

USER GUIDE

# Essential Studio for Xamarin

---

Version - v17.3.0.14 | Release Date - October 3, 2019

Welcome to Syncfusion Essential Studio for Xamarin.Forms .....	87
How to best read this user guide .....	87
Additional help resources .....	87
Create a support incident .....	87
System Requirements .....	87
Hardware Environment.....	87
Development Environment.....	87
Supported Platforms.....	87
Target Framework (Compile SDK Version) for Android .....	88
Download Installer .....	88
Download the Trial Version .....	88
Free Trial Page.....	88
Start Trial Page .....	89
Download the License Version.....	89
Installation and Upgrade.....	90
Installation using Web Installer .....	90
Installation using Offline Installer .....	99
Installing with UI .....	99
Installing in silent mode .....	105
Explore the libraries package .....	107
Mac Installation .....	107
Installation FAQ.....	116
Upgrade from one version to another version. ....	116
Upgrade from major version to service pack version .....	116
Upgrade from trial version to license version.....	116
Visual Studio Integration.....	117
Overview .....	117
Visual Studio Extensions .....	118
Create project .....	118
Toolbox control .....	123
Essential UI Kit.....	126
Check for Updates.....	130
Visual Studio Mac Extensions .....	131
NuGet Packages .....	139
Installing NuGet Packages.....	140



Explore Libraries Package .....	141
Troubleshooting.....	141
Themes.....	141
Overriding the default theme .....	145
Creating your own theme .....	149
Keys of Syncfusion Controls .....	149
SfChart.....	149
SfDateTimeRangeNavigator .....	152
SfKanban .....	153
SfListView .....	155
SfLinearProgressBar .....	156
SfCircularProgressBar.....	156
SfNumericTextBox.....	156
SfNumericUpDown .....	157
SfDiagram.....	158
SfMaskedEdit .....	158
SfTreeView .....	159
SfTextInputLayout.....	160
SfBackdropPage .....	162
SfCardView .....	162
SfPdfViewer.....	163
SfDataGrid.....	166
SfPopupLayout .....	168
SfPullToRefresh.....	170
SfDataForm .....	170
SfSchedule.....	171
SfCalendar .....	178
SfMaps .....	180
SfCircularGauge.....	181
SfLinearGauge .....	182
SfDigitalGauge.....	182
SfSunburstChart.....	183
SfSparkline .....	183
SfTreeMap.....	184
SfImageEditor.....	185

SfAutoComplete .....	185
SfBusyIndicator .....	186
SfButton .....	186
SfSwitch.....	187
SfBorder .....	188
SfCarousel .....	189
SfCheckBox.....	189
SfChip .....	189
SfChipGroup .....	190
SfComboBox.....	190
SfNavigationDrawer .....	192
SfPicker.....	192
SfRadioButton .....	193
SfRadialMenu .....	194
SfRangeSlider .....	194
SfRating .....	195
SfRotator .....	195
SfSegmentedControl .....	196
SfTabView .....	197
Switching between light theme and dark theme.....	197
UI Kit.....	198
Syncfusion XAML Pages for Xamarin.Forms .....	198
Supported Platforms.....	198
Getting started .....	198
Essential UI Kit for Xamarin.Forms Extension.....	198
How to render the added page.....	201
Requesting Screens and Reporting Bugs.....	201
Release Notes .....	201
UI Kit 1.0.0.0.....	201
UI Kit 2.0.0.0.....	201
SfAccordion .....	202
Getting started .....	202
Assembly deployment.....	202
Adding SfAccordion reference .....	202
Launching the accordion on each platform .....	204

Creating the Accordion .....	205
Animation duration .....	211
Animation easing .....	211
Auto scroll position .....	211
Expand mode .....	211
Item spacing .....	212
Appearance .....	212
Header icon position .....	212
Header background color customization .....	212
Icon color customization .....	213
Visual State Manager .....	213
Bindable Layout .....	216
Creating Data Model .....	216
Binding data to SfAccordion .....	217
Defining the AccordionItem .....	217
SfAutoComplete .....	220
Overview .....	220
Key Features .....	220
Getting Started .....	221
Adding SfAutoComplete reference .....	221
Initializing AutoComplete .....	224
Populating AutoComplete with Data .....	225
Configuring filter options .....	227
Populating Data .....	229
Populating String Data .....	229
Populating Business Objects .....	230
DataTemplateSelector .....	234
OnSelectTemplate .....	237
AutoComplete Modes .....	238
Suggesting Choices in List .....	238
Appending Suggestion to Text .....	240
Suggesting Choices and Appending Suggestions to Text .....	241
Multiple Selection .....	243
Token Representation .....	243
Delimiter .....	251

AutoComplete Filtering Options .....	257
Filtering Words that Starts with Input Text .....	257
Filtering Words that Contains the Input Text .....	260
Filtering Words that Equals the Input Text .....	263
Filtering Words that Ends with the Input Text.....	269
Diacritic Sensitivity .....	272
Header and Footer .....	274
Header Content.....	274
Footer Content.....	276
Highlighting matched text.....	279
First Occurrence .....	279
Multiple Occurrence .....	281
Maximum Display Item with Expander .....	282
No Results Found .....	285
Customizing NoResultsFoundText .....	286
Handling Selected Items .....	288
SelectedIndex.....	288
SelectedIndices .....	291
SelectedItem .....	294
Dealing with Suggestion Box.....	297
Suggestion box placement mode.....	297
Maximum suggestion box height.....	299
Opening suggestion box on focus .....	301
Delay opening suggestion box .....	303
Delay before searching algorithm starts.....	304
Avoid opening suggestion box .....	306
Prefix Characters Constraint .....	307
Watermark .....	309
Changing Watermark Text Color.....	310
Focus the control .....	312
Customizing AutoComplete .....	313
Customizing the Entry.....	314
Custom Template for Suggestion Items.....	315
Customizing the Suggestion Box.....	317
How to perform an operation when selecting an item from drop-down list? .....	329

AutomationId .....	333
SfAvatarView .....	334
Overview .....	334
Key features .....	335
Getting Started.....	335
Adding SfAvatarView reference .....	335
Launching an application on each platform with SfAvatarView .....	336
Creating an SfAvatarView control.....	337
Avatar Types.....	339
Default.....	339
Initials.....	340
Custom image .....	342
Character avatars .....	342
GroupView .....	343
Add initials only in GroupView.....	345
Add both image and initials in a GroupView .....	346
Add GroupView using MVVM .....	347
Visual Style .....	350
Custom .....	350
Circle .....	350
Square .....	354
Customization .....	358
Colors .....	358
Sizing .....	362
How to.....	364
Set badge view to avatar.....	364
SfBackdropPage .....	366
Overview .....	366
Key features .....	366
Getting Started.....	367
Adding SfBackdropPage reference .....	367
Launch an application on each platform with backdrop page .....	369
Initialize backdrop page .....	370
Add back layer content .....	370
Add front layer content.....	371

Reveal and conceal the back layer .....	372
Corner Shape Customization .....	373
Reveal Height Customization .....	374
Events.....	377
BackLayerStateChanged event .....	377
Header Configuration.....	377
Icon customization .....	377
SfBadgeView .....	379
Overview .....	379
Key features .....	379
Getting Started.....	380
Adding SfBadgeView reference .....	380
Launching the application on each platform with badge view .....	381
Adding namespace.....	382
Initializing badge view.....	382
Adding badge text .....	383
Adding content.....	383
Badge Settings.....	385
Font customization .....	385
Stroke customization .....	386
Text customization.....	387
Predefined styles.....	388
Badge background customization.....	389
Setting corner radius of the badge .....	390
Alignment of badge.....	391
Position customization.....	392
Setting badge offset .....	392
Predefined symbols.....	393
Animation.....	395
SfBarcode .....	396
OVERVIEW.....	396
Getting Started.....	397
Adding SfBarcode reference .....	398
Configure the Barcode control.....	400
SUPPORTED SYMBOLOGIES .....	402

One Dimensional Barcodes .....	402
Two Dimensional Barcodes .....	412
SYMBOLLOGY SETTINGS .....	414
One Dimensional Barcode settings .....	414
Two Dimensional Barcode Settings.....	415
BARCODE CUSTOMIZATION .....	419
Bar Customization .....	419
Text Customization .....	420
SfBorder .....	421
Overview .....	421
Key Features.....	421
Getting Started.....	421
Adding SfBorder reference .....	421
Launching an application on each platform with SfBorder. ....	423
Creating SfBorder control .....	424
Customization .....	425
Border color .....	426
Background color .....	426
Border width .....	427
Corner radius.....	427
Adding circular image .....	428
Shadow Effect .....	429
SfBusyIndicator .....	430
Overview .....	430
Key Features.....	431
Getting Started.....	431
Adding SfBusyIndicator reference .....	431
Launching the SfBusyIndicator on each platform .....	432
Create a Simple SfBusyIndicator .....	433
Setting Animation Type.....	434
EnableAnimation.....	435
Make Busy Animation Idle .....	436
Set Header.....	437
Title .....	437
TextColor .....	438

TextSize .....	439
TitlePlacement .....	440
Sizing .....	441
Animation Type.....	442
Animation duration.....	446
SfButton .....	447
Overview .....	447
Key features .....	447
Getting Started.....	448
Adding SfButton reference .....	448
Creating a simple SfButton .....	451
Setting caption .....	452
Toggle button.....	452
Button icon.....	452
Button background image.....	453
Visual States .....	453
Customization .....	455
Text Customization .....	455
Background Customization .....	456
Image Customization .....	458
Gradient background .....	459
Command.....	461
Shadow Effect .....	462
How to.....	463
Add the custom view for button.....	463
Cards .....	464
Overview .....	464
Key features .....	464
Getting Started.....	465
Adding cards reference .....	465
Launching the application on each platform with cards.....	467
Initialize cards .....	468
SwipeToDismiss.....	470
Customization in CardLayout .....	474
Customization in CardView .....	480



Initialize view model .....	482
Populate CardLayout with data .....	483
Define the appearance of SfCardView .....	483
Events.....	483
CardTapped .....	483
VisibleCardIndexChanging .....	484
VisibleCardIndexChanged .....	485
Calculate.....	486
Overview .....	486
Getting Started.....	486
Adding Calculate reference .....	487
Compute formula using CalcQuickBase .....	487
Compute formula using ICalcData .....	488
Choosing between CalcQuickBase and ICalcData .....	489
Cross Sheet Reference .....	489
Compute formulas in different region settings .....	490
Parse and Compute.....	490
Parsing.....	490
Computation .....	491
Error Messages.....	493
Formatting the Computed Results.....	494
Working with CalcQuickBase .....	495
Compute using values .....	495
Compute using Variables .....	495
Automatic calculations.....	496
Reset keys .....	497
Working with ICalcData .....	497
Methods and Events in ICalcData .....	497
Computation using ICalcData.....	499
Working with CalcEngine .....	500
Computation using CalcEngine .....	501
Data Objects maintained in CalcEngine .....	501
Properties in CalcEngine .....	502
Methods in CalcEngine.....	506
Working with Essential XlsIO .....	509

Open a Workbook using XlsIO .....	509
Enable and Disable Calculations in XlsIO .....	510
Set and Compute the values at runtime in the Worksheet .....	510
To compute particular cell in the worksheet .....	511
Ambiguity Issue .....	511
Table Formulas .....	511
Supported Formulas .....	512
Database Functions .....	512
Date and Time Functions .....	513
Engineering Functions .....	515
Financial Functions .....	517
Information Functions .....	519
Logical Functions .....	520
Lookup & Reference Functions .....	520
Math & Trigonometry functions .....	521
Statistical functions .....	525
Text Functions .....	532
Web Functions .....	533
Operators .....	534
Arithmetic Operators .....	534
Logical Operators .....	534
Text concatenation operator (Binary literal operator) .....	534
Operator precedence .....	535
Equal Sign, the Formula Character .....	535
Square Brackets, indexers in CalcQuickBase class .....	535
Named Ranges Support in Essential Calculate .....	536
Syntax to define a name .....	536
Add Named Ranges .....	536
Remove Named Ranges .....	536
Manage Named Ranges .....	536
Custom Function .....	537
LibraryFunction .....	537
Add Custom Function .....	537
Remove Custom Function .....	538
Replace Function .....	538

Performance and Limitations.....	539
To improve the performance .....	539
To avoid stack overflow exception .....	540
Limitations.....	541
SfCalendar .....	542
Overview .....	542
Key Features.....	542
Getting Started.....	542
Adding SfCalendar reference .....	543
Launching the SfCalendar on each platform.....	544
Create a Simple SfCalendar .....	545
Set Blackout Dates .....	546
Enable Multiple Selection .....	546
Restrict Dates .....	546
Calendar views .....	547
Month view .....	547
Trailing and leading days.....	548
Enable and disable past dates.....	549
Month view customization .....	550
Week view.....	554
Year view .....	555
Year view mode.....	556
Year view customization .....	557
Decade view .....	558
Decade view customization .....	559
Century view .....	560
Century view customization.....	561
Date Navigation .....	562
Programmatic Navigation .....	562
Events (Appointments) .....	564
Month Appointment Display .....	565
Month Appointment Indicator.....	566
Customize inline/agenda view using DataTemplate.....	566
Customize inline/agenda view using Template Selector .....	568
Getting inline/agenda view appointment details .....	570

Select Multiple Dates .....	572
Multi selection mode .....	572
Range selection mode.....	573
Multi range selection mode .....	574
Single selection mode .....	575
Programmatically clear the selected dates.....	576
Restrict Dates From Selection .....	576
Range of Min / Max Dates .....	576
Blackout Dates .....	577
Binding Properties in MVVM Pattern .....	580
Binding SelectedDate .....	580
Binding SelectedDates .....	581
Binding SelectedRange.....	581
Commands .....	583
Change First Day Of Week .....	586
Localization .....	586
Change default control language .....	587
Localizing custom strings from PCL.....	587
Localizing the custom texts using platform renderer .....	588
Right to left(RTL) .....	590
Customization of Calendar control .....	592
How to Perform an Operation while a Calendar Cell is Tapped? .....	592
How to Perform an Operation when the Selected Date Get Changed? .....	592
How to Perform an Operation when Navigate to Next Month? .....	593
How to Perform an Operation while Dealing with Appointments?.....	593
How to Customize Cell or Month View? .....	594
Create your own custom calendar month cell view .....	595
How to customize month view cell using a template? .....	596
How to Perform the Operation while long pressing the dateCell?.....	598
How to Resize the SfCalendar Control? .....	599
How to Customize the SfCalendar Header?.....	599
How to enable or disable the YearView in SfCalendar? .....	600
How to enable or disable the Horizontal and Vertical cell grid lines in SfCalendar?.....	600
Customize the year cell or year view .....	601
Customize the year view with custom UI .....	602

Deselect today selection on initial load .....	603
Feature Comparison across the different platforms in Xamarin. ....	603
Theming .....	607
Default theme (light theme) .....	608
Customizing the default theme .....	609
AutomationId .....	611
SfCarousel .....	615
Overview .....	615
Key Features:.....	616
Getting Started.....	616
Adding SfCarousel reference .....	616
Launching the SfCarousel on each platform .....	617
Create a Simple SfCarousel .....	618
Add Carousel Items .....	619
Setting the height and width of the carousel item .....	623
Set Desire Item to be Selected.....	624
Linear Arrangement .....	626
Populating Data.....	628
Through Binding.....	629
Through Carousel Item.....	631
LoadMore.....	633
AllowLoadMore.....	633
LoadMoreItemsCount .....	635
LoadMoreView .....	637
UI Virtualization .....	640
EnableVirtualization.....	640
Transformation .....	642
Tilt Non Selected Items .....	642
Set Gap between Items.....	643
Set Gap between Selected and unselected Item .....	644
Set Scaling for Carousel Items.....	645
Spacing between the Items in Linear mode.....	647
Animation.....	649
How to perform an operation while changing the carouselItem? .....	651
AutomationId .....	653

SfChart.....	654
Overview .....	654
Key features .....	655
Getting Started.....	655
Adding SfChart reference.....	655
Launching the application on each platform with chart.....	657
Initialize Chart .....	659
Initialize view model .....	660
Populate Chart with data .....	661
Add Title .....	662
Enable data labels .....	662
Enable legend.....	663
Enable tooltip.....	663
Populating Data.....	666
Chart Data Point.....	666
Custom Object .....	667
Title .....	668
Text Alignment.....	669
Axis.....	670
Category Axis.....	670
Numeric Axis .....	673
Date Time Axis .....	678
Date-time category axis .....	683
Logarithmic axis .....	684
Common axis features .....	687
Axis Crossing .....	704
Smart Axis Labels .....	706
Event .....	707
Chart Series .....	708
Multiple Series .....	708
Combination Series .....	710
Grouping Stacked Series .....	712
Animation.....	714
Transpose the Series (Vertical Chart) .....	714
Methods.....	715

Chart Types .....	715
Line Chart .....	715
Fast Line Chart.....	716
Area Chart .....	718
Spline Area Chart .....	719
Step Area Chart.....	721
Range Area Chart .....	722
Spline Range Area Chart .....	723
Stacked Area Chart.....	726
100% Stacked Area Chart.....	727
Column Chart .....	728
Histogram Chart.....	730
Range Column Chart .....	731
Stacked Column Chart.....	733
100% Stacked Column Chart.....	735
Bar Chart .....	736
Stacked Bar Chart.....	738
100% Stacked Bar Chart.....	739
Spline Chart.....	741
Step Line Chart.....	744
Bubble Chart .....	745
Scatter Chart .....	747
Fast Scatter Chart.....	748
OHLC Chart.....	749
Candle Chart.....	752
Radar Chart .....	755
Polar Chart .....	761
Pie Chart.....	767
Doughnut Chart.....	772
Pyramid Chart .....	779
Funnel Chart.....	783
Waterfall Chart.....	787
Data Markers .....	788
Customizing labels .....	789
Formatting label content .....	790

Label position .....	791
Smart labels.....	794
Customizing marker shapes .....	795
Apply series color .....	796
Connector line.....	797
Label template .....	798
Event .....	800
Legend.....	800
Customizing labels .....	801
Legend icons.....	802
Legend title .....	803
Toggle the series visibility .....	804
Legend visibility.....	804
Legend item visibility .....	805
Legend wrap.....	806
Positioning the legend .....	808
Populate the data based legend items for all series.....	809
ItemTemplate.....	809
Event .....	811
Color Palette .....	812
Apply palette for Chart .....	812
Apply palette for Series.....	813
Gradient Colors .....	816
Plotting Area Customization .....	819
Zooming and Panning .....	819
Enable Zooming .....	819
Zoom Mode.....	821
Auto Interval On Zooming.....	822
Events.....	822
Methods.....	824
Selection.....	826
Data Point Selection.....	826
Series Selection .....	828
Events.....	829
Methods.....	830



Trackball.....	831
Label Display Mode .....	832
Activation mode .....	833
Customizing appearance.....	833
Show/hide the trackball label in axis .....	836
Axis label alignment .....	837
Show/hide the series label.....	837
Label Template.....	838
Methods .....	841
Events.....	842
Tooltip .....	842
Customizing appearance .....	843
Tooltip Template .....	844
Methods .....	846
Strip Lines.....	847
What is strip line? .....	847
How to add strip lines? .....	847
Strip Line Recurrence .....	850
Customize Text.....	851
Segmented StripLine .....	853
Performance .....	856
Annotation .....	858
Adding Annotations .....	858
Positioning the annotation .....	859
Adding annotation for multiple axes .....	861
Text annotation.....	862
Shape annotation .....	865
View annotation.....	875
Event .....	876
Get the touch position in annotation .....	876
Technical Indicators .....	877
Adding technical indicators to chart .....	877
Indicator visibility .....	880
Average true range indicator .....	881
Simple moving average indicator.....	882

Relative strength index indicator .....	883
Accumulation distribution indicator .....	884
Momentum indicator .....	886
Stochastic indicator .....	887
Exponential moving average indicator .....	888
Triangular moving average indicator .....	889
Bollinger band indicator .....	890
MACD indicator .....	892
Exporting .....	893
Export as an image .....	893
Get the stream of Chart .....	894
GetStream .....	894
GetStreamAsync .....	894
How to .....	894
Transform axis value to pixel value and vice-versa .....	894
Get the touch position in chart .....	895
Get the data point collection based on region .....	895
Adding duplicate axis in SfChart .....	895
Customize the SfChart padding .....	896
Adding custom labels to SfChart axis .....	897
Localization .....	898
Localize at application level .....	899
SfCheckBox .....	900
Overview .....	900
Key features .....	900
Getting Started .....	901
Adding SfCheckBox reference .....	901
Create a Simple SfCheckBox .....	904
Setting caption .....	904
Change the check box state .....	905
Indeterminate .....	907
Visual Customization .....	909
Customizing shape .....	909
Customizing state color .....	910
Setting caption text appearance .....	911

LineBreakMode .....	912
TickColor Customization .....	912
Visual States .....	913
Event .....	914
StateChanged event.....	914
Chips.....	916
Overview .....	916
Key Features.....	916
Getting started .....	916
Adding Chips reference.....	917
Launching an application on each platform with chips .....	918
Initialize chips.....	920
Set layout for the control.....	921
Populating business objects.....	922
Set types of chip group .....	924
Populating chips.....	927
Populating business objects as items .....	927
Populating SfChip as items.....	927
Set the type for chip group .....	928
Input.....	928
Choice.....	928
Filter .....	928
Action .....	928
Customization of SfChip.....	930
ShowCloseButton .....	930
ShowSelectionIndicator .....	931
CloseButtonColor .....	933
SelectionIndicatorColor .....	934
BackgroundColor.....	935
BorderColor .....	936
BorderWidth .....	937
CornerRadius.....	938
FontAttributes.....	939
FontFamily.....	941
FontSize.....	942

TextColor .....	943
TextAlignment.....	944
ShowIcon.....	946
ImageSource .....	947
ImageWidth.....	948
ImageAlignment.....	949
Command.....	950
Customization of SfChipGroup.....	952
InputView .....	952
SelectedChipBackgroundColor.....	953
SelectedChipTextColor.....	955
ChipBackgroundColor .....	956
ChipBorderColor .....	957
ChipTextColor.....	958
ChipTextSize .....	960
ChipPadding .....	961
ChipBorderWidth .....	963
ItemHeight .....	964
ShowIcon.....	965
CloseButtonColor .....	968
SelectionIndicatorColor .....	969
ChipImageWidth .....	971
SfCircularGauge.....	973
Overview .....	973
Key features .....	973
Getting Started.....	975
Adding SfCircularGauge reference.....	975
Launching an application on each platform with SfCircularGauge.....	976
Reference Mono.Android.Export.....	977
Initialize gauge .....	978
Adding header.....	979
Configuring scales .....	979
Adding ranges .....	979
Adding a needle pointer.....	980
Adding a range pointer .....	980

Adding a marker pointer .....	981
Scales.....	983
Scale .....	983
Setting start and end values for scale .....	984
Setting start and sweep angles for scale .....	985
Setting interval for scale .....	985
Setting auto interval for scale .....	986
Setting scale direction for scale .....	987
Setting maximum labels.....	987
Setting multiple scales for scale.....	988
Events.....	990
Rim .....	991
Rim customization.....	992
Setting position for rim .....	993
Show rim .....	994
Tick Setting.....	995
Show ticks for scale.....	995
Ticks customization.....	995
Customize major ticks for scale .....	995
Customize minor ticks for scale .....	996
Setting position for ticks .....	997
Labels .....	999
Label color customization .....	999
Label font customization.....	999
Setting position for labels .....	1000
Setting number of decimal digits for labels .....	1000
Setting postfix and prefix for labels .....	1001
Edge label customization .....	1002
Show labels .....	1003
Setting auto angle for label.....	1004
Custom labels for circular gauge.....	1004
Ranges.....	1005
Setting start and end values for range.....	1005
Range customization.....	1006
Setting position for range .....	1007

Setting range color for labels .....	1009
Setting multiple ranges .....	1010
Setting gradient color for range .....	1012
Pointers .....	1014
Needle pointer .....	1014
Range pointer .....	1018
Marker pointer .....	1022
Header .....	1029
Adding header in circular gauge .....	1029
Setting position for header .....	1030
Customization of header .....	1030
Alignment of header .....	1031
Annotations .....	1032
Setting view annotation .....	1033
Setting image annotation .....	1039
Setting text annotation .....	1041
Set margin to annotation .....	1043
Alignment of annotation .....	1044
How to .....	1047
Changing the gauge size .....	1047
Gauge alignment .....	1048
SfComboBox .....	1049
Overview .....	1049
Key features .....	1050
Getting Started .....	1051
Adding SfComboBox reference .....	1051
Initializing ComboBox .....	1053
Populating ComboBox with data .....	1054
ComboBox modes .....	1056
Retrieving selected values .....	1059
Populating data .....	1061
Populating string data .....	1061
Populating business objects .....	1062
DataTemplateSelector .....	1066
OnSelectTemplate .....	1068

Handling Selected Items .....	1069
SelectedIndex.....	1069
SelectedIndices .....	1072
SelectedItem .....	1075
Diacritic Sensitivity.....	1078
Multiple selection .....	1080
Token representation .....	1080
Delimiter .....	1085
No Results Found .....	1087
Customizing NoResultsFoundText .....	1089
Highlighting matched text.....	1090
First occurrence .....	1090
Multiple occurrence.....	1091
Header and Footer .....	1092
Header content .....	1092
Footer Content.....	1094
ComboBox modes .....	1095
Editable combo box .....	1095
Non-Editable combo box .....	1096
Dealing with suggestion box .....	1097
Suggestion box placement mode.....	1097
Maximum suggestion box height.....	1099
Opening suggestion box on focus .....	1100
Delay opening suggestion box .....	1101
Delay before searching algorithm starts.....	1102
Avoid opening suggestion box .....	1103
Customizing ComboBox .....	1103
Customizing the entry .....	1103
CustomView for ComboBox.....	1107
Custom template for suggestion items.....	1108
Customizing the suggestion box .....	1110
DropDown button customization .....	1119
View for drop down button .....	1120
Watermark.....	1121
Changing Watermark Text Color.....	1122

Filtering .....	1122
Filtering types.....	1124
ComboBox modes .....	1133
Suggesting choices in List.....	1134
Appending suggestions to text .....	1135
Suggesting choices and appending suggestions to text.....	1136
AutomationId .....	1137
SfDataForm .....	1138
Getting started .....	1138
Assembly deployment.....	1138
Adding SfDataForm reference .....	1138
Launching the data form on each platform .....	1140
Creating the data form.....	1141
Setting data object.....	1144
Defining editors.....	1145
Layout options .....	1147
Loading DataForm inside StackLayout.....	1149
Loading DataForm with customized height and width.....	1152
Editing .....	1154
Working with the data form .....	1154
Auto-generating DataFormItems for the data field.....	1154
Customize auto generated fields .....	1155
Cancel DataFormItem generation of the data field.....	1155
Changing editor type.....	1156
Changing property settings.....	1156
Setting watermark .....	1157
Changing DataFormItem .....	1158
Adding or removing the data field displayed in the dataForm at runtime.....	1158
DataFormItemManager .....	1165
Binding with dynamic data object .....	1167
Adding custom DataFormItems .....	1170
Editing .....	1175
Supported editors and associated DataFormItem.....	1175
Changing editor for type .....	1177
Changing editor for property .....	1177



Customizing existing editor .....	1178
Creating new custom editor .....	1180
Support for Email editor .....	1183
Commit mode .....	1185
Update editor value based on another editor .....	1185
Converter .....	1186
Disable editing.....	1188
Two-way data binding.....	1188
Editors .....	1189
Text editor .....	1192
Multiline Text editor .....	1192
Numeric editor .....	1194
Date editor .....	1194
Time editor.....	1200
Segment editor .....	1204
CheckBox editor .....	1207
Switch Editor .....	1215
Drop down editor.....	1217
Picker editor .....	1223
NumericUpDown editor.....	1228
Password editor .....	1236
RadioGroup editor .....	1238
MaskedEditText editor.....	1241
AutoComplete editor .....	1244
Custom editor .....	1261
Validation .....	1263
Built in validations.....	1263
Validation mode.....	1267
Valid or positive message .....	1269
How to validate the property value based on another value .....	1272
Customize validation message using DataTemplate .....	1272
Customize validation message using DataTemplateSelector .....	1273
Layout.....	1275
Overview .....	1275
Linear layout support.....	1275

Grid layout support .....	1279
Label visibility .....	1283
Label position .....	1285
Loading images for label .....	1288
Changing order of the DataFormItem.....	1290
Grouping data fields.....	1292
Customizing DataFormLayoutManager .....	1307
Label width customization .....	1313
Spanning rows and columns .....	1315
Change DataFormItem visibility at runtime .....	1319
Change DataFormGroupItem visibility at runtime.....	1319
Programmatically scroll to specific editor.....	1320
Changing the height of DataFormItem. ....	1321
Floating label layout.....	1322
Layout options .....	1323
Supported editors .....	1324
Container types.....	1325
Leading view .....	1327
Trailing view .....	1329
Enable password visibility toggle for password editor .....	1330
Assistive label.....	1331
Appearance customization .....	1335
Customize the background color of editor view.....	1339
Font customization .....	1340
Limitations.....	1342
Unsupported editors.....	1342
Data Annotations .....	1342
Display attribute.....	1342
Validation attributes .....	1343
Bindable attribute .....	1343
Editable attribute .....	1343
ReadOnly attribute.....	1343
EnumDataType attribute .....	1343
DataType attribute.....	1343
DisplayFormat attribute.....	1344

CustomDataType attribute .....	1344
Custom attribute .....	1344
Localization .....	1345
Localizing data form item display values .....	1345
Localizing validation error messages .....	1346
Right to left (RTL) .....	1348
AutomationId .....	1350
SfDataGrid .....	1351
SfDataGrid .....	1351
Getting Started.....	1354
Assembly deployment.....	1354
NuGet configuration .....	1354
Adding SfDataGrid reference .....	1354
Launching the SfDataGrid on each platform.....	1356
Create a simple SfDataGrid .....	1357
Creating the project .....	1358
Adding SfDataGrid in Xamarin.Forms .....	1358
Create DataModel for the SfDataGrid .....	1359
Binding data to the SfDataGrid .....	1360
Defining columns .....	1362
Sorting .....	1362
Grouping .....	1363
Selection.....	1364
Launching the SfDataGrid inside a StackLayout.....	1364
Loading the SfDataGrid with customized height and width .....	1366
Linker issue in Xamarin.Forms.iOS .....	1367
Data Binding.....	1368
Binding with IEnumerable.....	1369
Binding with DataTable .....	1369
Binding complex properties .....	1370
View .....	1370
Columns .....	1377
Automatic columns generation.....	1377
Manually generate columns .....	1381
Resizing columns .....	1381

Column Types.....	1386
GridColumn .....	1386
GridTextColumn .....	1390
GridSwitchColumn .....	1393
GridImageColumn .....	1395
GridTemplateColumn.....	1398
CellTemplateSelector .....	1400
GridDateTimeColumn .....	1404
GridPickerColumn .....	1406
GridComboBoxColumn .....	1415
GridNumericColumn .....	1426
Row header .....	1428
Bind a view model property inside header template? .....	1428
ColumnSizer .....	1430
ColumnSizer.None.....	1431
ColumnSizer.LastColumnFill.....	1432
ColumnSizer.Star .....	1433
ColumnSizer.Auto .....	1434
How to.....	1435
Star column sizer ratio support.....	1441
Freeze Panes .....	1443
Freeze rows .....	1444
Freeze columns .....	1444
Grid Events.....	1445
GridTapped event .....	1445
GridDoubleTapped event.....	1446
GridLongPressed event .....	1446
GridViewCreated event.....	1447
GridLoaded event.....	1447
ValueChanged event .....	1448
ItemsSource changed event.....	1448
Create custom context menu using grid events .....	1449
Commands .....	1452
Sorting.....	1455
Programmatic sorting .....	1455

Tri-State sorting .....	1456
Multi-column sorting .....	1457
Sort column in double click.....	1458
Sorting events .....	1458
Custom sorting.....	1459
Animating sorting icon .....	1461
How to disable sorting .....	1462
Grouping .....	1463
Programmatic grouping .....	1463
MultiGrouping.....	1464
Indent column customizations.....	1465
Custom grouping.....	1467
Expand groups while grouping.....	1469
Expand or collapse the groups.....	1469
Display based grouping using GroupMode property.....	1470
Clearing or removing a group .....	1471
Events.....	1474
Animate group expand/collapse icon .....	1476
How to hide the grouped column in SfDataGrid.....	1476
Summary .....	1476
Caption summaries .....	1477
Group summary .....	1486
Table summaries .....	1494
Table summary template .....	1501
Formatting summary .....	1508
Aggregate types .....	1512
Custom summaries .....	1513
Overriding summary renderer .....	1515
Filtering .....	1521
View Filtering .....	1521
DataTable filtering .....	1526
Filter individual columns.....	1526
Filter based on conditions.....	1528
Clear filtering.....	1535
Editing .....	1536

Column editing.....	1536
Entering into edit mode.....	1537
Cursor behavior.....	1537
Support for IEditableObject.....	1538
Editing events.....	1541
Programmatically edit a cell.....	1542
How to.....	1543
Selection.....	1544
Selection modes.....	1544
Getting selected rows.....	1546
Programmatic selection.....	1547
Keyboard behavior.....	1549
Clear selection.....	1551
Row header selection.....	1551
Selection animation.....	1552
Events in selection.....	1553
Customizing Selection Appearance.....	1555
Binding selection properties.....	1560
Stacked Headers.....	1562
Adding child columns.....	1565
Removing child columns.....	1565
Changing stacked header row height.....	1565
Appearance.....	1566
Loading template in stacked column.....	1570
Load More.....	1573
Load more command.....	1573
Customize load more display text.....	1575
Customize load more view position.....	1575
Customize LoadMoreView.....	1576
Pull To Refresh.....	1577
Pull to refresh command.....	1577
Host the data grid inside pull-to-refresh.....	1580
Column Drag And Drop.....	1581
Column drag and drop events.....	1582
Cancel dragging of a particular column.....	1582

Cancel dropping when dragging for a particular column .....	1583
Cancel dropping for a particular column .....	1583
Cancel dropping at a particular position.....	1583
Cancel dropping of a particular column in a position .....	1584
Cancel drag and drop between frozen and non-frozen columns .....	1584
Customize column drag and drop indicators .....	1585
Cancel auto scrolling .....	1586
Row Drag and Drop.....	1587
Dragging scenarios .....	1587
Row drag-and-drop template .....	1588
Default template .....	1588
Customizing row drag-and-drop template .....	1589
Events in row drag-and-drop .....	1591
Cancel dragging of a particular row .....	1592
Cancel dropping when dragging over particular rows.....	1592
Cancel dropping of a particular row .....	1592
Cancel dropping at a particular position.....	1593
Cancel dropping of a particular row in a position.....	1593
Cancel drag and drop between frozen and non-frozen rows .....	1593
Reorder the underlying data.....	1594
Drop a grid row in the last position .....	1595
Customizing row drag-and-drop indicators .....	1595
Updating summaries when dragging and dropping a row between groups .....	1596
Cancel auto scrolling .....	1597
Swiping.....	1598
Swipe template .....	1598
Swipe events .....	1600
Loading multiple views as swipe template .....	1600
Customized swipe delete functionality.....	1603
Loading complex template for swiping.....	1605
How to cancel the swipe programmatically .....	1607
Limitations.....	1607
Styles.....	1608
Applying alternate row style .....	1610
Customizing the alternation count .....	1611

Border customization.....	1612
Header border color customization .....	1616
Customizing sort icons in the header.....	1617
Customizing resizing indicator .....	1619
Border width customization .....	1619
Conditional Styles.....	1619
Styling cells using column CellStyle.....	1620
Styling cells using converter .....	1620
Styling cells using QueryCellStyle event.....	1621
How to style a particular column .....	1623
How to style a particular cell based on RowIndex and ColumnIndex.....	1624
How to style a particular cell based on CellValue.....	1625
Styling cells using RowStyle event .....	1626
How to style a particular row based on RowIndex .....	1627
How to style a particular row based on RowData .....	1628
Paging.....	1630
Normal paging.....	1630
OnDemandPaging .....	1633
AppearanceManager .....	1634
Row Height Customization.....	1638
Customize HeaderRowHeight .....	1638
Customize RowHeight for all rows .....	1638
GridRowSizingOptions.....	1639
Reset row height at runtime .....	1640
Auto fit the grid rows based on content.....	1640
Customize header row height based on header content .....	1641
Customize caption summary row height .....	1642
Change TableSummaryRow height.....	1643
How to optimize performance when using QueryRowHeight event.....	1644
Limitations.....	1645
Scrolling.....	1645
Scrolling mode .....	1645
Programmatic scrolling .....	1648
Diagonal scrolling .....	1652
Vertical Over Scroll Mode .....	1652



Identifying scroll state changes.....	1654
Scrolling customization using Slider.....	1655
Unbound Rows.....	1655
Positioning unbound rows .....	1656
Populating data for unbound rows.....	1657
Refreshing the Unbound Rows at runtime .....	1659
Editing in unbound rows .....	1660
Styling in Unbound rows .....	1661
Changing unbound row height .....	1662
Get unbound rows .....	1663
Unbound Column .....	1663
Populating data for the unbound column .....	1664
Export to Excel .....	1667
Exporting Options .....	1669
Customize Exporting Excel Version.....	1681
Exporting Unbound rows .....	1699
Exporting unbound columns.....	1699
Saving options.....	1705
Export AllPages in datagrid .....	1705
Row Height and Column Width customization.....	1709
Events.....	1713
Cell customization in Excel while exporting.....	1713
Customize exported workbook and worksheet.....	1720
Exporting the selected rows of SfDataGrid.....	1727
Export to PDF .....	1728
ExportToPdf .....	1729
ExportToPdfGrid.....	1730
Exporting options.....	1732
Export paging .....	1768
Setting header and footer.....	1772
Change PDF page orientation .....	1774
Saving options.....	1776
Row height and column width customization .....	1780
Events.....	1784
Exporting customization .....	1784

Exporting Unbound rows .....	1789
Cell customization in PDF while exporting.....	1795
Exporting the selected rows of SfDataGrid .....	1804
Localization .....	1804
Add a .resx file.....	1805
Convert the platform specific language format to .NET format .....	1806
Apply the converted format.....	1810
AutomationId .....	1813
DataGrid .....	1813
DataPager.....	1816
DataSource.....	1819
DataSource.....	1819
Overview .....	1819
Getting started .....	1819
Adding DataSource reference.....	1820
Creating your first DataSource in Xamarin.Forms .....	1820
Sorting.....	1822
Grouping .....	1823
Binding DataSource to a ListView .....	1823
Defining the LiveDataUpdateMode .....	1825
Defining the source data type.....	1825
SfDateTimeRangeNavigator .....	1825
Overview .....	1825
Key features .....	1825
Getting started .....	1826
Adding SfDateTimeRangeNavigator reference .....	1826
Launching the application on each platform with range navigator.....	1827
Adding and configuring SfDateTimeRangeNavigator .....	1828
Handle range selection .....	1830
Content .....	1831
Selecting Range .....	1833
Selected Data .....	1834
Overlay Color.....	1834
Deferred Update .....	1834
RangeChanged Event .....	1835

Thumb .....	1835
Grid Lines .....	1836
Tooltip .....	1837
Tooltip Visibility.....	1837
Tooltip Format .....	1838
Appearance Customization .....	1839
Major and Minor Scales .....	1840
Intervals .....	1840
Appearance Customization .....	1841
Scale visibility .....	1843
MinorScaleLabelsCreated event .....	1845
MajorScaleLabelsCreated event .....	1845
Localization .....	1846
Localize at application level .....	1847
SfDiagram.....	1847
Overview .....	1847
Key features .....	1848
Getting started .....	1848
Adding SfDiagram reference .....	1848
Basic building blocks of Diagram.....	1850
Creating a simple flow chart .....	1851
Create a simple organizational chart .....	1853
Diagram.....	1855
Page settings .....	1855
Stencil:.....	1855
Node template: .....	1856
Diagram constraints .....	1856
Diagram style settings.....	1857
Zooming enhancement .....	1857
Node.....	1858
Create node.....	1858
Create a node with custom path/shape .....	1859
Accessing a node from the diagram instance .....	1860
Remove a node .....	1860
Customization: .....	1860

Constraints .....	1860
Connector.....	1861
Create connector .....	1861
Connections with nodes.....	1861
Connections with ports.....	1862
Using port.....	1862
Segments.....	1864
Straight.....	1864
Orthogonal .....	1865
Curve .....	1866
Decorator .....	1867
Remove connector .....	1868
Appearance .....	1869
Connector appearance.....	1869
Decorator appearance .....	1870
Annotations.....	1872
Create annotation .....	1872
Accessing an annotation from node and connector instance .....	1874
Remove an annotation.....	1874
Annotation customization .....	1875
Alignment.....	1876
Port.....	1878
Create ports for a node.....	1878
Accessing a port from the node instance .....	1880
Remove a port from the node .....	1880
Customization .....	1881
Stencil.....	1881
Add default shapes into stencil.....	1882
Add custom shapes into stencil .....	1883
Add category heading text.....	1885
Layouts .....	1888
Create class for data .....	1888
Initialize data source settings .....	1889
Organization layout.....	1889
BeginInitLayout.....	1891

BeginNodeRender .....	1892
Expand and collapse node .....	1892
Drag-and-drop support for directed tree layout .....	1893
Layout sibling spacing .....	1896
Mind map layout.....	1897
Initializing the mind map layout .....	1900
Mind map layout style .....	1901
Branch wise node style .....	1901
Level wise node style .....	1903
Repeat mode.....	1905
Free form layout .....	1905
Drawing mode.....	1906
Text node .....	1906
Connector.....	1907
Gridlines .....	1907
Gridlines visibility .....	1907
Customizing gridlines .....	1908
Snapping gridlines .....	1908
User handles .....	1909
User handles clicked event .....	1910
Customizing user handle position.....	1911
OverviewPanel .....	1912
Create overview panel .....	1912
Customizing view port rect .....	1914
SfDigitalGauge.....	1914
Overview .....	1914
Key features .....	1914
Getting Started.....	1915
Adding SfDigitalGauge reference.....	1915
Launching an application on each platform with SfDigitalGauge.....	1916
Adding namespace for the assemblies .....	1917
Initialize gauge .....	1917
Setting value for digital gauge .....	1918
Setting character type for digital gauge.....	1918
Configuring properties .....	1918

Customize character segments .....	1920
customize character size .....	1920
Setting character spacing .....	1921
Customize character segment stroke .....	1922
Customize disabled segment .....	1923
Customize background color of digital gauge .....	1923
Character types .....	1924
Seven segment .....	1924
Fourteen segment .....	1925
Sixteen segment .....	1925
EightCrossEightDotMatrix segment .....	1925
Display value types .....	1926
Setting value to number .....	1926
Setting value to alphabet .....	1926
Setting value to special characters .....	1927
SfEffectsView .....	1928
Overview .....	1928
Key features .....	1928
Getting Started .....	1928
Adding SfEffectsView reference .....	1928
Launching an application on each platform with SfEffectsView .....	1930
Initializing SfEffectsView .....	1930
Effects .....	1931
Highlight .....	1931
Ripple .....	1933
Scale .....	1934
Selection .....	1934
Rotation .....	1935
Combinations .....	1936
Features in SfEffectsView .....	1938
FadeOutRipple .....	1938
IsSelected .....	1940
ShouldIgnoreTouches .....	1941
Interaction in SfEffectsView .....	1941
TouchDownEffects .....	1941

LongPressEffects .....	1942
TouchUpEffects .....	1942
Methods .....	1943
ApplyEffects .....	1943
Reset .....	1944
Customization of SfEffectsView .....	1945
RippleAnimationDuration .....	1945
ScaleAnimationDuration .....	1946
RotationAnimationDuration .....	1946
InitialRippleFactor .....	1946
ScaleFactor .....	1948
HighlightColor .....	1949
RippleColor .....	1951
SelectionColor .....	1952
CornerRadius .....	1953
Angle .....	1955
SfExpander .....	1956
Getting started .....	1956
Assembly deployment .....	1956
Adding SfExpander reference .....	1956
Launching the expander on each platform .....	1957
Creating the Expander .....	1958
Animation duration .....	1962
Animation easing .....	1962
Expand and Collapse .....	1962
Appearance .....	1962
Header icon position .....	1962
Header background color customization .....	1963
Icon color customization .....	1963
Visual State Manager .....	1963
SfGradientView .....	1965
Overview .....	1965
Key Features .....	1965
Getting Started .....	1966
Adding SfGradientView reference .....	1966

Launching an application on each platform with SfGradientView. ....	1967
Adding Gradient View .....	1968
Customization .....	1971
GradientStops .....	1971
SfLinearGradientBrush .....	1971
SfRadialGradientBrush .....	1974
SfImageEditor.....	1977
Overview .....	1977
Key features .....	1977
Getting Started.....	1977
Adding SfImageEditor reference.....	1977
Launching the application in iOS.....	1978
Initializing image editor.....	1979
Loading an image to image editor .....	1980
Text .....	1982
Customize text with TextSettings .....	1982
Custom font family.....	1983
Multiline text and text alignment .....	1987
Text Rotation.....	1988
Restricting the edit text box pop-up window .....	1989
Shapes .....	1989
Customizing a shape with pen settings.....	1990
PenSettings .....	1990
Deleting a shape or text from view.....	1992
Transformation .....	1992
Rotation.....	1992
Flip.....	1993
Crop.....	1993
Image cropping ratio .....	1993
Save .....	1995
Save events .....	1995
Saving Image with Custom Size and Format .....	1996
Reset .....	1997
Reset events.....	1997
ImageLoaded Event.....	1998



ItemSelected Event .....	1998
ItemUnselected Event.....	1999
ImageEdited Event .....	1999
Undo and Redo .....	1999
Undo.....	1999
Redo .....	2000
Toolbar customization .....	2000
Customize toolbar items .....	2001
Adding HeaderToolBarItem .....	2001
Adding FooterToolBarItem .....	2001
Adding SubItems to the FooterToolBarItem .....	2002
ToolBarItemSelected event.....	2002
To hide/show toolbar.....	2003
To hide/show the toolbar item.....	2004
To customize the ColorPalette.....	2005
Default Color Selected Index.....	2005
ToolBarHeight Customization .....	2006
Zooming .....	2008
Enable zooming.....	2008
Maximum zoom level.....	2008
Panning mode .....	2008
Serialization and Deserialization.....	2009
Serialization.....	2009
Deserialization.....	2010
Moving shapes to front and back .....	2011
BringToFront .....	2011
SendToBack.....	2012
BringForward .....	2012
SendBackward.....	2012
Localization .....	2013
Change default language .....	2013
Using Resx file from PCL.....	2013
From platform-specific projects.....	2015
CustomView .....	2018
CustomViewSettings .....	2018

CustomView Rotation .....	2019
Image Filter .....	2020
Hue .....	2020
Saturation.....	2021
Brightness.....	2022
Contrast.....	2023
Blur .....	2024
Sharpen .....	2025
Automation ID.....	2027
SfKanban .....	2032
Overview .....	2032
Key features .....	2032
Getting Started.....	2033
Adding SfKanban reference .....	2033
Launching an application on each platform with kanban.....	2034
Initialize Kanban .....	2035
Initialize view model .....	2036
Binding data to SfKanban.....	2037
Defining columns .....	2037
Column .....	2039
Customizing Column Size .....	2039
Categorizing Columns .....	2040
Headers .....	2040
Expand/Collapse Column .....	2042
Enable/Disable Drag & Drop .....	2042
Items Count.....	2043
Work In-Progress Limit.....	2044
Workflows .....	2045
Cards .....	2046
Template .....	2047
Data template selector .....	2049
Placeholder .....	2051
Localization .....	2053
Localize at application level .....	2054
Events.....	2055

ItemTapped .....	2055
DragStart .....	2056
DragEnd .....	2056
DragEnter .....	2056
DragLeave .....	2057
DragOver .....	2057
ColumnsGenerated .....	2057
SfLinearGauge .....	2057
Overview .....	2057
Key Features.....	2058
Getting Started.....	2058
Adding SfLinearGauge reference .....	2058
Launching an application on each platform with SfLinearGauge. ....	2060
Reference Mono.Android.Export.....	2060
Initialize gauge .....	2061
Adding header.....	2062
Configuring scales .....	2062
Adding a symbol pointer .....	2063
Adding a bar pointer .....	2063
Adding ranges .....	2063
Scales.....	2066
Linear scale.....	2066
Setting minimum and maximum values for scale.....	2067
Setting interval for scale .....	2068
Setting maximum labels.....	2068
Scale customization .....	2069
Scale Offset .....	2070
Setting opposite position .....	2071
Setting scale direction.....	2071
Setting corner radius type for scale .....	2072
Multiple scales .....	2073
Setting gradient color for scale .....	2075
Tick Setting.....	2077
Ticks visibility .....	2077
Tick customization .....	2077

Setting minor ticks per interval.....	2079
Setting position for ticks .....	2079
Labels .....	2080
Label color customization .....	2080
Label font customization.....	2081
Setting position for labels .....	2082
Custom labels.....	2084
Labels visibility .....	2085
Ranges.....	2086
Setting start and end values for range.....	2086
Range customization.....	2087
Setting position for range .....	2088
Setting multiple ranges .....	2089
Setting gradient color for range.....	2090
Pointers .....	2091
Adding bar pointer to scale.....	2091
Bar pointer customization .....	2092
Setting corner radius type for bar pointer .....	2093
Setting gradient color for bar pointer.....	2094
Adding symbol pointer to scale .....	2095
Symbol pointer customization .....	2096
Positioning symbol pointer .....	2097
Setting symbol pointer position.....	2097
Setting offset for symbol pointer .....	2098
Change symbol pointer shapes.....	2099
Setting image shape for symbol pointer .....	2101
Adding multiple pointers .....	2102
Change Orientation.....	2103
Header.....	2104
Adding header to linear gauge.....	2104
Positioning the header.....	2105
Customizing header text .....	2106
Annotations.....	2107
Annotation .....	2107
Positioning the annotation .....	2108

Set margin to the annotation.....	2111
Alignment of annotation.....	2112
Setting scale index for annotation .....	2115
Multiple annotations .....	2117
SfListView .....	2119
SfListView .....	2119
Key features .....	2119
Advantages of the SfListView over Xamarin.Forms ListView.....	2120
Getting Started.....	2121
Assembly Deployment .....	2121
Adding SfListView reference .....	2121
System Requirements .....	2123
Launching the SfListView on Each Platform.....	2123
Create a Simple SfListView.....	2125
Creating the Project .....	2126
Adding the SfListView in Xamarin.Forms .....	2126
Creating Data Model for the SfListView.....	2126
Binding Data to the SfListView.....	2128
Defining an ItemTemplate .....	2129
Layouts .....	2130
Sorting.....	2131
Filtering .....	2132
Grouping .....	2133
Selection.....	2133
Header and Footer .....	2134
Working with SfListView .....	2135
Interacting with ListView Items .....	2135
Improving ListView performance.....	2138
Loading ListView inside ScrollView .....	2139
Loading ListView inside CarouselPage/Master detail page .....	2139
Context menu on items.....	2140
Paging.....	2144
Loading data from SQLite online database.....	2146
Loading data from SQLite offline database.....	2149
ListView with Prism Framework.....	2152

Scrolling ListView without virtualization.....	2153
Working with nested ListView .....	2155
Rendering ListView when loading in different layouts .....	2156
How to.....	2156
Dispose listview.....	2157
View Appearance .....	2158
Item template .....	2158
Data template selector .....	2158
Horizontal ListView .....	2161
Horizontal list inside vertical list .....	2164
Item size .....	2167
Item spacing .....	2167
Alternate row styling.....	2168
Rounded corner on items .....	2172
Drop shadow effect on items.....	2175
ListViewItem customization.....	2178
Accordion view.....	2180
show busy indicator on list view .....	2184
show busy indicator on list view items .....	2188
Show busy indicator on list view items using toggle switch .....	2194
Item animation on appearing .....	2195
Scrolling.....	2199
Programmatic scrolling .....	2199
Scrollbar visibility .....	2200
Identifying scroll state changes.....	2200
Identify when end of the list is reached on scrolling .....	2200
Maintain the scroll position while updating ItemsSource at runtime .....	2201
Item Size Customization.....	2202
Customize item size of a particular item on-demand.....	2202
AutoFit the items based on the content .....	2204
Updating the listview item size based on font at runtime.....	2207
Updating the Header and Footer height based on font at runtime .....	2209
Load images with autofit mode .....	2211
Limitations.....	2212
Layouts .....	2212

Linear Layout.....	2212
Grid Layout.....	2213
Customize span count based on platform .....	2214
Change span count based on screen size .....	2215
Item Drag and Drop .....	2216
Drag indicator view .....	2217
Drag item customization.....	2219
Event .....	2220
Auto scroll options .....	2220
Disable dragging for particular item .....	2221
Cancel dropping for the dragged item.....	2221
Reorder the underlying collection .....	2221
Delete item when dropping in particular view .....	2222
Skip dragging item into another group .....	2225
Drag and drop customization.....	2226
Sorting.....	2226
Programmatic sorting .....	2226
Custom sorting.....	2228
Sort the items on header tapped.....	2229
Sort the items along with grouping .....	2231
Sorting with grouping by year.....	2231
Sorting with grouping by month and year .....	2232
Filtering .....	2234
Programmatic filtering.....	2234
Getting the filtered data .....	2236
Clear filtering.....	2237
Sort the filtered items.....	2237
Grouping .....	2237
Programmatic grouping .....	2237
Custom grouping.....	2238
Sorting the groups.....	2241
Group header summary.....	2242
Multi-level grouping.....	2246
Group expand and collapse .....	2248
Stick group header .....	2253

Group header customization .....	2254
How To .....	2264
Header and Footer .....	2268
Header and footer customization .....	2268
Header and footer items appearance .....	2269
Sticky header and footer .....	2269
Sticky footer position .....	2270
How to .....	2271
Selection .....	2280
UI selection .....	2280
Programmatic selection .....	2281
Selected items .....	2282
Selected item customization .....	2282
Selected item style .....	2286
Events .....	2288
Key navigation .....	2289
MacOS support .....	2290
How to .....	2290
Limitation .....	2293
Swiping .....	2293
Overview .....	2293
Assigning left and right swipe templates .....	2293
Working with multiple views in swipe template .....	2296
Performing swipe delete operation .....	2299
Events .....	2300
Limitations .....	2303
How to reset swipe view automatically? .....	2304
How to swipe an item indefinitely? .....	2304
How to edit data by swiping? .....	2305
PullToRefresh .....	2306
SfListView inside the SfPullToRefresh .....	2306
SfListView inside the SfPullToRefresh with ScrollView .....	2308
Limitation .....	2309
How To .....	2309
Load More .....	2312



Load more automatically .....	2313
Load more manually .....	2314
Load more when user interacts .....	2315
Show loading indicator .....	2316
Load more view customization .....	2318
Disable load more at runtime .....	2322
Limitations.....	2323
How to.....	2323
Right to left(RTL) .....	2330
Limitation .....	2331
MVVM .....	2332
Commands .....	2332
Event to command.....	2333
Binding command of inner ListView to Model Command? .....	2333
Binding command of Button inside the ItemTemplate of Xamarin.Forms ListView to ViewModel Command? .....	2334
ListView with Prism Framework.....	2335
ListView with MVVMCross .....	2336
Binding properties in MVVM pattern .....	2338
SfMaps .....	2347
Overview .....	2347
Key features .....	2348
Getting Started.....	2348
Adding SfMaps reference.....	2348
Launching an application on each platform with SfMaps.....	2349
Adding namespace.....	2350
Initializing maps .....	2351
Adding layers.....	2351
Adding shape files .....	2351
GeoJSON support .....	2352
Data binding.....	2353
Adding markers.....	2353
Color mapping.....	2354
Adding legends.....	2354
Populate Data .....	2357

Data binding.....	2357
Layers .....	2360
Imagery layer .....	2360
Shape file layer.....	2360
Sublayer .....	2363
Displaying layer in the view .....	2364
Shape Type.....	2367
Polygon.....	2367
Polyline.....	2367
Points .....	2368
Markers .....	2374
Adding marker objects.....	2374
Customizing markers.....	2375
Custom marker.....	2381
Marker Alignment .....	2383
Events.....	2387
Bubble Marker .....	2388
Bubble data .....	2388
Adding bubbles .....	2388
Customizing bubble marker .....	2389
Customizing bubble size.....	2392
Color mapping.....	2396
Equal color mapping .....	2396
Range color mapping .....	2397
Data Labels.....	2399
Adding data labels.....	2399
Setting contrast color.....	2400
Customizing data labels .....	2401
To smartly align data labels .....	2403
To avoid overlap in data labels .....	2404
Legend.....	2405
Visibility.....	2405
Legend type.....	2406
Legend position.....	2406
Legend alignment.....	2407

Icon customization .....	2408
Item margin.....	2409
Legend label .....	2410
Tooltip .....	2416
Tooltip customization .....	2417
Tooltip font customization.....	2419
Custom template for tooltip .....	2420
Setting animation for tooltip .....	2422
Setting pointer length for tooltip.....	2422
Events.....	2423
Tooltip for bubbles.....	2424
Bubble tooltip customization.....	2425
Tooltip for markers .....	2427
Marker tooltip customization .....	2429
Sublayer .....	2431
Adding sublayers in ShapeFileLayer .....	2431
Adding sublayers in ImageryLayer .....	2433
User interaction .....	2434
Selection.....	2434
Zooming .....	2437
Panning .....	2437
Map Providers.....	2438
OpenStreetMap .....	2438
Bing Maps.....	2439
Set different bing map style.....	2439
AerialWithLabel.....	2440
Zooming and panning .....	2441
Reset zooming.....	2442
Set Geo coordinates points(center position).....	2442
Set markers in imagery layer .....	2444
Custom map providers.....	2445
Cache tile images in application memory .....	2446
Clear cached tile images from application memory .....	2446
Events.....	2447
How to.....	2448

Transform latitude and longitude value to pixel value and vice-versa.....	2448
SfMaskedEdit .....	2449
Overview .....	2449
Key features .....	2450
Getting Started.....	2450
Adding SfMaskedEdit reference .....	2450
Create a Simple SfMaskedEdit .....	2452
Masking the input .....	2453
Basic Features .....	2454
Setting Value .....	2454
Setting Prompt Character .....	2454
Setting Watermark.....	2455
MaskType .....	2455
Text .....	2455
Regex.....	2456
Localization .....	2458
Using Mask Characters as Literals.....	2458
Hiding Prompt Characters.....	2459
Validation .....	2459
Validation Mode.....	2459
HasError .....	2460
Events.....	2461
ValueChanged event .....	2461
MaskInputRejected event .....	2463
Formatting Value .....	2465
ExcludePromptAndLiterals.....	2465
IncludePrompt .....	2465
IncludeLiterals .....	2466
IncludePromptAndLiterals .....	2466
Formatting clipboard text .....	2468
ExcludePromptAndLiterals.....	2468
IncludePrompt .....	2468
IncludeLiterals .....	2468
IncludePromptAndLiterals .....	2469
Visual Customization.....	2469

BorderColor .....	2469
ErrorBorderColor .....	2469
Setting Appearance of Text .....	2470
Show password character .....	2471
Password Delay .....	2471
Password Delay Duration .....	2472
SfNavigationDrawer .....	2472
Overview .....	2472
Key Features .....	2473
Getting Started .....	2473
Adding SfNavigationDrawer reference .....	2473
Initialize SfNavigationDrawer .....	2476
Adjust Drawer Size .....	2476
Add Hamburger Menu for Toggling Drawer .....	2477
Set ListView as Drawer Content .....	2479
Main Content .....	2482
Multi Drawer .....	2486
Default drawer settings .....	2486
Secondary drawer settings .....	2490
Toggling method .....	2495
Navigation Pane Sides .....	2495
Left .....	2495
Right .....	2497
Top .....	2499
Bottom .....	2502
Side Pane Content .....	2504
Header Content .....	2504
Header Height .....	2506
Footer Content .....	2508
Footer Height .....	2510
Main Content .....	2512
Side Pane Sizing .....	2514
Drawer Height .....	2514
Drawer Width .....	2515
Swipe Gesture and Sensitivity .....	2517

Enabling Swipe Gesture .....	2517
Swipe Sensitivity .....	2517
Toggle Animations .....	2518
SlideOnTop.....	2518
Push.....	2520
Reveal.....	2522
Toggling Drawer .....	2524
Opening Drawer Programmatically.....	2524
AutomationId .....	2525
SfNumericTextBox.....	2525
Overview .....	2525
Key Features.....	2526
Getting Started.....	2526
Adding SfNumericTextBox reference.....	2526
Launching the SfNumericTextBox on each platform .....	2527
Create a Simple SfNumericTextBox .....	2528
Display Customization.....	2529
Parsing the Value .....	2530
Colors .....	2530
Number Formatting .....	2532
Format String .....	2532
Compute to Percentage .....	2534
Set EnableGroupSeparator .....	2535
Font Settings .....	2535
Localization .....	2536
Change Localization of Return key text .....	2537
Set Maximum Number of Decimal Digits.....	2537
Assign Nullable Value.....	2537
Set Hint Text.....	2538
Events and Interactivity .....	2538
Events.....	2538
Interactivity : ValueChangeMode .....	2539
Range Support.....	2541
Set SelectAllOnFocus.....	2541
SfNumericUpDown .....	2542

Overview .....	2542
Key Features.....	2542
Getting Started.....	2542
Adding SfNumericUpDown reference.....	2543
Launching the SfNumericUpDown on each platform .....	2544
Create a Simple SfNumericUpDown .....	2545
Set Value .....	2545
Parsing the Value .....	2545
Colors .....	2546
Number Formatting .....	2547
Format String .....	2548
Compute to Percentage .....	2549
Set EnableGroupSeparator .....	2550
Font Customization .....	2550
Localization .....	2551
Change Localization of Return key text .....	2551
Set Maximum Number of Decimal Digits.....	2551
Assign Nullable Value.....	2552
Set Hint Text.....	2552
Auto Reverse.....	2553
Continuous Spinning Between Ranges .....	2553
Set Increment.....	2554
Set IsEditable.....	2555
Set SelectAllOnFocus.....	2555
Events and Interactivity .....	2555
Events.....	2555
Interactivity : ValueChangeMode .....	2556
Spin Button Alignment.....	2559
UpDownButtonSetting Customization.....	2561
Additional customization properties of UpDownButtonSettings .....	2565
AutomationId .....	2567
PDF .....	2567
SfParallaxView.....	2567
Overview .....	2567
Key features .....	2568

Getting Started.....	2568
Adding SfParallaxView reference.....	2568
Launch an application in iOS.....	2569
Initialize parallax view.....	2570
Add content to the parallax view.....	2571
Bind source to the parallax view.....	2572
Customization .....	2575
Speed Customization .....	2575
Orientation.....	2575
Scrolling support for custom controls.....	2576
Scrollable ContentSize .....	2576
Scrolling event.....	2577
SfPdfViewer.....	2578
PDF Viewer.....	2578
Getting started.....	2579
Adding SfPdfViewer reference.....	2579
Creating a simple PDF Viewer application .....	2581
Resolving issue when deploying an application in ReleaseMode in the UWP platform .....	2581
Unloading PDF document from the Viewer.....	2585
How to get & set the current page number?.....	2585
How to get the total page number? .....	2585
How to navigate to next page and previous page? .....	2585
How to detect a page change? .....	2586
How to get and set the zoom for SfPdfViewer? .....	2586
How to get and set Horizontal and Vertical Offsets in PDF Viewer? .....	2586
AutomationId .....	2587
Single page view mode .....	2588
Switching between view modes .....	2591
Tracking the changes in the `PageViewMode` property .....	2594
Designing a custom toolbar .....	2594
Working with built-in toolbar .....	2596
How to disable/enable built-in toolbar.....	2596
How to disable/enable bookmark.....	2599
Search for a text instance .....	2600
How to initiate text search?.....	2600



How to identify if there is no instance of the text being searched?.....	2600
How to navigate to the next instance of the text? .....	2601
How to navigate to the previous instance of the text? .....	2601
How to identify if a complete cycle of text search is performed?.....	2601
How to cancel text search?.....	2602
How to track search progress? .....	2602
Implementing search bar with search features. ....	2603
Customizing search highlight color .....	2606
Select and copy text.....	2608
How to select text? .....	2609
How to enable or disable the context menu? .....	2609
How to modify the selection and its handle color? .....	2609
How to acquire selected text? .....	2609
How to acquire page number, page bounds and selected region?.....	2610
Working with Hyperlink navigation .....	2610
How to disable hyperlink navigation in PDF document using PDF viewer control? .....	2610
How to acquire the properties of clicked URI from PDF viewer? .....	2610
Working with Document Link Annotation (Table of content) .....	2611
How to disable document link navigation in PDF document using PDF viewer control?.....	2611
Working with ink annotation .....	2611
Inclusion of ink annotation .....	2611
How to identify whether the ink annotation is included? .....	2612
How to perform undo and redo operation during inking session? .....	2613
Deleting ink annotation .....	2614
Working with ink annotation settings.....	2615
How to select the ink annotation?.....	2616
How to deselect the ink annotation?.....	2618
How to move or resize the selected ink annotation? .....	2618
Working with handwritten signatures .....	2618
Adding handwritten signatures .....	2619
Customizing the appearance of handwritten signatures.....	2619
Similarity with Ink annotations .....	2620
Working with shape annotations.....	2621
Adding shape annotations .....	2621
Detecting tap on shape annotations.....	2622

Selecting shape annotations .....	2622
Deselecting shape annotations .....	2622
Customizing the appearance of shape annotations .....	2623
Moving or resizing shape annotations .....	2624
Deleting shape annotations .....	2625
Working with free text annotations.....	2626
Adding free text annotations .....	2626
Detecting tap on free text annotations .....	2627
Selecting free text annotations.....	2627
Deselecting free text annotations.....	2628
Customizing the appearance of free text annotations .....	2628
Moving or resizing free text annotations.....	2629
Deleting free text annotations .....	2630
Working with text markup annotation .....	2631
Highlight a text .....	2631
Underline a text .....	2632
Strikethrough a text .....	2633
Deleting a text markup annotation.....	2633
Editing the color of the text markup annotation .....	2635
Performing undo and redo.....	2635
Saving the PDF document .....	2636
Working with Bookmark Navigation .....	2636
Enabling and disabling bookmark feature .....	2636
Expand and collapse the bookmark pane from built-in toolbar .....	2637
Expand and collapse the bookmark pane programmatically.....	2637
Programmatically navigate to a bookmark destination.....	2638
Track the occurrence of the bookmark navigation.....	2638
Touch interaction events .....	2639
Tapped .....	2639
DoubleTapped.....	2639
LongPressed .....	2639
Working with PDF AcroForms .....	2640
How to restrict editing the form field values in the existing PDF document.....	2640
How to flatten and save the form fields data .....	2641
Working with ScrollHead .....	2641

Enabling and disabling ScrollHead .....	2641
Navigate to a desired page using ScrollHead.....	2641
Localization .....	2641
Add a .resx file.....	2642
Convert the platform specific language format to .NET format .....	2644
Apply the converted format.....	2648
Loading password protected PDFs.....	2649
Handling invalid passwords.....	2650
Importing and exporting form data .....	2650
Exporting form data .....	2650
Importing form data.....	2650
Exporting the pages of PDF document as images.....	2651
Exporting a single PDF page as image.....	2651
Exporting a single PDF page as image with custom scale factor .....	2651
Exporting a range of PDF pages as images.....	2651
Exporting a range of PDF pages as images with custom scale factor .....	2652
Working with magnification.....	2652
Set custom zoom percentage .....	2652
Get custom zoom factor .....	2652
Set maximum zoom percentage .....	2653
Set minimum zoom percentage.....	2653
SfPicker.....	2653
Overview .....	2653
Feature comparisons across iOS, Android, and UWP .....	2654
Getting Started.....	2655
Adding SfPicker reference.....	2655
Initialize picker on each platform .....	2656
Populating Items .....	2661
Binding data source .....	2661
Multi-column items.....	2662
Set items colors and font attributes customization.....	2664
Adding custom view to items.....	2667
DataTemplateSelector .....	2669
Dealing with Header and Footer.....	2672
Enable or disable header .....	2672

Set custom header .....	2673
Header customization .....	2673
Enable or disable footer .....	2675
Set custom footer .....	2675
Perform validation with default validation button .....	2675
Dealing with Columns .....	2676
Adjust column width .....	2676
Add caption .....	2677
Caption customization .....	2679
Cascading .....	2680
Looping .....	2683
EnableLooping .....	2683
How to restrict Looping in a particular column of the picker .....	2684
Cascading .....	2685
Time Picker .....	2689
Date Time Picker .....	2693
Date Picker .....	2699
Localization .....	2704
Localize at application level .....	2705
AutomationId .....	2706
SfPopupLayout .....	2706
Overview .....	2706
Key features .....	2706
Getting Started .....	2707
Assembly deployment .....	2708
NuGet configuration .....	2708
Adding SfPopupLayout reference .....	2708
Launching the SfPopupLayout on each platform .....	2709
Create a simple popup .....	2711
Adding SfPopupLayout in Xamarin.Forms .....	2711
Customize positioning .....	2713
Customizing layouts .....	2714
Customizing animations .....	2716
Popup Positioning .....	2717
Center positioning .....	2718

Absolute positioning .....	2721
Position popup at touch point .....	2723
Relative positioning.....	2725
Popup Size.....	2733
Full Screen.....	2737
Auto-size Popup .....	2741
Layout Customizations.....	2742
Popup with one button in the footer.....	2743
Popup with two buttons in the footer .....	2745
Popup without header .....	2746
Popup without Footer.....	2748
Popup without header and footer .....	2750
Customizing popup header .....	2751
Customizing popup footer .....	2753
Customizing popup content.....	2755
Styles .....	2757
Styling popup header .....	2757
Styling popup footer .....	2759
Border customization.....	2761
Styling overlay background.....	2763
Popup Animations.....	2765
Zoom .....	2765
Fade.....	2766
SlideOnLeft.....	2767
SlideOnRight.....	2768
SlideOnTop.....	2769
SlideOnBottom.....	2770
None.....	2771
Modal Window Popup .....	2772
Popup Events And Commands.....	2774
Opening event.....	2774
Opened event .....	2775
Closing event.....	2775
Closed event.....	2776
Accept command .....	2776

Decline command .....	2777
Localization .....	2778
Add a .resx file.....	2778
Convert the platform specific language format to .NET format .....	2779
Apply the converted format.....	2783
AutomationId .....	2786
AutomationId for popup view inner elements .....	2786
AutomationId for template content .....	2788
FAQ.....	2790
Load the SfPopupLayout in GridTappedEvent of the SfDataGrid .....	2790
Open SfPopupLayout in ItemTapped event of SfListView .....	2793
Show ListView as a popup.....	2797
Display popup when interacting with a switch.....	2801
Display popup in MVVM .....	2801
Load SfPopupLayout in Prism .....	2802
Change the close button icon .....	2805
How to close popup view.....	2806
How to change popup view background color .....	2807
How to prevent the Popup from being closed when pressing the back navigation button.....	2808
How to show overlay background always in Xamarin.Forms Popup? .....	2808
Progress Bar .....	2808
Overview .....	2808
Key features .....	2808
Getting Started.....	2809
Adding SfProgressBar reference .....	2809
Launching the application on each platform with progress bar .....	2810
Initializing the progress bar.....	2811
Enabling indeterminate state.....	2813
Enable segments .....	2814
Apply colors.....	2815
States .....	2816
Determinate .....	2816
Indeterminate .....	2816
Buffer .....	2817
Range .....	2818

Custom Content .....	2819
Segments.....	2821
Appearance .....	2823
Angle .....	2823
Range colors.....	2824
Thickness.....	2828
Corner radius.....	2830
Color customization .....	2831
Animation.....	2833
Easing effects .....	2833
Indeterminate Easing Effects .....	2834
Events.....	2834
ValueChanged .....	2834
ProgressCompleted.....	2835
Step Progress Bar .....	2836
Overview .....	2836
Key features .....	2836
Getting Started.....	2836
Adding SfStepProgressBar reference .....	2837
Launching the application on each platform with StepProgressBar.....	2838
Initializing the StepProgressBar .....	2839
Orientation.....	2840
Horizontal.....	2840
Vertical .....	2840
Description.....	2841
StepProgressBar overview .....	2841
Text .....	2842
Formatted text .....	2843
Customize description.....	2845
Status .....	2846
Customizing Step view style based on their status.....	2848
SfPullToRefresh .....	2854
Overview .....	2854
Use Case Scenarios.....	2854
Key Features.....	2855

Getting Started.....	2856
Assembly deployment.....	2856
NuGet configuration .....	2856
Adding SfPullToRefresh reference .....	2857
Launching the SfPullToRefresh on each platform.....	2858
Create a sample with SfPullToRefresh .....	2859
Creating the project .....	2859
Adding SfPullToRefresh in Xamarin.Forms .....	2860
Features .....	2862
PullableContent.....	2862
TransitionMode.....	2863
RefreshContentThreshold .....	2865
PullingThreshold .....	2865
IsRefreshing.....	2866
ProgressBackgroundColor.....	2866
ProgressStrokeColor .....	2866
ProgressStrokeWidth .....	2866
RefreshContentWidth .....	2866
RefreshContentHeight .....	2867
Programmatic Support.....	2867
Host SfDataGrid as pullable content.....	2867
Host SfListView as pullable content.....	2872
Pulling and refreshing template.....	2876
Events.....	2879
Pulling.....	2880
Refreshing .....	2880
Refreshed .....	2880
SfRadioButton .....	2881
Overview .....	2881
Key features .....	2881
Getting Started.....	2881
Adding SfRadioButton reference .....	2882
Create a Simple SfRadioButton .....	2884
Setting caption .....	2885
Change the radio button state.....	2885



Visual Customization.....	2886
Customizing state color.....	2886
Setting caption text appearance.....	2887
LineBreakMode.....	2888
Visual States.....	2888
Grouping.....	2889
Group Key.....	2889
SfRadioGroup.....	2891
Event.....	2891
StateChanged event.....	2891
SfRadialMenu.....	2893
Overview.....	2893
Key features.....	2893
Getting Started.....	2894
Adding SfRadialMenu reference.....	2894
Creating a simple radial menu.....	2896
Populating Items.....	2898
Through radial menu items.....	2898
Through ItemsSource and ItemTemplate.....	2901
Animation duration.....	2903
IsOpen.....	2904
Selection color of radial menu.....	2905
Separator thickness and color in radial menu.....	2906
Rim color and rim radius in radial menu.....	2907
DisplayMemberPath.....	2908
SfRadialMenuItem Customization.....	2909
Items.....	2909
Text.....	2910
ItemHeight.....	2911
ItemWidth.....	2911
BackgroundColor.....	2912
FontFamily.....	2913
FontSize.....	2914
FontAttribute.....	2914
Image.....	2915

VisibleSegmentCount.....	2916
BackgroundImage .....	2917
FontIconText .....	2918
IconFontColor.....	2918
IconFontSize .....	2919
IconFontFamily.....	2920
View .....	2921
Layout types.....	2922
Default.....	2922
Custom .....	2923
Code snippet for VisibleSegmentCount and SegmentIndex .....	2926
Placing and Dragging RadialMenu .....	2928
Dragging RadialMenu.....	2928
DragEvents .....	2930
Placing RadialMenu.....	2932
CenterButton Customization .....	2934
CenterButtonText and CenterButtonBackText.....	2934
CenterButtonTextColor and CenterButtonBackTextColor.....	2935
CenterButtonBackgroundColor.....	2936
CenterButtonRadius.....	2937
CenterButtonFontFamily and CenterButtonBackFontFamily .....	2938
CenterButtonFontSize and CenterButtonBackFontSize.....	2940
CenterButtonFontAttributes.....	2941
CenterButtonBorderColor.....	2943
CenterButtonBorderThickness.....	2944
CenterButtonPlacement .....	2946
Center button view and center back button view .....	2947
EnableCenterButtonAnimation.....	2948
Events.....	2950
Perform an action while navigating to next level .....	2950
Perform an action while opening the radial menu .....	2951
Perform an action while closing the radial menu .....	2952
Perform an action while tapping the center back button .....	2953
Perform an action while tapping the radial menu item.....	2954
AutomationId .....	2955

SfRangeSlider .....	2956
Overview .....	2956
Key features .....	2957
Getting Started.....	2957
Adding SfRangeSlider reference .....	2957
Additional step for UWP .....	2958
Set Range .....	2959
Restricting Values.....	2960
Adding Snapping Mode.....	2960
Range .....	2960
Set Show Range.....	2960
Set Range values .....	2961
ValueChangeMode.....	2961
Value .....	2962
Customizing ticks .....	2962
TickPlacement.....	2962
Customizing tick color .....	2964
TrackBar Customization .....	2965
Formatting String .....	2969
Formatting types.....	2969
Culture Localization.....	2970
Selection Value Configuration .....	2971
Set Minimum Value.....	2971
Set Maximum Value .....	2971
Set Tick Frequency .....	2971
Set Interval between Snap Points.....	2971
Set Snapping Mode .....	2972
Orientation.....	2972
Horizontal.....	2972
Vertical .....	2973
Customizing labels .....	2974
Show Value Label .....	2975
Set Custom Label.....	2975
Value Placement .....	2976
Label Placement.....	2977

Customizing label font .....	2977
LabelColor .....	2979
ToolTip Support.....	2980
Set ToolTip Precision.....	2980
Set ToolTip Placement .....	2980
Tooltip color .....	2981
How to Perform an Action while Selecting a Value? .....	2982
How to Perform an Action when the Range Get Changing?.....	2982
How to get notifications when a thumb drag is started and completed? .....	2983
How to trigger the ThumbTouchDown event? .....	2983
How to trigger the ThumbTouchUp event? .....	2984
AutomationId .....	2984
SfRating .....	2984
Overview .....	2984
Key Features.....	2985
Getting Started.....	2985
Adding SfRating reference .....	2985
Adding namespace.....	2987
Initialize Rating.....	2987
Set Number of Rating Items.....	2987
Set Value .....	2988
Precision.....	2988
Precision Mode .....	2989
Standard.....	2989
Half .....	2989
Exact.....	2990
Tooltip .....	2990
Set Tooltip Placement.....	2990
Set ToolTip Precision.....	2992
Appearance and Styling .....	2992
Set Size .....	2992
Set Number of Items.....	2993
Set Space between Items.....	2993
Rating Settings .....	2993
Restrict User Selection.....	2994

Appearance Customization .....	2994
Set Fill Color .....	2995
Set Stroke Color .....	2995
Set Stroke Width .....	2996
Custom Views.....	2997
Add SfRating items.....	2997
Set selected view .....	2998
Set unselected view .....	2998
Add Items .....	2998
Enable custom items.....	2999
View Range Selection.....	3000
AutomationId .....	3001
SfRichTextEditor .....	3001
Overview .....	3001
Key features .....	3002
Getting Started.....	3002
Adding SfRichTextEditor reference .....	3002
Launching the application on each platform with Rich Text Editor.....	3004
Initializing the Rich Text Editor .....	3005
SfRotator .....	3006
Overview .....	3006
Key Features.....	3006
Getting Started.....	3006
Adding SfRotator reference .....	3006
Launching the SfRotator on each platform.....	3008
Create a Simple SfRotator .....	3008
Add Rotator Items.....	3009
Setting Navigation Mode .....	3015
Customizing Position.....	3017
Populating Data.....	3019
Through Binding.....	3019
Through Rotator Item .....	3022
DataTemplateSelector .....	3024
Adding Looping and Delays.....	3026
Toggle AutoPlay .....	3027

Setting Navigation Delay .....	3028
Looping Items.....	3030
Enable swiping .....	3031
Navigation Modes .....	3033
Items / Dot Strip Positions .....	3037
Sliding Direction .....	3039
Horizontal.....	3039
Vertical .....	3041
Header Visibility .....	3043
Placement Modes .....	3044
DotsPlacement.....	3044
Customization .....	3046
DotsBorder Color .....	3046
Selected Dot Color .....	3048
Unselected Dot Color .....	3050
Loading URL Images .....	3052
Events.....	3054
Selection Changed.....	3054
ItemTapped .....	3054
AutomationId .....	3057
SfSchedule.....	3058
Overview .....	3058
Key features .....	3059
Getting Started.....	3059
Adding SfSchedule reference .....	3059
Create a simple application with SfSchedule.....	3061
Creating a new project.....	3062
Adding SfSchedule to the project .....	3062
Changing Schedule Views .....	3063
Binding data to SfSchedule control.....	3069
Header.....	3074
Header Height .....	3074
Appearance .....	3074
Loading Custom Headers .....	3076
Header Date Format .....	3077

Header Tapped Event.....	3078
DayView .....	3079
ViewHeader Appearance .....	3079
Change Time Interval .....	3085
Change Time Interval Height.....	3086
Change Working hours.....	3088
Changing StartHour and EndHour.....	3089
Timeslot Appearance .....	3092
Non-Accessible timeslots .....	3095
Change first day of week.....	3096
Time Label Formatting .....	3096
Time Label Appearance.....	3097
Time Label Size .....	3098
Selection.....	3099
WeekView .....	3103
ViewHeader Appearance .....	3103
Change Time Interval .....	3110
Change Time Interval Height.....	3111
Change Working hours.....	3113
Changing StartHour and EndHour.....	3114
Timeslot Appearance .....	3117
Non-Accessible timeslots .....	3120
Change first day of week.....	3121
Time Label Formatting .....	3122
Time Label Appearance.....	3123
Time Label Size .....	3124
Selection.....	3125
WorkWeekView .....	3129
ViewHeader Appearance .....	3129
Change Time Interval .....	3136
Change Time Interval Height.....	3137
Change Working hours.....	3139
Changing StartHour and EndHour.....	3140
Changing NonWorking Days .....	3143
Timeslot Appearance .....	3143

Non-Accessible timeslots .....	3146
Change first day of week.....	3148
Time Label Formatting .....	3149
Time Label Appearance.....	3150
Time Label Size.....	3151
Selection.....	3152
Timeline view .....	3156
Timeline view days count.....	3157
Timeline view based on day, week, work week, and month.....	3158
Customized working hours .....	3158
Special time regions .....	3161
Time interval .....	3162
Time interval height .....	3163
Nonworking days .....	3165
First day of week .....	3165
Appointment height.....	3166
View header tapped event.....	3168
View header customization .....	3168
Timeslot customization.....	3172
Time label customization .....	3173
Selection.....	3177
Month View .....	3181
Month Appointment indicator.....	3182
Month Appointment display mode.....	3183
Month InlineView.....	3186
Agenda View .....	3188
Month Navigation direction.....	3196
Restricted days in Month .....	3196
First day of Week in Month .....	3197
Week Number of the Year in Month .....	3198
Week Number Appearance.....	3199
View Header Appearance .....	3200
MonthCell Appearance .....	3204
Getting Inline Appointment details .....	3212
InlineView Appearance .....	3213



InlineAppointment Appearance.....	3217
Selection.....	3219
Today Background Color .....	3226
Custom Font.....	3228
Appointments .....	3233
Mapping .....	3234
Spanned Appointments .....	3236
All Day Appointments .....	3239
Recurrence Appointment.....	3241
Recurrence Pattern Exceptions.....	3249
Get visible appointments.....	3261
Suspend and resume the appointment update .....	3262
Appearance Customization .....	3263
Customize appearance using Custom View .....	3267
Customize appearance using DataTemplate .....	3269
Customize appearance using DataTemplateSelector .....	3270
Selection.....	3275
Minimum Appointment Height.....	3279
Resource view .....	3280
Resource view visibility .....	3282
Assigning events for resources .....	3282
Assigning custom events to resources.....	3283
Mapping .....	3284
Selection mode .....	3286
Programmatic resource selection .....	3287
Changing resource view height.....	3288
Visible resource count.....	3289
Resource item tapped event.....	3290
Customization .....	3291
Appointment Drag and Drop .....	3295
Handle dragging based on the appointment .....	3296
Get the dragging appointment position .....	3297
Handle appointment dropping .....	3298
Customizing the Drag and Drop environment .....	3299
Customize appearance of dragging Time Indicator .....	3300

Time Zone .....	3301
Create appointments in different time zones.....	3305
Display Appointments based on client's time zone .....	3306
Display appointments based on schedule time zone .....	3306
Display appointments at same time everywhere regardless of client's time zone .....	3306
Updating StartTime and EndTime after drag and drop appointment based on Time Zone.....	3307
Date Navigations.....	3307
Enabling Navigation .....	3307
Programmatically change to specific dates .....	3307
Programmatically change to adjacent dates. ....	3308
Range for visible dates.....	3308
VisibleDatesChanged event .....	3309
Localization .....	3310
Change default control language.....	3310
Change custom texts in the control.....	3311
Localizing custom strings from PCL.....	3313
Right to left(RTL) .....	3319
Theming .....	3320
Default theme (light theme) .....	3321
Customizing the default theme .....	3325
AutomationId .....	3329
SfSegmentedControl.....	3336
Overview .....	3336
Key Features.....	3336
Getting Started.....	3336
Assembly deployment.....	3337
Adding SfSegmentedControl reference .....	3337
Launching the SfSegmentedControl on each platform.....	3338
Creating an application with segmented control. ....	3339
Creating a project.....	3340
Adding data/Items to SfSegmentedControl.....	3344
Customizing segmented control appearance .....	3358
Customizing selection indicator appearance.....	3359
Handle click events .....	3359
Populating data source .....	3360

String collection .....	3360
Segment items .....	3361
Custom views .....	3362
Selection changed .....	3364
User interface.....	3364
Selected Index through programmatically.....	3364
Display mode.....	3365
Text .....	3365
Image.....	3366
Image with text .....	3367
How to set the font icons using ttf file?.....	3369
Indicating the selected item .....	3369
Selection text color .....	3369
Selection strip .....	3370
Handling multiple segments .....	3373
Visible segment counts .....	3373
Segment width .....	3374
Customization .....	3374
Text appearance.....	3374
Border .....	3376
Scrolling in segmented control programmatically .....	3380
AutomationId .....	3380
How to.....	3381
Clear the default selection in SfSegmentedControl.....	3381
Autoscrolling the selected segment item .....	3382
SfShimmer .....	3383
Overview .....	3383
Key features .....	3383
Getting Started.....	3383
Adding SfShimmer reference .....	3383
Launching an application on each platform with SfShimmer .....	3385
Initializing shimmer .....	3385
Loading shimmer content .....	3386
Shimmer Types.....	3387
Built-in types .....	3387

Custom view.....	3388
Customization of Shimmer .....	3390
WaveDirection .....	3390
Color.....	3391
WaveColor.....	3391
WaveWidth .....	3392
AnimationDuration .....	3393
SfSparkline .....	3393
Overview .....	3393
Key features .....	3394
Getting Started.....	3394
Adding SfSparkline reference.....	3394
Launching an application on each platform with SfSparkline.....	3395
Initialize view model .....	3396
Populate Sparkline with data .....	3397
Sparkline Types .....	3398
Line Sparkline .....	3398
Column Sparkline .....	3399
Area Sparkline .....	3400
WinLoss Sparkline .....	3401
Range Band .....	3402
Markers .....	3403
Customize data points .....	3404
Sparkline Axis .....	3406
SfSunburstChart .....	3407
Overview .....	3407
Key features .....	3408
Getting Started.....	3408
Adding SfSunburstChart reference .....	3408
Launching an application on each platform with SfSunburstChart. ....	3410
Initialize sunburst chart .....	3411
Initialize view model .....	3411
Populate sunburst chart with data .....	3414
Add title.....	3415
Add legend .....	3416

Add data labels.....	3417
Levels.....	3420
Group member path .....	3420
Legend.....	3422
Visibility .....	3422
Position .....	3423
Legend icon types .....	3424
Icon size customization .....	3425
Label style .....	3426
Item margin.....	3428
Toggle selection .....	3429
Data label .....	3430
Overflow Mode .....	3431
Rotation Mode .....	3433
Customization .....	3435
Label text color.....	3436
Drill-down .....	3437
Positioning Toolbar .....	3438
Toolbar alignment.....	3439
Selection.....	3441
Selection type.....	3441
Selection display mode .....	3445
Events.....	3448
Tooltip .....	3450
Customization .....	3451
Font customization .....	3452
Custom template .....	3453
Animation.....	3455
Duration .....	3456
Customization .....	3457
Palettes .....	3457
Angle .....	3459
Radius.....	3460
Inner radius .....	3462
Stroke customization .....	3463

SfSwitch.....	3465
SfSwitch.....	3465
Overview .....	3465
Key features .....	3466
Getting Started.....	3467
Adding SfSwitch reference.....	3467
Launching an application on each platform with SfSwitch.....	3468
Initializing SfSwitch .....	3469
Performing an action based on state.....	3470
Visual Types .....	3470
Default.....	3470
Material.....	3471
Cupertino .....	3471
Fluent .....	3472
Custom .....	3472
States .....	3472
On.....	3472
Off .....	3473
Indeterminate .....	3473
Disabled On.....	3474
Disabled Off.....	3474
Disabled Indeterminate .....	3475
Orientation.....	3475
Horizontal.....	3475
Vertical .....	3476
Customization with visual states .....	3476
Solid colors .....	3476
Gradients.....	3479
Sizing .....	3481
Images .....	3483
Limitation .....	3484
How to.....	3484
Show busy indicator to perform async action .....	3484
Change thumb color alone based on its state and devices .....	3485
Change thumb color alone based on its state with Material theme for all devices.....	3487

Set color for disabled state .....	3488
Change busy indicator color.....	3490
Set all state in same look .....	3491
Set RTL to switch control .....	3492
Get default color of the switches in all three state. ....	3494
AutomationId .....	3498
SfTabView .....	3498
Overview .....	3498
Key features .....	3499
Getting Started.....	3499
Assembly deployment.....	3499
Adding SfTabView reference.....	3499
Launching the tab view on each platform .....	3501
Create a simple tab view.....	3502
Populating tab items .....	3503
Adding ListView in tab view .....	3505
UseCase Sample with Contacts Information stored as a ListView in TabView Control .....	3505
Binding data to list view.....	3505
Swiping.....	3507
Tab Items.....	3507
Share the header space equally .....	3509
Selection Changed.....	3511
Enable swiping .....	3512
Display Type .....	3515
How to change the selection color for text and font icons?.....	3517
How to customize text appearance of the header title? .....	3520
How to set and customize the font icons' appearance in the header? .....	3523
Selection Indicator Strip.....	3526
Positioning of header .....	3529
Handling overflow tabs .....	3531
How to customize the more button?.....	3534
Custom Header .....	3537
How to handle the events for custom view with tab view .....	3542
Image Source .....	3542
Nested Tab Items .....	3544

CenterButtonSettings .....	3548
Customize CenterButtonSettings .....	3549
CenterButtonTapped event .....	3549
Custom CenterButton .....	3550
TabItemTapped .....	3551
AutomationId .....	3552
SfTreeMap .....	3553
Overview .....	3553
Key features .....	3554
Getting Started .....	3554
Adding SfTreeMap reference .....	3554
Initializing TreeMap .....	3556
Populating TreeMap items .....	3556
Grouping TreeMap items using levels .....	3559
Customizing the appearance of TreeMap by range .....	3559
LeafItemSetting .....	3560
Enabling legends .....	3560
Labels for legends .....	3560
DataBinding .....	3563
TreeMap Levels .....	3564
Flat Level .....	3564
Hierarchical Level .....	3567
Layout .....	3569
Squarified .....	3569
SliceAndDiceAuto .....	3570
SliceAndDiceHorizontal .....	3572
SliceAndDiceVertical .....	3573
Customization .....	3575
Color .....	3575
Tooltip .....	3581
Selection .....	3585
ItemTemplate .....	3587
TreeMap Elements .....	3589
Legend .....	3590
Header .....	3591



Data labels.....	3592
TreeMap Events .....	3599
ItemSelected .....	3599
Drilldown.....	3600
Header customization .....	3601
SfTreeView .....	3611
SfTreeView .....	3611
Key features .....	3612
Getting Started.....	3612
Assembly Deployment .....	3612
Adding SfTreeView reference .....	3613
Launching the TreeView on Each Platform .....	3614
Create a tree view .....	3615
Creating the Project .....	3616
Adding the tree view in Xamarin.Forms .....	3616
Populating Nodes without data source - Unbound Mode.....	3617
Creating Data Model for the tree view .....	3620
Bind to a hierarchical data source - Bound Mode .....	3623
Creating hierarchical Data Model for tree view.....	3624
Bind to a Hierarchy Property Descriptors data source - Bound mode .....	3628
Defining a template to expander and content view .....	3629
Interacting with a tree view .....	3633
Selection.....	3633
Data Population .....	3634
Populating Nodes by data binding - Bound Mode .....	3634
Populating Nodes without data binding - Unbound Mode .....	3641
Appearance .....	3644
ItemTemplate.....	3644
BindingContext for ItemTemplate .....	3645
ItemTemplate Selector.....	3646
Indentation.....	3649
ExpanderWidth .....	3649
ExpanderPosition .....	3649
Level based styling .....	3649
Animation.....	3652

Expand and Collapse .....	3652
Expand Action Target .....	3652
Auto Expand Mode .....	3653
Programmatic Expand and Collapse .....	3653
Expand and Collapse using Keyboard .....	3654
Events.....	3654
Interactivity .....	3654
Interacting with TreeView items.....	3654
Scrolling.....	3655
Bring Into View.....	3656
Scrollbar Visibility.....	3657
Selection.....	3657
UI Selection .....	3658
Programmatic Selection.....	3660
Selected items.....	3660
Selected item style.....	3660
Events.....	3661
Key Navigation .....	3661
FocusBorderColor .....	3662
FocusBorderThickness .....	3662
Limitation .....	3662
Checkbox.....	3662
Working with Checkbox in BoundMode .....	3662
Working with Checkbox in UnboundMode.....	3665
CheckBox State .....	3668
Get or Set Checked Items.....	3669
Events.....	3669
MVVM .....	3670
Binding properties in MVVM pattern .....	3670
Commands .....	3675
Event to command.....	3678
Checkbox items binding in MVVM.....	3679
Load on demand .....	3680
Handling expander visibility.....	3684
On-demand loading of child items.....	3685

Item Height Customization .....	3686
Customize Item Height.....	3687
Customize Item height using `QueryNodeSize` event .....	3687
Autofit item height on dynamic changes .....	3689
Limitations.....	3691
Right to left(RTL) .....	3691
Limitations.....	3693
SfTextInputLayout.....	3693
Overview .....	3693
Key features .....	3693
Getting Started.....	3694
Adding SfTextInputLayout reference .....	3694
Launching an application on each platform with text input layout.....	3695
Initializing text input layout .....	3696
Enabling password visibility toggle .....	3697
Supported input views .....	3697
Entry.....	3698
Editor.....	3698
Masked edit .....	3698
Numeric text box.....	3699
Autocomplete .....	3700
Combo box .....	3700
Container type .....	3701
Filled.....	3702
Outlined .....	3702
None.....	3704
Adding custom icons.....	3704
Leading view .....	3704
Trailing view .....	3705
Font Customization .....	3706
Hint.....	3706
Helper text .....	3707
Error text.....	3707
Counter label.....	3708
Adding assistive labels .....	3709

Helper text .....	3709
Error message .....	3710
Character counter .....	3710
Reserve spaces for assistive labels.....	3711
States and Colors .....	3712
Focused color .....	3712
Unfocused color .....	3712
Error color .....	3713
Container color .....	3713
Disabled state.....	3714
Right-to-Left.....	3715
Fixed hint position.....	3716
Filled.....	3716
Outlined .....	3716
None.....	3717
How to.....	3717
Customize the thickness of stroke .....	3717
Customize the font for input view .....	3718
AutomationId .....	3718
Utilities .....	3719
Project Template.....	3719
Create Syncfusion Xamarin Application.....	3719
Project Configuration: .....	3720
Toolbox .....	3723
Add Syncfusion Xamarin (Xamarin.Forms) Controls in your Project .....	3723

## Welcome to Syncfusion Essential Studio for Xamarin.Forms

Essential Studio for Xamarin.Forms is a comprehensive collection of enterprise-grade Xamarin.Forms components for building modern Mobile applications. It includes all the UI controls that are typically required for building line-of-business (LOB) applications including Charts, Gauge, Maps and much more.

### How to best read this user guide

- The best way to read the User guide is to start with the Getting Started section of the documentation for the component that you need. The Getting Started guide gives information needed to know before writing the code. This is the only section recommended for end-to-end reading before writing the code. All the other information can be referred as and when needed.
- Now, that you are familiar with the basics of using the component, the next step would be to start integrating the component into your application. A good starting point would be to refer to the code examples in the sample browser. It is very likely that you would find a code example that resembles your intended usage scenario.
- After you have integrated the component into your application, it is likely that you would need additional information on specific features and API. Search the specific topic by using the search box available at the top of the user guide.
- Another valuable resource is the API reference that provides detailed information on the classes and its members.

### Additional help resources

The Knowledge Base section contains responses to some of the most common questions that other customers had asked in the past. You can search for topics that are not covered in the user guide.

Similar to the Knowledge Base, the Forum section also contains responses to questions of other customers asked in the past.

### Create a support incident

In case, you are not able to find the information that you are looking for in the self-help resources mentioned above, then please contact us by creating a support ticket.

## System Requirements

### Hardware Environment

- Processor: x86 or x64
- RAM : 512 MB (minimum), 1 GB (recommended)
- Hard disk: up to 1.5 GB of available space may be required. However, 250 MB free space is required in boot drive even if you are installing the setup in other drive.

### Development Environment

Please find the recommended development environment for Xamarin platform in the following link.

<https://docs.microsoft.com/en-us/xamarin/cross-platform/get-started/requirements>

### Supported Platforms

The following table lists the platforms supported by our Xamarin.Forms controls.

Platform	Device Types	Controls	Supported versions
Android	Phone, Tablet	SfPdfViewer, SfCheckBox	API level 21 and later versions
		All other controls	API level 19 and later versions
iOS	iPhone, iPod, iPad	All controls	iOS 9.0 and later versions
UWP	Desktop, Tablet	All controls except SfEffectsView and SfShimmer	Windows 10 devices
macOS	iMac, Mac book, Mac mini	SfChart, SfDataGrid and SfListView	macOS 10.11 and later versions
WPF	Desktop	SfChart, SfListView, SfSchedule, SfBorder, SfButton, SfChip, SfSwitch, SfRadioButton, SfCheckBox, SfGradientView, SfSegmentedControl, SfTextInputLayout	Windows 10

#### Target Framework (Compile SDK Version) for Android

For Android platform, the target framework or compile SDK version of the application should be equal or greater than the latest API available at the time of our release. Please find more details about setting the target framework in the following link.

<https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/android-api-levels?tabs=windows#target-framework>

## Download Installer

You can download the installer from [Syncfusion.com](https://www.syncfusion.com) website.

## Download the Trial Version

There are two ways to download our 30-day trial.

### Free Trial Page

1. You can evaluate our 30-day free trial from [Free Trial](#) page.
2. Once you fill the required form or made the login using the your Syncfusion registered account you can download the trial installer setup in the confirmation page.
3. You can download the latest version trial installer.
4. You can unlock the installer using the unlock key, also you can unlock the installer using the Syncfusion registered login credential.
5. You can download the trial installer using the [Trials & Downloads](#) page under your registered account at any time before the trial expire. (Refer the below screenshot).

## Trial Downloads and Unlock Keys

Trial SKU Name	
Essential Studio	
Trial Version :	
<a href="#">What's New</a>   <a href="#">Generate License File</a> ?   <a href="#">Get Unlock Key</a> ?	<div>Download 1</div> <div>More Download Options 2</div>

### Start Trial Page

1. You can evaluate our 30-day free trial from [Start Trial](#) page.
2. You should login using your Syncfusion account to access this page.
3. You can start your trail by clicking on the required product.

**Note:** If you already using the trail products and it's not expired, you couldn't start the trial again for same product.

4. After you started the trial, you can download the latest version trial installer using the [Trials & Downloads](#) page.
5. In [Trials & Downloads](#) page, you can find your current active trial products. Trials, which you done in both Free Trial Page and Start trial pages are listed here.
6. Use the Download (element 1 in below screenshot) button to download the installer of respective product.
7. Online installer can be downloaded from this page.
8. No need of unlock key to unlock the online installer.
9. You can unlock the installer using the Syncfusion registered login credential.

## License Downloads and Unlock Keys

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

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

### Download the License Version

1. You can find your available licensed products which under your registered Syncfusion account in [License & Downloads](#) page.
2. You can find all the licenses (both active licenses and expired licenses) which are under your account.
3. You can find the licenses listed based on SKU names.

4. Use the Download (element 1 in below screenshot) button to download the installer of respective product.
5. Latest version installer will be downloaded from this page.
6. You can navigate to [Downloads Older Versions](#) (element 2 in below screenshot) to download older version installers.
7. From 16.2 version online installer will be downloaded by default, and earlier versions offline installer will be downloaded.

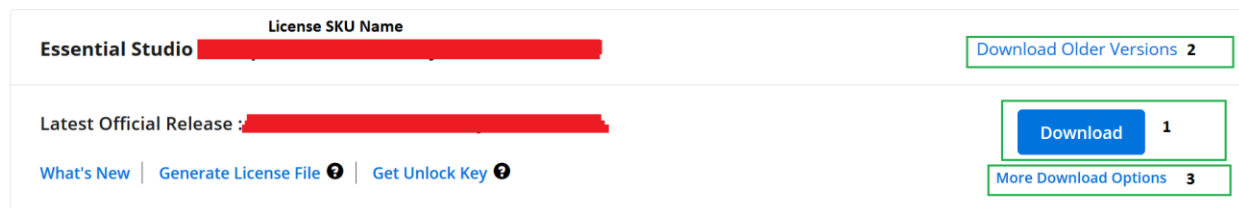
---

**Note:** Online Installer have been introduced from the release version 16.2.

---

8. You can navigate to More Downloads Options (element 3 in below screenshot) to download other setups.
9. EXE and Zip format available to download for Windows OS. Both are Offline Installer.
10. No need of unlock key to unlock the online installer.
11. You can unlock the installer using the unlock key for versions earlier to 16.2, also you can unlock the installer using the Syncfusion registered login credential.

## License Downloads and Unlock Keys




---

**Note:** You can generate the license key for your licensed products from [License & Downloads](#) page. This license key will be required only from release version 16.2. To know more about License key, refer this [help topic](#).

---

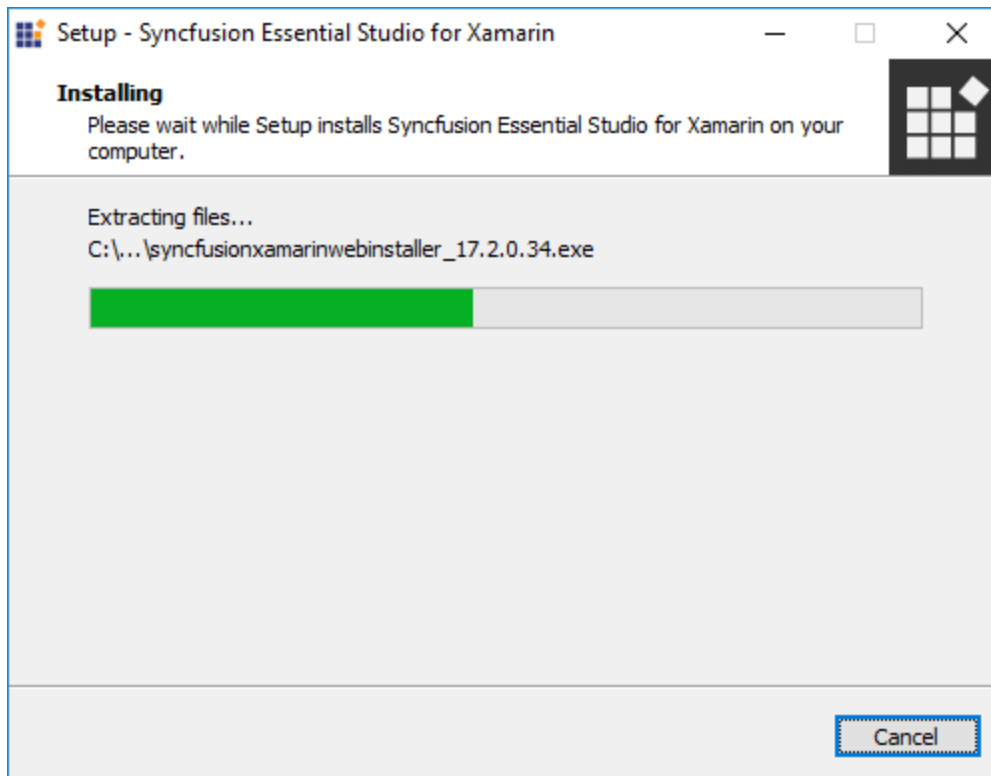
## Installation and Upgrade

### Installation using Web Installer

The following procedure illustrates how to install Essential Studio for Xamarin Online Installer.

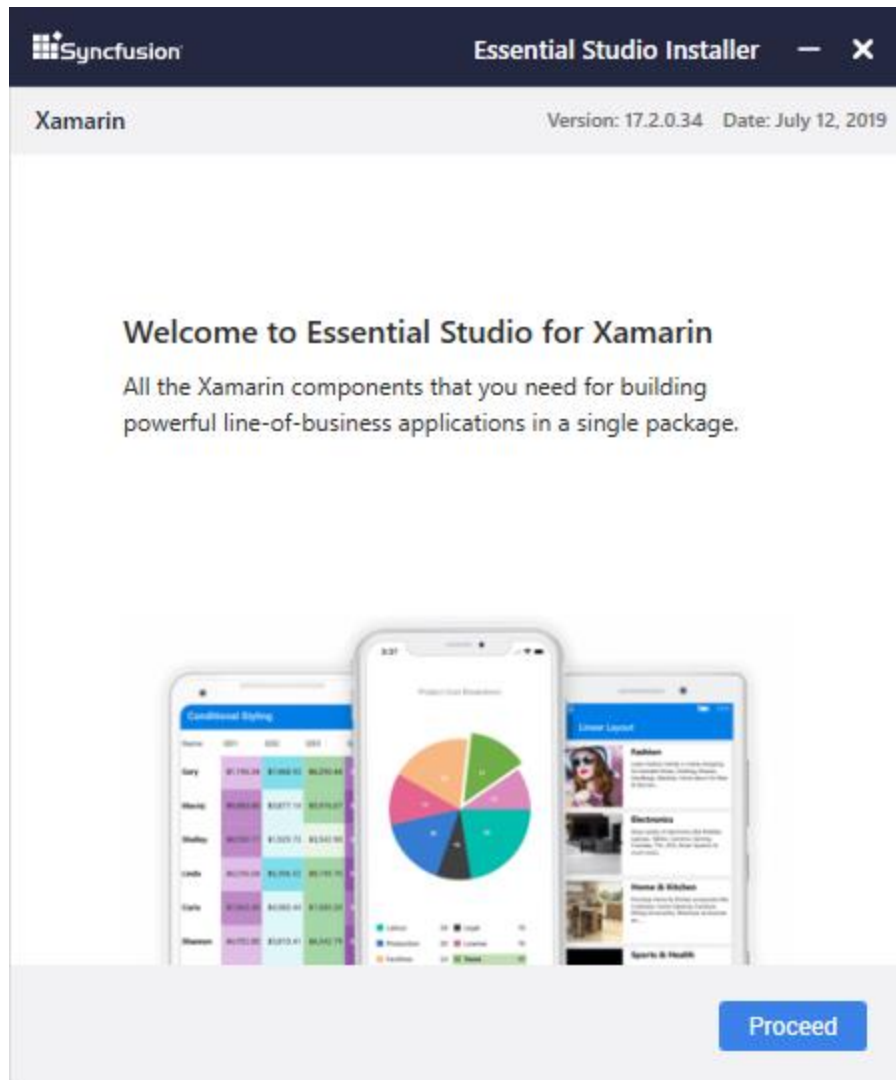
1. Double-click the Syncfusion Xamarin Online Installer setup file. The Setup Wizard opens and extracts the package automatically.



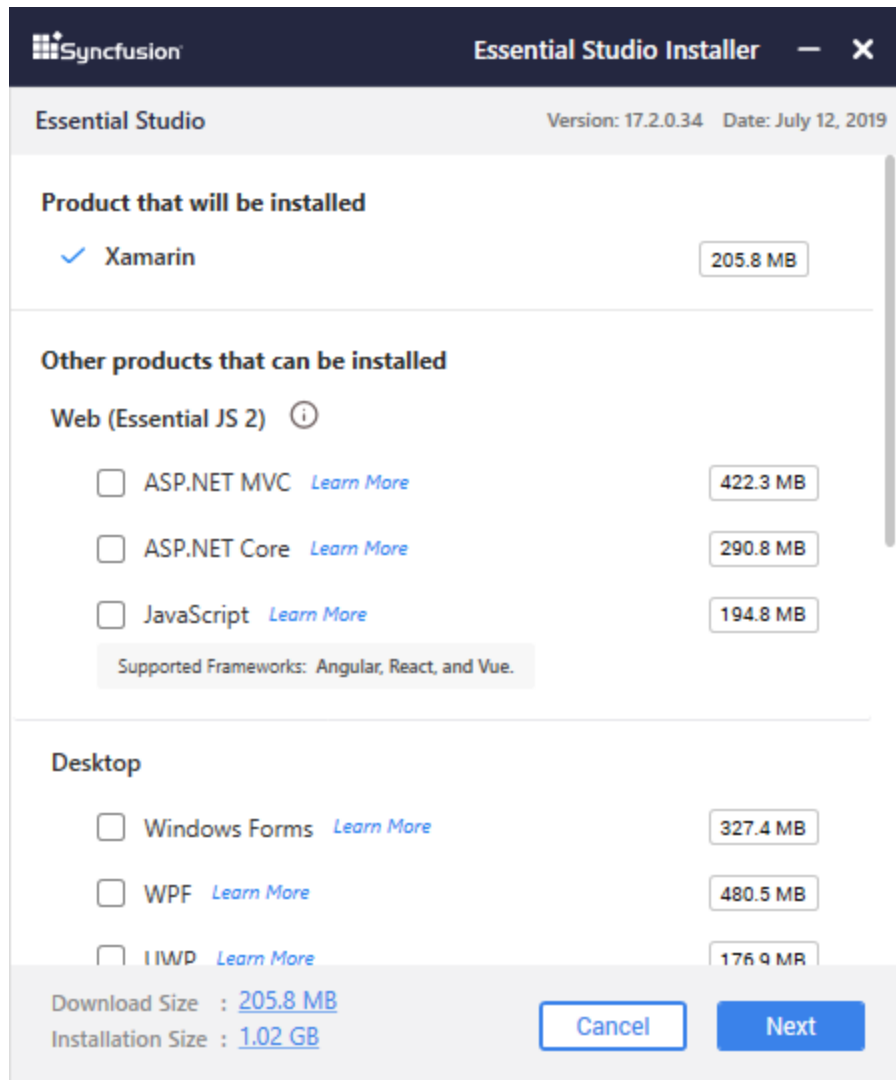


**Note:** The Setup wizard extracts the syncfusionxamarinwebinstaller\_{version}.exe dialog, displaying the unzip operation of the package.

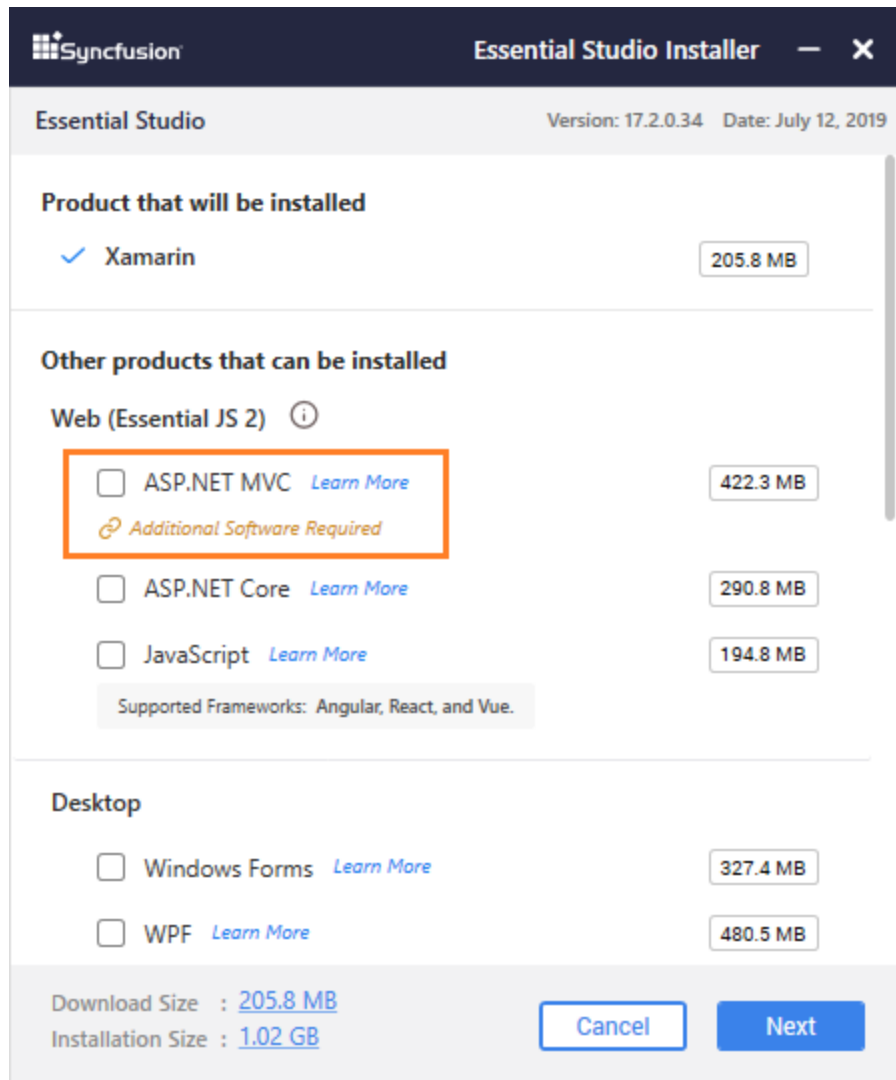
2. Welcome wizard of the Syncfusion Online Installer will be displayed. Click Proceed.



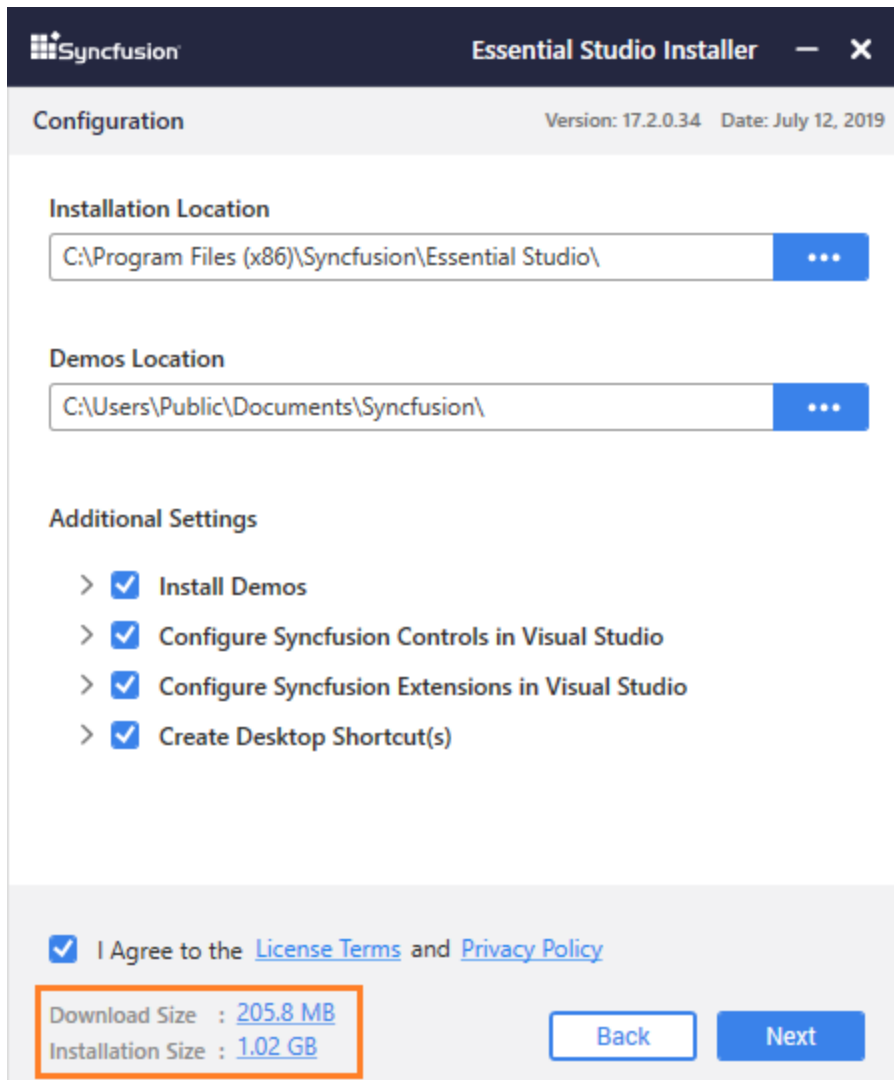
3. Platform Selection wizard will be displayed. Here you can select the required platforms to be installed. Click Next.



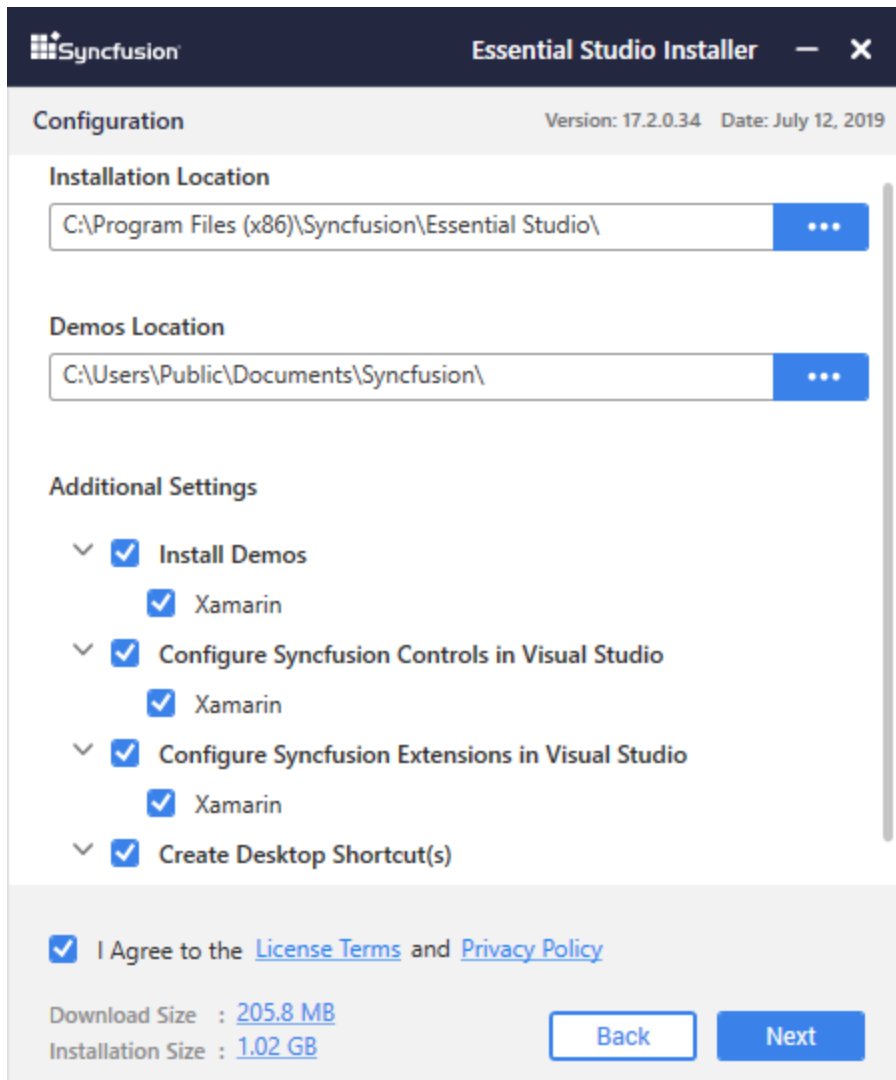
**Information:** If the required software of the selected platform was not already installed, **Additional Software Required** alert will be displayed. However, you can continue the installation and install the required software later.



**Note:** You can check the Estimated size of the Download and Installation by clicking the **Download Size and Installation Size** link.



4. Configuration wizard will be displayed. Here you can change the Install and samples location. Also, you can change the Additional settings by platform basis. To install using the default configuration, click Next.



**Note:** From the 2018 Volume 2 release, Syncfusion has changed the install and samples location

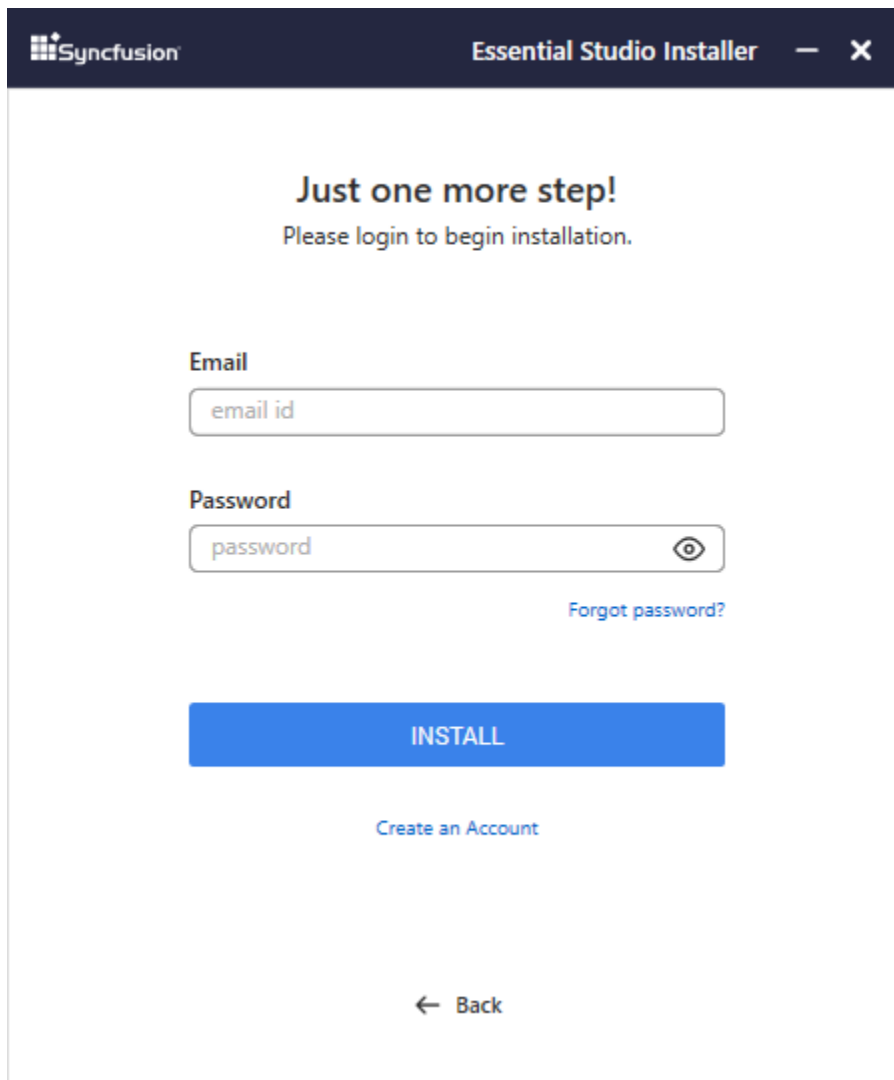
**Default Install location:** {ProgramFilesFolder}\Syncfusion\{Platform}\{version}

**Default Samples location:** C:\Users\Public\Documents\Syncfusion\{platform}\{version}

However, you can change the locations by clicking browse button.

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

5. After reading the License Terms and Conditions, check the **I agree to the License Terms and Privacy Policy** check box. Click Next.
6. Login wizard will be displayed. You should enter your Syncfusion Direct-Trac login credentials. If you don't have Syncfusion Direct-Trac login credentials, then you can click on **Create an Account**. Else if you forgot your password, click on **Forgot Password** to create new password. Click Install.



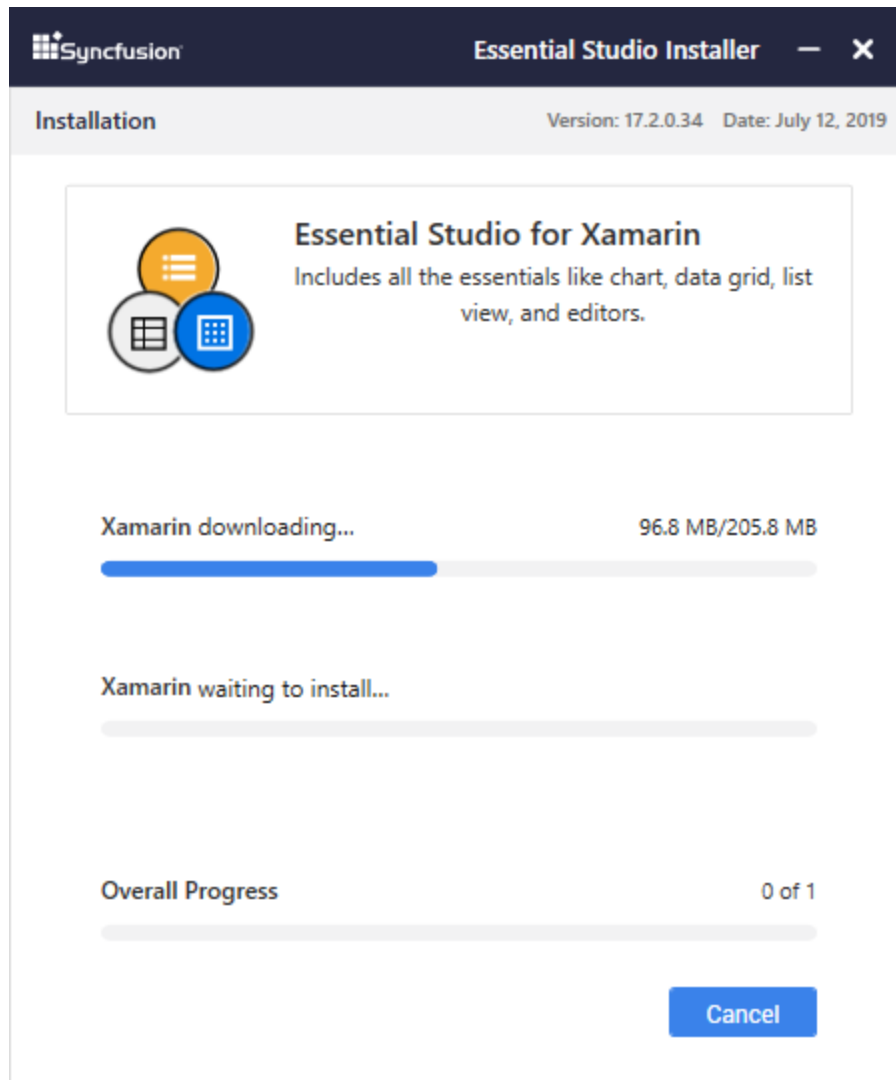
The screenshot shows the 'Essential Studio Installer' window. At the top, there's a dark blue header with the Syncfusion logo on the left and the text 'Essential Studio Installer' followed by a minus sign and a close button (X) on the right. The main content area is white and contains the following elements: a heading 'Just one more step!' in bold, followed by the instruction 'Please login to begin installation.' Below this, there are two input fields: 'Email' with a placeholder 'email id' and 'Password' with a placeholder 'password' and an eye icon for toggling visibility. To the right of the password field is a link 'Forgot password?'. Below the input fields is a large blue button labeled 'INSTALL'. Underneath the 'INSTALL' button is a link 'Create an Account'. At the bottom left, there is a back arrow icon followed by the text 'Back'.

---

**Information:** The selected platforms will be installed based on your Syncfusion License (Trial or Licensed).

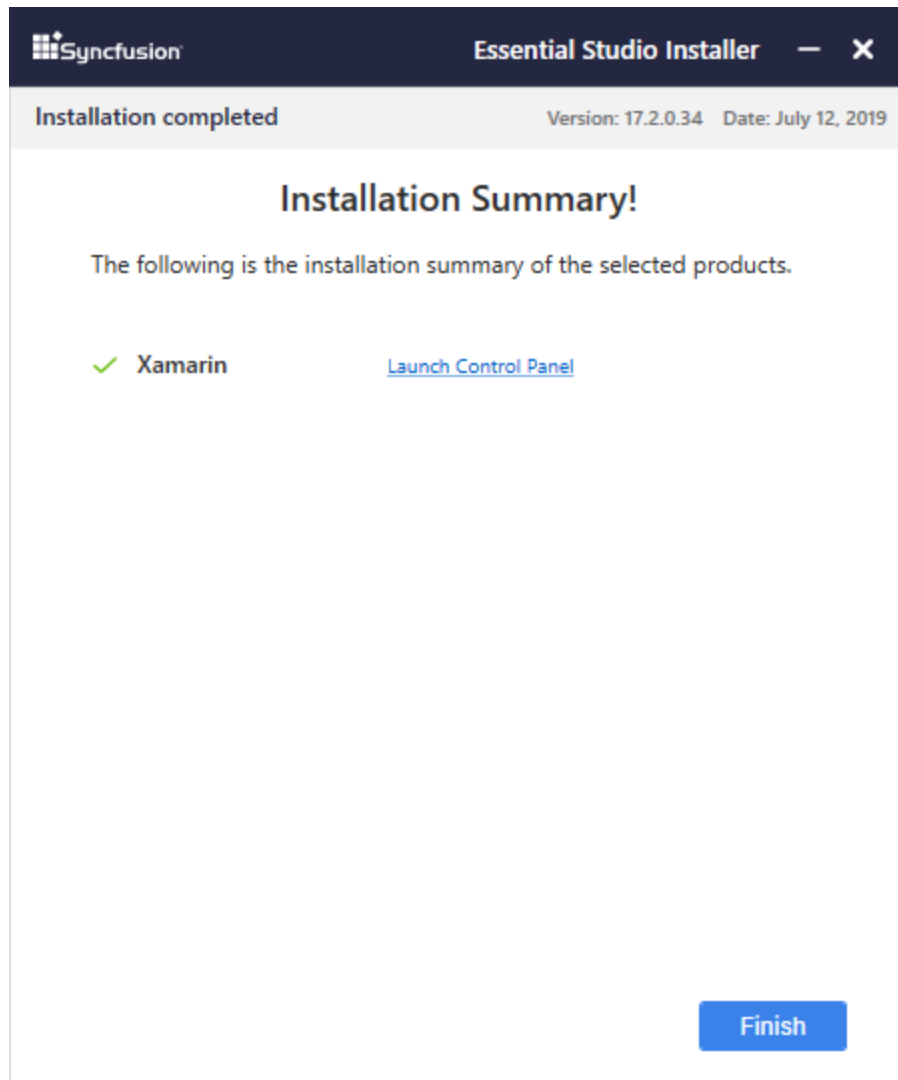
---

7. Download and Installation progress will be displayed.



8. Once the Installation is complete, **Installation Summary** wizard will be displayed. Here you can check the list of platforms which are installed successfully and failed. Click Finish to exit the Installation Summary wizard.





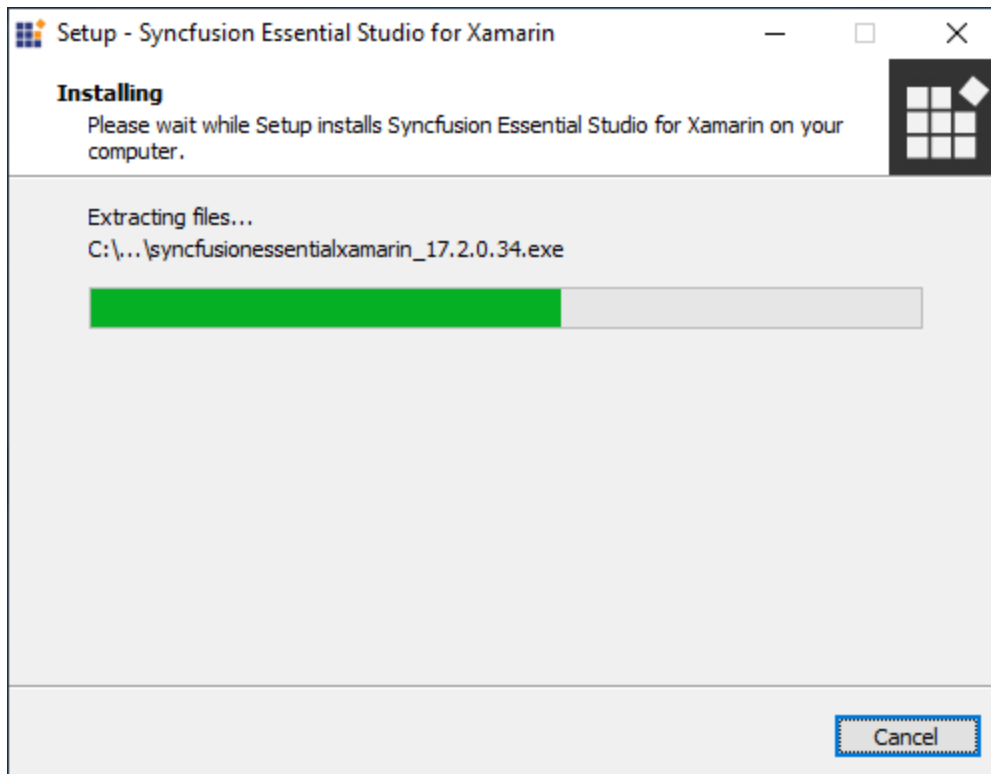
- Click **Launch Control Panel** to open the Syncfusion Control Panel.

## Installation using Offline Installer

### Installing with UI

The following procedure illustrates how to install Essential Studio Xamarin platform.

1. Close all the running Visual Studio instances.
2. Double-click the Syncfusion Xamarin platform Setup file. The Setup Wizard opens and extracts the package automatically.



---

**Note:** The Setup wizard extracts the syncfusionessentialxamarin\_(version).exe dialog, displaying the unzip operation of the package.

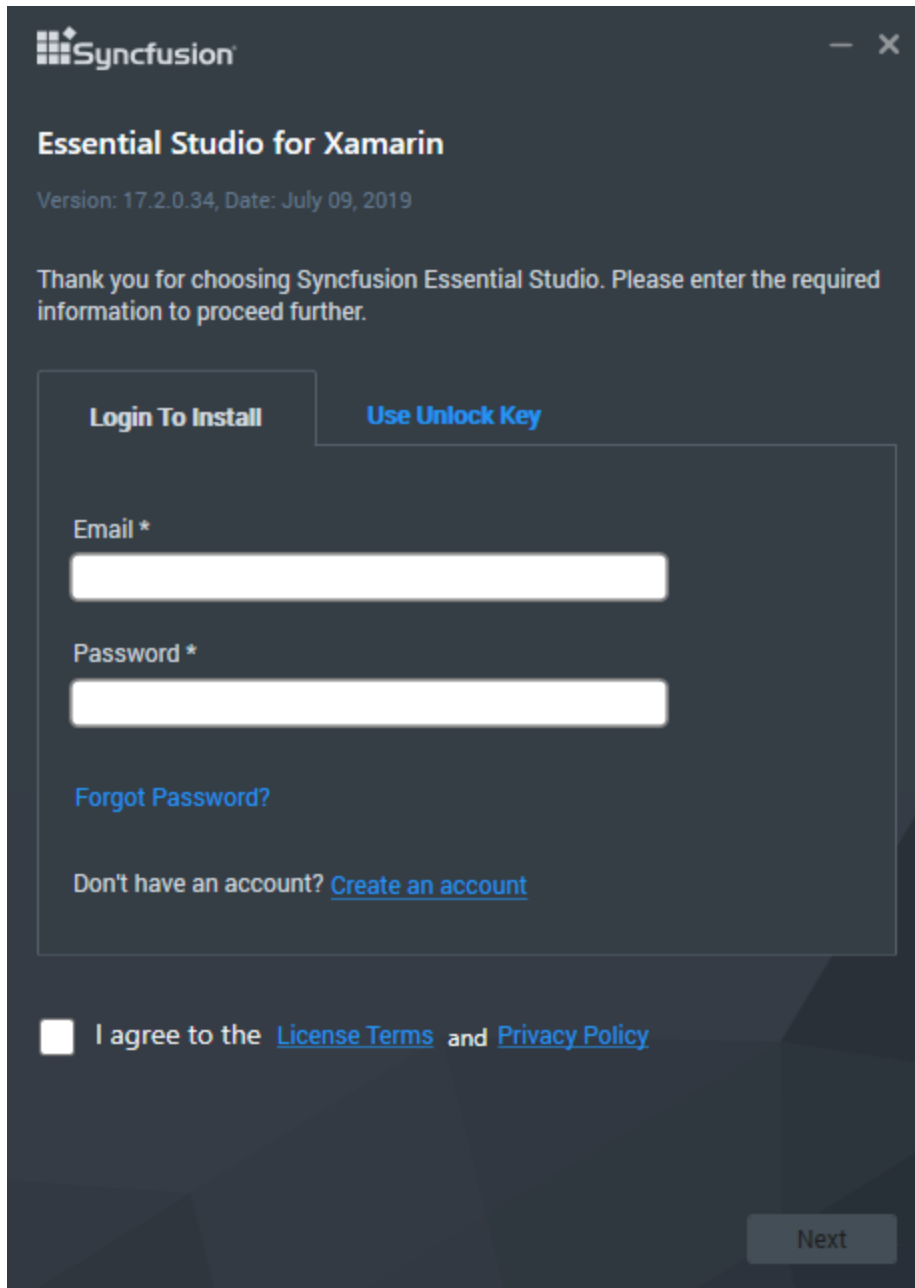
---

3. You have two options to unlock the Syncfusion setup:

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

### Login To Install

You should enter your Syncfusion Direct-Trac login credentials. If you don't have Syncfusion Direct-Trac login credentials, then you can click on Sign Up to create a new account. Else if you forgot your password, click on Reset Password to create a new password. Here Email address and Password is validated and the Platform Selection window opens.



**Syncfusion**

## Essential Studio for Xamarin

Version: 17.2.0.34, Date: July 09, 2019

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

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

Email \*

Password \*

[Forgot Password?](#)

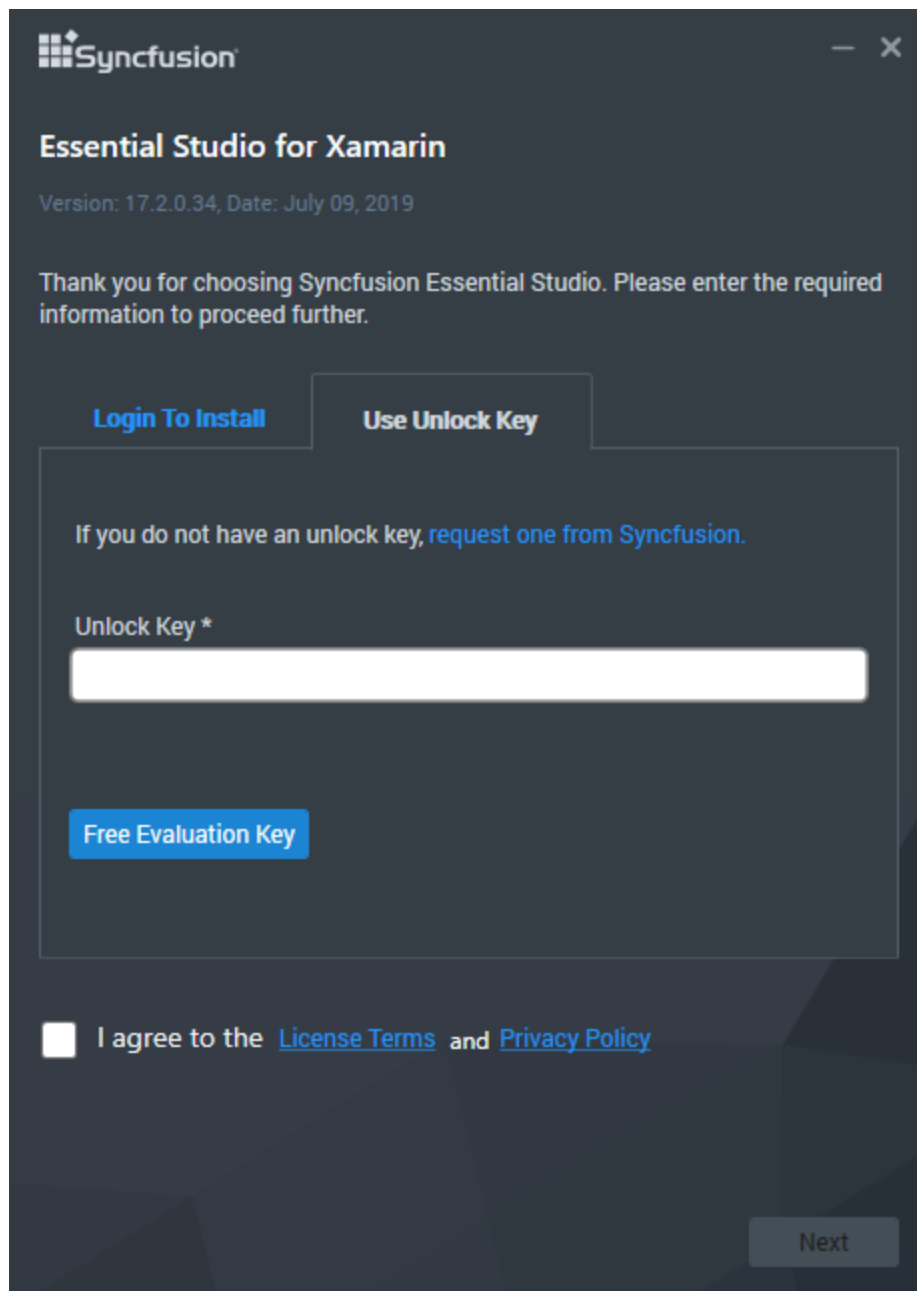
Don't have an account? [Create an account](#)

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

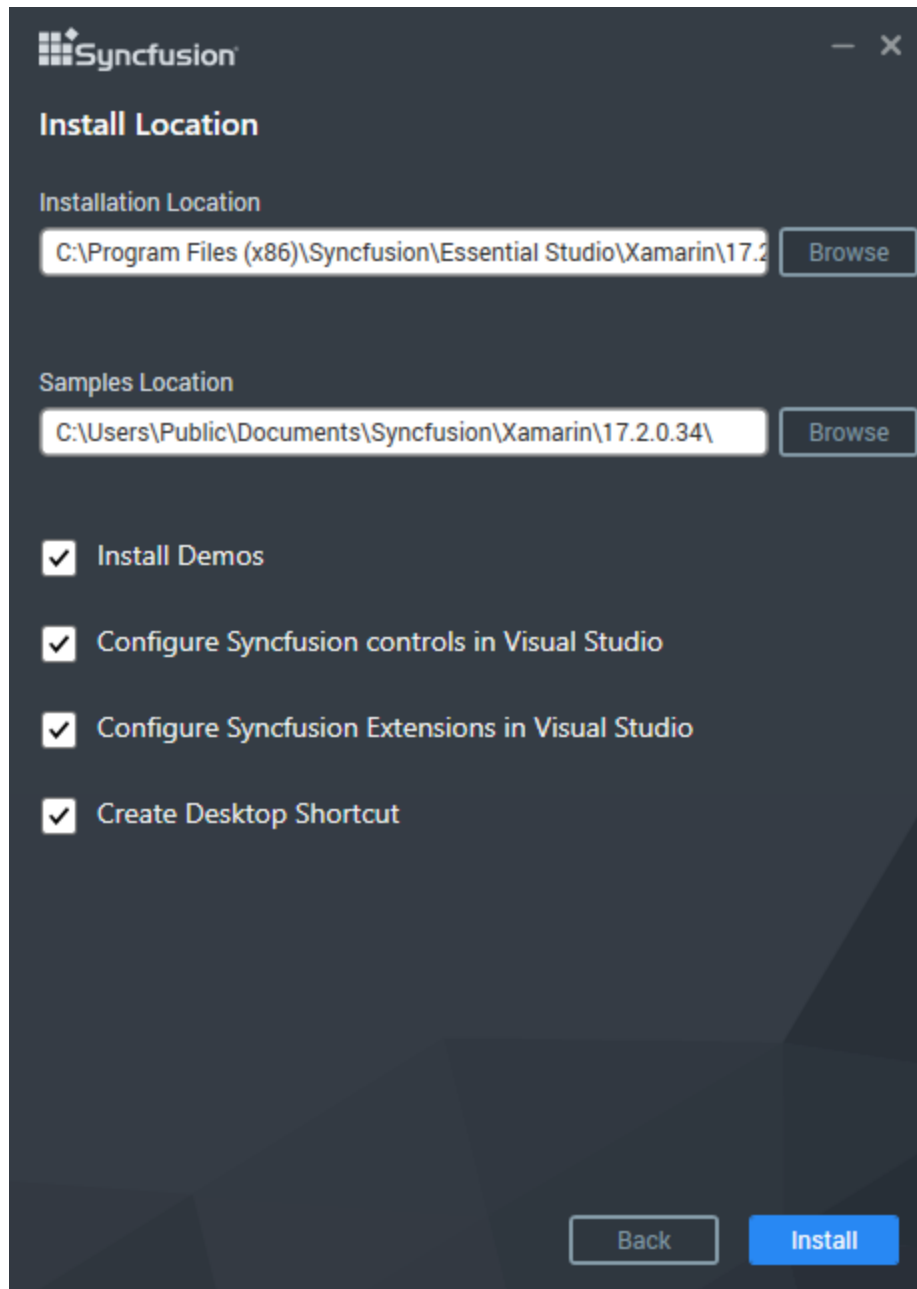
Next

### Use Unlock Key

You should use the Syncfusion License/Trial key. Trial key is valid for 30 days and the installer won't accept the expired trial key. Licensed customer can generate key from [here](#).



4. After reading the License Terms and Conditions, check the **I agree to the License Terms and Conditions** check box.
5. Click Next. Select the Installation, Samples Folder and Advanced Options screen opens. To install in the displayed default location, click Install



---

**Note:** From the 2018 Volume 2 release, Syncfusion has changed the install and samples location

---

**Default Install location:** {ProgramFilesFolder}\Syncfusion\{Platform}\{version}

**Default Samples location:** C:\Users\Public\Documents\Syncfusion\{platform}\{version}

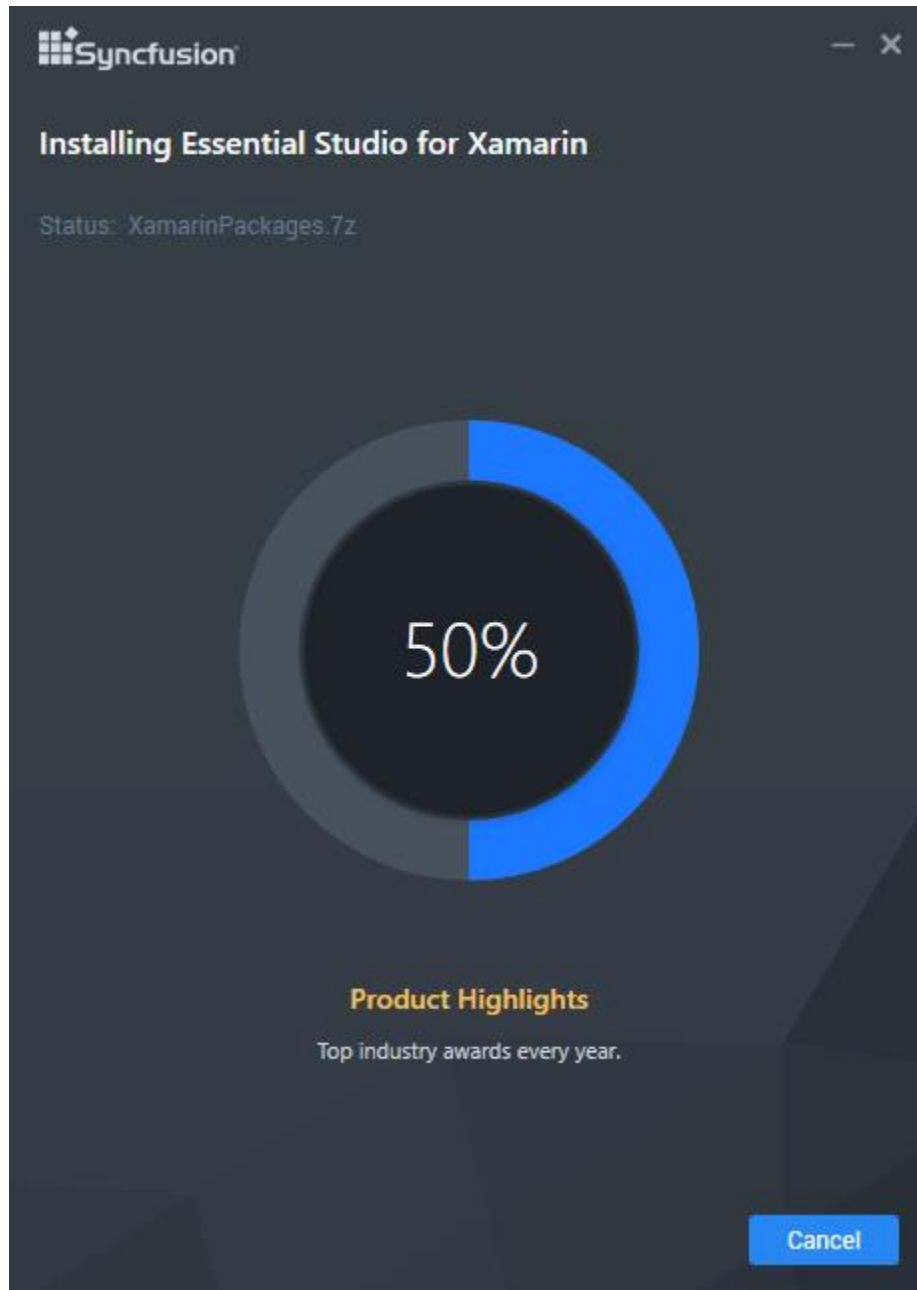
However, you can change the locations by clicking browse button.

- Select the **Install Syncfusion Samples** check box to install Syncfusion samples, or leave the check box clear, when you do not want to install Syncfusion samples.
- Select the **Configure Syncfusion controls in Visual Studio Toolbox** check box to configure the Syncfusion controls in the Visual Studio toolbox, or clear this check box when you do not want to configure the Syncfusion controls in the Visual Studio toolbox during setup installation. Note

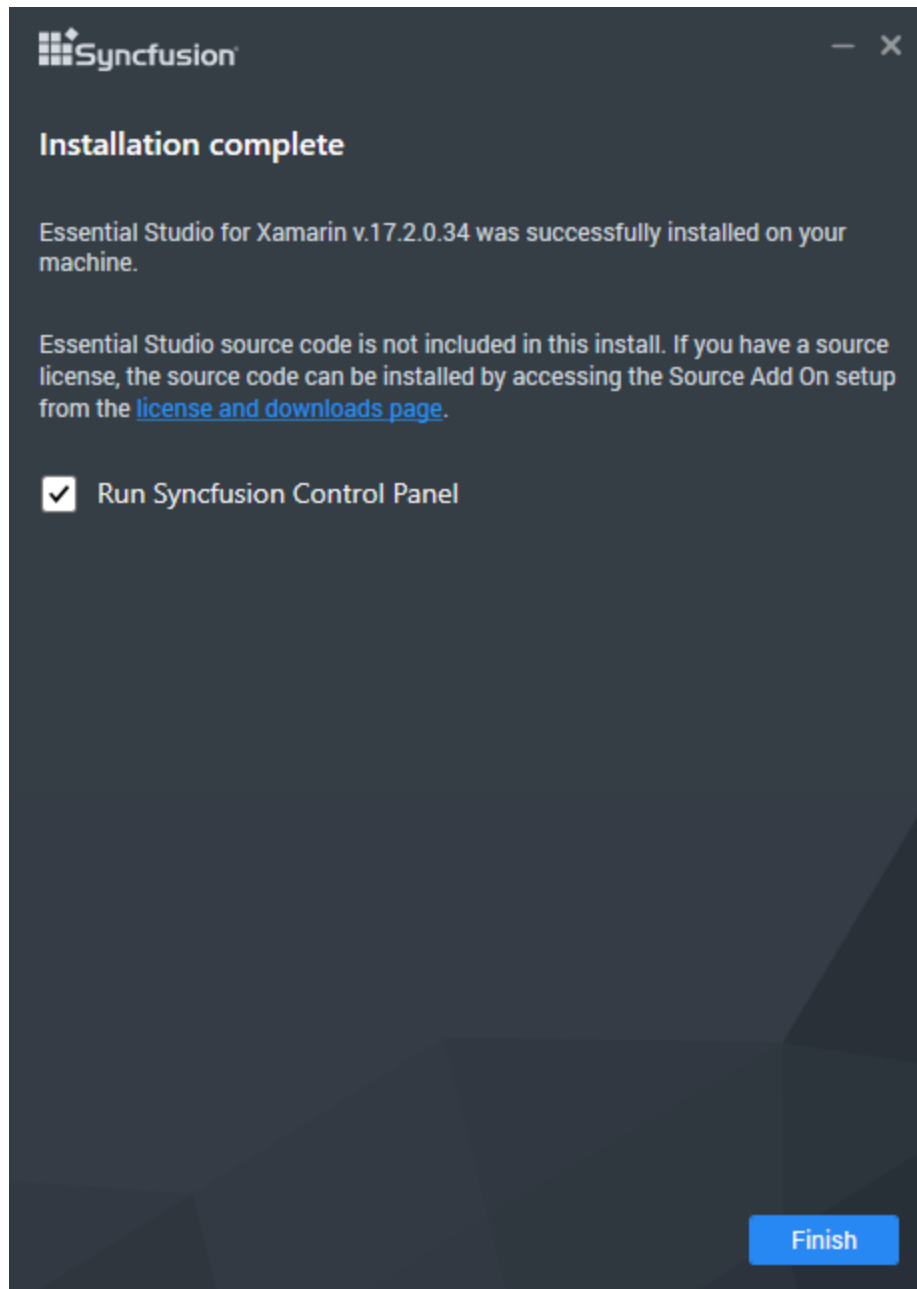
that you must also select the Register Syncfusion assemblies in GAC check box when you select this check box.

- Select the **Install Syncfusion Extensions** checkbox to configure the Syncfusion Extensions in Visual Studio or clear this check box when you do not want to configure the Syncfusion Extensions in Visual Studio.

6. Click Install.



7. The Completed screen is displayed once the Xamarin platform is installed.



8. Select the **Run Syncfusion Control Panel** check box to launch the Syncfusion Control Panel after installing.
9. Click Finish. Syncfusion Xamarin platform is installed in your system and the Syncfusion Essential Studio [Syncfusion Control Panel](#) is launched automatically.

#### Installing in silent mode

The Syncfusion Essential Studio Platform Installer supports installing/uninstalling the setup through Command Line. The following sections illustrate this ability.

### Command Line Installation

Follow the steps below to install through Command Line in Silent mode.

1. Double-click the Syncfusion Essential Studio platform setup file. The Setup Wizard opens and extracts the package automatically.
2. The syncfusionessentialxamarin\_(version).exe file is extracted into the Temp folder.
3. Run %temp%. The Temp folder will open. The syncfusionessentialxamarin\_(version).exe file is available in one of the folders.
4. Copy the extracted syncfusionessentialxamarin\_(version).exe file in local drive.
5. Cancel the Wizard.
6. Open the Command Prompt in administrator mode and pass the following arguments.

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

---

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

---

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

7. Setup is installed.

---

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

---

### Command Line Uninstallation

Syncfusion Essential Studio supports uninstalling the setup through Command Line in Silent mode. The following steps illustrate this.

1. When you do not have the extracted setup (syncfusionessentialxamarin\_(version).exe) then follow the steps from 2 to 7.
2. Double-click the Syncfusion Essential Studio platform setup file. The Setup Wizard opens and extracts the package automatically.
3. The syncfusionessentialxamarin\_(version).exe file is extracted into the Temp folder.
4. Run %temp%. The Temp folder will open. The syncfusionessentialxamarin\_(version).exe file is available in one of the folders.
5. Copy the syncfusionessentialxamarin\_(version).exe file in local drive.
6. Cancel the Wizard.
7. Open the Command Prompt in administrator mode and pass the following arguments.

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

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

8. Setup is uninstalled.



---

**Note:** x.x.x.x needs to be replaced with the Essential Studio version installed in your machine.

---

### Explore the libraries package

You can find the Syncfusion libraries, samples and NuGet from the installed location in Windows.

{Essential Studio installed location}\Syncfusion\Essential Studio\{version}\Xamarin

- **“lib”** folder - e.g., C:\Program Files (x86)\Syncfusion\Essential Studio\16.2.0.41\Xamarin\lib

It contains all the required libraries for Xamarin.iOS, Xamarin.Android, and Xamarin.Forms projects.

- **“nuget”** folder - e.g., C:\Users\Public\Documents\Syncfusion\16.2.0.41\Xamarin\nuget

It contains the above libraries as NuGet packages. The same NuGet packages also can be configured from online [nuget.org](http://nuget.org).

- **“sample”** folder - e.g., C:\Users\Public\Documents\Syncfusion\16.2.0.41\Xamarin\sample

It contains the sample applications for our controls in Xamarin.iOS, Xamarin.Android, and Xamarin.Forms platforms in iOS, Android, and Forms folders, respectively.

The "Forms" directory includes,

- Individual control sample folders: It contains the samples for individual controls such as SfChart, SfDataGrid, etc. Since they represent the individual controls, these samples are light-weighted. You can check the samples for your required controls alone faster with minimum deployment time.
- **“nuget”** folder: It contains the compiled assemblies of the above samples as NuGet package. It is referred in the common sample browser as explained in the next step.
- **“SampleBrowser”** folder: It contains common sample browser, which refers the individual control's samples as NuGet package. Run this to see the demo samples of all the controls in single application.
- It also contains showcase samples such as Patient Monitor, Server Monitor, and Invoice.

### Add reference to the project

You can then add the assembly references to the respective projects such as PCL, XForms.Droid, XForms.iOS and XForms.UWP. You can find the dependencies for each control from this [link](#).

---

**Information:** After adding the reference, currently, an additional step is required for iOS and UWP projects. For example, if we are using SfKanban, we need to call the Init method of SfKanbanRenderer as shown in this [KB article](#).

---

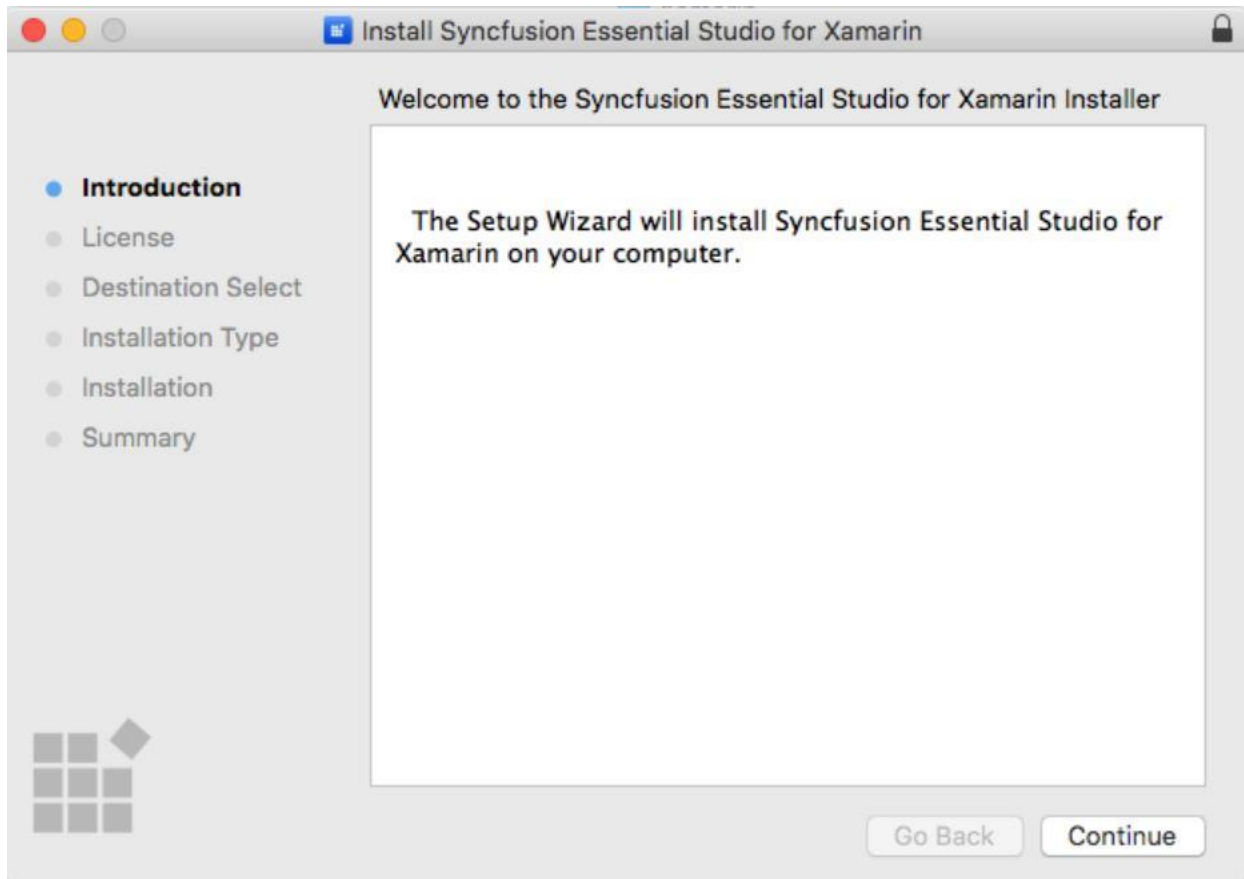
**Information:** For UWP alone, one more additional step is required if the project is built in release mode with .NET Native tool chain enabled. For example, if we are using SfKanban, you can refer the [KB article](#) for more details.

---

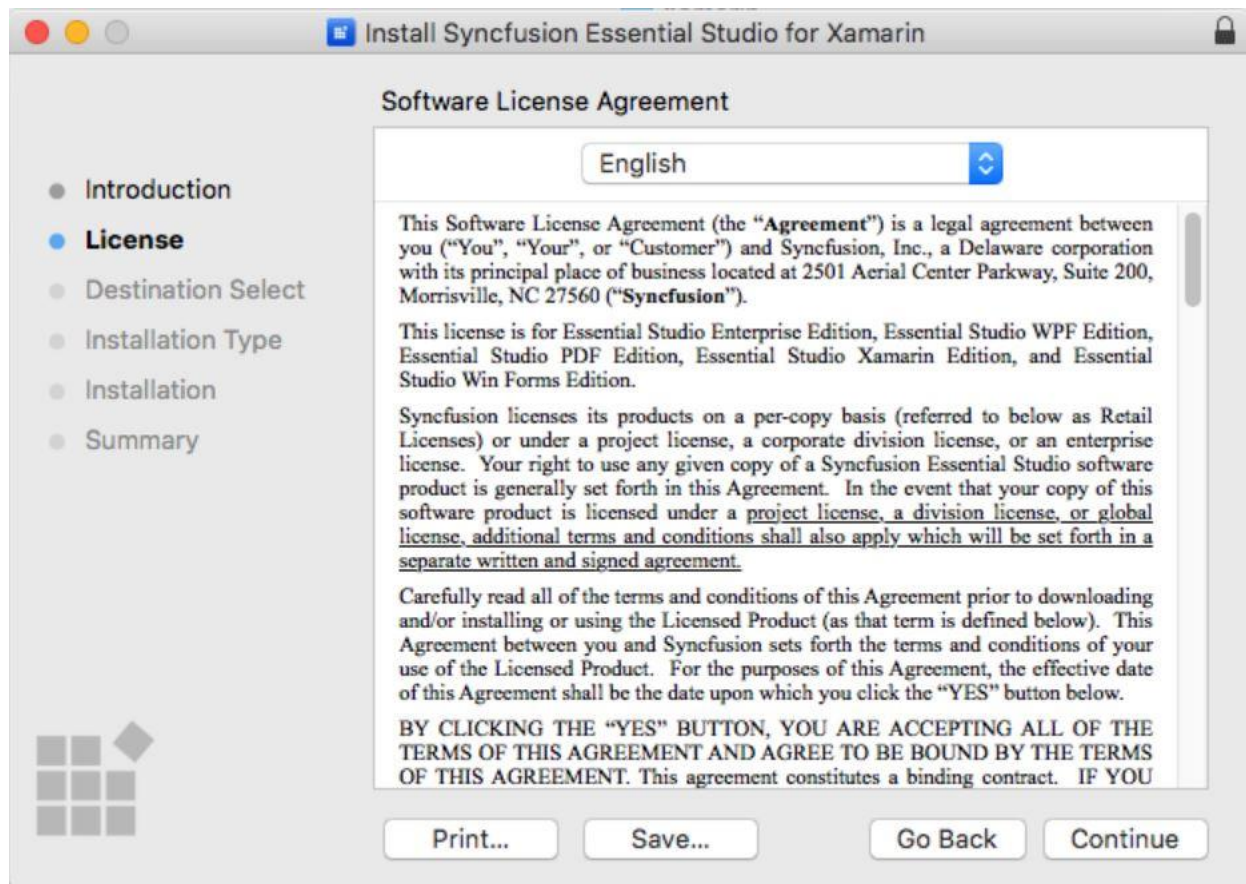
### Mac Installation

The following procedure illustrates how to install Xamarin Mac installer.

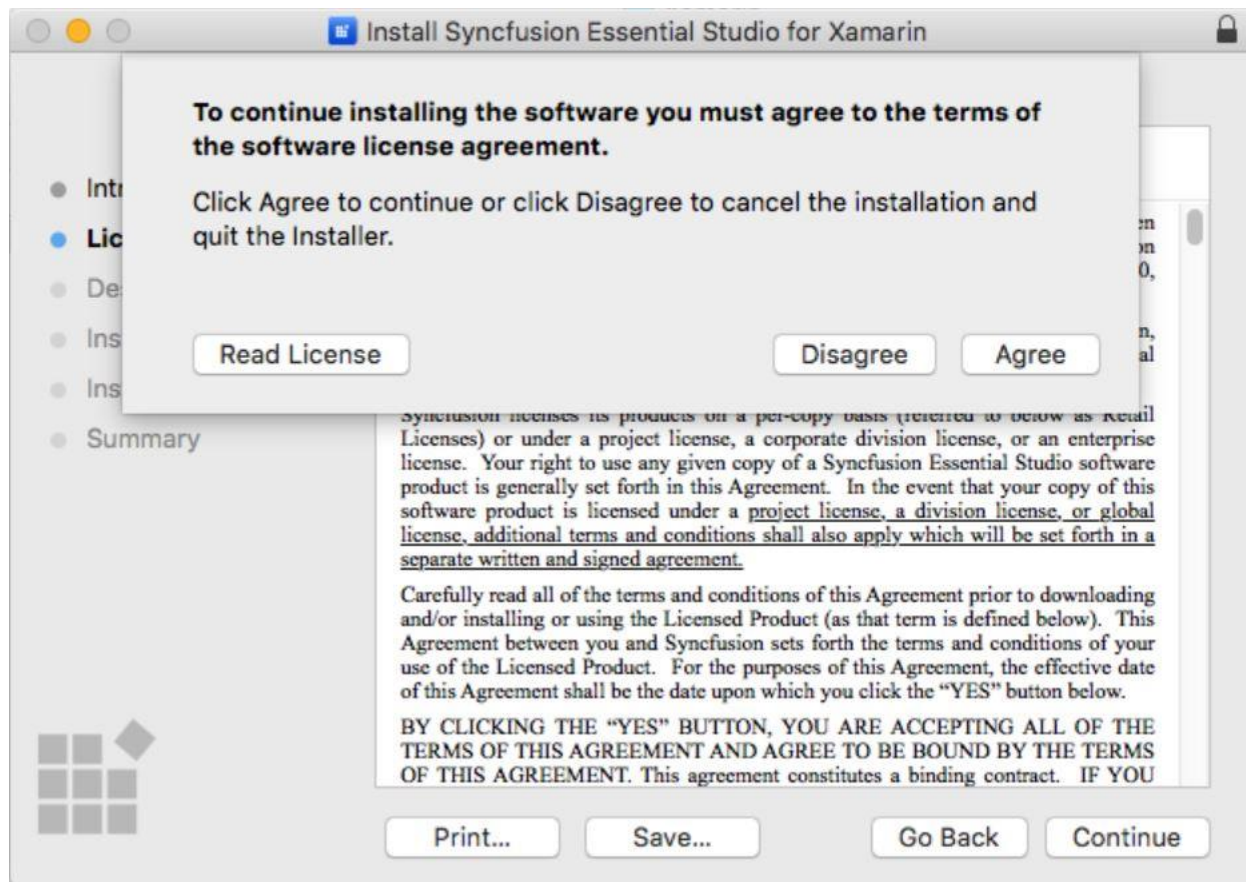
1. Double-click the Syncfusion Xamarin Mac Setup(.pkg) file. The Setup Wizard opens. Click Continue.



2. Software License Agreement window opens. Click Continue.

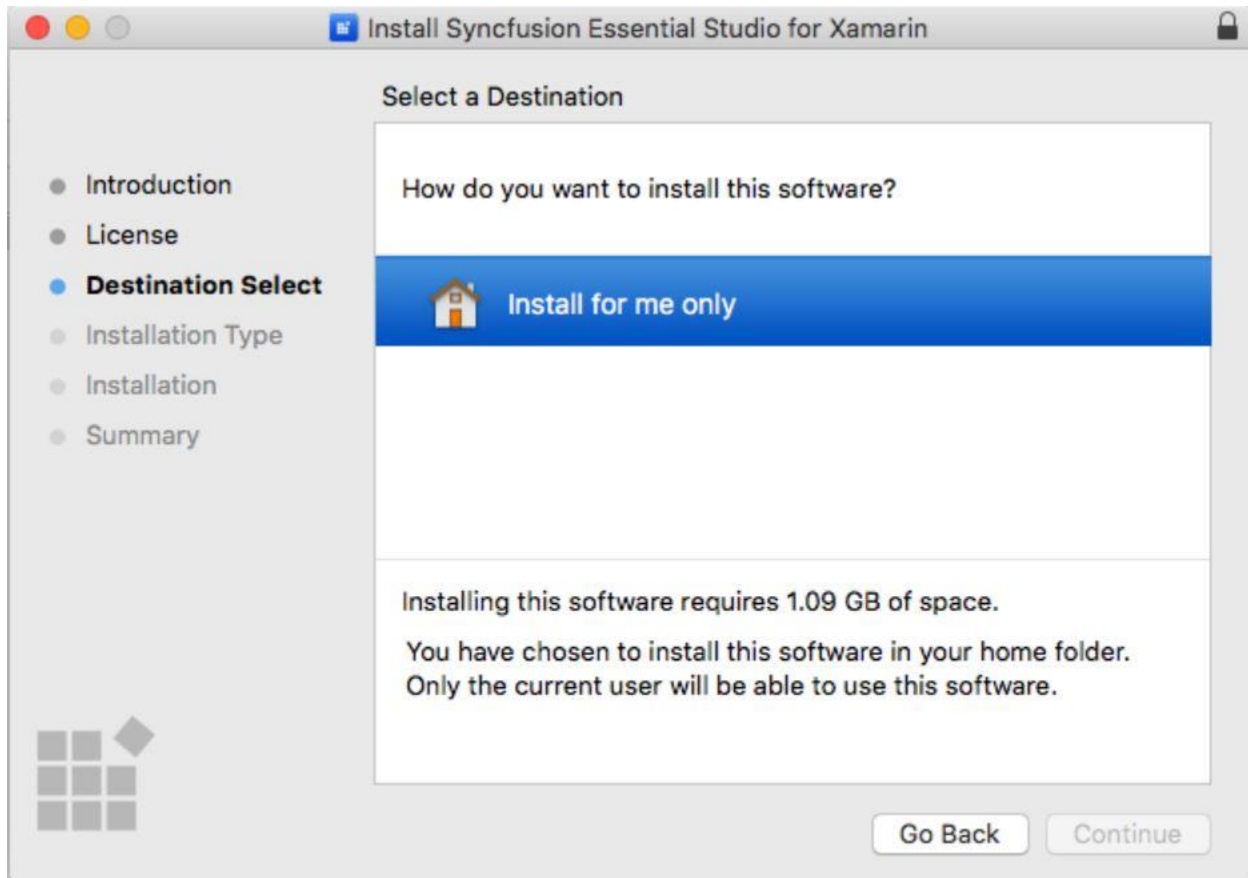


3. Confirmation window will be displayed for the License Agreement. Click **Agree**.

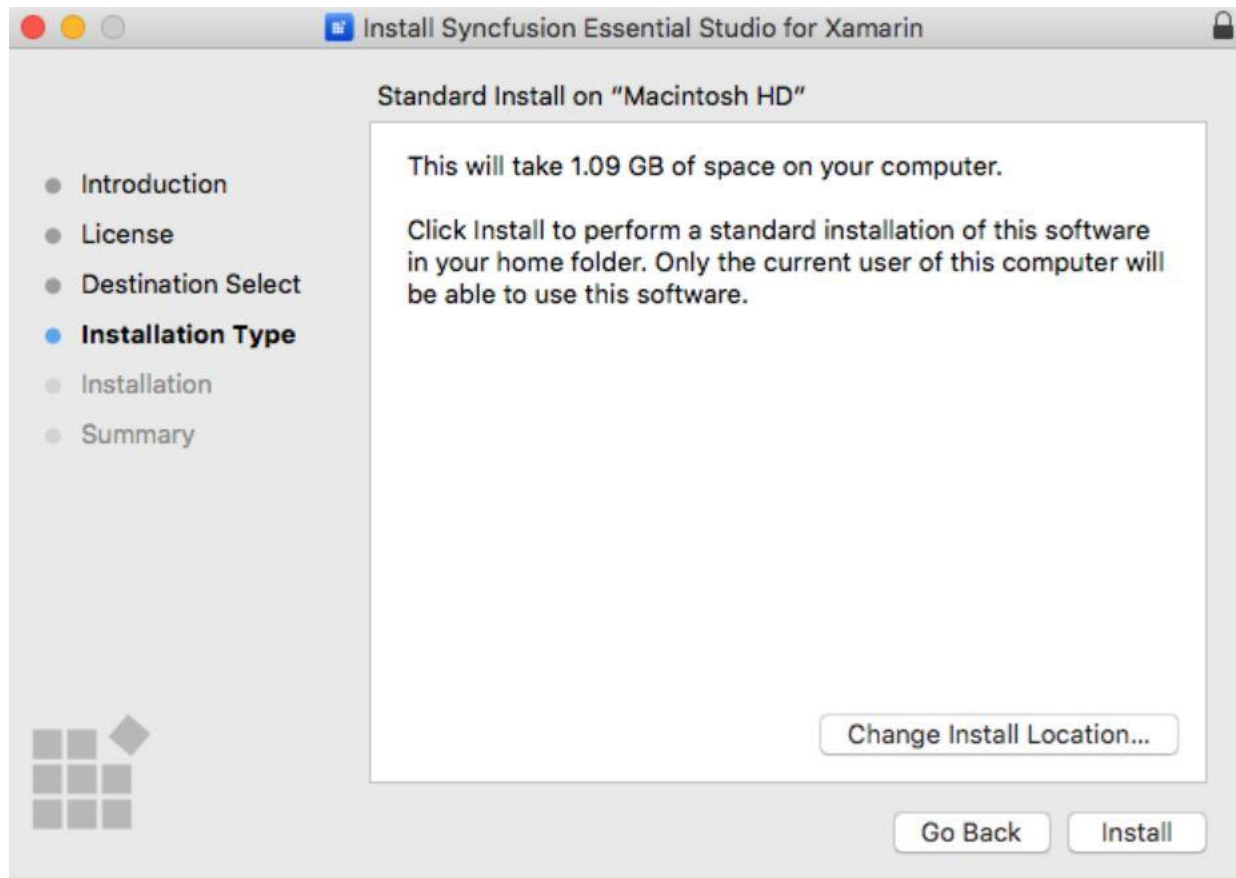


**Note:** Unlock key is not required for installing the Mac installer. Syncfusion Mac installer can be used for developing purposes without registering the Unlock key.

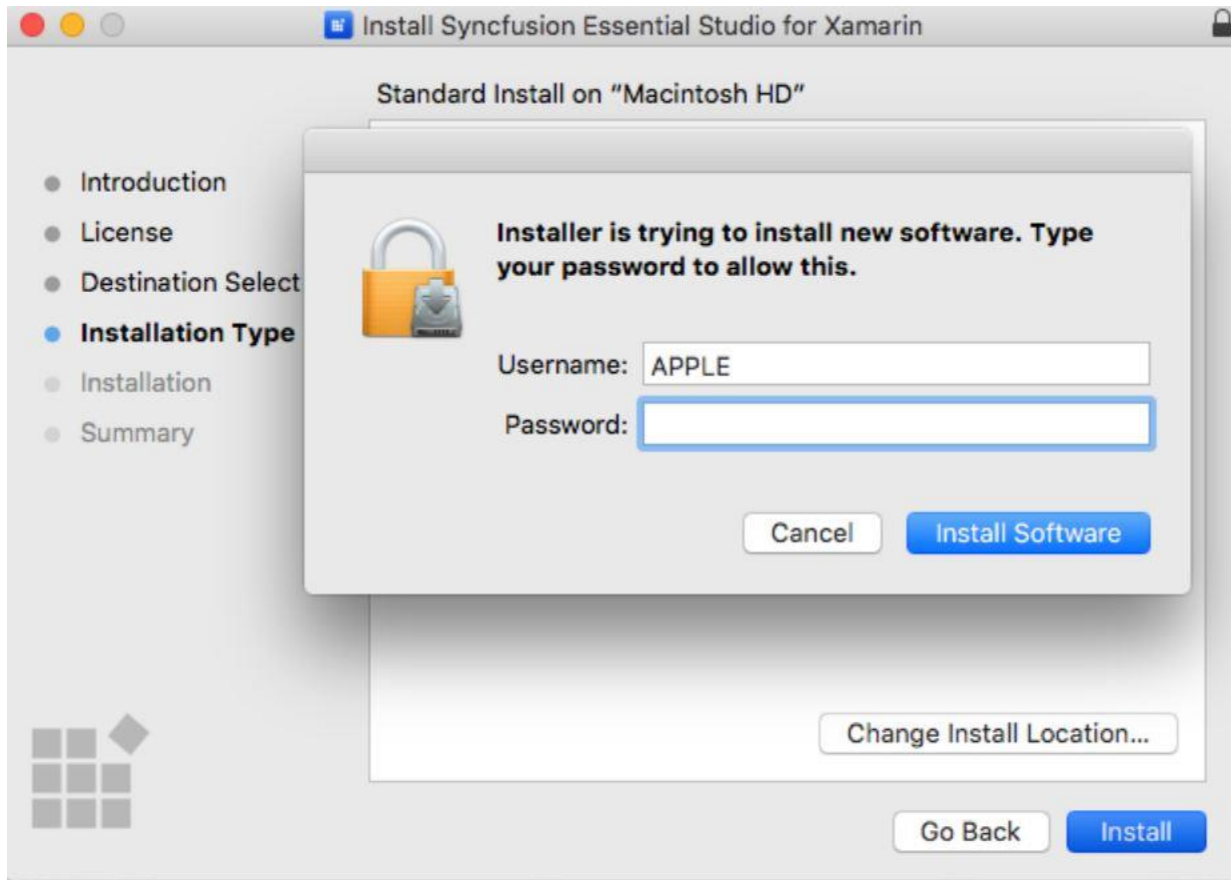
4. Destination Select windows opens. Click Continue.



5. Installation Type window opens. Click Install.

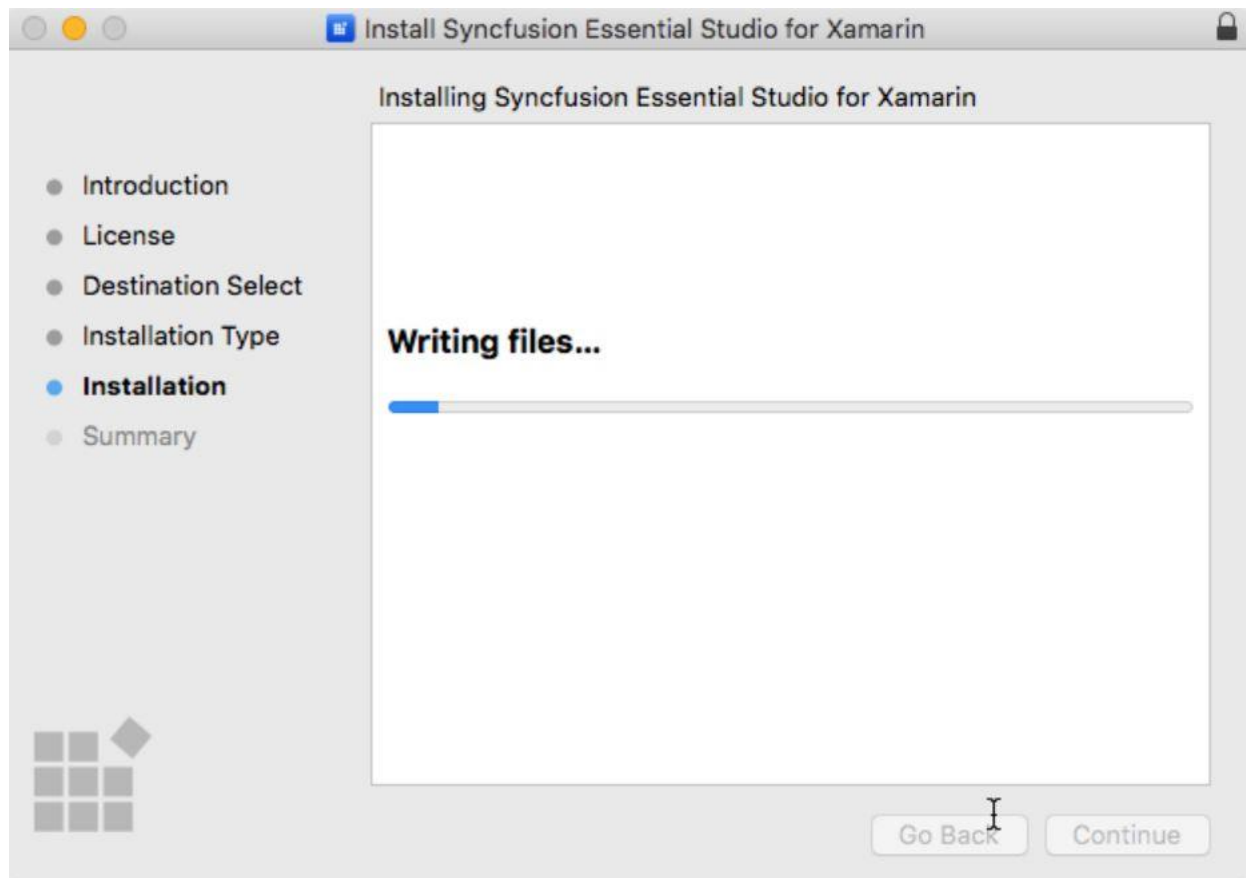


6. Authentication window opens. Provide the password and click **Install Software**



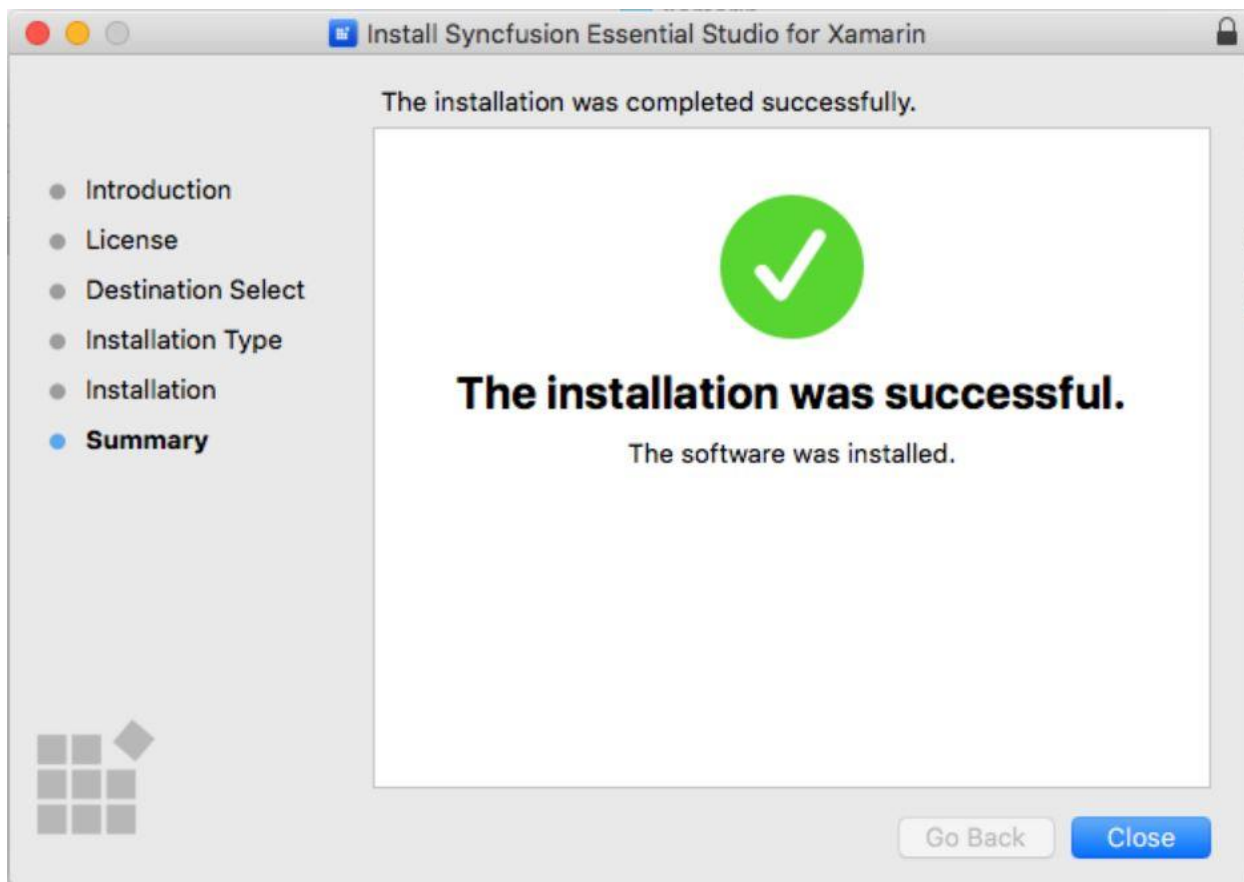
7. Installation will be started in your machine.





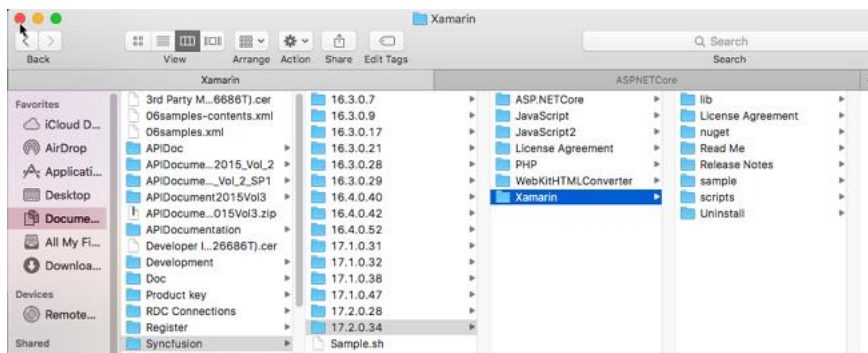
8. Completed screen will be displayed once the installation is finished. Click Close to exit the installation wizard.





By default, Mac installer will install the files in following location.

**Location:** {Documents}\Syncfusion\{version}\{platform}



#### [Explore the libraries package](#)

You can find the Syncfusion libraries, samples and NuGet from the installed location in Mac.

{Essential Studio installed location}\Syncfusion\Essential Studio\{version}\Xamarin

- **“lib”** folder - e.g., /Users/labuser/Documents/Syncfusion/16.2.0.41/Xamarin/lib

It contains all the required libraries for Xamarin.iOS, Xamarin.Android, and Xamarin.Forms projects.

- **“nuget”** folder - e.g., /Users/labuser/Documents/Syncfusion/16.2.0.41/nuget

It contains the above libraries as NuGet packages. The same NuGet packages also can be configured from online [nuget.org](https://nuget.org).

- **"sample"** folder - e.g., /Users/labuser/Documents/Syncfusion/16.2.0.41/sample

It contains the sample applications for our controls in Xamarin.iOS, Xamarin.Android, and Xamarin.Forms platforms in iOS, Android, and Forms folders, respectively.

The "Forms" directory includes,

- Individual control sample folders: It contains the samples for individual controls such as SfChart, SfDataGrid, etc. Since they represent the individual controls, these samples are light-weighted. You can check the samples for your required controls alone faster with minimum deployment time.
- "nuget" folder: It contains the compiled assemblies of the above samples as NuGet package. It is referred in the common sample browser as explained in the next step.
- "SampleBrowser" folder: It contains common sample browser, which refers the individual control's samples as NuGet package. Run this to see the demo samples of all the controls in single application.
- It also contains showcase samples such as Patient Monitor, Server Monitor, and Invoice.

#### *Add reference to the project*

You can then add the assembly references to the respective projects such as PCL, XForms.Droid, XForms.iOS. You can find the dependencies for each control from this [link](#).

---

**Information:** After adding the reference, currently, an additional step is required for iOS and UWP projects. For example, if we are using SfKanban, we need to call the Init method of SfKanbanRenderer as shown in this [KB article](#).

---

#### Installation FAQ

Refer [this](#) topic for more information regarding the issues related to installation.

#### Upgrade from one version to another version.

You can upgrade to the latest version by downloading and installing the platforms you require from [this](#) link.

#### Upgrade from major version to service pack version

Syncfusion provides a new Volume release once in every three months which has exciting new features. For that Volume release, there may be one or two Service Pack releases. The issues in the Volume release will be addressed in the Service Pack releases. You can download and install the latest Service Pack setup [here](#).

It is not required to install the Volume release setup before installing the Service Pack release setup. As Volume and Service Packs releases works independently, you can directly install the latest Service Pack setup which contains major issue fixes.

#### Upgrade from trial version to license version

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

- Uninstall the trial version and install the fully licensed build from the [License & Downloads](#) section of our website.

- Replace the currently used trial license key with a paid license key that can be generated from the [License & Downloads](#) section of our website. Refer to [this](#) topic for more information regarding registering the license in the application.

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

## Visual Studio Integration

### Overview

The Syncfusion Xamarin Visual Studio Extensions can be accessed through the Syncfusion Menu to create and configure the project with Syncfusion references in Visual Studio.

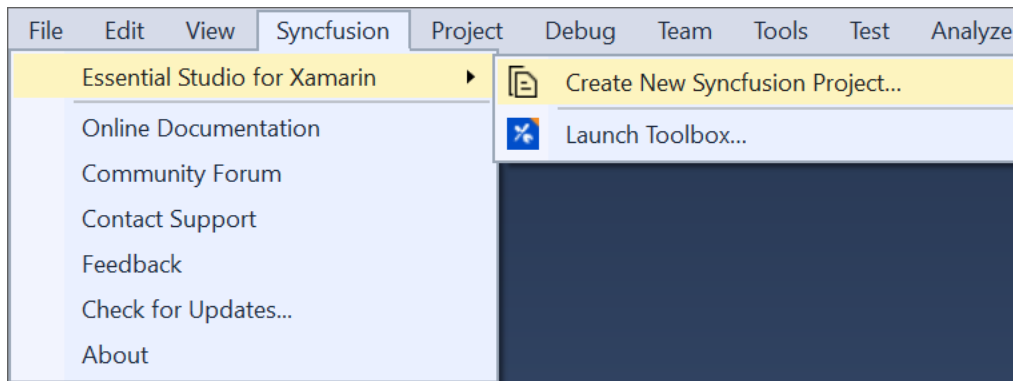
**Note:** Syncfusion Extension is published in Visual Studio Marketplace. You can download Xamarin Extensions [here](#).

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

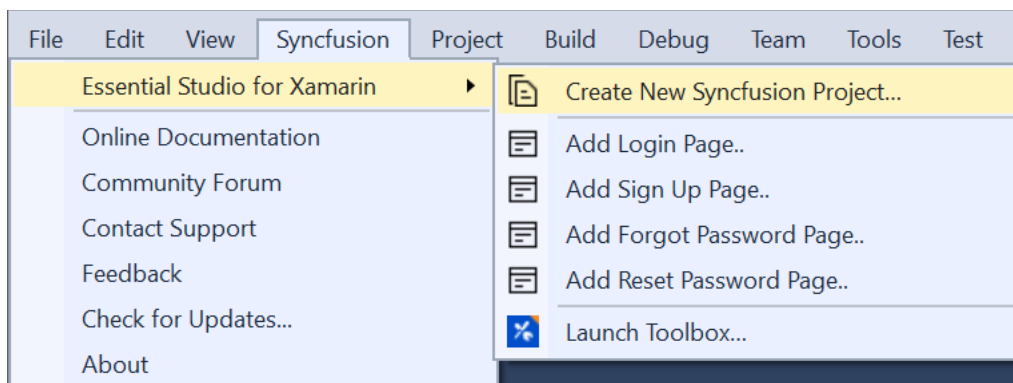
The Syncfusion provides the following extension supports in Visual Studio:

1. [Create Project](#): To create a Syncfusion Xamarin application by adding the required Syncfusion NuGet based on the control chosen.
2. [Item Template](#): Add the predefined Syncfusion items (Pages) in Xamarin.Forms and install the required Syncfusion NuGet packages.
3. [Toolbox](#): Add the Syncfusion Xamarin components in Xamarin.Forms designer from Syncfusion Toolbox.

### No project selected in Visual Studio



### Selected Microsoft/Syncfusion Xamarin (Xamarin.Forms) application in Visual Studio



---

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

---

## Visual Studio Extensions

### Create project

Syncfusion provides the Visual Studio Project Templates for the Syncfusion Xamarin platform to create the Syncfusion Xamarin Application by adding the required Syncfusion NuGet packages for the control chosen.

---

**Information:** The Syncfusion Xamarin Project Templates are available from v16.2.0.41.

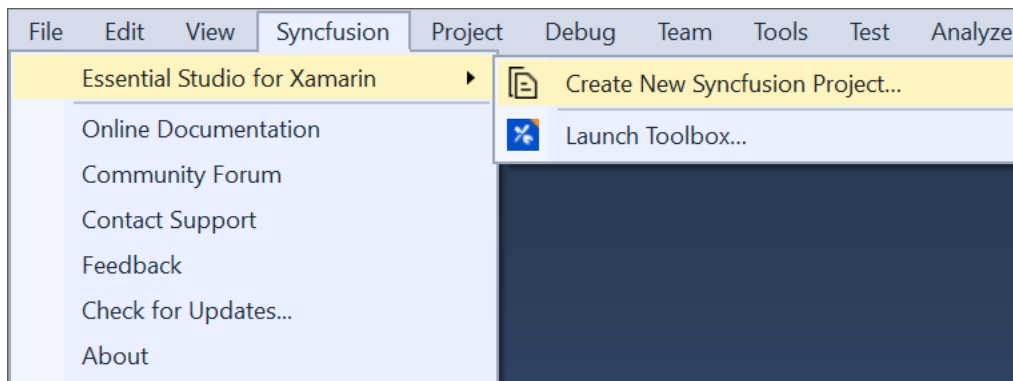
---

Use the following steps to create the **Syncfusion Xamarin Application** through the **Visual Studio 2017**:

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

#### Option 1:

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



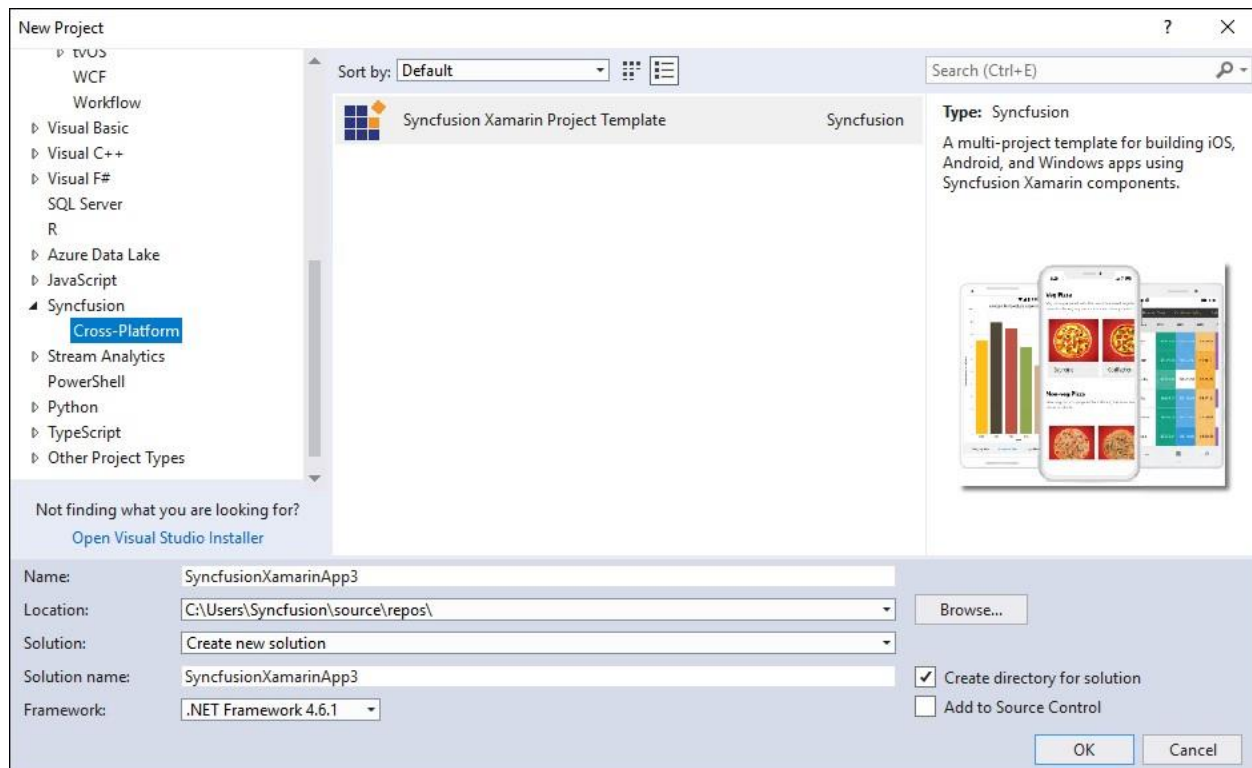
---

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

---

#### Option 2:

Choose **File > New > Project** and navigate to **Syncfusion > Cross-Platform > Syncfusion Xamarin Project Template** in **Visual Studio**.



2. Name the **Project**, choose the destination location, and set the Framework of the project, and then click **OK**. The Project Configuration Wizard appears.
3. Choose the options to configure the Syncfusion Xamarin Application by using the following Project Configuration dialog, choose the Project, Android, iOS, and UWP by on/off.

**New Project Configuration Wizard** Version 16.2.0.41

**Project Preference**

**Android** ☒

Minimum Android Version: Android 7.1 (API Level 25)

Target Android Version: Use Compile using SDK version

**iOS** ☒

Target Device: Universal

Target Version: 11.3

**UWP** ☒

Assemblies From: NuGet

**Choose controls**

Search here

- ☐ Autocomplete
- ☐ Bar Code
- ☐ Busy Indicator
- ☐ Calendar
- ☐ Carousel
- ☐ Chart
- ☐ Checkbox
- ☐ Circular Gauge
- ☐ Combo Box

Total Selected Controls: 0 [Clear All](#)

[Syncfusion](#) | [Help](#) | [Support](#) **Create** **Cancel**

### Project Configuration:

**Assemblies From:** Load the Syncfusion Xamarin reference to Xamarin Application, either NuGet or Installed Location.

**Note:** Installed location option will be available only when the Syncfusion Xamarin setup has been installed.

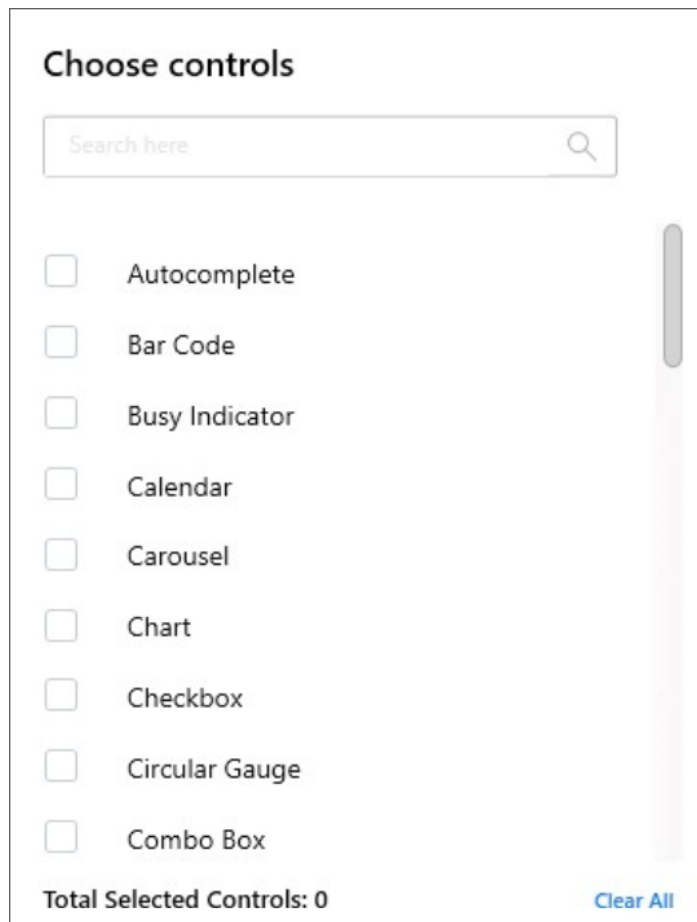
### Android

1. **Minimum Android Version:** Select the oldest Android version that you want to support your application.
2. **Target Android Version:** Select the version of Android to run your application.

### iOS

1. **Target Device:** Select the device of Xamarin.iOS project either Unified, iPhone/iPod, or iPad. 2. **Target Version:** Choose the version of Xamarin.iOS Project.

**Choose controls:** Choose the Xamarin application needs to create with the Syncfusion controls.



4. Click **Create**, the Syncfusion Xamarin Application has been created.

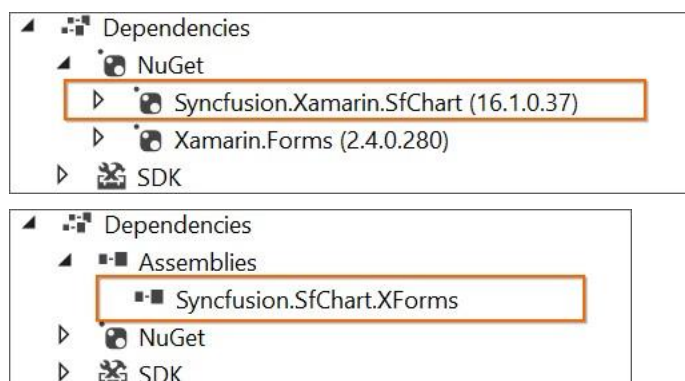
---

**Note:** Choose any one of the project type and controls from Project Configuration Wizard.

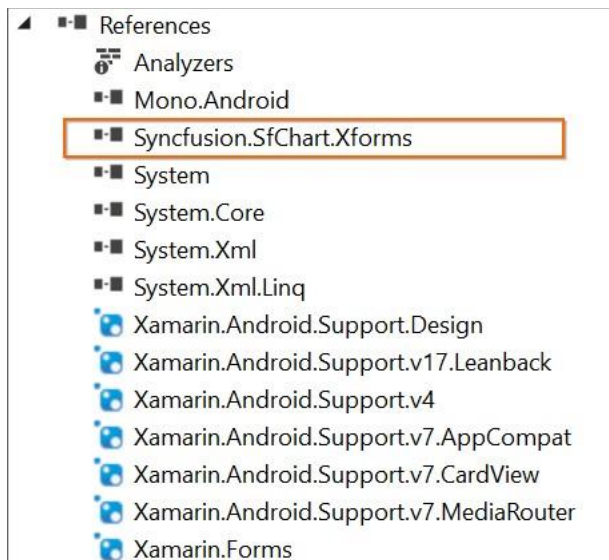
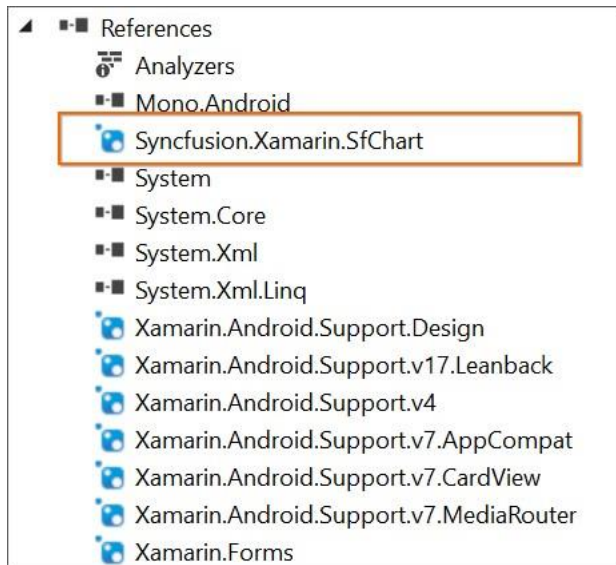
---

5. Required Syncfusion NuGet/Assemblies and configuration have been added to the project based on the control chosen.

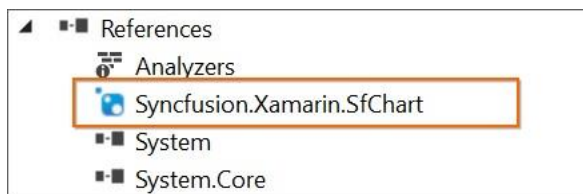
#### Net Standard /PCL



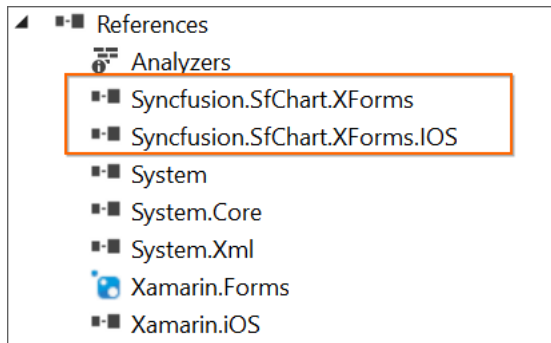
#### Android



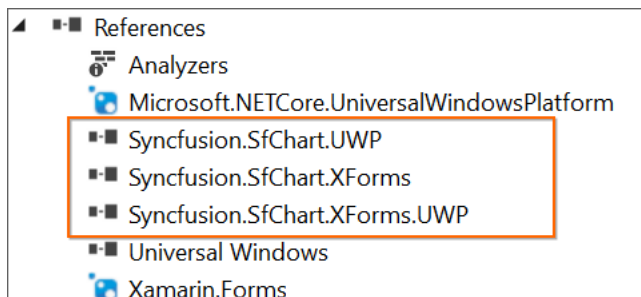
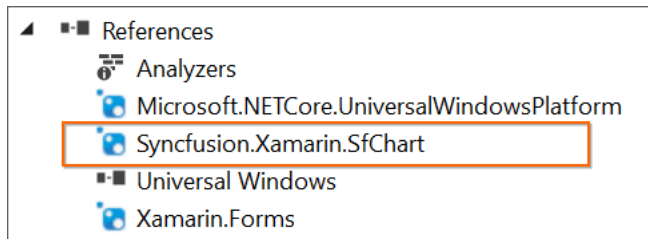
## iOS



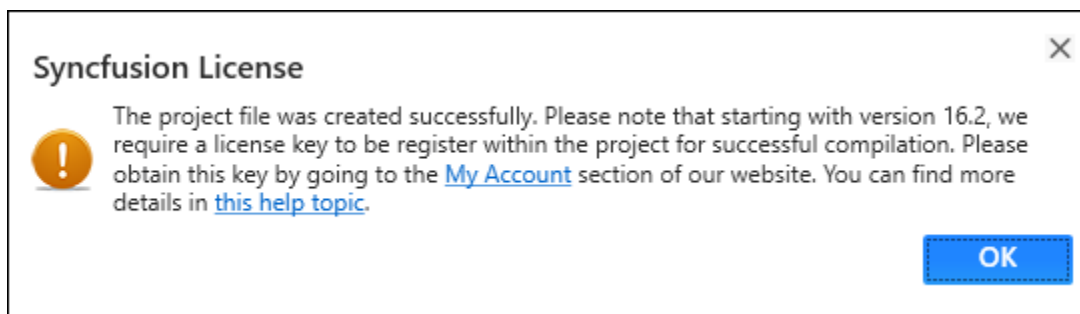




## UWP



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



## Toolbox control

Syncfusion provides the customized **Visual Studio Toolbox** for the Syncfusion Xamarin platform to add the Syncfusion Xamarin (Xamarin.Forms) controls in your project. It supports Microsoft Visual Studio 2017 and later. The Syncfusion Xamarin toolbox helps you use the Syncfusion controls without coding in the Visual Studio designer.

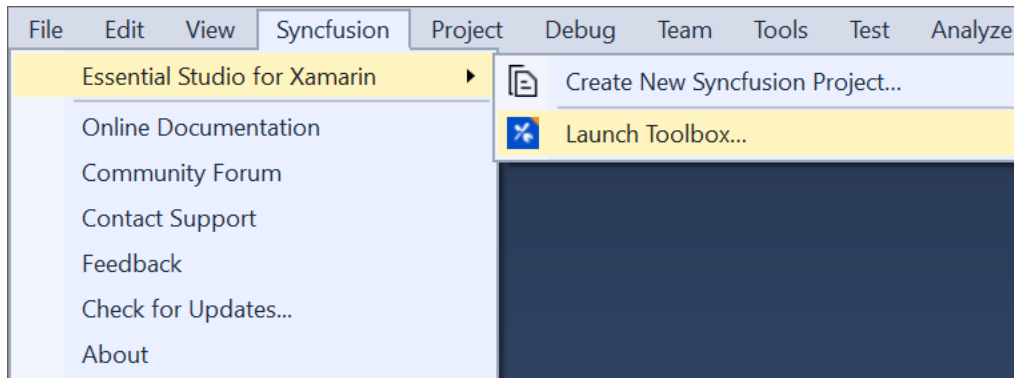
**Information:** The Syncfusion Xamarin Toolbox is available from v16.2.0.41.

Create the Xamarin or Syncfusion Xamarin project. The following steps help you add the Syncfusion controls through the Visual Studio Toolbox:

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

**Option 1:**

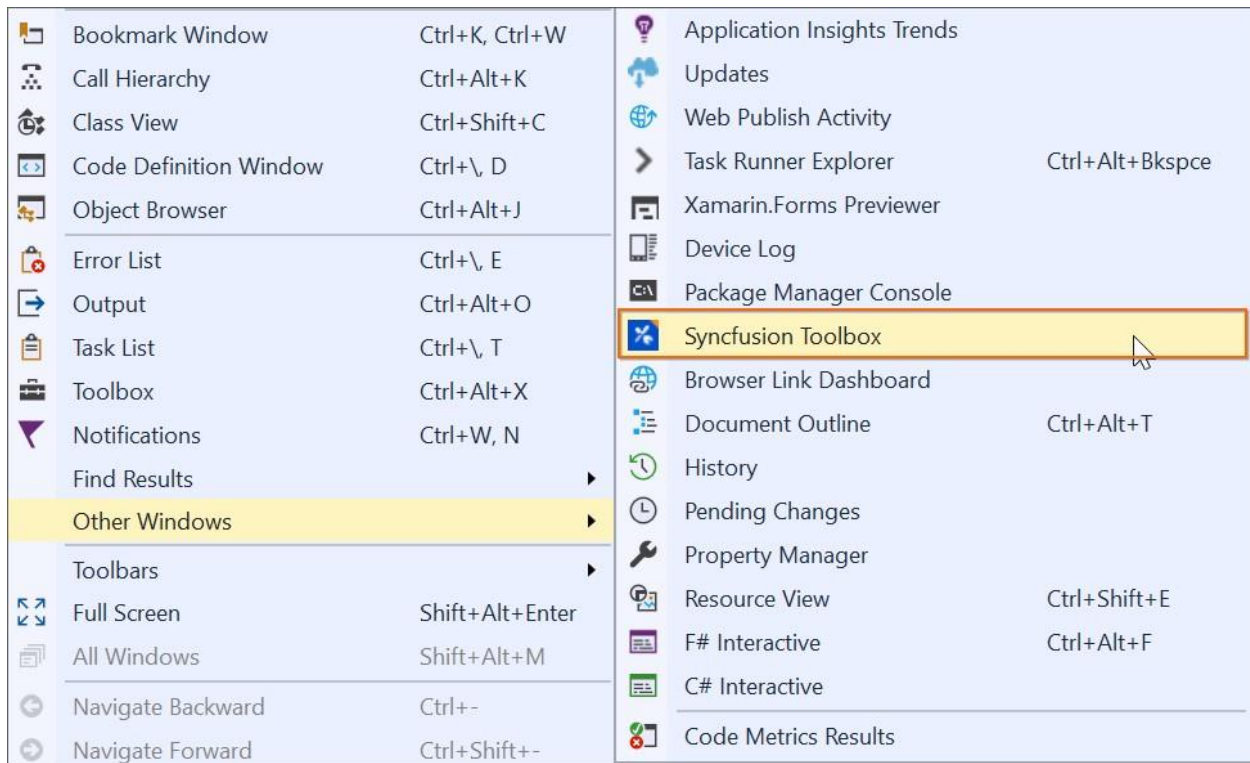
Click **Syncfusion Menu** and choose **Essential Studio for Xamarin > Launch Toolbox...** in **Visual Studio**.



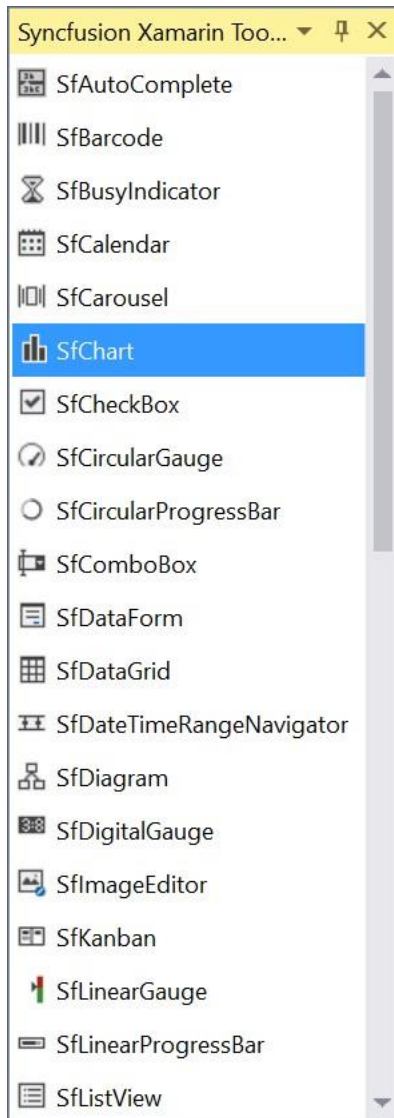
**Note:** In Visual Studio 2019, Syncfusion menu is available under Extensions in Visual Studio menu.

**Option 2:**

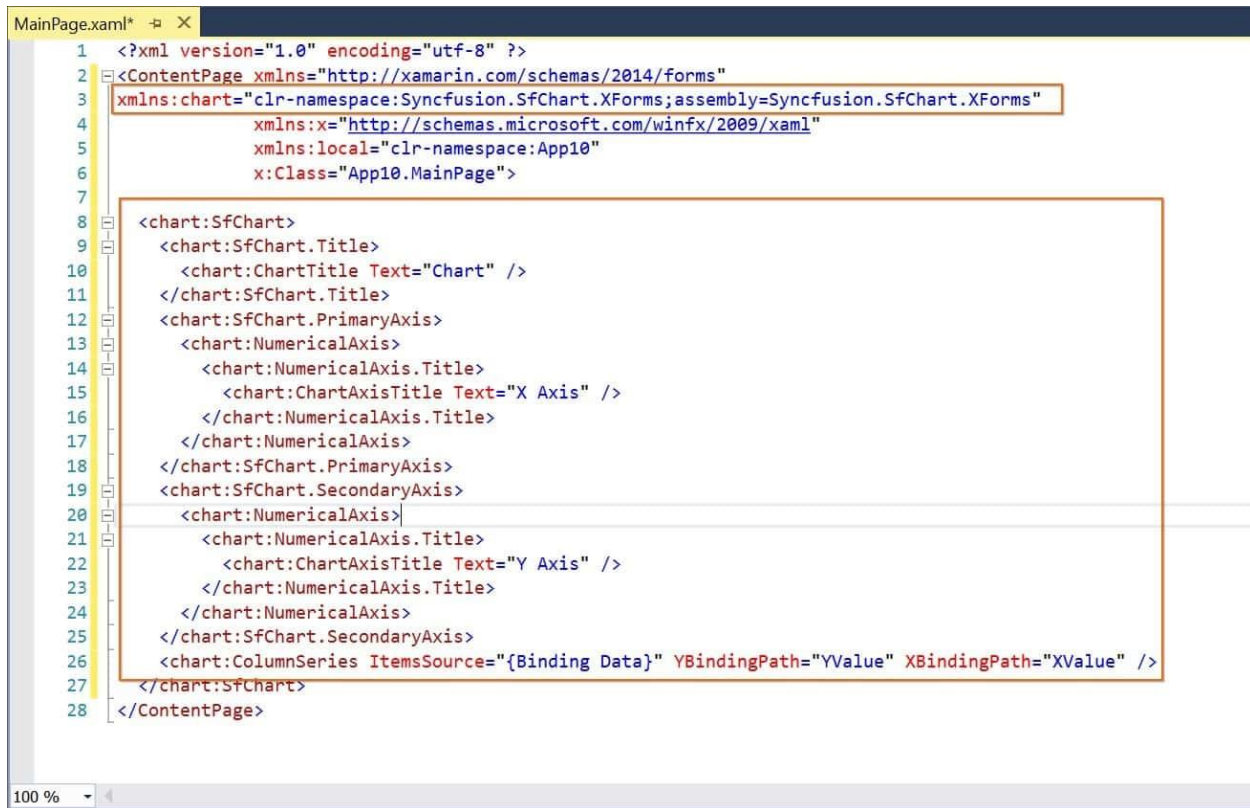
Choose **View > Other Windows > Syncfusion Toolbox** in **Visual Studio**.



2. Click **Syncfusion Toolbox** menu item, the Syncfusion Toolbox wizard appears. The Syncfusion control will be enabled when opening the Xamarin.Forms designer page. There is no Syncfusion control appears until open the appropriate .xaml file from the Xamarin shared/.NET Standard /PCL project.



3. The required Syncfusion controls design (.xaml) snippet and namespace will be added by drag and drop the required control from the toolbox to the designer.



Also, the required control Syncfusion Xamarin NuGet packages will be installed automatically when drag and drop the control to the designer to render the Syncfusion control.

### Essential UI Kit

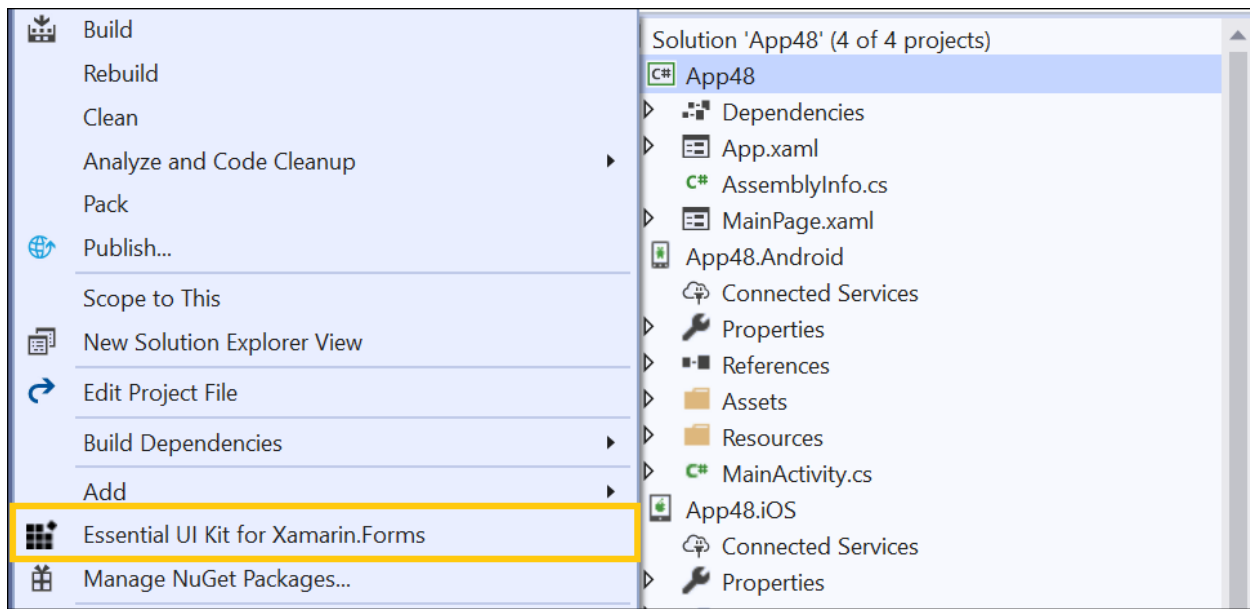
Essential UI Kit for Xamarin.Forms includes predefined XAML templates for Xamarin.Forms apps. The UI kit allows to build a user interface in a cross-platform application. It provides a clear separation of View, View Model, and Model classes, so integrating your business logic and making changes in the existing view is simple.

#### Installation of Xamarin UI Kit Extension

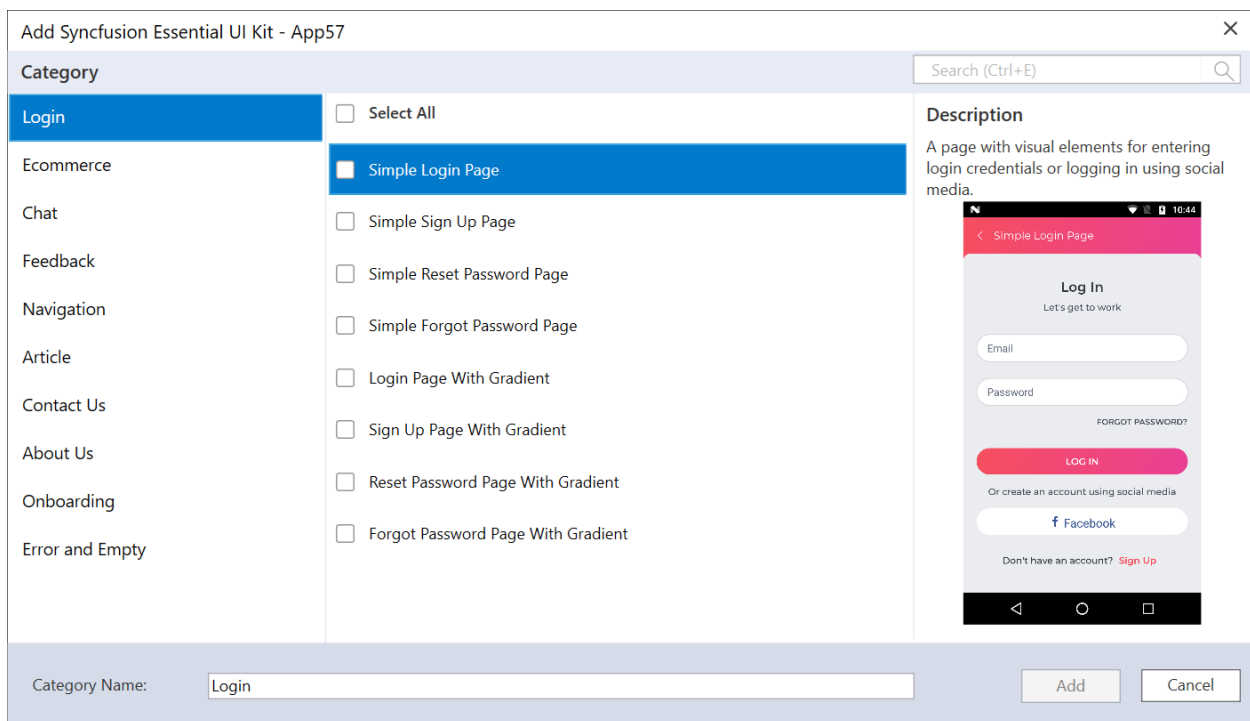
Download the necessary [Xamarin UI Kit Extension](#) from the market place and install them in your Visual Studio. So that, you can use the Syncfusion Extension from Syncfusion menu of your project.

#### Creating Syncfusion Xamarin application

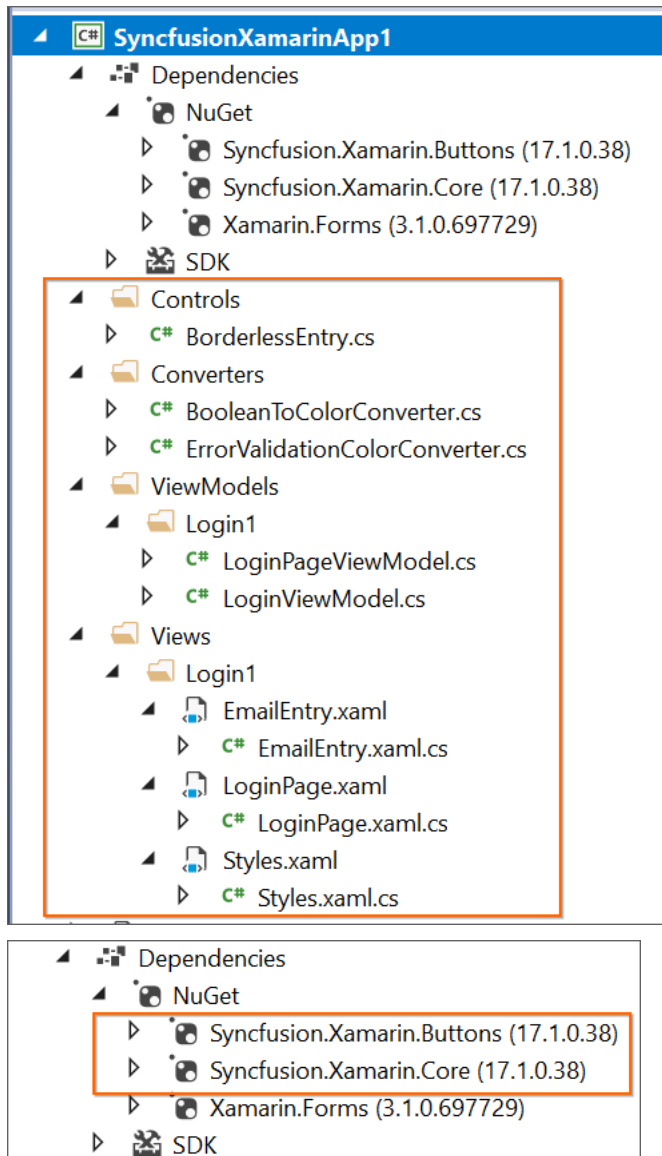
1. Open a new or existing Xamarin application.
2. Right-click on your **Xamarin.Form** project from the **Solution Explorer** and select the **Syncfusion UI for Xamarin** option.

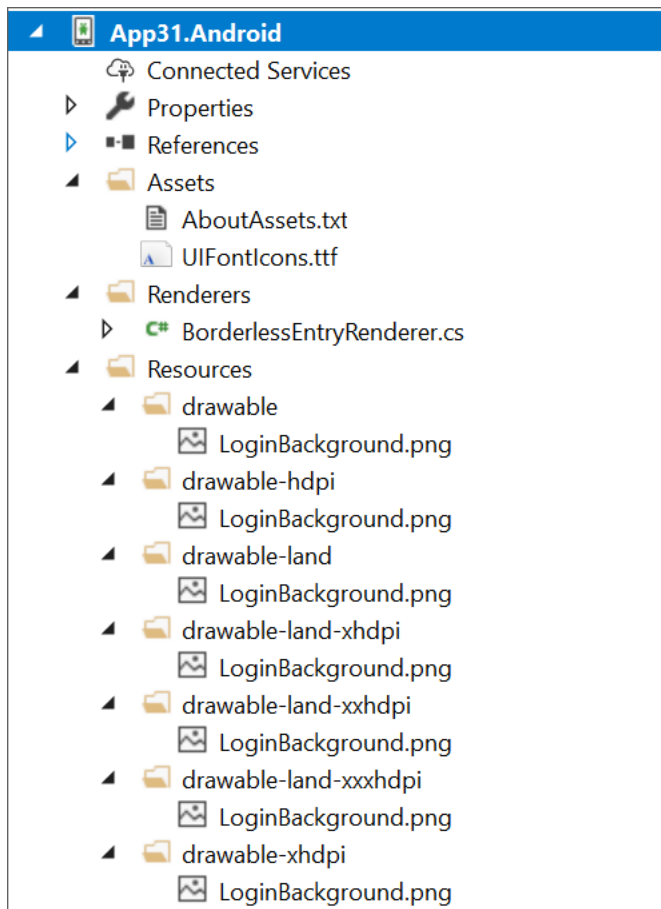


3. Then the Category dialog window will open with its predefined template



4. Now, add the desired pages of the any number of predefined categories
5. The selected pages will be added along View, ViewModel, Model classes, resource files and Syncfusion NuGet package reference,

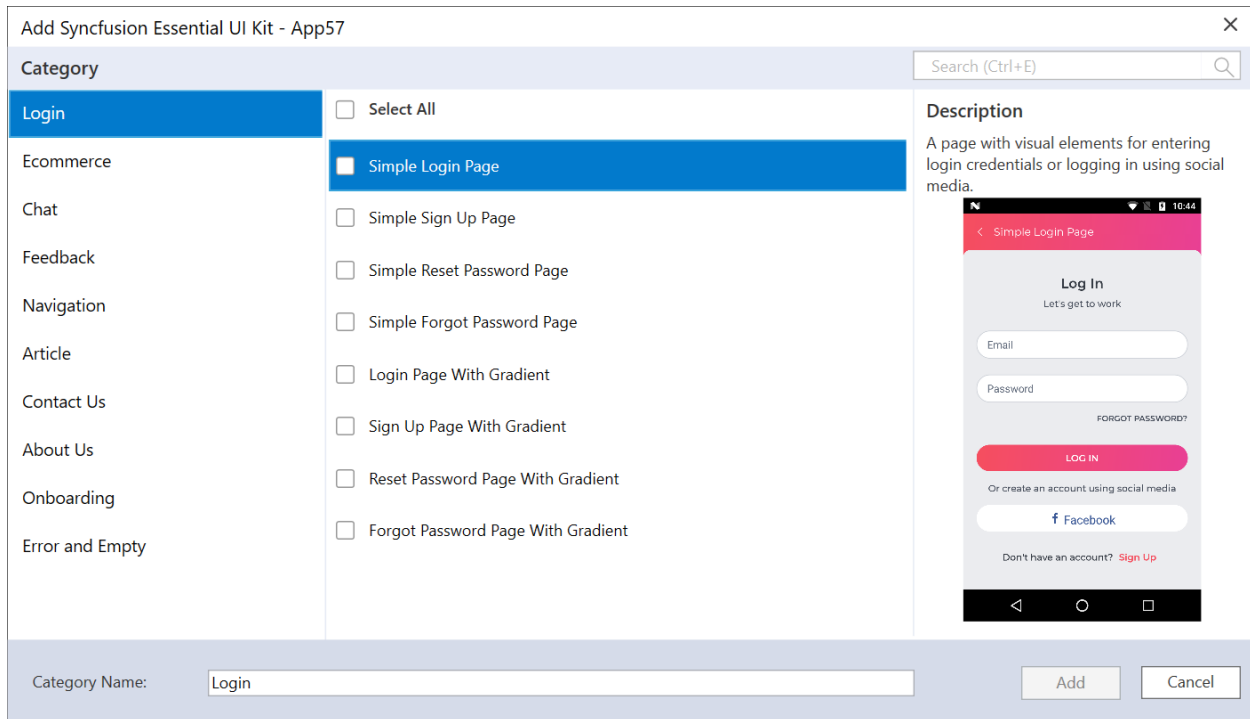




#### *Template Selection and naming*

The name of the categories such as Login, Chat, E-Commerce page can be changed while adding it to your project. Therefore, the corresponding category will be displayed with the provided name in the View, ViewModel and Model classes of the application.

The main advantage of category dialog window is to choose multiple pages from multiple categories of the UI templates at the time of addition.



#### Run the UI template item

To set the desired UI Template item as the start page of your application, Open the **App.xaml.cs** and make the following changes.

MainPage=new application name.Views.

**Item name**.selected template page name();

Example: If you added Login Page,

MainPage=new App1.Views.Login.LoginPage();

#### Check for Updates

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

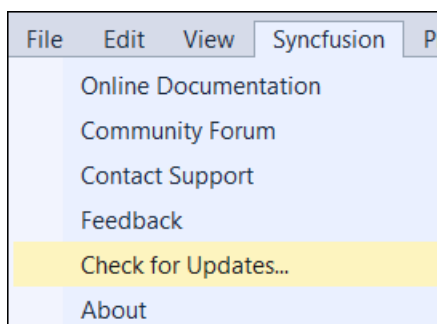
---

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

---

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

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



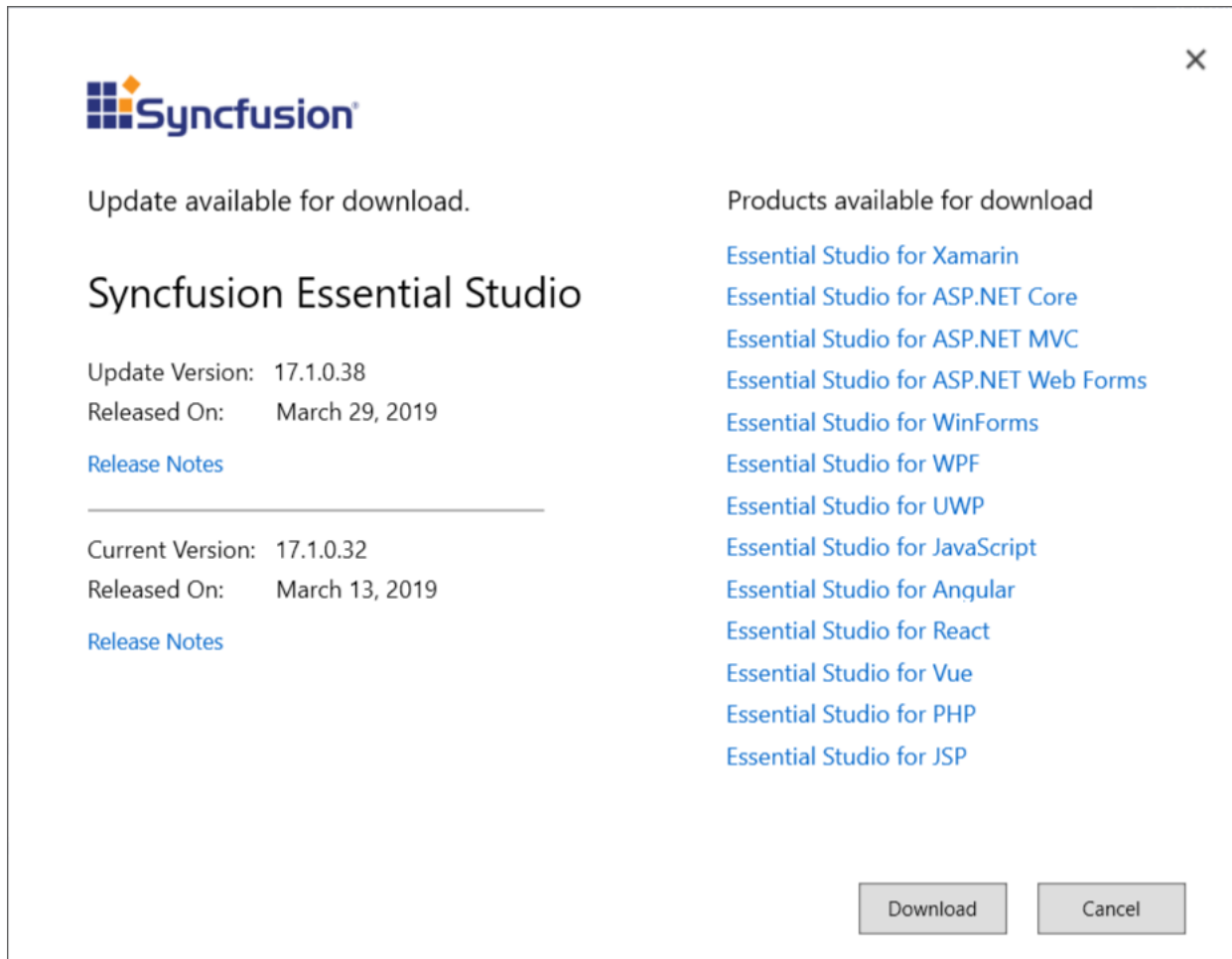


---

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

---

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



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

## Visual Studio Mac Extensions

### Overview

The Syncfusion Essential Studio for Xamarin Visual Studio for Mac extensions that allow you to create the Xamarin application in Visual Studio for Mac with the Syncfusion components.

---

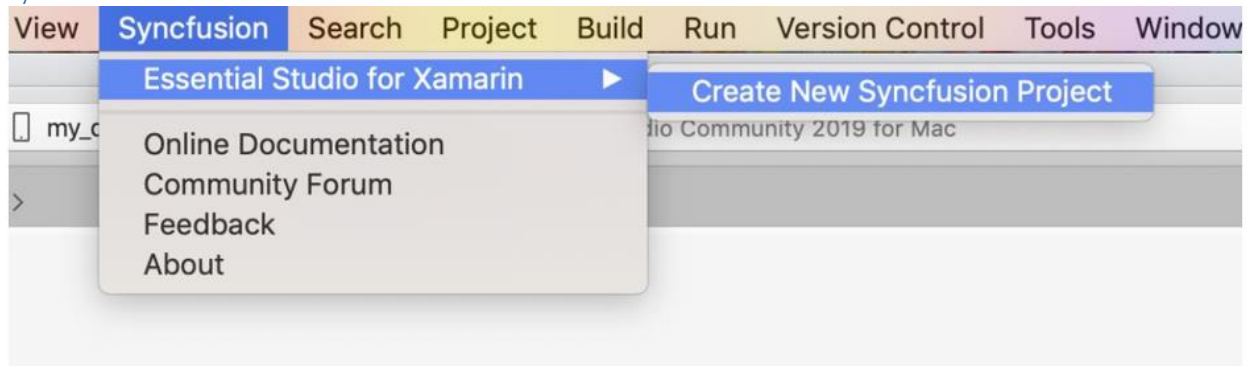
**Information:** The Syncfusion Xamarin Mac Extensions Support for Visual Studio for Mac 2019 and it is available from v17.3.0.9.

---

The Syncfusion provides the following extension supports in Visual Studio for Mac:

[Project template](#): To create a Syncfusion Xamarin application by adding the required Syncfusion NuGet based on the control chosen.

## Syncfusion Menu



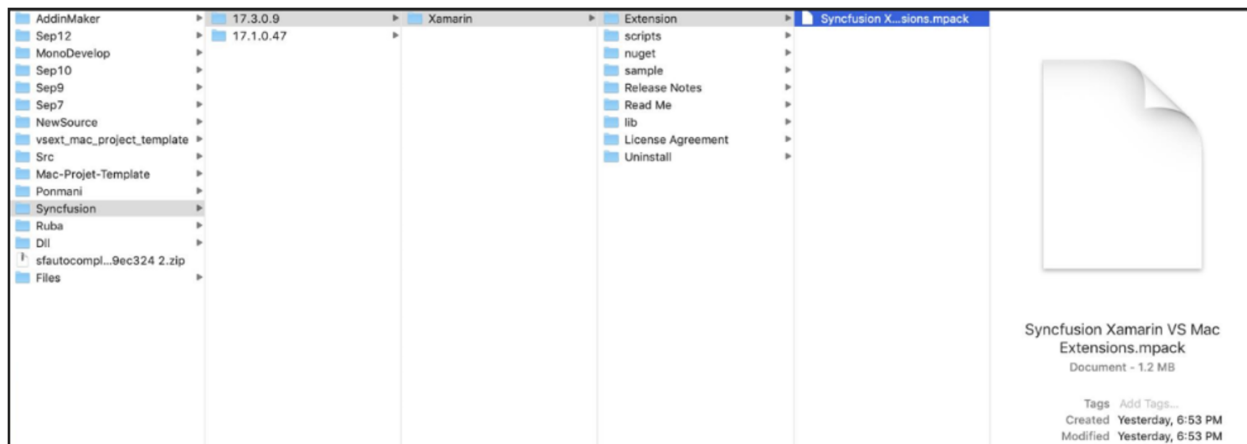
## Installation

The following procedure illustrates how to install Syncfusion Xamarin Extension in Visual Studio for Mac.

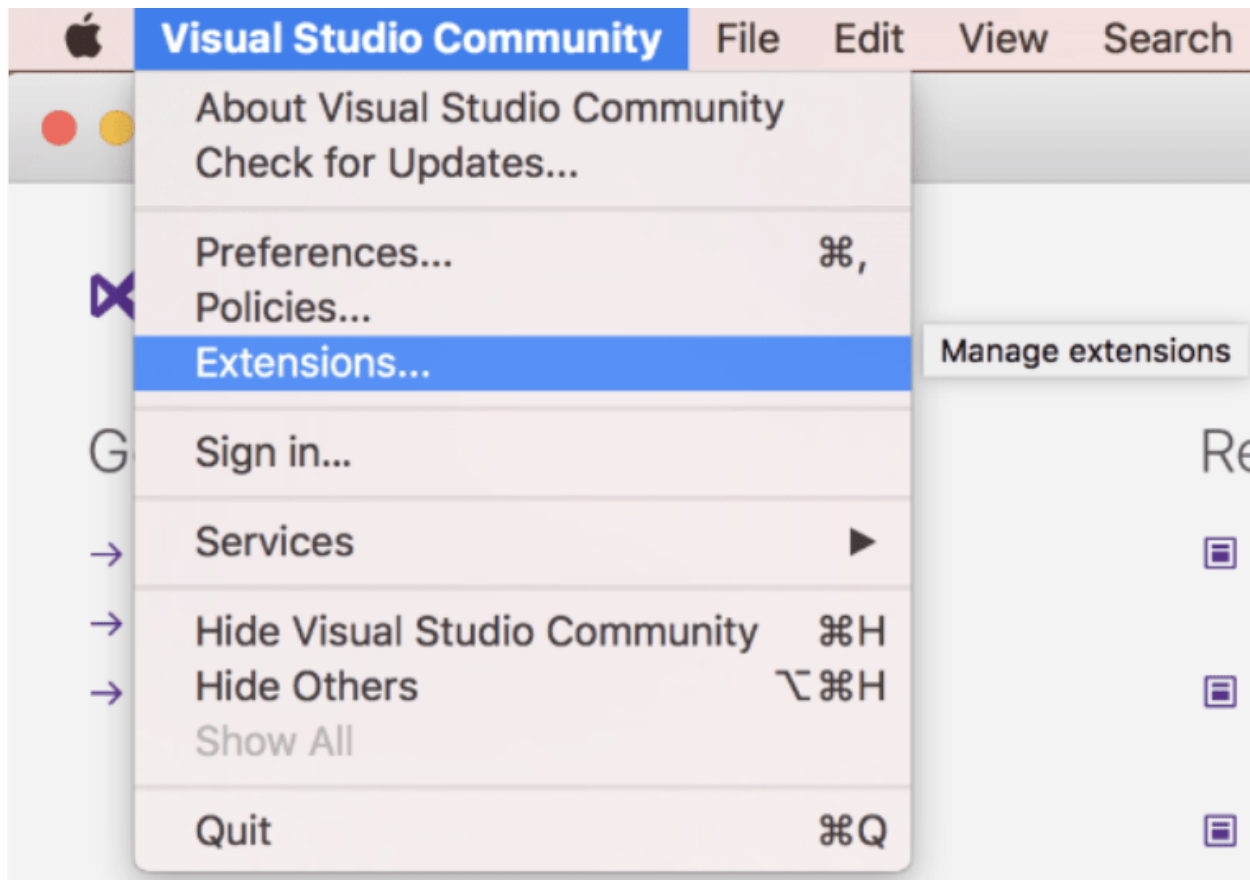
Follow [this guideline](#) to install the Essential Studio for Xamarin Mac build if not installed, then you will get the Visual Studio for Mac Extensions in installed location.

**Location:**

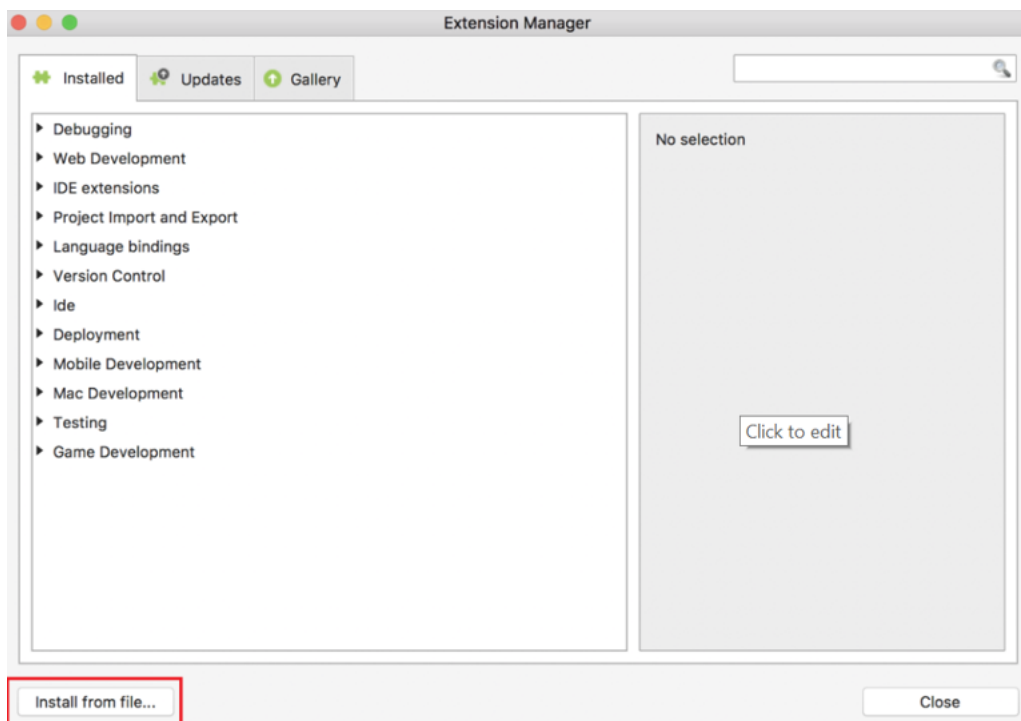
Eg: {Documents}\Syncfusion\ {version}\Xamarin\Extension\Syncfusion Xamarin VS Mac Extensions.mpack



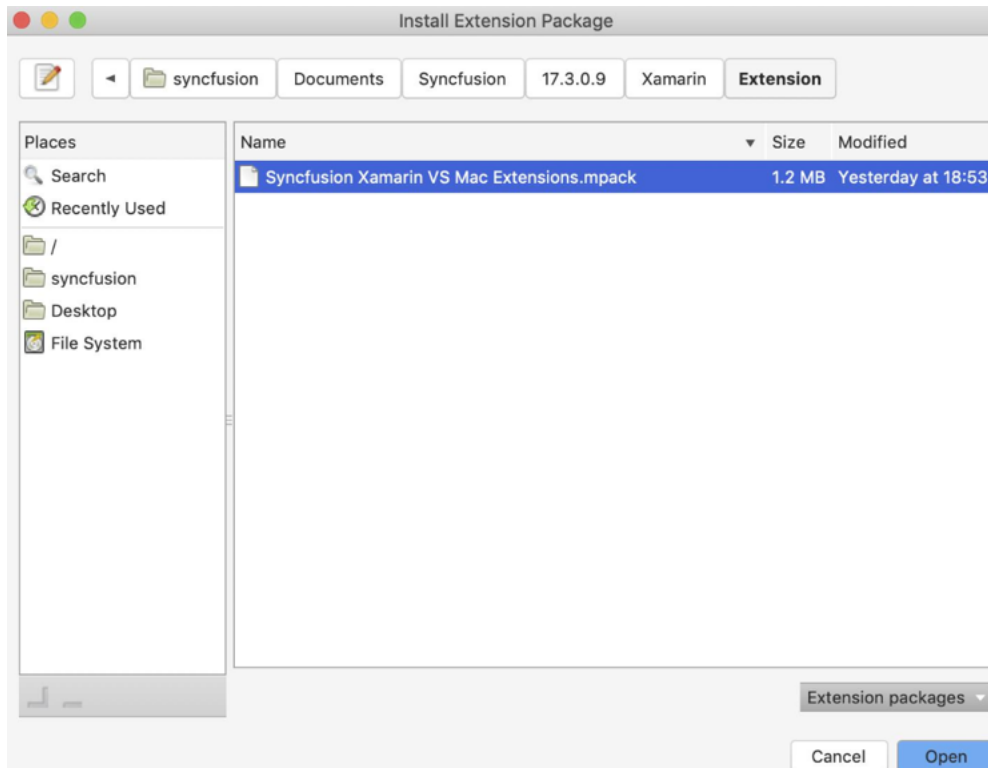
2. Open the Visual Studio 2019 for Mac.
3. Click the Visual Studio Community/Professional/Enterprise and select the Extensions...



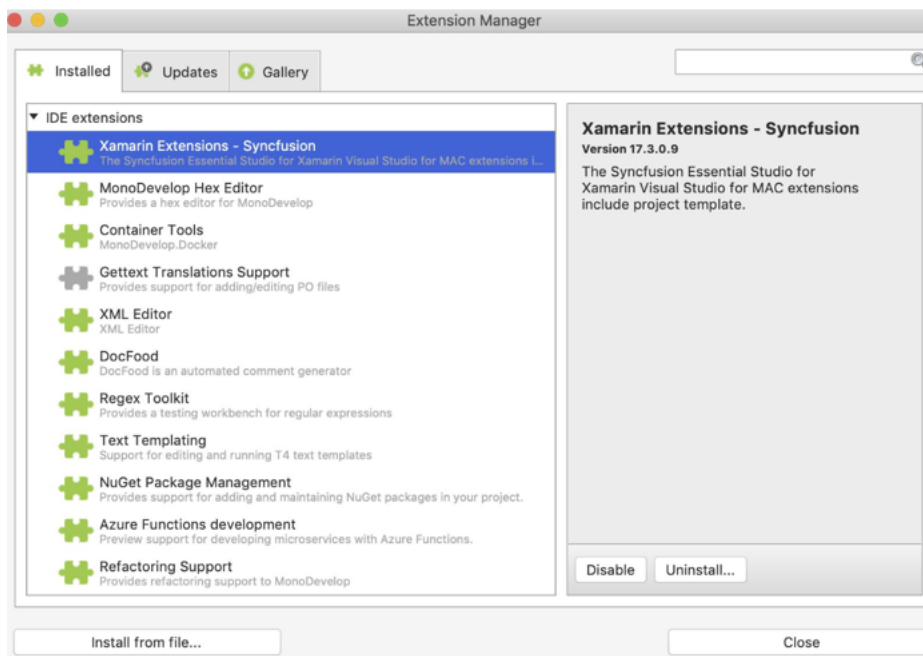
4. The Extension Manager window will open. Select the Install from file...



5. The Install Extension Package dialog will open. Navigate to Syncfusion Xamarin Mac Extensions file(.mpack) location which explained in above and click open to install.



6. Once the extension installed, the Syncfusion Xamarin Extension will listed in the IDE extension section.



### Create Project

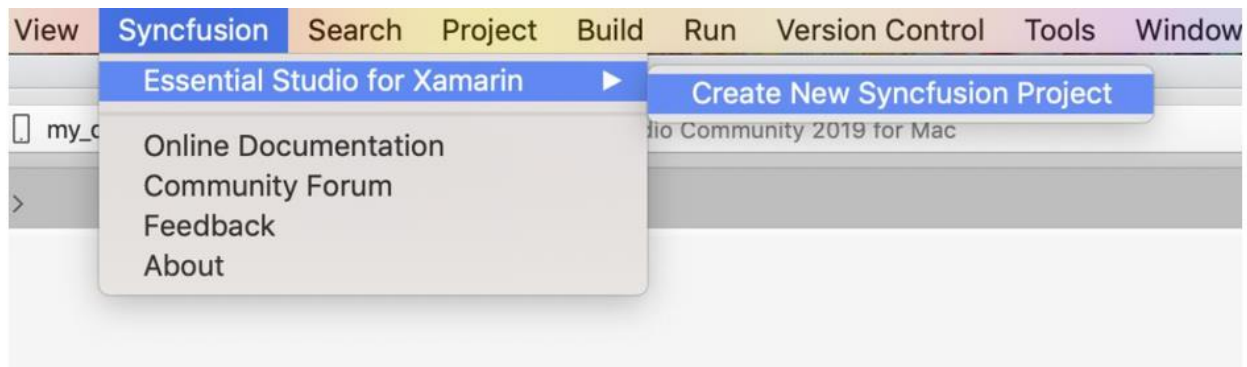
Syncfusion provides the Visual Studio for Mac Project Templates for Xamarin platform to create the Syncfusion Xamarin Application by adding the required Syncfusion NuGet packages.

Use the following steps to create the Syncfusion Xamarin Application through the Visual Studio 2019 for Mac:

1) To create a Syncfusion Xamarin project, follow either one of the options below in Visual Studio for Mac.

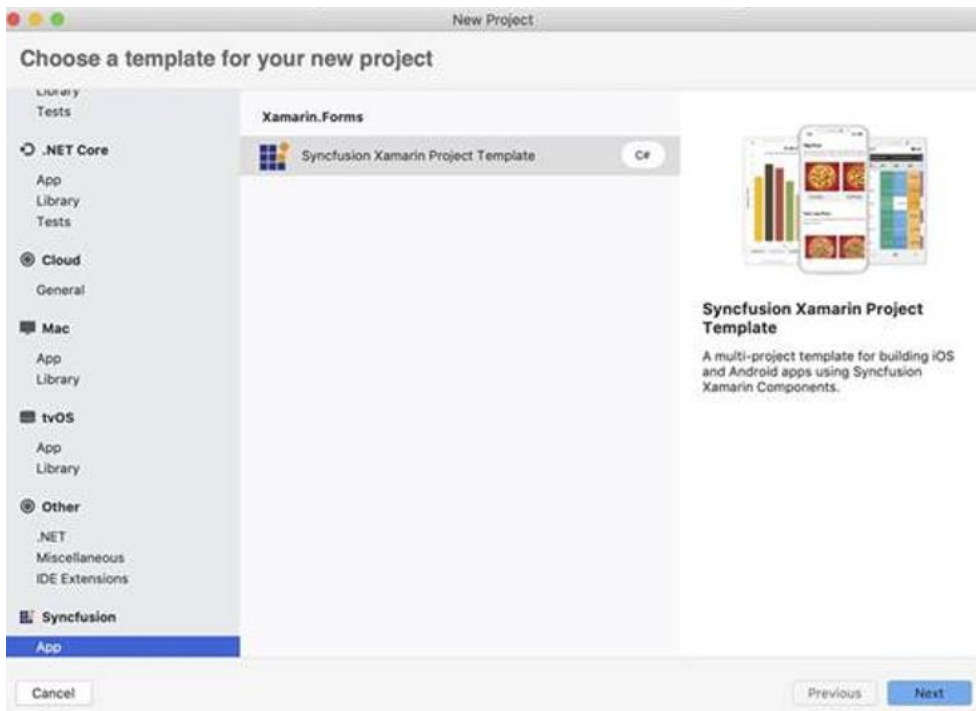
Option 1:

Click **Syncfusion Menu** and choose **Essential Studio for Xamarin > Create New Syncfusion Project**.

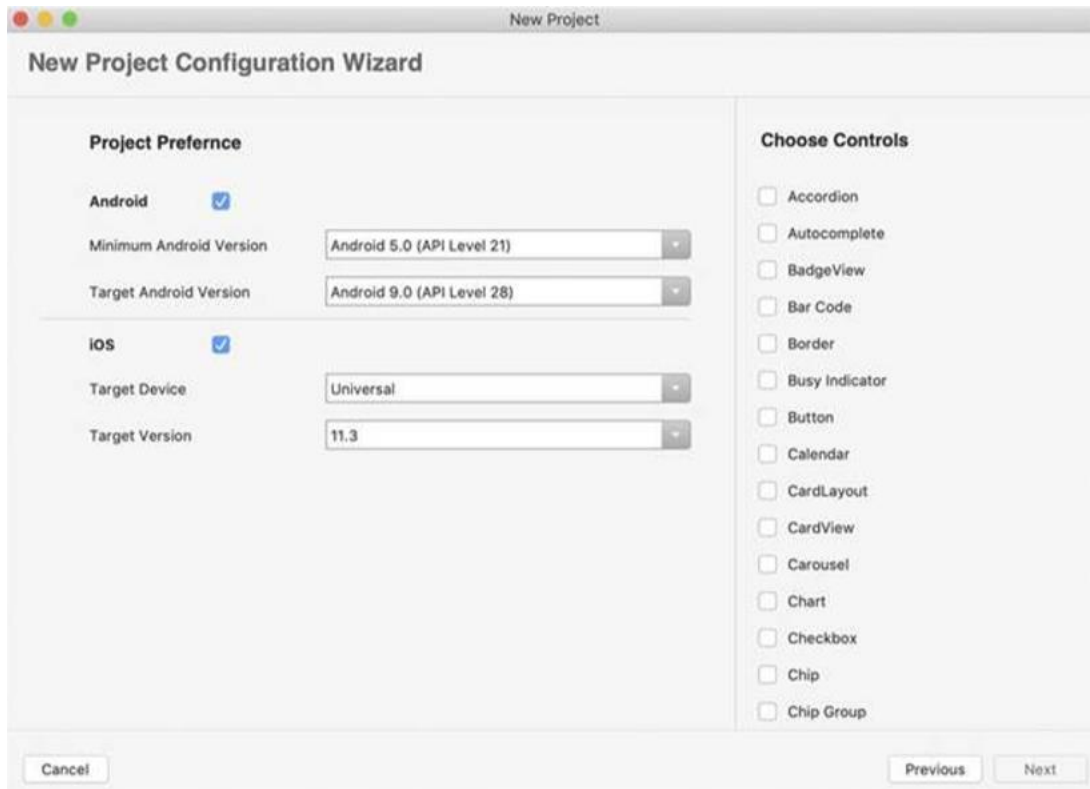


Option 2:

Choose **File > New > Project** and navigate to **Syncfusion > App > Syncfusion Xamarin Project Template**.



2) Configure the Syncfusion Xamarin Application by using the following Project Configuration dialog, choose the project Android and iOS by toggling respected checkbox and Syncfusion components.



#### Android:

- 1. Minimum Android Version:** Select the oldest Android version that you want to support your application.
- 2. Target Android Version:** Select the version of Android to run your application.

#### iOS:

- 1. Target Device:** Select the device of Xamarin.iOS project either Unified, iPhone/iPod, or iPad
- 2. Target Version:** Choose the version of Xamarin.iOS Project.

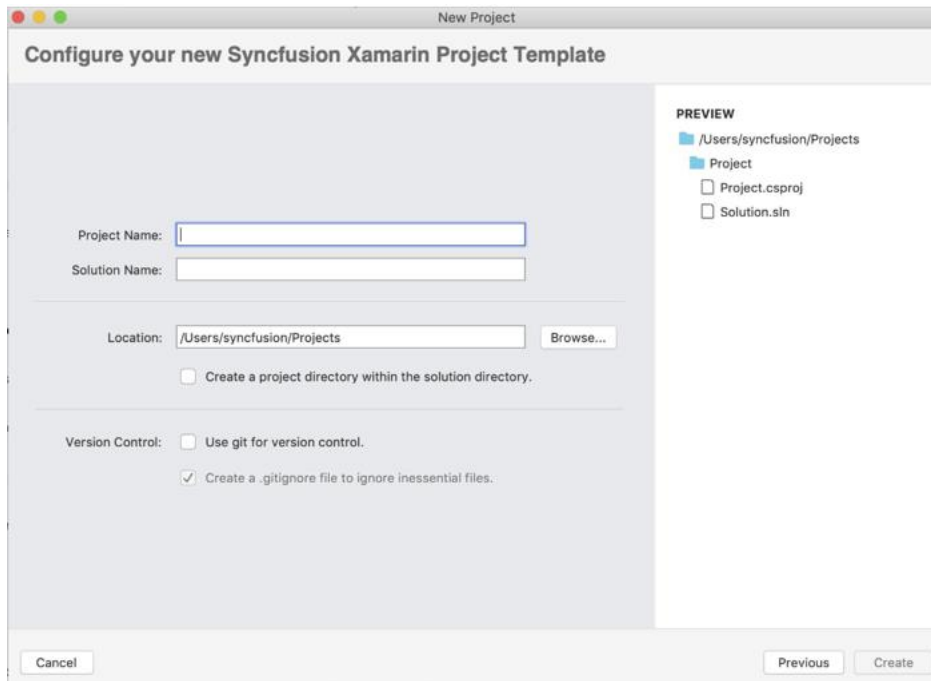
#### Choose controls:

Choose the Xamarin application needs to create with the Syncfusion controls.

**Choose Controls**

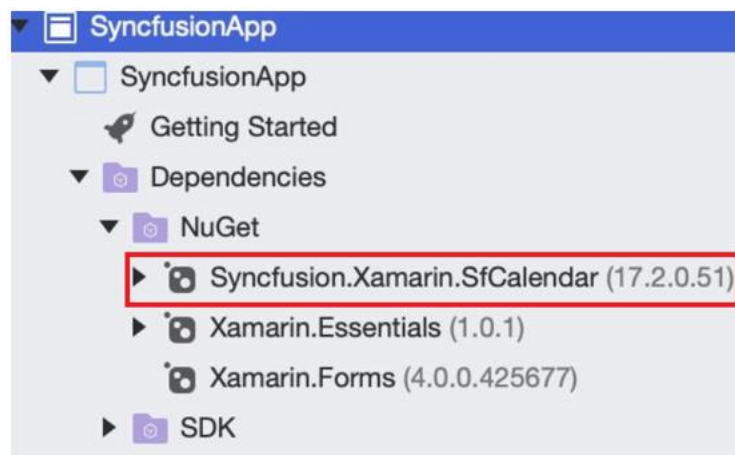
- ☐ Accordion
- ☐ Autocomplete
- ☐ BadgeView
- ☐ Bar Code
- ☐ Border
- ☐ Busy Indicator
- ☐ Button
- ☐ Calendar
- ☐ CardLayout
- ☐ CardView
- ☐ Carousel
- ☐ Chart
- ☐ Checkbox
- ☐ Chip
- ☐ Chip Group

3) Name the project and click the Create.



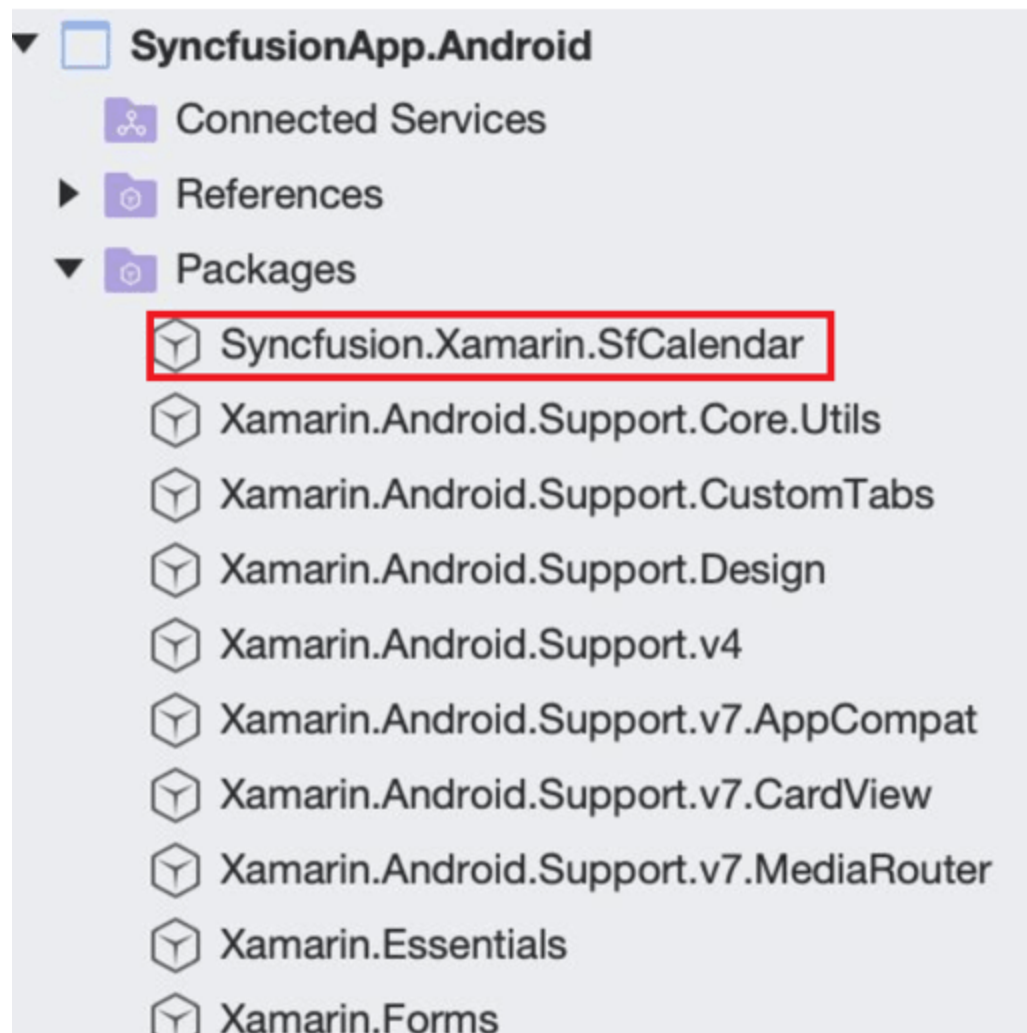
4) Required Syncfusion NuGet and configuration have been added to the project based on the Syncfusion component chosen.

#### Net Standard:

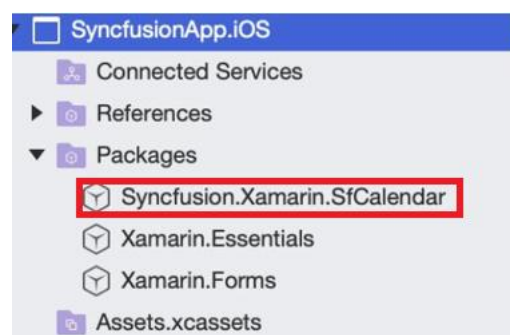


#### Android:





iOS:



Now, you can follow the [user guide documentation](#) to use the Syncfusion Xamarin components.

### NuGet Packages

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

From v15.4.0.17 onwards, the Syncfusion Xamarin.Forms NuGet packages are published in [nuget.org](https://www.nuget.org).

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

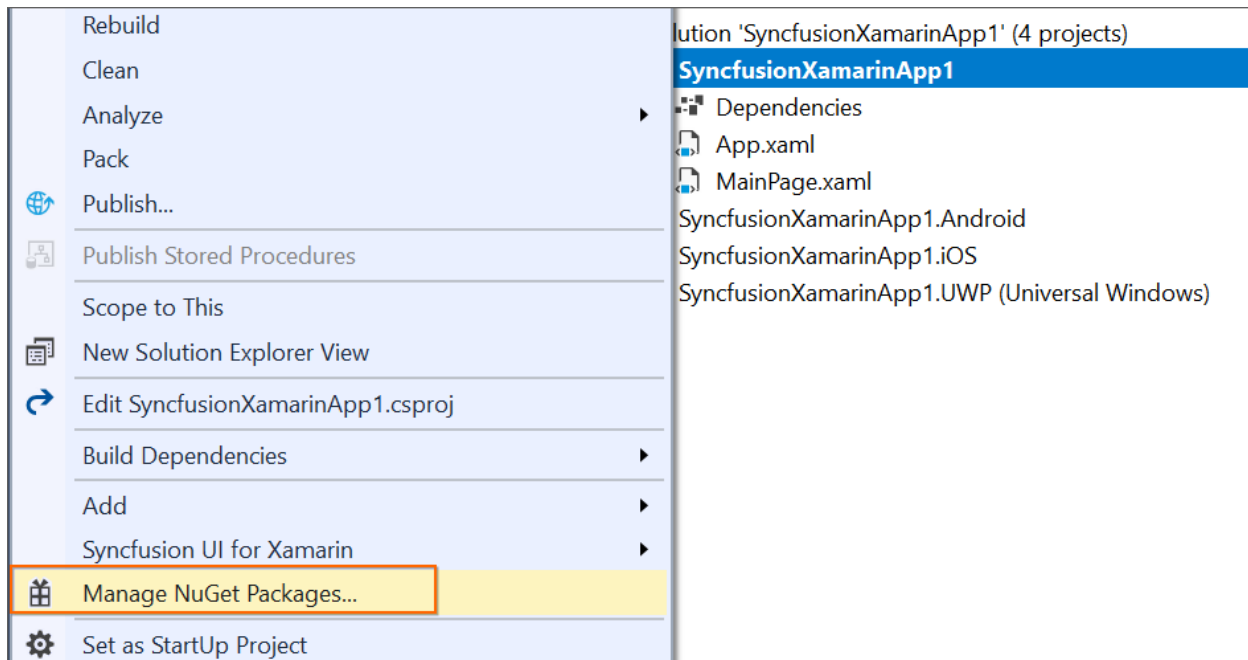
**Note:** Starting from v17.1.0.32 (2018 Volume 1), Syncfusion will no longer publish NuGet packages at [nuget.syncfusion.com](http://nuget.syncfusion.com).

## Installing NuGet Packages

### Using NuGet Package Manager

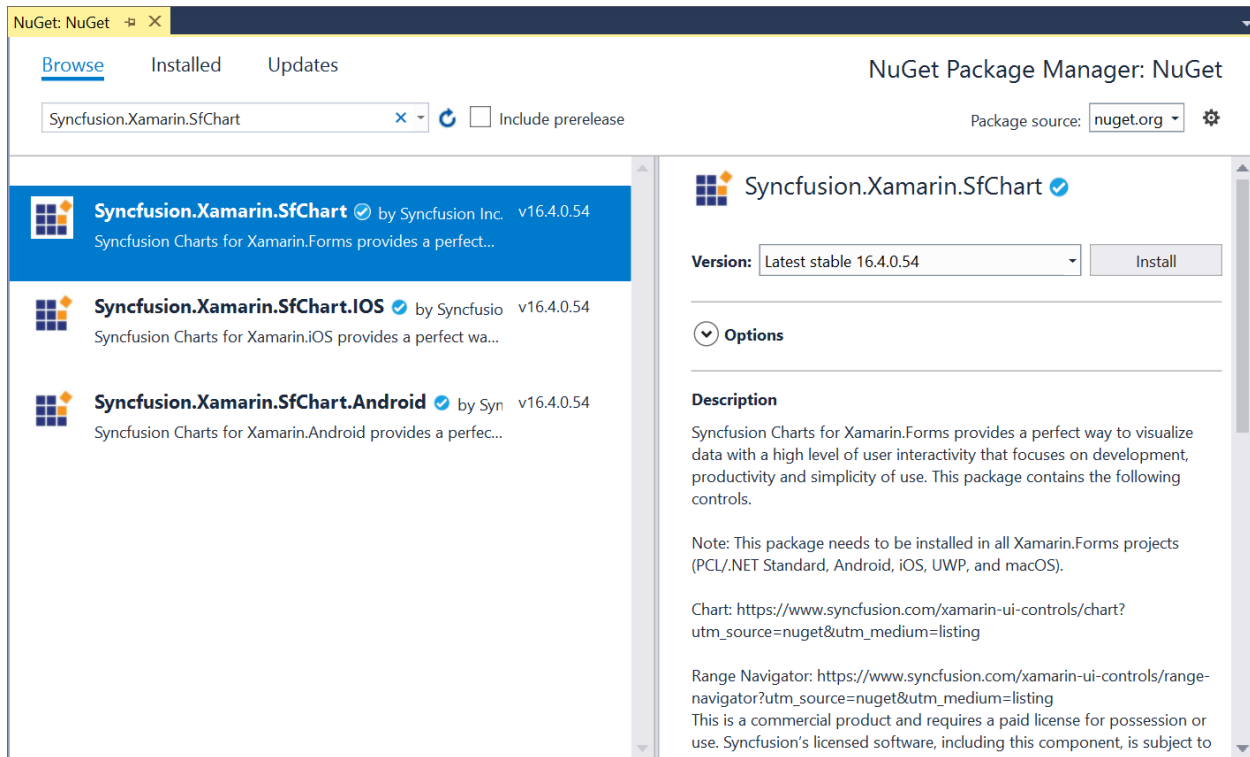
The NuGet Package Manager can be used to search and install NuGet packages in the Visual Studio solution or project:

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



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

2. Select the NuGet.org from the **Package Source** drop-down.



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

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

#### Using Package Manager Console

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

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

## Explore Libraries Package

## Troubleshooting

## Themes

### XML

```
<Application xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms"
xmlns:gauge="clr-
namespace:Syncfusion.SfGauge.XForms;assembly=Syncfusion.SfGauge.XForms"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
...>
<Application.Resources>
<ResourceDictionary>
```

```

<ResourceDictionary.MergedDictionaries>
<!-- Theme resource dictionary -->
<syncTheme:DarkTheme />
<!-- Control style resource dictionaries -->
<gauge:SfCircularGaugeStyles />
<buttons:SfButtonStyles />
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Application.Resources>
...
</Application>

```

### Merging the dictionaries

You can merge the theme resource dictionary and control style resource dictionaries in the following two ways:

#### Manual merging

For manual merging, both the theme resource dictionary and each control style resource dictionary need to be merged for the required controls in the application resources as follows.

#### XML

```

<Application xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms"
xmlns:gauge="clr-
namespace:Syncfusion.SfGauge.XForms;assembly=Syncfusion.SfGauge.XForms"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
...>
<Application.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<!-- Theme resource dictionary -->
<syncTheme:DarkTheme />
<!-- Control style resource dictionaries -->
<gauge:SfCircularGaugeStyles />
<buttons:SfButtonStyles />
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</Application.Resources>
...
</Application>

```

#### Automatic merging

When using more number of Syncfusion controls in an application, to make the process easier for merging the control style dictionaries of the controls, the SyncfusionThemeDictionary class has been provided for automatic merging. When the theme resource dictionary is merged to this dictionary, control style resource dictionaries will be merged automatically. However, only the styles for the controls used in the application will be merged.

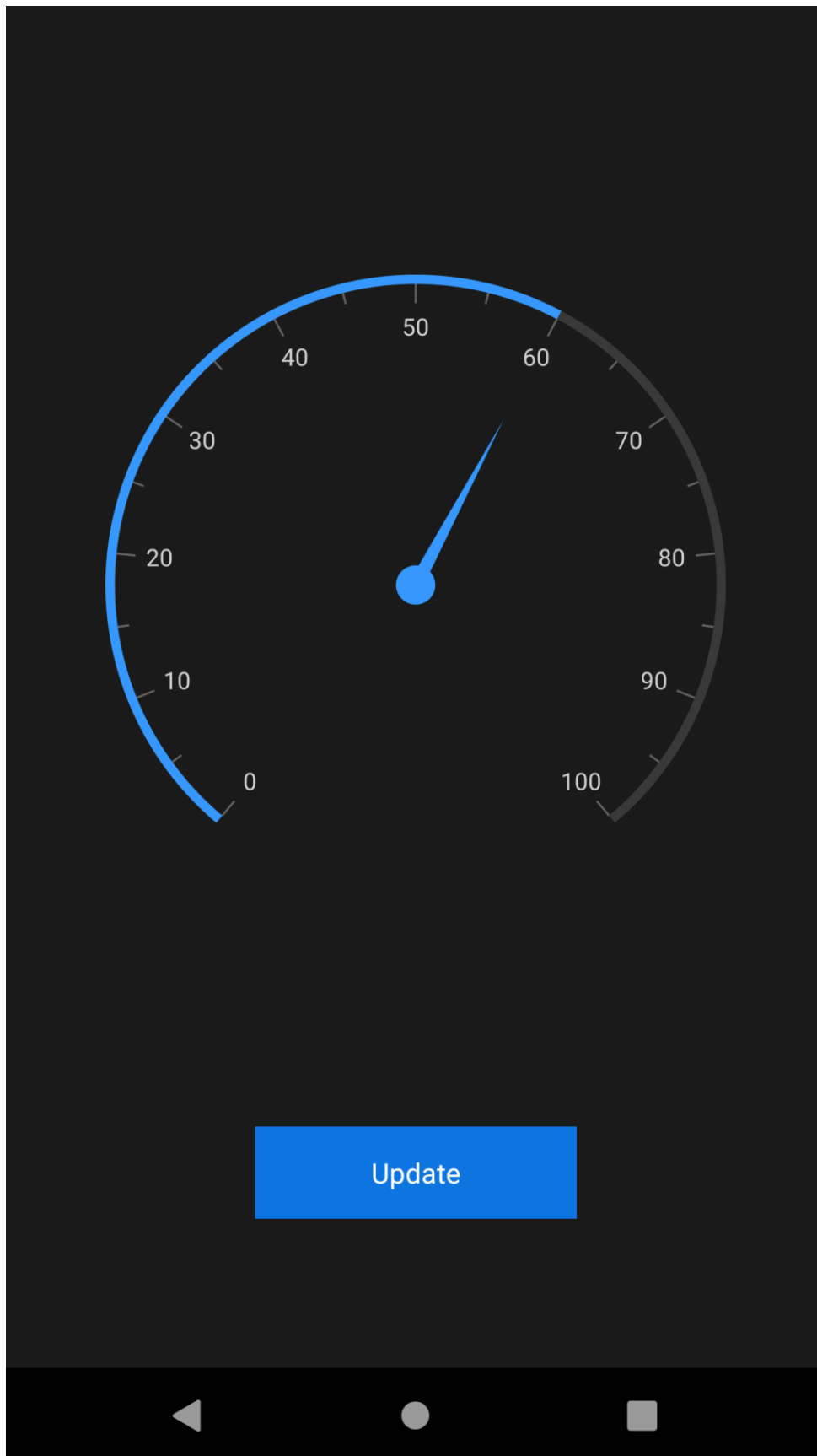
#### XML

```

<Application xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms"
...>

```

```
<Application.Resources>
<syncTheme:SyncfusionThemeDictionary>
<syncTheme:SyncfusionThemeDictionary.MergedDictionaries>
<!-- Theme resource dictionary -->
<syncTheme:DarkTheme />
</syncTheme:SyncfusionThemeDictionary.MergedDictionaries>
</syncTheme:SyncfusionThemeDictionary>
</Application.Resources>
....
</Application>
```



### Overriding the default theme

The theme resource dictionary contains a set of keys that are mapped to the style in control style dictionaries. The default appearance of themes can be customized by overriding the key values.

The following section explains how to override both the primary and control specific keys.

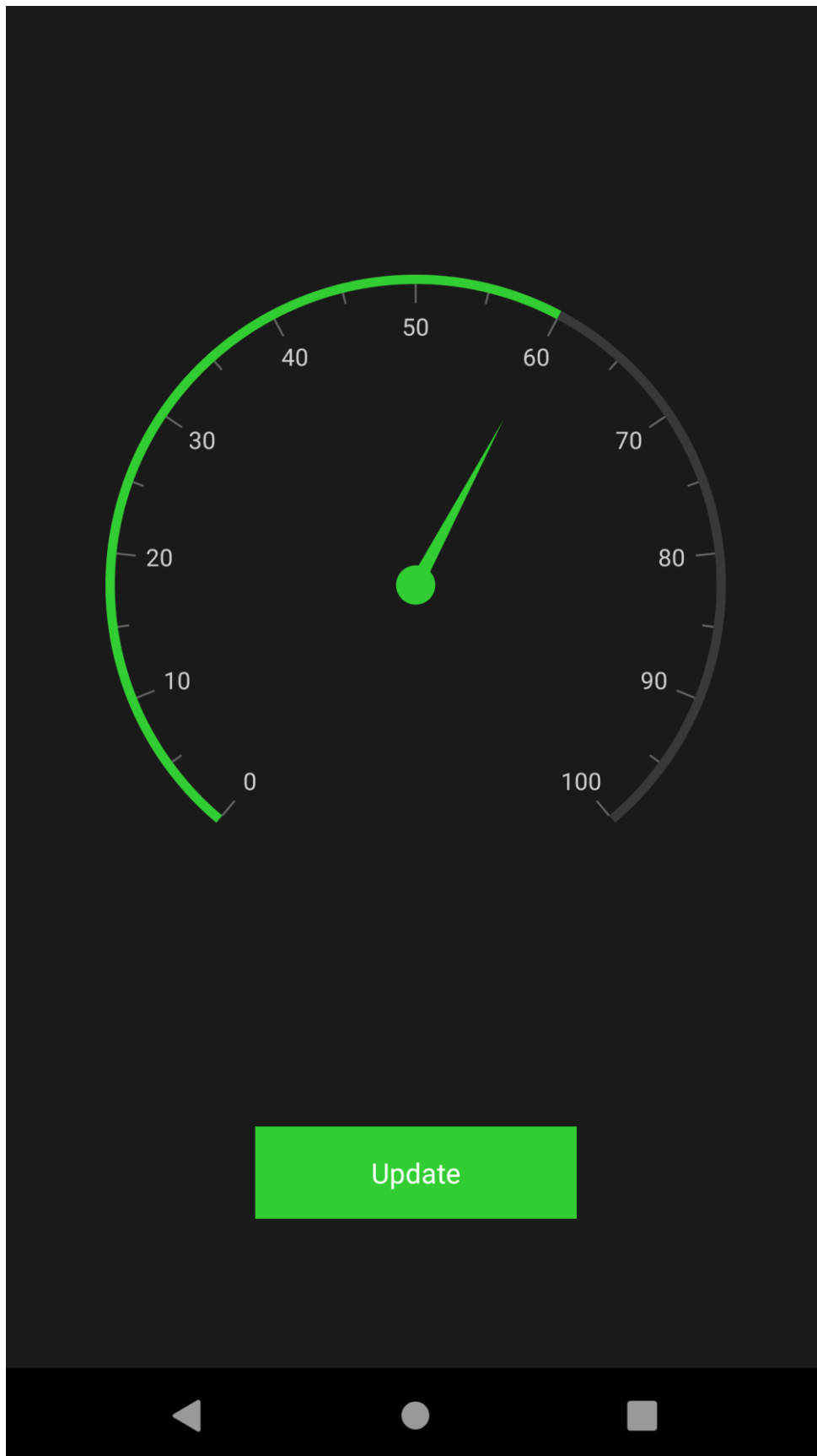
#### Overriding the primary keys

The theme resource dictionary contains the following set of primary keys that are mapped to the UI elements of all the controls. To override the primary colors of theme, change the values for these keys as required. You can find the keys and the UI elements to which they are mapped to all the controls in this [documentation](#).

- SyncPrimaryColor
- SyncPrimaryLightColor
- SyncPrimaryDarkColor
- SyncPrimaryForegroundColor
- SyncPrimaryLightForegroundColor
- SyncPrimaryDarkForegroundColor
- SyncSuccessColor
- SyncErrorColor
- SyncWarningColor
- SyncInfoColor

### XML

```
<Application xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms"
...>
<Application.Resources>
<syncCore:SyncfusionThemeDictionary>
<syncCore:SyncfusionThemeDictionary.MergedDictionaries>
<syncCore:DarkTheme />
<ResourceDictionary>
<Color x:Key="SyncPrimaryColor">LimeGreen</Color>
<Color x:Key="SyncPrimaryLightColor">LimeGreen</Color>
</ResourceDictionary>
</syncCore:SyncfusionThemeDictionary.MergedDictionaries>
</syncCore:SyncfusionThemeDictionary>
</Application.Resources>
...
</Application>
```



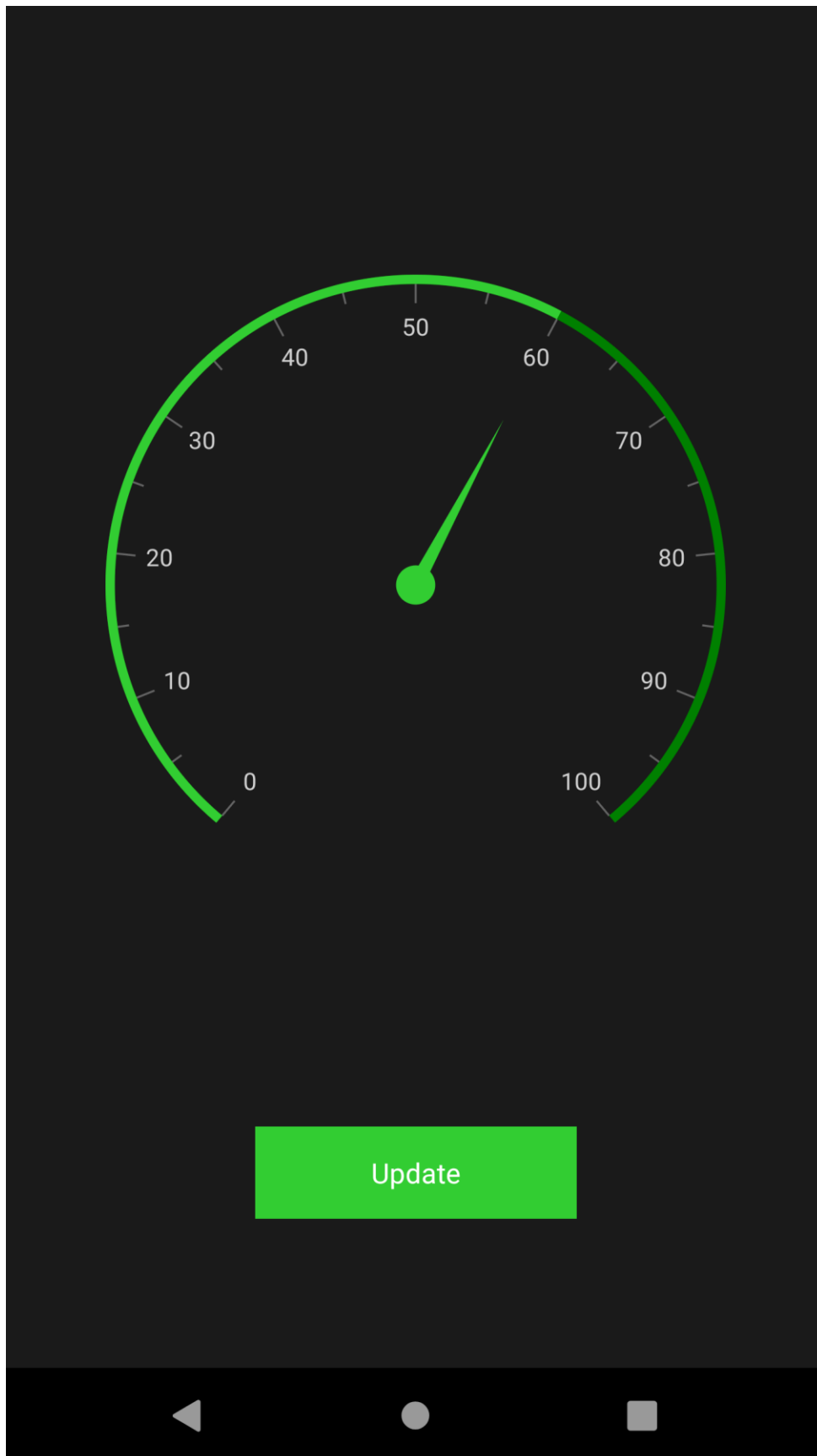


### *Overriding the control specific keys*

In addition to the primary keys, the theme resource dictionary also contains the keys that are specific to each controls; these keys can also be overridden. You can find the keys and the UI elements to which they are mapped to all the controls in this [documentation](#).

### **XML**

```
<Application xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms"
...>
<Application.Resources>
<syncCore:SyncfusionThemeDictionary>
<syncCore:SyncfusionThemeDictionary.MergedDictionaries>
<syncCore:DarkTheme />
<ResourceDictionary>
<Color x:Key="SyncPrimaryColor">LimeGreen</Color>
<Color x:Key="SyncPrimaryLightColor">LimeGreen</Color>
<Color x:Key="SfCircularGaugeScaleRimColor">Green</Color>
</ResourceDictionary>
</syncCore:SyncfusionThemeDictionary.MergedDictionaries>
</syncCore:SyncfusionThemeDictionary>
</Application.Resources>
....
</Application>
```



### Creating your own theme

As an alternative approach to the above methods, you can also create your own theme. To create own theme, first, you need to merge the resource, whose key name should be "ControlName" + "Theme" based on the controls, e.g., SfChartTheme and SfTextInputLayoutTheme. You can find this key for each control in this [documentation](#). After merge the control name, you need to merge the required color resources with keys based on the UI elements that need to be customized. You can find the keys and the UI elements to which they are mapped to all the controls in this [documentation](#).

Using this approach, you can create your own theme to all the controls or for specific controls you need.

### XML

```
<Application xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms"
...>
<Application.Resources>
<syncCore:SyncfusionThemeDictionary>
<syncCore:SyncfusionThemeDictionary.MergedDictionaries>
<ResourceDictionary>
<x:String x:Key="SfCircularGaugeTheme">CustomTheme</x:String>
<Color x:Key="SyncPrimaryColor">LimeGreen</Color>
<Color x:Key="SyncPrimaryLightColor">LimeGreen</Color>
<Color x:Key="SfCircularGaugeScaleRimColor">Green</Color>
</ResourceDictionary>
</syncCore:SyncfusionThemeDictionary.MergedDictionaries>
</syncCore:SyncfusionThemeDictionary>
</Application.Resources>
....
</Application>
```

### Keys of Syncfusion Controls

This page lists the keys for each control and the element to which it is mapped for all the controls.

#### SfChart

Theme Dictionary	Keys	Description
SfChartStyles	SfChartTheme	By merging this key in application resources, you can customize the appearance of SfChart without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CommonTheme Transparent Transparent ....{% endhighlight %}
	SyncPrimaryColor	Stroke color of annotation.
		Text color of annotation label.

		Background color of the axis label of VerticalLine and HorizontalLine annotations.
		Border color of the axis label of VerticalLine and HorizontalLine annotations.
	SfChartTitleTextColor	Text color of the chart title.
	SfChartTitleBorderColor	Border color of the chart title.
	SfChartTitleBackgroundColor	Background color of the chart title.
	SfChartBackgroundColor	Background color of the chart.
	SfChartSelectionRectFillColor	Fill color of the selection rectangle when selection zooming the chart.
	SfChartSelectionRectStrokeColor	Stroke color of the selection rectangle when selection zooming the chart.
	SfChartAnnotationFillColor	Fill color of annotation.
	SfChartAnnotationLabelBorderColor	Border color of the annotation label.
	SfChartAnnotationLabelBackgroundColor	Background color of the annotation label.
	SfChartAnnotationAxisLabelTextColor	Text color of axis label of VerticalLine and HorizontalLine annotations.
	SfChartTrackballLineStrokeColor	Stroke color of the trackball line.
	SfChartTrackballBackgroundColor	Background color of the trackball label.
	SfChartTrackballBorderColor	Border color of the trackball label.
	SfChartTrackballTextColor	Text color of the trackball label.
	SfChartTrackballAxisLabelBackgroundColor	Background color of the axis label of trackball.
	SfChartTrackballAxisLabelBorderColor	Border color of the axis label of trackball.
	SfChartTrackballAxisLabelTextColor	Text color of axis label of trackball.

	SfChartDataPointSelectionColor	Color of the selected segment of the series.
	SfChartSeriesSelectionColor	Color of the selected series.
	SfChartStriplineBackgroundColor	Background color of the strip line.
	SfChartStriplineBorderColor	Border color of the strip line.
	SfChartStriplineTextColor	Text color of the strip line label.
	SfChartStriplineLabelBorderColor	Border color of the strip line label.
	SfChartStriplineLabelBackgroundColor	Background color of the strip line label.
	SfChartMajorGridLineColor	Color of the axis major grid line.
	SfChartMinorGridLineColor	Color of the axis minor grid line.
	SfChartMajorTickLineColor	Color of the axis major tick line.
	SfChartMinorTickLineColor	Color of the axis minor tick line.
	SfChartAxisLineColor	Color of the axis line.
	SfChartAxisTitleTextColor	Color of the axis title text.
	SfChartAxisTitleBackgroundColor	Background color of the axis title.
	SfChartAxisTitleBorderColor	Border color of the axis title.
	SfChartAxisLabelColor	Color of the axis label.
	SfChartAxisLabelBackgroundColor	Background color of the axis label.
	SfChartAxisLabelBorderColor	Border color of the axis label.
	SfChartLegendTitleTextColor	Color of the Legend title text.
	SfChartLegendTitleBorderColor	Border color of the legend title.
	SfChartLegendTitleBackgroundColor	Background color of the legend title.
	SfChartLegendLabelColor	Color of the legend label.

## SfDateTimeRangeNavigator

Theme Dictionary	Keys	Description
SfDateTimeRangeNavigatorStyles	SfDateTimeRangeNavigatorTheme	By merging this key in application resources, you can customize the appearance of SfDateTimeRangeNavigator without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CommonTheme Transparent White ....{% endhighlight %}
	SyncPrimaryColor	Border color of tooltip and thumb.
		Line color of thumb.
		Background color of tooltip.
	SyncPrimaryLightColor	Selected label text color of scale.
	SfDateTimeRangeNavigatorBackgroundColor	Background color of the date-time range navigator.
	SfDateTimeRangeNavigatorOverlayColor	Overlay color of unselected region of the date time range navigator.
	SfDateTimeRangeNavigatorLabelTextColor	Text color of the date time range navigator label.
	SfDateTimeRangeNavigatorLabelBackgroundColor	Background color of the date time range navigator label.

	SfDateTimeRangeNavigatorLabelBorderColor	Border color of the date time range navigator label.
	SfDateTimeRangeNavigatorSelectedLabelBackgroundColor	Background color of the date time range navigator selected label.
	SfDateTimeRangeNavigatorSelectedLabelBorderColor	Border color of the date time range navigator selected label.
	SfDateTimeRangeNavigatorGridLineColor	Color of the date time range navigator grid lines.
	SfDateTimeRangeNavigatorTickLineColor	Color of the date time range navigator tick lines.
	SfDateTimeRangeNavigatorThumbBackgroundColor	Background color of the date time range navigator thumb.
	SfDateTimeRangeNavigatorThumbBorderColor	Border color of the date time range navigator thumb.
	SfDateTimeRangeNavigatorThumbLineColor	Color of the date time range navigator thumb line.
	SfDateTimeRangeNavigatorTooltipTextColor	Text color of the date time range navigator tooltip.

## SfKanban

Theme Dictionary	Keys	Description
SfKanbanStyles	SfKanbanTheme	By merging this key in application resources, you can customize the appearance of SfKanban without

		merging common theme resource and control style resource dictionaries. {% highlight xaml %} CommonTheme Transparent White Smoke ....{% endhighlight %}
	SfKanbanBackgroundColor	Background color of kanban.
	SfKanbanMinValidationColor	Color value of the error bar for minimum limit validation.
	SfKanbanMaxValidationColor	Color value of the error bar for maximum limit validation.
	SfKanbanValidationColor	Color value of the error bar for validation.
	SfKanbanPlaceholderBackgroundColor	Background color of the placeholder.
	SfKanbanPlaceholderBorderColor	Border color of the placeholder.
	SfKanbanPlaceholderTextColor	Text color of the placeholder.
	SfKanbanPlaceholderSelectedBackgroundColor	Background color of the selected placeholder.
	SfKanbanPlaceholderSelectedBorderColor	Border color of the selected placeholder.
	SfKanbanPlaceholderSelectedTextColor	Text color of the selected placeholder.
	SfKanbanCardBackgroundColor	Background color of the card.
	SfKanbanCardBorderColor	Border color of the card.
	SfKanbanCardTextColor	Text color of the card.
	SfKanbanCardTitleColor	Color of card title.
	SfKanbanCardTagBackgroundColor	Background color of the card tag.
	SfKanbanCardTagTextColor	Text color of the card tag.
	SfKanbanHeaderTitleTextColor	Text color of the header.
	SfKanbanHeaderBackgroundColor	Background color of the header.



	SfKanbanHeaderCollapsedBackgroundColor	Background color of the collapsed header.
	SfKanbanHeaderInfoTextColor	Text color of the header info.

## SfListView

Theme Dictionary	Keys	Description
SfListViewStyles	SfListViewTheme	By merging this key in application resources, you can customize the appearance of the SfListView without merging common theme resource and control style resource dictionaries.{% highlight xaml %} CustomTheme LightGreen DarkGreen ....{% endhighlight %}
	SfListViewBackgroundColor	Background color of ListView.
	SfListViewForegroundColor	Text color of the ListViewItem, when it in the default state.
	SfListViewSelectionBackgroundColor	Background color of the ListViewItem when it is in selection.
	SfListViewFocusBorderThickness	Border thickness of the ListViewItem when it is in selection.
	SfListViewFocusBorderColor	Border color of the ListViewItem when it is in selection.
	SfListViewGroupHeaderForegroundColor	Text color of the GroupHeaderItem, when it in the default state.
	SfListViewLoadMoreForegroundColor	Text color of the LoadMoreItem When it is in the default state.
	SyncPrimaryLightColor	Color of the LoadMoreIndicator when the indicator is in the progressed state.
	SfListViewLoadMoreBackgroundColor	Background color of the LoadMoreItem when it is in the default state.

	SfListViewLoadMoreOpacity	Percentage of transparency level for the text in the LoadMoreItem when it is in the default state.
--	---------------------------	--

## SfLinearProgressBar

Theme Dictionary	Keys	Description
SfProgressBarStyles	SfProgressBarTheme	By merging this key in application resources, you can customize the appearance of SfLinearProgressBar without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SyncPrimaryColor	Color of the progress indicator in the linear progress bar.
	SfProgressBarTrackColor	Color of the track in the linear progress bar.
	SfProgressBarSecondaryProgressColor	Color of the secondary progress(buffer) indicator in the linear progress bar.

## SfCircularProgressBar

Theme Dictionary	Keys	Description
SfProgressBarStyles	SfProgressBarTheme	By merging this key in application resources, you can customize the appearance of SfCircularProgressBar without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SyncPrimaryColor	Color of the progress indicator in the circular progress bar.
	SfProgressBarTrackColor	Color of the track in the circular progress bar.

## SfNumericTextBox

Theme Dictionary	Keys	Description
------------------	------	-------------

SfNumericTextBoxStyles	SfNumericTextBoxTheme	By merging this key in application resources, you can customize the appearance of SfNumericTextBox without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SyncPrimaryColor	Border color of the numeric text box.
	SfNumericTextBoxTextColor	Text color of the numeric text box.
	SfNumericTextBoxBackgroundColor	Background color of the numeric text box.
	SfNumericTextBoxBorderColor	Border color of the numeric text box.

## SfNumericUpDown

Theme Dictionary	Keys	Description
SfNumericUpDownStyles	SfNumericUpDownTheme	By merging this key in application resources, you can customize the appearance of SfNumericUpDown without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SyncPrimaryColor	Border color of numeric up down.
	SfNumericUpDownTextColor	Text color of the numeric up down.
	SfNumericUpDownBackgroundColor	Background color of numeric up down.
	SfNumericUpDownButtonBackgroundColor	Background color of the up and down buttons.

	SfNumericUpDownButtonForegroundColor	Foreground color of the up and down buttons.
	SfNumericUpDownBorderColor	Border color of the numeric up down.
	SfNumericUpDownHighlightedBackgroundColor	Sets the background color for the up-down button when it is pressed.
	SfNumericUpDownHighlightedFontColor	Sets the font color for the up-down button when it is pressed.

## SfDiagram

Theme Dictionary	Keys	Description
SfDiagramStyles	SfDiagramTheme	By merging this key in application resources, it is possible to customize the appearance of the SfDiagram without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SfDiagramStencilHeaderBackgroundColor	Background color of the stencil header.
	SfDiagramStencilHeaderForegroundColor	Foreground color of the stencil header.
	SfDiagramDialogHeaderForegroundColor	Foreground color of header in dialog box.
	SfDiagramDialogBackgroundColor	Background color of header in dialog box.
	SfDiagramLabelBackgroundColor	Background color of Label in dialog box.
	SfDiagramStencilBorderColor	Border color of stencil.
	SfDiagramStencilBackgroundColor	Background color of stencil.

## SfMaskedEdit

Theme Dictionary	Keys	Description
------------------	------	-------------

SfMaskedEditStyles	SfMaskedEditTheme	By merging this key in application resources, you can customize the appearance of SfMaskedEdit without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SyncPrimaryColor	Border color of the masked edit.
	SfMaskedEditTextColor	Text color of the masked edit.
	SfMaskedEditBackgroundColor	Background color of the masked edit.
	SfMaskedEditWatermarkColor	Watermark color of the masked edit.

## SfTreeView

Theme Dictionary	Keys	Description
SfTreeViewStyles	SfTreeViewTheme	By merging this key in application resources, you can customize the appearance of SfTreeView without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue White ....{% endhighlight %}
	SfTreeViewBackgroundColor	Background color of TreeView.
	SfTreeViewSelectionBackgroundColor	Background color of TreeViewItem when it is in selection.
	SfTreeViewContentForegroundColor	Text color of content view label when it is in default state.
	SyncPrimaryLightColor	Text color of content view label when it is in selection.
	SfTreeViewExpanderColor	Color of expander icon.
	SfTreeViewFocusBorderColor	Border color of focused item.
	SfTreeViewFocusBorderThickness	Border thickness of focused item.

## SfTextInputLayout

Theme Dictionary	Keys	Description
SfTextInputLayoutStyles	SfTextInputLayoutTheme	By merging this key in application resources, it is possible to customize the appearance of the SfTextInputLayout without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SyncPrimaryLightColor	Color of hint when text input layout is focused.
	SyncPrimaryColor	Color of line/border when text input layout is focused.
	SyncErrorColor	Color of hint when the input view contains error and is focused.
		Color of border when input view contains error.
		Color of counter label when input view contains error.
	SfTextInputLayoutHintColor	Color of hint when the text input layout is in the default state.
	SfTextInputLayoutDisabledHintColor	Color of hint when text input layout is disabled.
	SfTextInputLayoutFloatedHintDisabledColor	Color of hint when text input layout is disabled,

		and text inside the input view is floated.
	SfTextInputLayoutLineColor	Color of line/border when text input layout is in the default state.
	SfTextInputLayoutFilledLineDisabledColor	Color of line when text input layout is in the disabled state.
	SfTextInputLayoutOutlinedLineDisabledColor	Color of border when text input layout is in the disabled state.
	SfTextInputLayoutCounterLabelColor	Color of counter label when text input layout is in the default state.
	SfTextInputLayoutCounterLabelDisabledColor	Color of counter label when text input layout is in the disabled state.
	SfTextInputLayoutContainerBackgroundColor	Color of container background when the text input layout is in the default state.
	SfTextInputLayoutContainerBackgroundDisabledColor	Color of container background when text input layout is in the disabled state.
	SfTextInputLayoutHelperTextColor	Color of helper text when text input layout is in the default state.
	SfTextInputLayoutHelperTextDisabledColor	Color of helper text when text input layout is in the disabled state.
	SfTextInputLayoutErrorTextColor	Color of error text when text input layout is in the default state.

	SfTextInputLayoutErrorTextDisabledColor	Color of error text when text input layout is in the disabled state.
	SfTextInputLayoutPasswordToggleFocusedColor	Color of password toggle button when the input view is focused.
	SfTextInputLayoutPasswordToggleDisabledColor	Color of password toggle button when the text input layout is in the disabled state.
	SfTextInputLayoutPasswordToggleUnfocusedColor	Color of password toggle button when the input view is unfocused.
	SfTextInputLayoutFloatedHintUnfocusedColor	Color of floated hint text when input view is unfocused.

### SfBackdropPage

Theme Dictionary	Keys	Description
SfBackdropPageStyles	SfBackdropPageTheme	By merging this key in application resources, you can customize the appearance of SfBackdropPage without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Red ....{% endhighlight %}
	SfBackdropPageFrontLayerBackgroundColor	Background color of the front layer.

### SfCardView

Theme Dictionary	Keys	Description
SfCardViewStyles	SfCardViewTheme	By merging this key in application resources, you can customize the appearance of SfCardView without merging common theme resource and control style resource



		dictionaries. {% highlight xaml %} CustomTheme Red ....{% endhighlight %}
	CardViewContentBackgroundColor	Background color of the card view content.

## SfPdfViewer

Theme Dictionary	Keys	Description
SfPdfViewerStyles	SfPdfViewerTheme	By merging this key in application resources, you can customize the appearance of SfPdfViewer without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CommonTheme Cyan Red ....{% endhighlight %}
	SfPdfViewerButtonIconEnabledColor	Text color of the button when it is in enabled state.
	SfPdfViewerButtonIconDisabledColor	Text color of the button when it is in disabled state.
	SfPdfViewerButtonPressedColor	Pressed color of the toolbar buttons.
	SfPdfViewerToolbarBackgroundColor	Background color of the toolbar.
	SfPdfViewerToolbarBorderColor	Border color of the toolbar.
	SfPdfViewerToolbarSeparatorColor	Toolbar separator color.
	SfPdfViewerEntryTextColor	Text color of entry.
	SfPdfViewerEntryBorderColor	Border color of entry.

	SfPdfViewerEntryPlaceholderColor	Placeholder color of entry.
	SfPdfViewerEntryBackgroundColor	Background color of entry.
	SfPdfViewerBookmarkItemBackgroundColor	Background color of the bookmark items.
	SfPdfViewerBookmarkTitleBarBackgroundColor	Background color of the bookmark title bar.
	SfPdfViewerBookmarkTitleBarTextColor	Text color of the bookmark title bar.
	SfPdfViewerTextElementColor	Text color of the PDF Viewer text labels.
	SfPdfViewerMoreMenuBackgroundColor	Background color of the more menu bar.
	SfPdfViewerMoreMenuTextDisabledColor	Text color of the more menu items when it is in disabled state.
	SfPdfViewerMoreMenuTextEnabledColor	Text color of the more menu items when it is in enabled state.
	SfPdfViewerScrollHeadColor	Background color of the scroll head label.
	SfPdfViewerScrollHeadBorderColor	Border color of the scroll head.
	SfPdfViewerScrollHeadPageNumberViewBackgroundColor	Background color of the page number alert dialog.
	SfPdfViewerScrollHeadPageNumberViewSeparatorColor	Separator color of the page number alert dialog.
	SfPdfViewerSignaturePadTitleColor	Title color of the signature pad.
	SfPdfViewerSignaturePadColor	Background color of the signature pad view.

	SfPdfViewerSignaturePadHeaderFooterColor	Background color of header and footer of signature pad.
	SfPdfViewerSignatureBorderColor	Border color of the signature pad.
	SfPdfViewerSignaturePadConfirmationButtonDisabledColor	Text color of the signature pad confirmation button when it is in disabled state.
	SfPdfViewerSignaturePadCancelButtonEnabledColor	Text color of the signature pad cancel button when it is in enabled state.
	SfPdfViewerSignaturePadCancelButtonDisabledColor	Text color of the signature pad cancel button when it is in disabled state.
	SfPdfViewerSignaturePadClearButtonEnabledColor	Text color of the signature pad clear button when it is in enabled state.
	SfPdfViewerSignaturePadClearButtonDisabledColor	Text color of the signature pad clear button when it is in disabled state.
	SfPdfViewerEditWindowPlaceHolderColor	Placeholder color of the edit text window.
	SfPdfViewerSearchBarBackgroundColor	Background color of the search bar.
	SfPdfViewerFreeTextConfirmationButtonDisabledColor	Text color of the FreeText confirmation button when it is in disabled state.

	SfPdfViewerFreeTextCancelButtonEnabledColor	Text color of the FreeText cancel button when it is in enabled state.
	SfPdfViewerFreeTextCancelButtonDisabledColor	Text color of the FreeText cancel button when it is in disabled state.
	SfPdfViewerThicknessViewBackgroundColor	Background color of the thickness selector toolbar.
	SfPdfViewerThicknessViewButtonBackgroundColor	Background color of the thickness button in thickness selector toolbar.
	SfPdfViewerThicknessViewButtonBorderDisabledColor	Border color of the thickness button when it is in disabled state.
	SfPdfViewerAlertDialogBackgroundColor	Background color of the page number alert dialog.
	SfPdfViewerAlertDialogTitleColor	Title color of the page number alert dialog.
	SfPdfViewerAlertDialogTextColor	Text color of the page number alert dialog.
	SfPdfViewerAlertDialogBorderColor	Border color of the page number alert dialog.

## SfDataGrid

Theme Dictionary	Keys	Description
SfDataGridStyles	SfDataGridTheme	By merging this key in application resources, it is possible to customize the appearance of the SfDataGrid without merging common theme resource and control style resource dictionaries. {% highlight xaml %}

		CustomTheme Blue Green ....{%endhighlight %}
	SfDataGridBackgroundColor	Background color of the remaining area other than rows/columns is rendered in DataGrid.
	SfDataGridHeaderBackgroundColor	Background color of the row and column headers in the SfDataGrid.
	SfDataGridHeaderForegroundColor	Foreground color of the row and column headers in the SfDataGrid.
	SfDataGridHeaderBorderColor	Border color of the row and column header cells in the SfDataGrid.
	SfDataGridRecordBackgroundColor	Background color of the grid cells in the SfDataGrid.
	SfDataGridRecordForegroundColor	Foreground color of the grid cells in the SfDataGrid.
	SfDataGridBorderColor	Border color of the grid cells in the SfDataGrid.
	SfDataGridSelectionBackgroundColor	Background color of the selected row in SfDataGrid.
	SfDataGridSelectionForegroundColor	Foreground color of the selected row in SfDataGrid.
	SfDataGridAlternatingRowBackgroundColor	Background color of the alternate rows in SfDataGrid.
	SfDataGridRowDragViewBackgroundColor	Background color of the row drag view.
	SfDataGridRowDragViewForegroundColor	Foreground color of the row drag view.
	SfDataGridColumnDragViewBackgroundColor	Background color of the column drag view.
	SfDataGridColumnDragViewForegroundColor	Foreground color of the column drag view.

	SfDataGridColumnDragViewBorderColor	Border color of the column drag view.
	SfDataGridTableSummaryBackgroundColor	Background color of the <a href="#">TableSummaryRow</a> in SfDataGrid.
	SfDataGridTableSummaryForegroundColor	Foreground color of the <a href="#">TableSummaryRow</a> in SfDataGrid.
	SfDataGridCaptionSummaryRowBackgroundColor	Background color of the <a href="#">CaptionSummaryRow</a> in SfDataGrid.
	SfDataGridCaptionSummaryRowForegroundColor	Foreground color of the <a href="#">CaptionSummaryRow</a> in SfDataGrid.
	SfDataGridGroupSummaryRowBackgroundColor	Background color of the <a href="#">GroupSummaryRow</a> in SfDataGrid.
	SfDataGridGroupSummaryRowForegroundColor	Foreground color of the <a href="#">GroupSummaryRow</a> in SfDataGrid.
	SfDataGridLoadMoreViewBackgroundColor	Background color of the LoadMore view in SfDataGrid.
	SfDataGridLoadMoreViewForegroundColor	Foreground color of the LoadMore view in SfDataGrid.
	SfDataGridIndentBackgroundColor	Background color of the indent column in SfDataGrid.

### SfPopupLayout

Theme Dictionary	Keys	Description
SfPopupLayoutStyles	SfPopupLayoutTheme	By merging this key in application resources, it is possible to customize the appearance of the SfPopupLayout without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Blue ....{% endhighlight %}

	SfPopupLayoutHeaderBackgroundColor	Background color of the header view in SfPopupLayout.
	SfPopupLayoutFooterBackgroundColor	Background color of the footer view in SfPopupLayout.
	SfPopupLayoutAcceptButtonBackgroundColor	Background color of the accept button in Xamarin.Forms.Android and Xamarin.Forms.iOS platform.
		Note: Background color of the accept button for Xamarin.Forms.UWP platform is obtained from SyncPrimaryColor key value.
	SfPopupLayoutDeclineButtonTextColor	Text color of the decline button in Xamarin.Forms.UWP platform.
		Note: Text color of the decline button for Xamarin.Forms.Android and Xamarin.Forms.iOS platform is obtained from SyncPrimaryLightColor key value.
	SfPopupLayoutBorderColor	Border color of the SfPopupLayout in Xamarin.Forms.Android and Xamarin.Forms.iOS platform.
		Note: Border color of the SfPopupLayout for Xamarin.Forms.UWP platform is obtained from SyncPrimaryColor key value.
	SfPopupLayoutDeclineButtonBackgroundColor	Background color of the decline button in Xamarin.Forms.UWP platform.
		Note: Background color of the decline button for Xamarin.Forms.Android and Xamarin.Forms.iOS platform is obtained from SyncPrimaryLightColor key value.

	SfPopupLayoutHeaderTextColor	Text color of the header in SfPopupLayout.
--	------------------------------	--

### SfPullToRefresh

Theme Dictionary	Keys	Description
SfPullToRefreshStyles	SfPullToRefreshTheme	By merging this key in application resources, it is possible to customize the appearance of the SfPullToRefresh without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue ....{% endhighlight %}
	SfPullToRefreshBackgroundColor	Background color of the circle view in SfPullToRefresh.

### SfDataForm

Theme Dictionary	Keys	Description
SfDataFormStyles	SfDataFormTheme	By merging this key in application resources, it is possible to customize the appearance of the SfDataForm without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Red Pink ....{% endhighlight %}
	SyncPrimaryColor	Border color of dataform editors when focused and date time picker's background color in dataform.
	SfDataFormBackgroundColor	Background color of the dataform.
	SfDataFormEditorBackgroundColor	Background color of the dataform editors.
	SfDataFormEditorDisabledBackgroundColor	Disabled editor's background color in dataform.
	SfDataFormEditorTextColor	Text color of the dataform editors.



	SfDataFormEditorPlaceholderColor	Placeholder color of the editor in dataform.
	SfDataFormEditorDisabledTextColor	Disabled editor`s text color in dataform.
	SfDataFormLabelTextColor	Text color of the dataform labels.
	SfDataFormLabelDisabledTextColor	Disabled label`s text color in dataform.
	SfDataFormLabelBackgroundColor	Background color of the dataform labels.
	SfDataFormLabelDisabledBackgroundColor	Disabled label`s background color in dataform.
	SfDataFormEditorBorderColor	Border color of the dataform editors.

## SfSchedule

Theme Dictionary	Keys	Description
SfScheduleStyles	SfScheduleTheme	By merging this key in application resources, it is possible to customize the appearance of the SfSchedule without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Red Blue Blue ....{% endhighlight %}
	SyncPrimaryColor	Change the color of today selection background in schedule month view.
	SyncPrimaryLightColor.	Change the color of current day(Sun,Mon,etc)

		text in schedule day, week and work week view.
		Change the color of current date(10,11,etc) text in schedule day, week and work week view.
	SyncPrimaryForegroundColor	Change the color of current day text in schedule month view.
	SfScheduleBackgroundColor	Change the color of schedule background.
	SfScheduleDayViewNonWorkingHoursTimeSlotColor	Change the color of non working hour time slot in schedule day view.
	SfScheduleDayViewNonWorkingHoursTimeSlotBorderColor	Change the color of non working hour time slot border in schedule day view.
	SfScheduleDayViewTimeSlotColor	Change the color of time slot in schedule day view.
	SfScheduleDayViewTimeSlotBorderColor	Change the color of time slot border color of schedule day view.
	SfScheduleDayViewVerticalLineColor	Change the color of time slot vertical lines at schedule day view.
	SfScheduleDayViewAllDayAppointmentLayoutColor	Change the color of all day appointment layout color, when

		enable the all day appointment in schedule day view.
	SfScheduleWeekViewNonWorkingHoursTimeSlotColor	Change the color of non working hour time slot in schedule week view.
	SfScheduleWeekViewNonWorkingHoursTimeSlotBorderColor	Change the color of non working hour time slot border in schedule week view.
	SfScheduleWeekViewTimeSlotColor	Change the color of time slot in schedule week view.
	SfScheduleWeekViewTimeSlotBorderColor	Change the color of time slot border in schedule week view.
	SfScheduleWeekViewVerticalLineColor	Change the color of time slots vertical line in schedule week view.
	SfScheduleWeekViewAllDayAppointmentLayoutColor	Change the color of all day appointment layout color, when enable the all day appointment in schedule week view.
	SfScheduleWorkWeekViewNonWorkingHoursTimeSlotColor	Change the color of non working hour timeslot in schedule work week view.
	SfScheduleWorkWeekViewNonWorkingHoursTimeSlotBorderColor	Change the color of non working hour time slot border in schedule work week view.

	SfScheduleWorkWeekViewTimeSlotColor	Change the color of time slot in schedule work week view.
	SfScheduleWorkWeekViewTimeSlotBorderColor	Change the color of time slot border in schedule work week view.
	SfScheduleWorkWeekViewVerticalLineColor	Change the color of time slot vertical line in schedule work week view.
	SfScheduleWorkWeekViewAllDayAppointmentLayout Color	Change the color of all day appointment layout color, when enable the all day appointment in schedule work week view.
	SfScheduleMonthViewTodayBackground	Change the color of current date circle background at schedule month view.
	SfScheduleMonthViewSelectionTextColor	Change the color of select month cell text at schedule month view.
	SfScheduleDayViewLabelTimeLabelColor	Change the color of time slot labels(2pm,3pm,etc) at schedule day view.
	SfScheduleWeekViewLabelTimeLabelColor	Change the color of time slot labels(2pm,3pm,etc) at schedule week view.
	SfScheduleWorkWeekViewLabelTimeLabelColor	Change the color of time slot

		labels(2pm,3pm,etc) at schedule work week view.
	SfScheduleHeaderBackgroundColor	Change the color of header layout in schedule.
	SfScheduleHeaderTextColor	Change the color of header text in schedule.
	SfScheduleViewHeaderDayTextColor	Change the color of view header day text(Sunday,Mondays,etc) in schedule.
	SfScheduleViewHeaderBackgroundColor	Change the color of view header layout in schedule.
	SfScheduleViewHeaderDateTextColor	Change the color of view header date text(10,11,etc) in schedule.
	SfScheduleSelectionBackgroundColor	Change the color of selection background at schedule.
	SfScheduleSelectionBorderColor	Change the color of selection border at schedule.
	SfScheduleAppointmentTextColor	Change the color of appointment text in schedule day, week and work week view.
	SfScheduleAppointmentBorderColor	Change the border color of appointment layout in schedule day,

		week and work week view.
	SfScheduleAppointmentSelectionBorderColor	Change the border color of appointment layout while select in schedule day, week and work week view.
	SfScheduleAppointmentSelectionTextColor	Change the text color of appointment while select in schedule day, week and work week view.
	SfScheduleMonthCellTextColor	Change the color of month cell text in schedule month view.
	SfScheduleMonthCellBackgroundColor	Change the color of month cell background in schedule month view.
	SfScheduleMonthCellTodayBackgroundColor	Change the current day background color of month cell in schedule month view.
	SfScheduleMonthCellTodayTextColor	Change the current date text color of month cell in schedule month view.
	SfScheduleMonthCellPreviousMonthTextColor	Change the previous month date text color of month cell in schedule month view.

	SfScheduleMonthCellPreviousMonthBackgroundColor	Change the previous month background color of month cell in schedule month view.
	SfScheduleMonthCellNextMonthBackgroundColor	Change the next month background color of month cell in schedule month view.
	SfScheduleMonthCellNextMonthTextColor	Change the next month date text color of month cell in schedule month view.
	SfScheduleMonthViewWeekNumberTextColor	Change the week number text color, when enable the week number view in schedule month view.
	SfScheduleMonthViewWeekNumberBackgroundColor	Change the week number background color, when enable the week number view in schedule month view.
	SfScheduleMonthAgendaViewSubjectFontColor	Change the appointment subject text color, when enable the agenda view in schedule month view.
	SfScheduleMonthAgendaViewBackgroundColor	Change the agenda view layout background color of schedule month view.

	SfScheduleMonthAgendaViewTimeFontColor	Change the appointment time text color, when enable the agenda view in schedule month view.
	SfScheduleMonthAgendaViewDateFontColor	Change the selected date text color, when enable the agenda view in schedule month view.
SfScheduleTimeIndicatorTextColor	Change the time indicator text color, when enable drag and drop in schedule day, week and work week view.	

## SfCalendar

Theme Dictionary	Keys	Description
SfCalendarStyles	SfCalendarTheme	By merging this key in application resources, it is possible to customize the appearance of the SfCalendar without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Green White Blue Blue Green ....{% endhighlight %}
	SyncPrimaryLightColor.	Change the color of current day text in calendar month view.
		Change the color of month header text in calendar year view.
	SfCalendarInlineTextColor	Change the color of inline appointment text in calendar month view.
	SfCalendarBlackoutColor	Change the color of black out date denoting view in calendar month view.



	SfCalendarDateSelectionColor	Change the color of selected date background in calendar month view.
	SfCalendarInlineBackgroundColor	Change the color of inline layout background in calendar month view.
	SfCalendarSelectedDayTextColor	Change the color of select date text color in calendar month view.
	SfCalendarBorderColor	Change the border color(vertical/horizontal line) of calendar month view.
	SfCalendarWeekDayBackgroundColor	Change the color of week day(Monday to Friday month cell) background in calendar month view.
	SfCalendarWeekDayTextColor	Change the color of week day(Monday to Friday month cell) text in calendar month view.
	SfCalendarDisabledBackgroundColor	Change the color of disabled month cell date background in calendar month view.
	SfCalendarDisabledTextColor	Change the color of disabled month cell date text in calendar month view.
	SfCalendarPreviousMonthBackgroundColor	Change the color of previous month cell date background in calendar month view.
	SfCalendarPreviousMonthTextColor	Change the color of previous month cell date text in calendar month view.
	SfCalendarCurrentMonthBackgroundColor	Change the color of current month cell background in calendar month view.
	SfCalendarCurrentMonthTextColor	Change the color of current month cell date text in calendar month view.
	SfCalendarWeekEndTextColor	Change the color of week end(Sunday & Saturday month cell) text in calendar month view.

	SfCalendarDayHeaderTextColor	Change the color day header (Sunday,Monday,etc) text color in calendar month view.
	SfCalendarDayHeaderBackgroundColor	Change the color day header (Sunday,Monday,etc) background color in calendar month view.
	SfCalendarWeekEndBackgroundColor	Change the color of week end(Sunday & Saturday month cell) background in calendar month view.
	SfCalendarHeaderBackgroundColor	Change the color of header layout background in calendar.
	SfCalendarHeaderTextColor	Change the color of header layout date text in calendar.
	SfCalendarYearViewLayoutBackground	Change the color of year view layout background in calendar.
	SfCalendarYearViewHeaderBackground	Change the color of header layout background in calendar year view.
	SfCalendarYearViewMonthLayoutBackground	Change the color of month layout(Jan,Feb,etc) background in calendar year view.
	SfCalendarYearViewMonthHeaderBackground	Change the color of month layout header(Jan,Feb,etc) background in calendar year view.
	SfCalendarYearViewDateTextColor	Change the color of month date (10,11,etc) text in calendar year view.
	SfCalendarYearViewHeaderTextColor	Change the color of month header text(Jan,Feb,etc) in calendar year view.

## SfMaps

Theme Dictionary	Keys	Description
SfMapsStyles	SfMapsTheme	By merging this key in application resources, it is possible to customize the appearance of the SfMaps without merging common theme resource and control style resource

		dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SfMapsBackgroundColor	Color of maps background.
	SfMapsSelectedShapeColor	Color of selected shape in maps.
	SfMapsSelectedShapeStrokeColor	Color of selected shape stroke color in maps.
	SfMapsBubbleMarkerFillColor	Color of bubble marker in maps.
	SfMapsLegendTextColor	Color of legend text in maps.
	SfMapsDataLabelTextColor	Color of data label text in maps.
	SfMapsMarkerLabelColor	Color of marker label in maps.
	SfMapsMarkerIconColor	Color of marker icon in maps.
	SfMapsTooltipBackgroundColor	Color of tooltip background in maps.
	SfMapsTooltipStrokeColor	Color of tooltip stroke in maps.
	SfMapsTooltipTextColor	Color of tooltip text in maps.

## SfCircularGauge

Theme Dictionary	Keys	Description
SfCircularGaugeStyles	SfCircularGaugeTheme	By merging this key in application resources, it is possible to customize the appearance of the SfCircularGauge without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SfCircularGaugeBackgroundColor	Color of circular gauge background.
	SfCircularGaugeNeedlePointerColor	Color of needle pointer in circular gauge.
	SfCircularGaugeSymbolPointerColor	Color of symbol pointer in circular gauge.

	SfCircularGaugeRangePointerColor	Color of range pointer in circular gauge.
	SfCircularGaugeRangeColor	Color of range in circular gauge.
	SfCircularGaugeHeaderColor	Color of header in circular gauge.
	SfCircularGaugeMinorTickColor	Color of minor tick in circular gauge.
	SfCircularGaugeMajorTickColor	Color of major tick in circular gauge.
	SfCircularGaugeKnobColor	Color of knob in circular gauge.
	SfCircularGaugeKnobStrokeColor	Color of knob stroke in circular gauge.
	SfCircularGaugeTailColor	Color of tail in circular gauge.
	SfCircularGaugeTailStrokeColor	Color of tail stroke in circular gauge.

## SfLinearGauge

Theme Dictionary	Keys	Description
SfLinearGaugeStyles	SfLinearGaugeTheme	By merging this key in application resources, it is possible to customize the appearance of the SfLinearGauge without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SfLinearGaugeLabelColor	Color of label in linear gauge.
	SfLinearGaugeRangeColor	Color of range in linear gauge.
	SfLinearGaugeHeaderColor	Color of header in linear gauge.
	SfLinearGaugeSymbolColor	Color of symbol pointer in linear gauge.
	SfLinearGaugeTickColor	Color of tick in linear gauge.
	SfLinearGaugeBarColor	Color of bar pointer in linear gauge.

## SfDigitalGauge

Theme Dictionary	Keys	Description
------------------	------	-------------

SfDigitalGaugeStyles	SfDigitalGaugeTheme	By merging this key in application resources, it is possible to customize the appearance of the SfDigitalGauge without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SfDigitalGaugeCharacterStrokeColor	Color of character stroke in digital gauge.

### SfSunburstChart

Theme Dictionary	Keys	Description
SfSunburstChartStyles	SfSunburstChartTheme	By merging this key in application resources, it is possible to customize the appearance of the SfSunburstChart without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SfSunburstChartBackgroundColor	Color of sunburst background.
	SfSunburstSelectionColor	Color of selection when segment is focused in sunburst.
	SfSunburstSelectionStrokeColor	Color of selection stroke when segment is focused in sunburst.
	SfSunburstTooltipBorderColor	Color of tooltip border in sunburst.
	SfSunburstTooltipBackgroundColor	Color of tooltip background in sunburst.
	SfSunburstTooltipTextColor	Color of tooltip text in sunburst.

### SfSparkline

Theme Dictionary	Keys	Description
SfSparklineStyles	SfSparklineTheme	By merging this key in application resources, it is possible to customize the appearance of the SfSparkline without merging common theme resource and control style resource dictionaries. {% highlight xaml %}

		CustomTheme Blue Green ....{% endhighlight %}
	SfSparklineAreaBackgroundColor	Color of area sparkline background.
	SfLineSparklineBackgroundColor	Color of line sparkline background.
	SfColumnSparklineBackgroundColor	Color of column sparkline background.
	SfWinLossSparklineBackgroundColor	Color of win loss sparkline background.
	SfSparklineMarkerBaseColor	Color of marker base in sparkline.
	SfSparklineAxisStrokeColor	Color of axis stroke in sparkline.
	SfSparklineRangeBandColor	Color of range band in sparkline.
	SfWinLossSparklineStrokeColor	Color of stroke color in sparkline.
	SfWinLossSparklineColor	Color of win loss in sparkline.
	SfColumnSparklineStrokeColor	Color of stroke in column sparkline.
	SfColumnSparklineColor	Color of column sparkline.

## SfTreeMap

Theme Dictionary	Keys	Description
SfTreeMapStyles	SfTreeMapTheme	By merging this key in application resources, it is possible to customize the appearance of the SfTreeMap without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SfTreemapGroupBackgroundColor	Color of group background in treemap.
	SfTreemapLeafItemBorderColor	Color of leaf item border in maps.
	SfTreemapTooltipBackgroundColor	Color of tooltip background in treemap.
	SfTreemapTooltipStrokeColor	Color of tooltip stroke in treemap.

	SfTreemapTooltipTextColor	Color of tooltip text in treemap.
--	---------------------------	-----------------------------------

### SfImageEditor

Theme Dictionary	Keys	Description
SfImageEditorStyles	SfImageEditorTheme	By merging this key in application resources, it is possible to customize the appearance of the SfImageEditor without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Green ....{% endhighlight %}
	SfImageEditorToolBarBackgroundColor	Color of background in image editor.
	SfImageEditorToolBarTextColor	Color of tool bar text in image editor.

### SfAutoComplete

Theme Dictionary	Keys	Description
SfAutoCompleteStyles	SfAutoCompleteTheme	By merging this key in application resources, you can customize the appearance of SfAutoComplete without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Red Green ....{% endhighlight %}
	SyncPrimaryColor	Color of indicator text in AutoComplete.
	SyncPrimaryLightColor	Color of highlighted text in AutoComplete.
	SfAutoCompleteBackgroundColor	Background color of AutoComplete.
	SfAutoCompleteTextColor	Text color of AutoComplete.
	SfAutoCompleteBorderColor	Border color of AutoComplete.

	SfAutoCompleteClearButtonColor	Color of the close button when enter the input text.
	SfAutoCompleteWatermarkColor	Color of AutoComplete's watermark text.
	SfAutoCompleteDropDownBackgroundColor	Background color of the drop-down box.
	SfAutoCompleteDropDownTextColor	Text color of items in the drop-down box.
	SfAutoCompleteTokenBackgroundColor	Background color of token.
	SfAutoCompleteTokenTextColor	Color of token's text.
	SfAutoCompleteTokenDeleteButtonColor	Color of token's delete button.

## SfBusyIndicator

Theme Dictionary	Keys	Description
SfBusyIndicatorStyles	SfBusyIndicatorTheme	By merging this key in application resources, you can customize the appearance of SfBusyIndicator without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Black ....{% endhighlight %}
	SyncPrimaryColor	Color of the SfBusyIndicator text.

## SfButton

Theme Dictionary	Keys	Description
SfButtonStyles	SfButtonTheme	By merging this key in application resources, you can customize the appearance of SfButton without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Black White ....{% endhighlight %}
	SyncPrimaryColor	Background color of the SfButton.
		Border color of the SfButton.
	SyncPrimaryDarkColor	Color of the button when it is in the pressed state.



	SyncPrimaryForegroundColor	Text color of the SfButton.
--	----------------------------	-----------------------------

## SfSwitch

Theme Dictionary	Keys	Description
SfSwitchStyles	SfSwitchTheme	By merging this key in application resources, you can customize the appearance of SfSwitch without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Black White ....{% endhighlight %}
	SyncPrimaryColor	Thumb color of the material switch in On state.
		Thumb border color of the material switch in On state.
		Track color of the Cupertino switch in On state.
		Track border color of the Cupertino switch in On state.
		Track color of the Fluent switch in On state.
		Track border color of the Fluent switch in On state.
	SyncPrimaryForegroundColor	Thumb color of the Cupertino switch in On state.
		Thumb border color of the Cupertino switch in On state.
		Thumb color of the Fluent switch in On state.
		Thumb border color of the Fluent switch in On state.

	MaterialOffThumbColor	Thumb color of the material switch in off state.
	MaterialOffTrackColor	Track color of the material switch in off state.
	MaterialOffThumbBorderColor	Thumb border color of the material switch in off state.
	MaterialOffTrackBorderColor	Track border color of the material switch in off state.
	MaterialIndeterminateTrackColor	Indeterminate track color of the material switch.
	MaterialIndeterminateTrackBorderColor	Indeterminate track border color of the material switch.
	CupertinoOffThumbBorderColor	Thumb border color of the Cupertino switch in off state.
	CupertinoOffTrackColor	Track color of the Cupertino switch in off state.
	CupertinoOffTrackBorderColor	Track border color of the Cupertino switch in off state.
	FluentOffThumbColor	Thumb color of the fluent switch in off state.
	FluentOffTrackColor	Track color of the fluent switch in off state.
	FluentOffThumbBorderColor	Thumb border color of the fluent switch in off state.
	FluentOffTrackBorderColor	Border color of the fluent switch in off state.

### SfBorder

Theme Dictionary	Keys	Description
SfBorderStyles	SfBorderTheme	By merging this key in application resources, you can customize the appearance of SfBorder without merging common theme resource and control style resource

		dictionaries. {% highlight xaml %} CustomTheme LightGray Gray ....{% endhighlight %}
	SfBorderBackgroundColor	Background color of the border.
	SfBorderBorderColor	Border color of the border control.

### SfCarousel

Theme Dictionary	Keys	Description
SfCarouselStyles	SfCarouselTheme	By merging this key in application resources, you can customize the appearance of SfCarousel without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme White ....{% endhighlight %}
	SfCarouselBackgroundColor	Background color of the SfCarousel.

### SfCheckBox

Theme Dictionary	Keys	Description
SfCheckBoxStyles	SfCheckBoxTheme	By merging this key in application resources, you can customize the appearance of SfCheckBox without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme White Black ....{% endhighlight %}
	SfCheckBoxUncheckedColor	Color of unchecked item.
	SyncPrimaryColor	Color of checked item.
	SfCheckBoxBackgroundColor	Background color of the SfCheckBox.
	SfCheckBoxTextColor	Text color of the SfCheckBox.

### SfChip

Theme Dictionary	Keys	Description
SfChipStyles	SfChipTheme	By merging this key in application resources, you can customize the appearance of SfChip without merging common theme resource and control style resource dictionaries. {% highlight xaml %}

		CustomTheme LightGray Transparent ....{% endhighlight %}
	SfChipTextColor	Color of the text in the SfChip.
	SfChipCloseButtonColor	Color of the close button in SfChip.
	SfChipSelectionIndicatorColor.	Color of the selection indicator in SfChip.
	SfChipBackgroundColor	Background color of the SfChip.
	SfChipBorderColor	Border color of the SfChip.

### SfChipGroup

Theme Dictionary	Keys	Description
SfChipGroupStyles	SfChipGroupTheme	By merging this key in application resources, you can customize the appearance of SfChipGroup without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Blue Transparent ....{% endhighlight %}
	SfChipGroupTextColor	Color of the text in SfChipGroup.
	SfChipGroupCloseButtonColor	Color of the close button in SfChipGroup.
	SfChipSelectionIndicatorColor.	Color of the selection indicator in SfChipGroup.
	SfChipGroupBackgroundColor	Background color of SfChipGroup.
	SfChipGroupBorderColor	Border color of SfChipGroup.
	SyncPrimaryColor	Background color of the selected chip in SfChipGroup.
	SyncPrimaryForegroundColor	Text color of the selected chip in SfChipGroup.

### SfComboBox

Theme Dictionary	Keys	Description
SfComboBoxStyles	SfComboBoxTheme	By merging this key in application resources,

		you can customize the appearance of SfComboBox without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme White Black ....{% endhighlight %}
	SyncPrimaryColor	Color of indicator text in ComboBox.
	SyncPrimaryLightColor	Color of highlighted text in ComboBox.
	SfComboBoxBackgroundColor	BackgroundColor of SfComboBox.
	SfComboBoxTextColor	Text color of SfComboBox.
	SfComboBoxBorderColor	Border color of SfComboBox.
	SfComboBoxClearButtonColor	Color of the close button in SfComboBox.
	SfComboBoxWatermarkColor	Color of the watermark's text in SfComboBox.
	SfComboBoxDropDownBackgroundColor	Background color of the suggestion box in SfComboBox.
	SfComboBoxSelectedDropDownItemColor	Color of the suggestion box's item in SfComboBox.
	SfComboBoxDropDownTextColor	Color of the suggestion box's text in SfComboBox.
	SfComboBoxTokenBackgroundColor	Background color of the token in SfComboBox.

	SfComboBoxTokenTextColor	Color of the token's text in SfComboBox.
	SfComboBoxTokenDeleteButtonColor	Color of the token's delete button in SfComboBox.
	SfComboBoxDropDownButtonBackgroundColor	Background color of the drop-down button in SfComboBox.
	SfComboBoxDropDownButtonTextColor	Color of the drop-down button's text in SfComboBox.
	SfComboBoxHighlightedDropDownButtonBackgroundColor	Highlighted Background color of the drop-down button in SfComboBox.

### SfNavigationDrawer

Theme Dictionary	Keys	Description
SfNavigationDrawerStyles	SfNavigationDrawerTheme	By merging this key in application resources, you can customize the appearance of SfNavigationDrawer without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Black ....{% endhighlight %}
	SfNavigationDrawerBackgroundColor	Background color of the SfNavigationDrawer.

### SfPicker

Theme Dictionary	Keys	Description
SfPickerStyles	SfPickerTheme	By merging this key in application resources, you can customize the appearance of SfPicker without merging common theme resource and control style resource dictionaries. {% highlight

		xaml %} CustomTheme Black Red ....{% endhighlight %}
	SyncPrimaryColor	Background color of header in SfPicker.
	SyncPrimaryLightColor	Color of SelectedItem's text in SfPicker.
	SyncPrimaryForegroundColor	Color of Header's text in SfPicker.
	SfPickerUnselectedItemTextColor	Color of UnSelectedItem's text in SfPicker.
	SfPickerColumnHeaderTextColor	Color of column header's text in SfPicker.
	SfPickerColumnHeaderBackgroundColor	Background color of theColumn header in SfPicker.
	SfPickerSelectionBackgroundColor	Background color of the selected item in SfPicker.
	SfPickerBackgroundColor	Background color of the SfPicker.
	SfPickerBorderColor	Border color of the SfPicker.

### SfRadioButton

Theme Dictionary	Keys	Description
SfRadioButtonStyles	SfRadioButtonTheme	By merging this key in application resources, you can customize the appearance of SfRadioButton without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme White Black ....{% endhighlight %}
	SyncPrimaryColor	Color of checked item.
	SfRadioButtonUncheckedColor	Color of unchecked item.
	SfRadioButtonBackgroundColor	Background color of the SfRadioButton.
	SfRadioButtonTextColor	Text color of the SfRadioButton.

## SfRadialMenu

Theme Dictionary	Keys	Description
SfRadialMenuStyles	SfRadialMenuTheme	By merging this key in application resources, you can customize the appearance of SfRadialMenu without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Gray White ....{% endhighlight %}
	SfRadialMenuPressedColor	Selection color of SfRadialMenu.
	SfRadialMenuCenterButtonBorderColor	Color of the center button's border in SfRadialMenu.
	SfRadialMenuCenterButtonBackgroundColor	Color of the center button's background in SfRadialMenu.
	SfRadialMenuCenterButtonTextColor	Color of the center button's text in SfRadialMenu.
	SfRadialMenuRimColor	Color of the rim when tap the center button of SfRadialMenu.
	SfRadialMenuSeparatorColor	Color of separator that appears between two items in SfRadialMenu.
	SfRadialMenuTextColor	Color of RadialMenu's text.
	SfRadialMenuIconFontColor	Color of font icon in SfRadialMenu.

## SfRangeSlider

Theme Dictionary	Keys	Description
SfRangeSliderStyles	SfRangeSliderTheme	By merging this key in application resources, you can customize the appearance of SfRangeSlider without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme White Gray ....{% endhighlight %}
	SyncPrimaryColor	Selection color of track in SfRangeSlider.



		Knob color in SfRangeSlider
	SfRangeSliderBackgroundColor	Background color of SfRangeSlider.
	SfRangeSliderTickColor	Color for ticks in SfRangeSlider.
	SfRangeSliderLabelColor	Color of SfRangeSlider's label.
	SfRangeSliderUnselectedTrackColor	Color of unselected track in SfRangeSlider.
	SfRangeSliderToolTipBackgroundColor	Background color of tooltip in SfRangeSlider.
	SfRangeSliderToolTipTextColor	Color of tooltip's text in SfRangeSlider.

### SfRating

Theme Dictionary	Keys	Description
SfRatingStyles	SfRatingTheme	By merging this key in application resources, you can customize the appearance of SfRating without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme White Green ....{% endhighlight %}
	SyncPrimaryColor	Stroke color of rated items.
		Fill color of rated items.
	SfRatingToolTipBackgroundColor	Background color of the tooltip in SfRating.
	SfRatingToolTipTextColor	Text color inside the tooltip in SfRating.
	SfRatingUnratedFillColor	Fill color for unrated items in SfRating.
	SfRatingUnratedStrokeColor	Stroke color for unrated items in SfRating.

### SfRotator

Theme Dictionary	Keys	Description
SfRotatorStyles	SfRotatorTheme	By merging this key in application resources,

		you can customize the appearance of SfRotator without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme White Gray ....{% endhighlight %}	
	SyncPrimaryColor	Color of selected dots in SfRotator.	
	SfRotatorUnselectedDotColor	Color of Unselected dots in SfRotator.	
	SfRotatorDotsBorderColor		Color of dots border in SfRotator.

### SfSegmentedControl

Theme Dictionary	Keys	Description
SfSegmentedControlStyles	SfSegmentedControlTheme	By merging this key in application resources, you can customize the appearance of SfSegmentedControl without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Gray White ....{% endhighlight %}
	SyncPrimaryForegroundColor	Text color of selected item in the SfSegmentedControl.
	SfSegmentFontIconFontColor	Color of font icon in the SfSegmentedControl.
	SfSegmentFontColor	Color of font in the SfSegmentedControl.
	SfSegmentedControlBorderColor	Color of segment's border in the SfSegmentedControl.

	SfSegmentColor	Color of segment in the SfSegmentedControl.
	SyncPrimaryColor	Background color of the selected item in the SfSegmentedControl.

### SfTabView

Theme Dictionary	Keys	Description
SfTabViewStyles	SfTabViewTheme	By merging this key in application resources, you can customize the appearance of SfTabView without merging common theme resource and control style resource dictionaries. {% highlight xaml %} CustomTheme Red ....{% endhighlight %}
	SfTabItemFontColor	Font color of title in SfTabView.
		Font color of font icon in SfTabView.
	SyncPrimaryColor	Background color of tab header in SfTabView.
	SyncPrimaryForegroundColor	Selection color of tab item in SfTabView.
		Color of selection indicator in SfTabView.

### Switching between light theme and dark theme

To switch to light theme from dark theme, use the following code snippet.

#### C#

```
void UpdateTheme(object sender, System.EventArgs e)
{
    ICollection<ResourceDictionary> mergedDictionaries =
    Application.Current.Resources.MergedDictionaries;
    var darkTheme = mergedDictionaries.OfType<DarkTheme>().FirstOrDefault();
    if (darkTheme != null)
    {
        mergedDictionaries.Remove(darkTheme);
    }
    mergedDictionaries.Add(new LightTheme());
}
```

Similarly, to switch to dark theme from light theme, remove the light theme resource, and add the dark theme resource dictionary.

## UI Kit

### Syncfusion XAML Pages for Xamarin.Forms

The [Syncfusion Essential UI Kit](#) is a collection of easy-to-use, extendable, and adaptable XAML pages that allows you to quickly create a Xamarin.Forms application by providing generic pre-defined screens. Now, the UI Kit has screens for the following categories:

- Forms
- Catalog
- Detail
- Chat
- Reviews and Ratings
- Contact Us
- About Us
- Navigation
- Error and Empty
- Transaction
- Bookmark
- History
- Social
- Profile

The screens are developed with MVVM pattern, which separates the UI and business logic results in a clean, professional, and scalable representation of user interface in your Xamarin.Forms applications. You can get the UI Kit application from [Google Play Store](#) and the complete source code is available in [GitHub repository](#)

### Supported Platforms

- Android 5.0 (or API level 21) and later versions.
- iOS 9.0 and later versions.
- UWP Build 17763 and later versions

---

**Note:** The required minimum version of Xamarin.Forms is 4.0.0.425677.

---

To get started with Syncfusion XAML Pages for Xamarin.Forms, refer to [Getting Started](#)

### Getting started

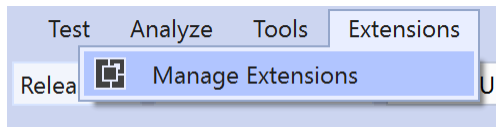
The UI Kit screens can be added in your application by the following two ways:

1. Using **Essential UI Kit for Xamarin.Forms** Visual Studio extension.
2. Copying the files from our open source [GitHub repository](#).

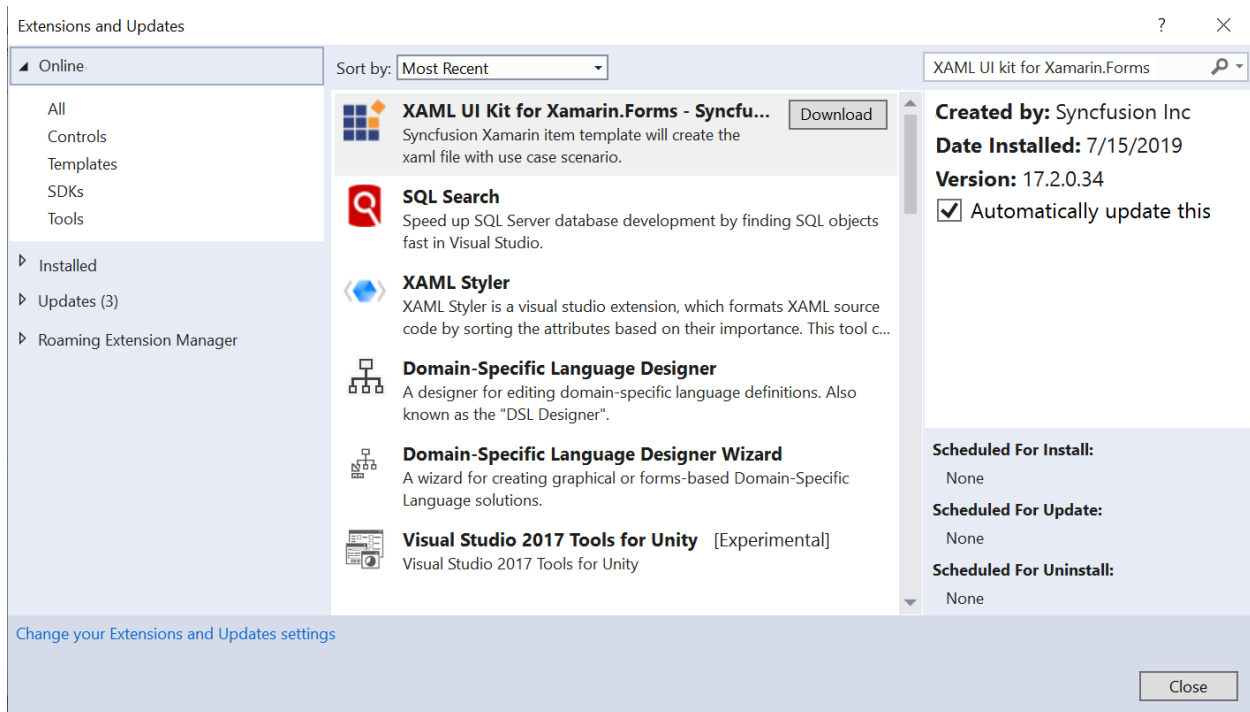
### Essential UI Kit for Xamarin.Forms Extension

This is the easiest way to add the pre-defined screens to your application. The following steps explain how to add screens to an application with our extension:

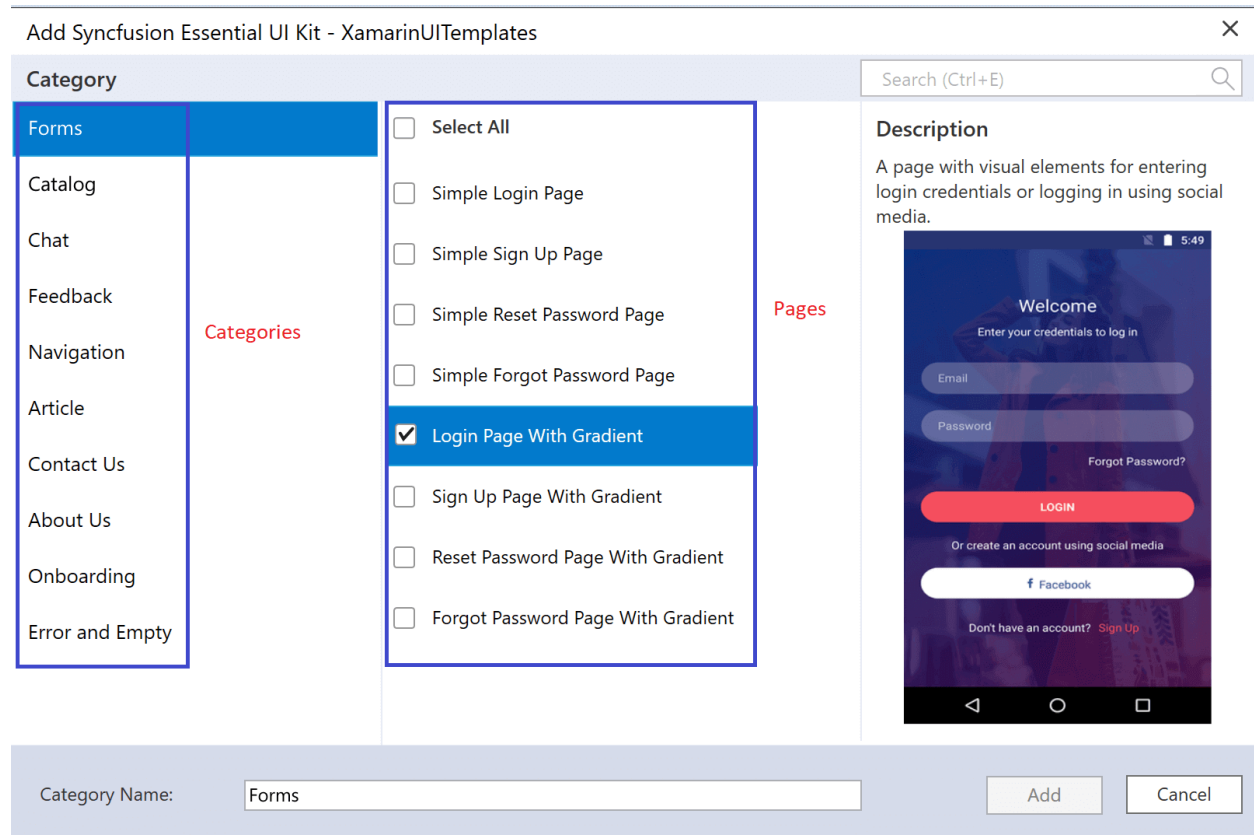
1. Open Visual Studio.
2. Go to Extension, and then click Manage Extensions as shown in the following screenshot.



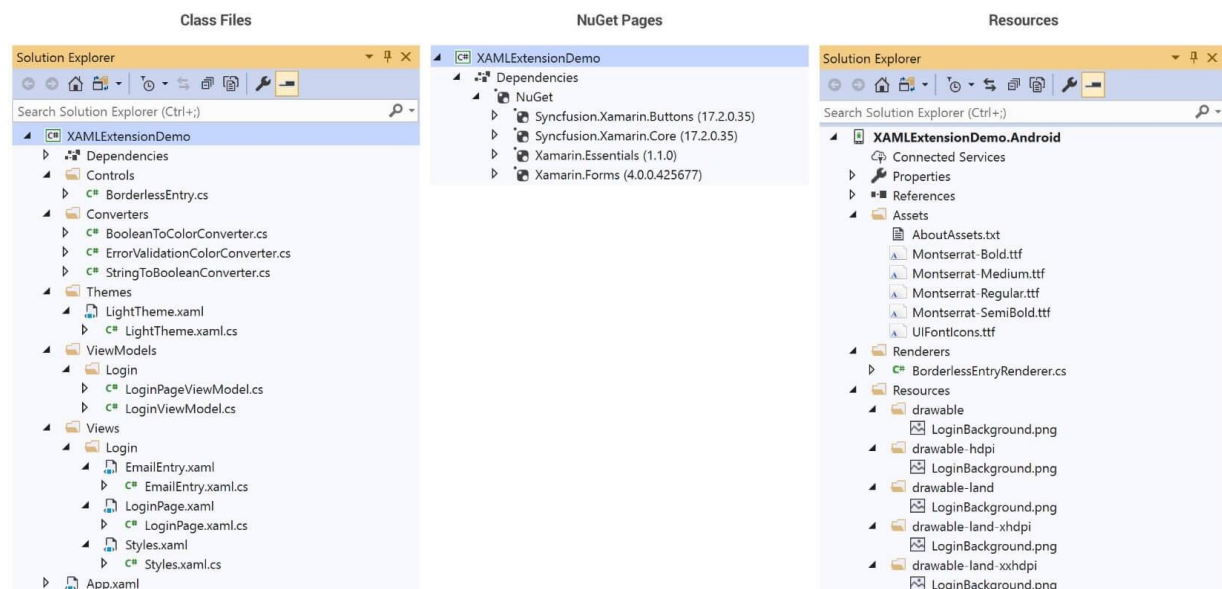
3. Search for **Essential UI Kit for Xamarin.Forms**, and then install it.



4. Restart the Visual Studio and allow it to complete the installation.
5. Now, open an existing Xamarin.Forms application or create a new application as per your requirements.
6. Right-click the Xamarin.Forms [NETStandard] project, and you can see the **Essential UI Kit for Xamarin.Forms** option.
7. Select the category and pages you need to add in your application. In the following screenshot, the **Login Page with Gradient** screen has been selected from the **Login** category.



- Clicking the 'Add' button will include the selected page to your project. The necessary class files, resources, and NuGet package references will automatically be added to your project as shown in the following screenshot.



### How to render the added page

In a Xamarin.Forms demo application, you must make the added page as the start-up page in the App.xaml.cs file.

Example: If you added the Login Page, then you must invoke the page as demonstrated in the following code.

#### C#

```
MainPage = new SampleFormsApplication.Views.Login.LoginPage();
```

In real-world applications, you may need to do the following to use these XAML pages:

1. Update the services for fetching the data from remote server or local database.
2. Wire up the navigation and update the business logics in view models.

### Requesting Screens and Reporting Bugs

If you would like to request a new screen or report a bug in existing screens, create a feature request or submit a bug through our [feedback portal](#)

---

**Note:** Now, **Essential UI Kit for Xamarin.Forms** Visual Studio extension is supported in the Windows operating system only.

---

### Release Notes

The UI Kit contains elegantly designed XAML pages for Xamarin.Forms apps. These pages are compatible with Android, iOS, and UWP platforms, and these pages use the MVVM design pattern to provide trouble-free integration.

To know more, refer to this [Read me](#) file.

#### UI Kit 1.0.0.0

The UI Kit version 1.0.0.0 has 45 screens for the following categories with RTL and themes support.

- Login
- ECommerce
- Chat
- Feedback
- Contact Us
- About Us
- Article
- Navigation
- Error and Empty

#### UI Kit 2.0.0.0

The UI Kit version 2.0.0.0 has 12 new screens for the following categories.

S.No	Categories	Pages
1	Forms	1. Add Contact Page 2. Tabbed Login Page 3. Add Card Page

2	Detail	1. Article Detail Page
3	Navigation	1. Bottom Navigation Page
4	Transaction	1. Payment Success Page 2. Payment Failure page
5	Bookmark	1. Wish List Page
6	History	1. Transaction History page 2. History - My Orders Page
7	Social	1. Social Profile with Connections Page 2. Social Profile with Card page

## SfAccordion

### Getting started

The Accordion control allows content to be organized in a vertically stacked list of items that is collapsible. Each item can also be expanded to reveal the content associated with that item. This section provides a quick overview for working with the **SfAccordion** for Xamarin.Forms.

### Assembly deployment

After installing Essential Studio for Xamarin, you can find all the required assemblies in the installation folders, {Syncfusion Essential Studio Installed location} \Essential Studio\16.x.x.x\Xamarin\lib

Eg: C:\Program Files (x86) \Syncfusion\Essential Studio\16.1.0.24\Xamarin\lib

---

**Note:** Assemblies can be found in unzipped package location in Mac.

---

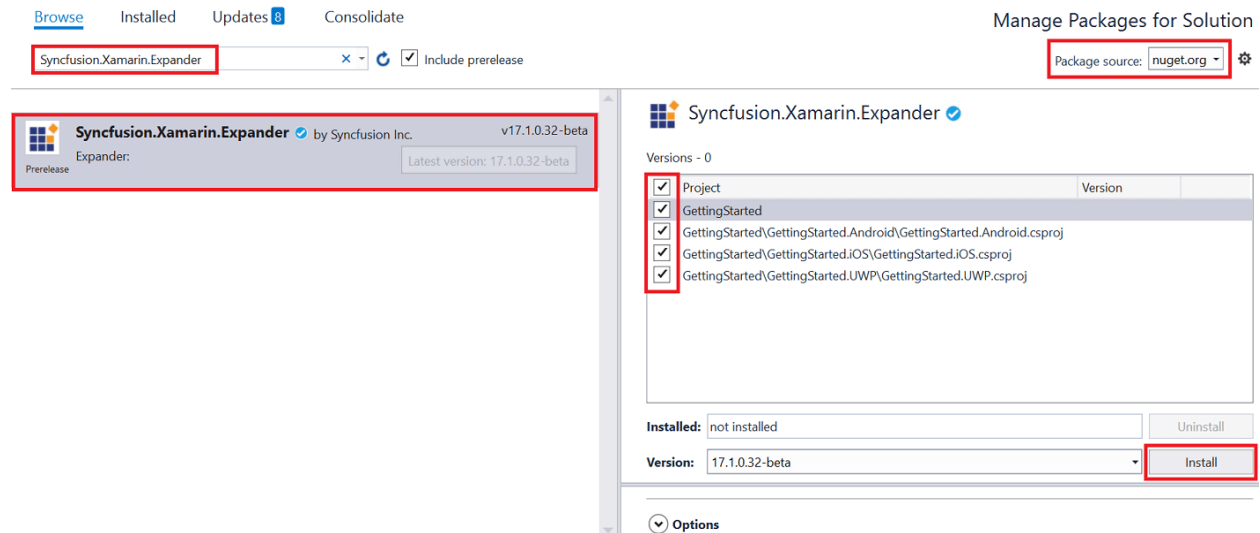
### Adding SfAccordion reference

You can add SfAccordion reference using one of the following methods:

#### Method 1: Adding SfAccordion reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Expander). To add SfAccordion to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Expander](https://www.nuget.org/packages/Syncfusion.Xamarin.Expander), and then install it.





**Note:** Install the same version of Expander NuGet in all the projects.

### Method 2: Adding SfAccordion reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfAccordion control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfAccordion assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Expander.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Expander.XForms.dll Syncfusion.Expander.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Expander.XForms.dll Syncfusion.Expander.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Expander.XForms.dll Syncfusion.Expander.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Launching the accordion on each platform

To use the accordion in an application, each platform application must initialize the accordion renderer. This initialization step varies from platform to platform and is discussed in the following sections:

#### Android

The Android launches the accordion without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

#### iOS

To launch the accordion in iOS, call the `SfAccordionRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework initialization and before the `LoadApplication` is called, as demonstrated in the following code example:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.XForms.iOS.Accordion.SfAccordionRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

#### Universal Windows Platform (UWP)

The UWP launches the accordion without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

#### ReleaseMode issue in UWP platform

The known Framework issue in UWP platform is that the custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the accordion assemblies in `App.xaml.cs` file in UWP project as in the following code snippet:

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
```

```
//Now, add all the assemblies your app uses
assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.Accordion.SfAccordionRenderer).GetTypeInfo().Assembly);
// replaces Xamarin.Forms.Forms.Init(e);
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
...
}
```

## Creating the Accordion

This section explains how to create a simple Xamarin.Forms application with [SfAccordion](#). The control should be configured entirely in C# code or by using XAML markup.

- Creating the project.
- Adding accordion in Xamarin.Forms.
- Defining accordion Items.

### Creating the project

Create a new blank (.Net Standard) application in Xamarin Studio or Visual Studio for Xamarin.Forms.

### Adding accordion in Xamarin.Forms:

To add the accordion to your application, follow the steps:

1. Add required assemblies as discussed in assembly deployment section.
2. Import the control namespace as `xmlns:accordion="clr-namespace:Syncfusion.XForms.Accordion;assembly=Syncfusion.Expander.XForms"` in XAML Page.
3. Create an instance of accordion control and add as content for content page.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
xmlns:syncfusion="clr-namespace:Syncfusion.XForms.Accordion;assembly=Syncfusion.Expander.XForms">
<ContentPage.Content>
<syncfusion:SfAccordion x:Name="accordion"/>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using Syncfusion.XForms.Accordion;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        SfAccordion accordion;
        public MainPage()
        {
```

```
InitializeComponent();
accordion = new SfAccordion();
}
}
}
```

### Defining accordion Items

**SfAccordion** is a layout control with vertically stacked list of accordion [Items](#) that comprise of [Header](#) and [Content](#). You can load any View in Header and Content. User can expand or collapse the Content view by tapping Header.

Here, Labels are loaded in Header and Content of accordion items.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.Accordion;assembly=Syncfusion.Expander.XForms">
<ContentPage.Content>
<syncfusion:SfAccordion Grid.Row="1">
<syncfusion:SfAccordion.Items>
<syncfusion:AccordionItem>
<syncfusion:AccordionItem.Header>
<Label TextColor="#495F6E" Text="Cheese burger" HeightRequest="50"
VerticalTextAlignment="Center"/>
</syncfusion:AccordionItem.Header>
<syncfusion:AccordionItem.Content>
<Grid Padding="10,10,10,10" BackgroundColor="#FFFFFF">
<Label TextColor="#303030" Text="Hamburger accompanied with melted cheese.
The term itself is a portmanteau of the words cheese and hamburger. The
cheese is usually sliced, then added a short time before the hamburger
finishes cooking to allow it to melt." HeightRequest="50"
VerticalTextAlignment="Center"/>
</Grid>
</syncfusion:AccordionItem.Content>
</syncfusion:AccordionItem>
<syncfusion:AccordionItem>
<syncfusion:AccordionItem.Header>
<Label TextColor="#495F6E" Text="Veggie burger" HeightRequest="50"
VerticalTextAlignment="Center"/>
</syncfusion:AccordionItem.Header>
<syncfusion:AccordionItem.Content>
<Grid Padding="10,10,10,10" BackgroundColor="#FFFFFF">
<Label TextColor="#303030" Text="Veggie burger, garden burger, or tofu
burger uses a meat analogue, a meat substitute such as tofu, textured
vegetable protein, seitan (wheat gluten), Quorn, beans, grains or an
assortment of vegetables, which are ground up and formed into patties."
HeightRequest="50" VerticalTextAlignment="Center"/>
</Grid>
</syncfusion:AccordionItem.Content>
</syncfusion:AccordionItem>
</syncfusion:AccordionItem>
```

```

<syncfusion:AccordionItem.Header>
<Label TextColor="#495F6E" Text="Barbecue burger" HeightRequest="50"
VerticalTextAlignment="Center"/>
</syncfusion:AccordionItem.Header>
<syncfusion:AccordionItem.Content>
<Grid Padding="10,10,10,10" BackgroundColor="#FFFFFF">
<Label TextColor="#303030" Text="Prepared with ground beef, mixed with
onions and barbecue sauce, and then grilled. Once the meat has been turned
once, barbecue sauce is spread on top and grilled until the sauce
caramelizes." HeightRequest="50" VerticalTextAlignment="Center"/>
</Grid>
</syncfusion:AccordionItem.Content>
</syncfusion:AccordionItem>
<syncfusion:AccordionItem>
<syncfusion:AccordionItem.Header>
<Label TextColor="#495F6E" Text="Chili burger" HeightRequest="50"
VerticalTextAlignment="Center"/>
</syncfusion:AccordionItem.Header>
<syncfusion:AccordionItem.Content>
<Grid Padding="10,10,10,10" BackgroundColor="#FFFFFF">
<Label TextColor="#303030" Text="Consists of a hamburger, with the patty
topped with chili con carne." HeightRequest="50"
VerticalTextAlignment="Center"/>
</Grid>
</syncfusion:AccordionItem.Content>
</syncfusion:AccordionItem>
</syncfusion:SfAccordion.Items>
</syncfusion:SfAccordion>
</ContentPage.Content>
</ContentPage>

```

## C#

```

public MainPageCS()
{
InitializeAccordionItems();
this.Content = Accordion;
}
private void InitializeAccordionItems()
{
Accordion = new SfAccordion();
Accordion.Items.Add(GenerateInvoiceHeaderAccordion());
Accordion.Items.Add(GenerateInvoiceItemsAccordion());
Accordion.Items.Add(GeneratePaymentDetailsAccordion());
}
private AccordionItem GenerateInvoiceHeaderAccordion()
{
var item = new AccordionItem();
var headerGrid = new Grid() { Padding = new Thickness(10, 0, 0, 10) };
var headerLabel = new Label() { TextColor = Color.FromHex("#495F6E"), Text =
"Invoice Date", HeightRequest = 50, VerticalTextAlignment =
TextAlignment.Center };
headerGrid.Children.Add(headerLabel);
var contentGrid = new Grid() { Padding = new Thickness(10, 0, 0, 10),
BackgroundColor = Color.FromHex("#FFFFFF") };

```

```

var contentLabel = new Label() { TextColor = Color.FromHex("#303030"), Text
= "11.03 AM, 15 January 2019", HeightRequest = 50, VerticalTextAlignment =
TextAlignment.Center };
contentGrid.Children.Add(contentLabel);
item.Header = headerGrid;
item.Content = contentGrid;
return item;
}
private AccordionItem GenerateInvoiceItemsAccordion()
{
var item = new AccordionItem();
var headerGrid = new Grid() { Padding = new Thickness(10, 0, 10, 10) };
var headerLabel = new Label() { TextColor = Color.FromHex("#495F6E"), Text =
"Item (s)", HeightRequest = 50, VerticalTextAlignment = TextAlignment.Center
};
headerGrid.Children.Add(headerLabel);
var contentGrid = new Grid() { Padding = new Thickness(10, 0, 10, 10),
BackgroundColor = Color.FromHex("#FFFFFF") };
contentGrid.RowDefinitions.Add(new RowDefinition() { Height = new
GridLength(1, GridUnitType.Auto) });
contentGrid.RowDefinitions.Add(new RowDefinition() { Height = new
GridLength(1, GridUnitType.Auto) });
contentGrid.RowDefinitions.Add(new RowDefinition() { Height = new
GridLength(1, GridUnitType.Auto) });
contentGrid.RowDefinitions.Add(new RowDefinition() { Height = new
GridLength(1, GridUnitType.Auto) });
contentGrid.RowDefinitions.Add(new RowDefinition() { Height = new
GridLength(1, GridUnitType.Auto) });
contentGrid.ColumnDefinitions.Add(new ColumnDefinition() { Width = new
GridLength(1, GridUnitType.Star) });
contentGrid.ColumnDefinitions.Add(new ColumnDefinition() { Width = new
GridLength(1, GridUnitType.Star) });
var contentLabel0 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "2018 Subaru Outback" };
var contentLabel2 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "Door Edge Guard Kit" };
var contentLabel11 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "All-Weather Mats" };
var contentLabel3 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "Rear Bumper Cover" };
var contentLabel4 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "Total Amount Paid", FontAttributes = FontAttributes.Bold };
var contentLabel5 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "$35,705.00", HorizontalTextAlignment = TextAlignment.End };
var contentLabel6 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "$105.00", HorizontalTextAlignment = TextAlignment.End };
var contentLabel7 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "$100.00", HorizontalTextAlignment = TextAlignment.End };
var contentLabel8 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "$95.00", HorizontalTextAlignment = TextAlignment.End };
var contentLabel9 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "$36,000.00", HorizontalTextAlignment = TextAlignment.End, FontAttributes
= FontAttributes.Bold };
contentGrid.Children.Add(contentLabel0, 0, 0);
contentGrid.Children.Add(contentLabel11, 0, 1);
contentGrid.Children.Add(contentLabel2, 0, 2);
contentGrid.Children.Add(contentLabel3, 0, 3);

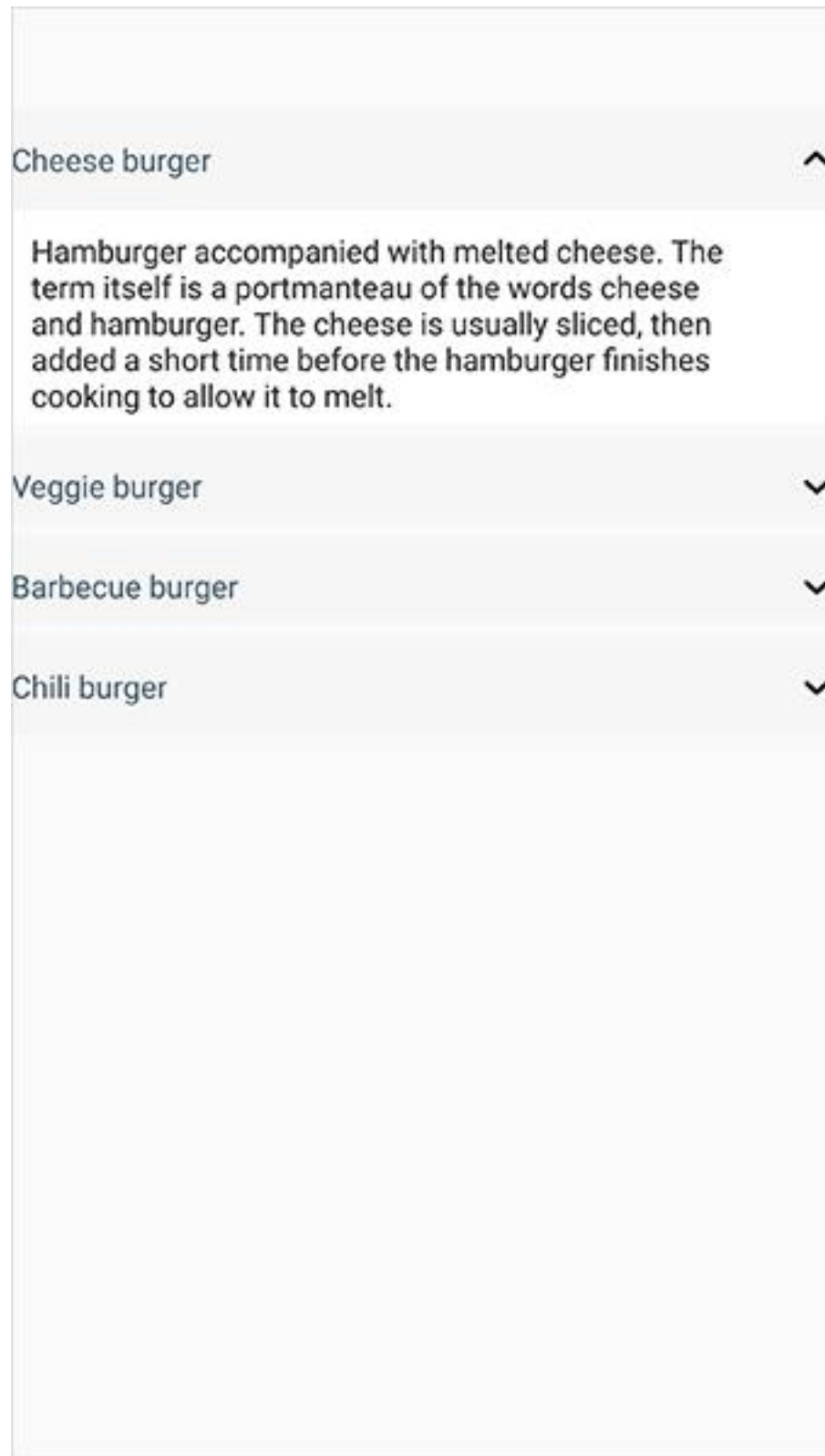
```

```

contentGrid.Children.Add(contentLabel4, 0, 4);
contentGrid.Children.Add(contentLabel5, 1, 0);
contentGrid.Children.Add(contentLabel6, 1, 1);
contentGrid.Children.Add(contentLabel7, 2, 1);
contentGrid.Children.Add(contentLabel8, 3, 1);
contentGrid.Children.Add(contentLabel9, 4, 1);
item.Header = headerGrid;
item.Content = contentGrid;
return item;
}
private AccordionItem GeneratePaymentDetailsAccordion()
{
var item = new AccordionItem();
var headerGrid = new Grid() { Padding = new Thickness(10, 0, 10, 10) };
var headerLabel = new Label() { TextColor = Color.FromHex("#495F6E"), Text =
"Payment Details", HeightRequest = 50, VerticalTextAlignment =
TextAlignment.Center };
headerGrid.Children.Add(headerLabel);
var contentGrid = new Grid() { Padding = new Thickness(10, 0, 10, 10),
BackgroundColor = Color.FromHex("#FFFFFF") };
contentGrid.RowDefinitions.Add(new RowDefinition() { Height = new
GridLength(1, GridUnitType.Auto) });
contentGrid.RowDefinitions.Add(new RowDefinition() { Height = new
GridLength(1, GridUnitType.Auto) });
contentGrid.RowDefinitions.Add(new RowDefinition() { Height = new
GridLength(1, GridUnitType.Auto) });
contentGrid.ColumnDefinitions.Add(new ColumnDefinition() { Width = new
GridLength(1, GridUnitType.Star) });
contentGrid.ColumnDefinitions.Add(new ColumnDefinition() { Width = new
GridLength(1, GridUnitType.Star) });
var contentLabel0 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "Card Payment" };
var contentLabel1 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "Third-Party coupons" };
var contentLabel2 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "Total Amount Paid", FontAttributes = FontAttributes.Bold };
var contentLabel3 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "$31,000.00", HorizontalTextAlignment = TextAlignment.End };
var contentLabel4 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "$5,000.00", HorizontalTextAlignment = TextAlignment.End };
var contentLabel5 = new Label() { TextColor = Color.FromHex("#303030"), Text
= "$36,000.00", HorizontalTextAlignment = TextAlignment.End, FontAttributes
= FontAttributes.Bold };
contentGrid.Children.Add(contentLabel0, 0, 0);
contentGrid.Children.Add(contentLabel1, 0, 1);
contentGrid.Children.Add(contentLabel2, 0, 2);
contentGrid.Children.Add(contentLabel3, 1, 0);
contentGrid.Children.Add(contentLabel4, 1, 1);
contentGrid.Children.Add(contentLabel5, 1, 2);
item.Header = headerGrid;
item.Content = contentGrid;
return item;
}

```

Now, run the application to render the following output.



You can download accordion sample for Xamarin.Forms from here [AccordionGettingStarted](#).



### Animation duration

**SfAccordion** allows to customize the expanding and collapsing of accordion item by using [AnimationDuration](#) property. By default, the animation duration is 250 milliseconds.

#### XML

```
<syncfusion:SfAccordion x:Name="accordion" AnimationDuration="150" />
```

#### C#

```
accordion.AnimationDuration = 150;
```

### Animation easing

**SfAccordion** allows to customize the rate of change of parameter over time or animation style of accordion item by using [AnimationEasing](#) property. By default, the animation easing is **Linear**.

#### XML

```
<syncfusion:SfAccordion x:Name="accordion" AnimationEasing="SinOut" />
```

#### C#

```
accordion.AnimationEasing =  
Syncfusion.XForms.Expander.AnimationEasing.SinOut;
```

### Auto scroll position

**SfAccordion** allows to customize the scroll position of the expanded accordion item by using [AutoScrollPosition](#) property. By default, the auto scroll position is **MakeVisible**.

#### XML

```
<syncfusion:SfAccordion x:Name="accordion" AutoScrollPosition="Top"/>
```

#### C#

```
accordion.AutoScrollPosition =  
Syncfusion.XForms.Accordion.AutoScrollPosition.Top;
```

### Expand mode

**SfAccordion** allows to expand single or multiple items by using the [ExpandMode](#) property. By default, the expand mode is **Single**.

#### XML

```
<syncfusion:SfAccordion x:Name="accordion" ExpandMode="Multiple" />
```

#### C#

```
accordion.ExpandMode = Syncfusion.XForms.Accordion.ExpandMode.Multiple;
```

### Item spacing

**SfAccordion** allows to customize the vertical spacing between the accordion items by using [ItemSpacing](#) property. The default value is **6.0d**.

#### XML

```
<syncfusion:SfAccordion x:Name="accordion" ItemSpacing="8.0d" />
```

#### C#

```
accordion.ItemSpacing = 8.0d;
```

### Appearance

The Accordion allows customizing appearance of the Icon, and provides different functionalities to the end-user.

#### Header icon position

**SfAccordion** allows to customize the position of the header icon in accordion item by using [HeaderIconPosition](#) property. By default, the header Icon position is **End**.

#### XML

```
<syncfusion:SfAccordion x:Name="accordion" HeaderIconPosition="Start" />
```

#### C#

```
accordion.HeaderIconPosition =  
Syncfusion.XForms.Expander.IconPosition.Start;
```

#### Header background color customization

**SfAccordion** allows to customize the background color of the expander header by using [HeaderBackgroundColor](#) property.

#### XML

```
<syncfusion:SfAccordion x:Name="accordion">  
  <syncfusion:SfAccordion.Items>  
    <syncfusion:AccordionItem HeaderBackgroundColor="Pink"/>  
  </syncfusion:SfAccordion.Items>  
</syncfusion:SfAccordion>
```

#### C#

```
SfAccordion accordion;  
public MainPage()  
{  
  InitializeComponent();  
  InitializeAccordionItems();  
  this.Content = accordion;  
}  
private void InitializeAccordionItems()  
{  
  accordion = new SfAccordion();
```

```
accordion.Items.Add(GenerateAccordionItem());
}
public AccordionItem GenerateAccordionItem()
{
    var item = new AccordionItem();
    item.HeaderBackgroundColor = Color.Pink;
    return item;
}
```

### Icon color customization

[SfAccordion](#) allows to customize the color of the expander icon by using [IconColor](#) property. By default, [IconColor](#) is black.

#### XML

```
<syncfusion:SfAccordion x:Name="accordion">
<syncfusion:SfAccordion.Items>
<syncfusion:AccordionItem IconColor="Accent"/>
</syncfusion:SfAccordion.Items>
</syncfusion:SfAccordion>
```

#### C#

```
SfAccordion accordion;
public MainPage()
{
    InitializeComponent();
    InitializeAccordionItems();
    this.Content = accordion;
}
private void InitializeAccordionItems()
{
    accordion = new SfAccordion();
    accordion.Items.Add(GenerateAccordionItem());
}
public AccordionItem GenerateAccordionItem()
{
    var item = new AccordionItem();
    item.IconColor = Color.Accent;
    return item;
}
```

### Visual State Manager

The appearance of the [SfAccordion](#) can be customized using the following two [VisualStates](#):

- Expanded
- Collapsed

#### XML

```
<syncfusion:SfAccordion x:Name="accordion">
<syncfusion:SfAccordion.Items>
<syncfusion:AccordionItem IconColor="Accent">
```

```

<syncfusion:AccordionItem.Header>
<Label TextColor="#495F6E" Text="Cheese burger" HeightRequest="50"
VerticalTextAlignment="Center"/>
</syncfusion:AccordionItem.Header>
<syncfusion:AccordionItem.Content>
<Grid Padding="10,10,10,10" BackgroundColor="#FFFFFF">
<Label TextColor="#303030" Text="Hamburger accompanied with melted cheese.
The term itself is a portmanteau of the words cheese and hamburger. The
cheese is usually sliced, then added a short time before the hamburger
finishes cooking to allow it to melt." HeightRequest="50"
VerticalTextAlignment="Center"/>
</Grid>
</syncfusion:AccordionItem.Content>
<VisualStateManager.VisualStateGroups>
<VisualStateGroupList>
<VisualStateGroup>
<VisualState Name="Expanded">
<VisualState.Setters>
<Setter Property="HeaderBackgroundColor" Value="Red"/>
</VisualState.Setters>
</VisualState>
<VisualState Name="Collapsed">
<VisualState.Setters>
<Setter Property="HeaderBackgroundColor" Value="Green"/>
</VisualState.Setters>
</VisualState>
</VisualStateGroup>
</VisualStateGroupList>
</VisualStateManager.VisualStateGroups>
</syncfusion:AccordionItem>
</syncfusion:SfAccordion>

```

**C#**

```

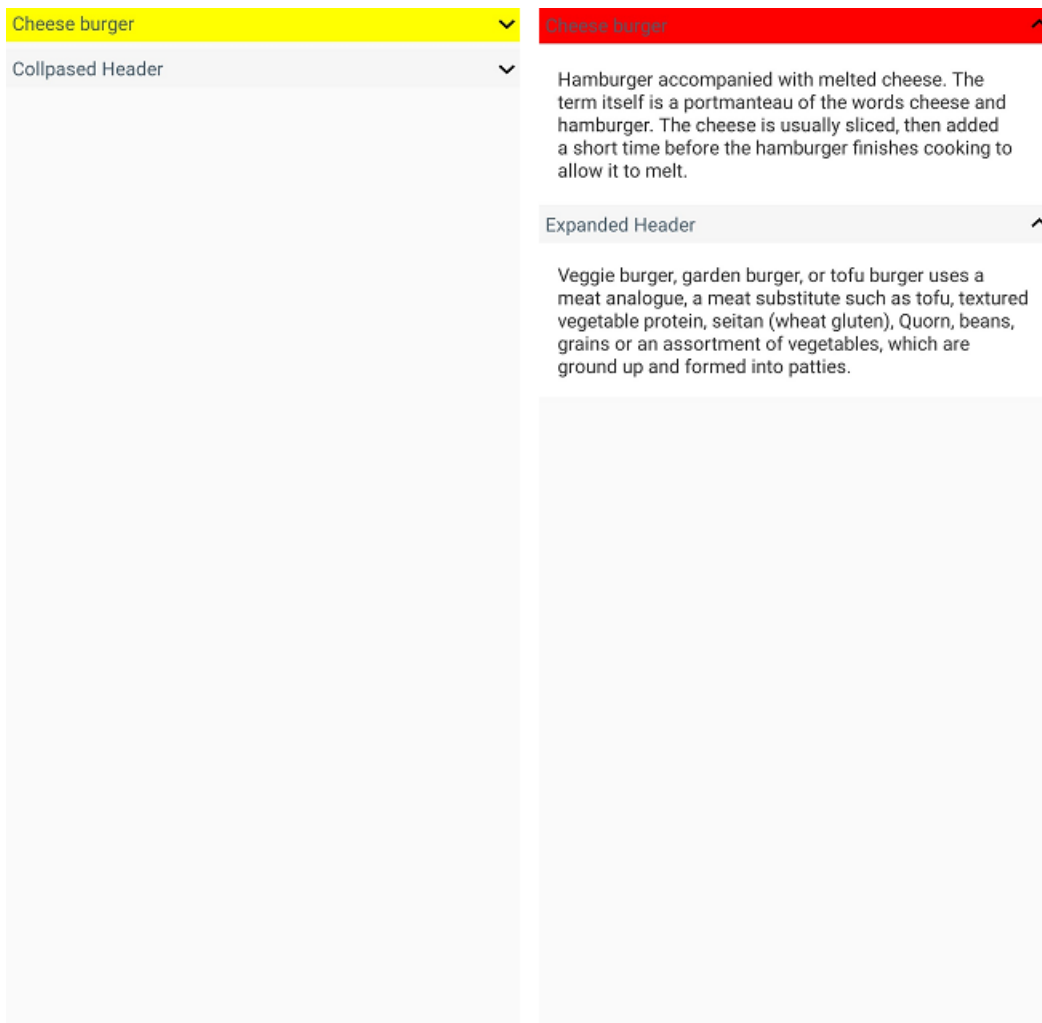
SfAccordion accordion;
public MainPage()
{
InitializeComponent();
InitializeAccordionItems();
this.Content = accordion;
}
private void InitializeAccordionItems()
{
accordion = new SfAccordion();
accordion.Items.Add(GenerateAccordionItem());
}
public AccordionItem GenerateAccordionItem()
{
var item = new AccordionItem();
item.IconColor = Color.Accent;
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState expanded = new VisualState
{
Name = "Expanded"

```

```

};
expanded.Setters.Add(new Setter { Property =
AccordionItem.HeaderBackgroundColorProperty, Value = Color.Red });
expanded.Setters.Add(new Setter { Property =
AccordionItem.HeaderBackgroundColorProperty, Value = Color.Red });
VisualState collapsed = new VisualState
{
    Name = "Collapsed"
};
collapsed.Setters.Add(new Setter { Property =
AccordionItem.HeaderBackgroundColorProperty, Value = Color.Green });
collapsed.Setters.Add(new Setter { Property =
AccordionItem.HeaderBackgroundColorProperty, Value = Color.Green });
commonStateGroup.States.Add(expanded);
commonStateGroup.States.Add(collapsed);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(item, visualStateGroupList);
return item;
}

```



You can download the entire source of this demo from [here](#).

## Bindable Layout

The [SfAccordion](#) allows to set a collection of items by setting `BindableLayout.ItemsSource` and `BindableLayout.ItemTemplate` properties. The Accordion supports Bindable Layout in Xamarin.Forms version 3.5 and above.

### Creating Data Model

Create a simple data model to bind the data for `SfAccordion` as shown in the following code example in a new class file, and save it as `ItemInfo.cs` file:

#### C#

```
public class ItemInfo
{
    public string Name { get; set; }
    public string Description { get; set; }
}
```

Create a model repository class with `ItemInfo` 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 `ItemInfoRepository.cs` file:

#### C#

```
public class ItemInfoRepository
{
    public ObservableCollection<ItemInfo> Info { get; set; }
    public ItemInfoRepository()
    {
        Info = new ObservableCollection<ItemInfo>();
        Info.Add(new ItemInfo() { Name = "Cheese burger", Description = "Hamburger accompanied with melted cheese. The term itself is a portmanteau of the words cheese and hamburger. The cheese is usually sliced, then added a short time before the hamburger finishes cooking to allow it to melt." });
        Info.Add(new ItemInfo() { Name = "Veggie burger", Description = "Veggie burger, garden burger, or tofu burger uses a meat analogue, a meat substitute such as tofu, textured vegetable protein, seitan (wheat gluten), Quorn, beans, grains or an assortment of vegetables, which are ground up and formed into patties." });
        Info.Add(new ItemInfo() { Name = "Barbecue burger", Description = "Prepared with ground beef, mixed with onions and barbecue sauce, and then grilled. Once the meat has been turned once, barbecue sauce is spread on top and grilled until the sauce caramelizes." });
        Info.Add(new ItemInfo() { Name = "Chili burger", Description = "Consists of a hamburger, with the patty topped with chili con carne." });
    }
}
```

Set the ViewModel instance as `BindingContext` of your page to bind properties of ViewModel to `SfAccordion`.

#### XML

```
<ContentPage.BindingContext>
<local:ItemInfoRepository/>
</ContentPage.BindingContext>
```

**C#**

```
this.BindingContext = new ItemInfoRepository();
```

## Binding data to SfAccordion

SfAccordion can be bounded with data by setting the ItemsSource property of BindableLayout.

The following code example binds the collection created in previous step to the Bindable.ItemsSource property:

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-namespace:Syncfusion.XForms.Accordion;assembly=Syncfusion.Expander.XForms"
xmlns:local="clr-namespace:AccordionBindableLayout"
x:Class="AccordionBindableLayout.MainPage">
<syncfusion:SfAccordion x:Name="Accordion"
BindableLayout.ItemsSource="{Binding Info}"/>
</ContentPage>
```

**C#**

```
SfAccordion Accordion = new SfAccordion();
BindableLayout.SetItemsSource(Accordion, viewModel.Info);
```

## Defining the AccordionItem

The SfAccordion accepts AccordionItem as its child element. The appearance of each AccordionItem can be defined by setting the BindableLayout.ItemTemplate property.

**XML**

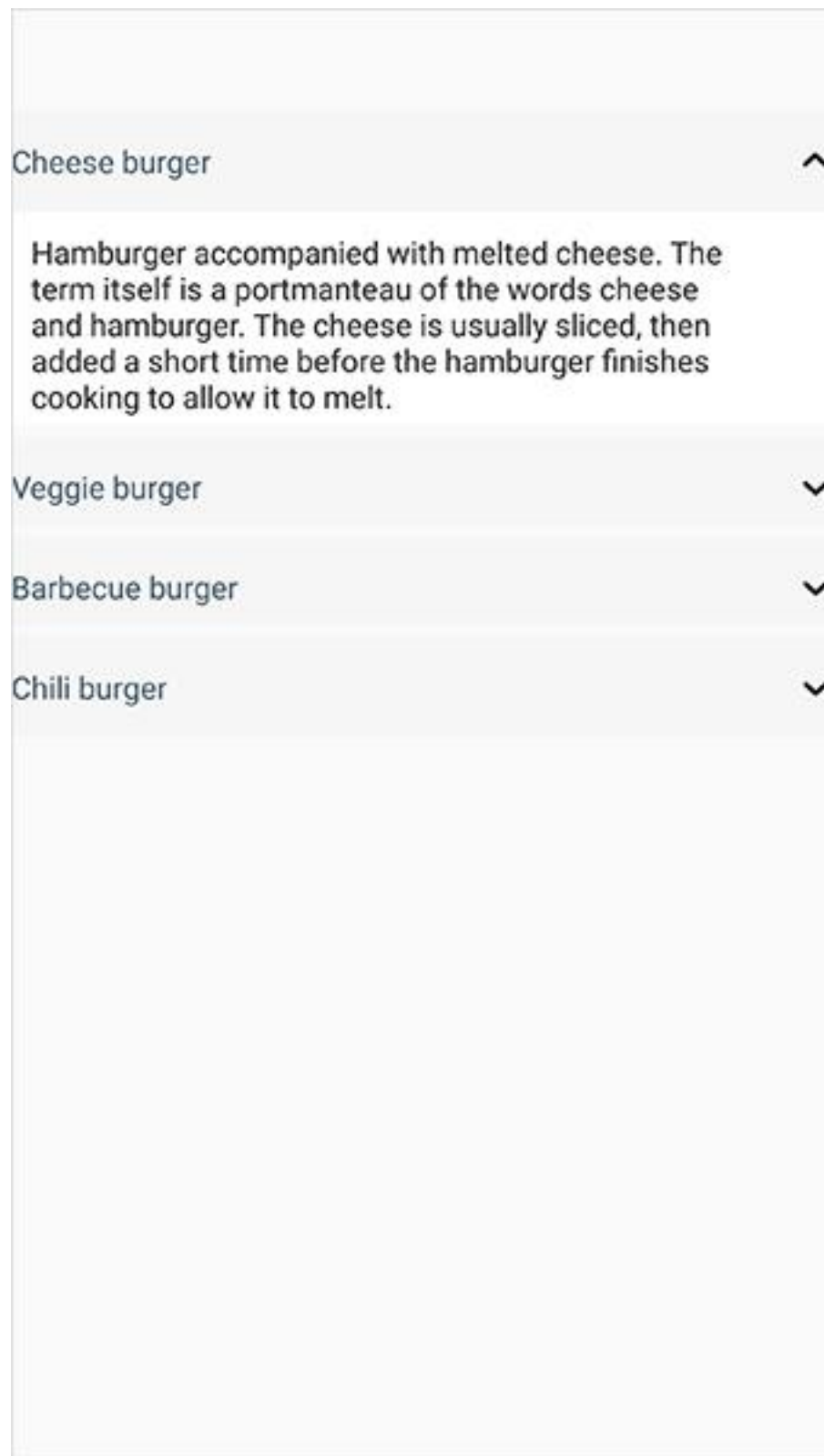
```
<syncfusion:SfAccordion x:Name="Accordion"
BindableLayout.ItemsSource="{Binding Info}">
<BindableLayout.ItemTemplate>
<DataTemplate>
<syncfusion:AccordionItem>
<syncfusion:AccordionItem.Header>
<Label Text="{Binding Name}"/>
</syncfusion:AccordionItem.Header>
<syncfusion:AccordionItem.Content>
<Label Text="{Binding Description}"/>
</syncfusion:AccordionItem.Content>
</syncfusion:AccordionItem>
</DataTemplate>
</BindableLayout.ItemTemplate>
</syncfusion:SfAccordion>
```

**C#**

```
SfAccordion Accordion = new SfAccordion();
DataTemplate ItemTemplate = new DataTemplate(() =>
```

```
{
    AccordionItem accordionItem = new AccordionItem();
    var header = new Label();
    header.SetBinding(Label.TextProperty, new Binding("Name"));
    accordionItem.Header = header;
    var content = new Label();
    content.SetBinding(Label.TextProperty, new Binding("Description"));
    accordionItem.Content = content;
    return accordionItem;
});
BindableLayout.SetItemTemplate(Accordion, ItemTemplate);
BindableLayout.SetItemsSource(Accordion, viewModel.Info);
```



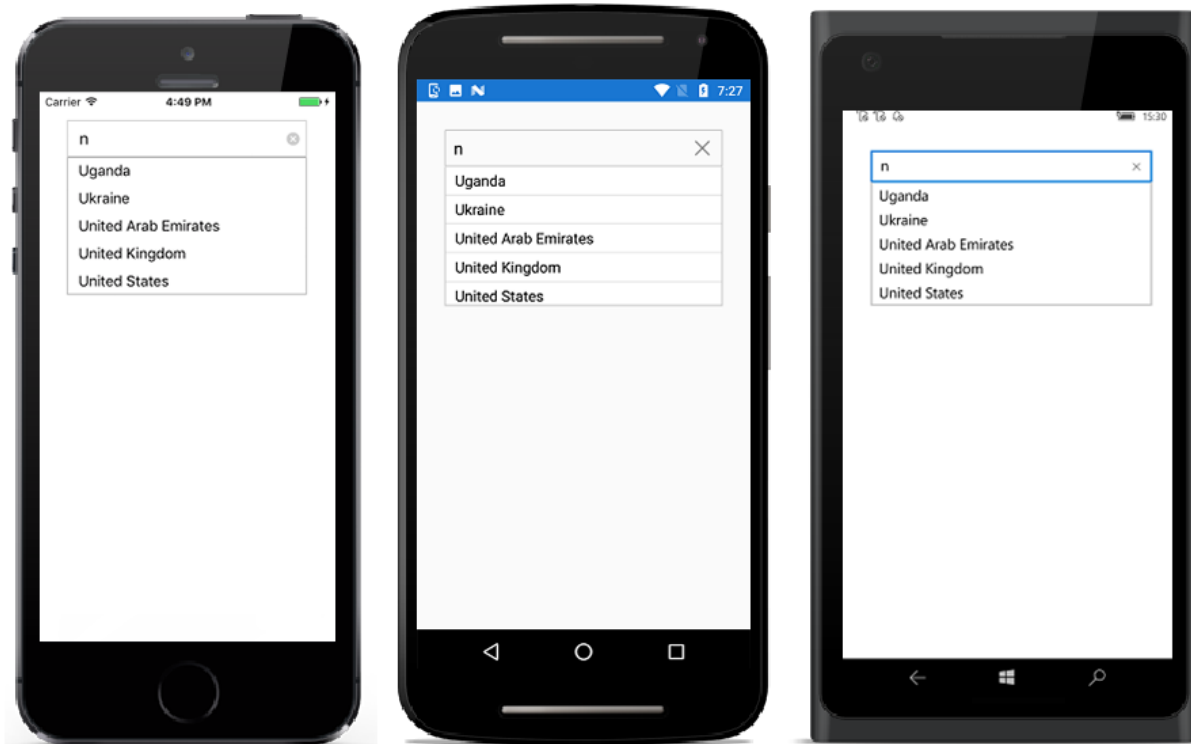


You can download the entire source of this demo from [here](#).

## SfAutoComplete

### Overview

Essential AutoComplete for Xamarin.Forms provides a simpler way to complete the text based on the characters that the user has entered before. It also provides option to choose a suggestion from drop down or append a suggestion to the text directly.



### Key Features

- AutoComplete modes - AutoComplete provides 3 different ways to display the filtered suggestions. They are displaying suggestion through drop down list, appending the first suggestion to text and both of these.
- Suggestion modes - Suggestions can be filtered in 9 different modes, like StartsWith, EndWith, Contains, Equals, Custom etc. AutoComplete provides both case sensitive and insensitive modes.
- Watermark - Supports explanatory text inside AutoComplete control until the user inputs text. Watermark is restored again if user clears the text in control.
- Popup delay & Minimum prefix characters - Filtering process can be delayed by providing the minimum character count, after which AutoComplete starts giving suggestions. Displaying the filtered suggestions through drop down list can also be delayed for some time duration.
- Customization support - AutoComplete provide options to customize both the Entry and Suggestion drop down.
- Multiple Selection - AutoComplete provides 2 different ways to select multiple items from the suggestion list. They are using Token representation and Delimiter. In Token mode, the text can

be wrapped in two ways. They are Wrap and None. In Wrap mode text will be wrap to next line. When using None, the text will be wrapped horizontally.

- Load More - Restrict the number of suggestions displayed and have the remaining items loaded by selecting LoadMore.
- Header and Footer - Header and Footer content can be given in the top and bottom of the Suggestion list in SfAutoComplete.
- Diacritic Sense - AutoComplete provides populating the items from a language with letters containing diacritics, and search for them with English characters from an en-US keyboard.
- Highlighting Text - AutoComplete provides highlighting the matching text in the Suggestion list based on the input given in it.
- Custom filter - AutoComplete provides the user to filter the item in the Suggestion list based on their filtering condition.

## Getting Started

This section explains the steps to create AutoComplete, populate it with data and filter the suggestions. This section covers only the minimal features that are needed to get started with the AutoComplete.

To get start quickly with Xamarin.Forms SfAutoComplete, you can check on this video:

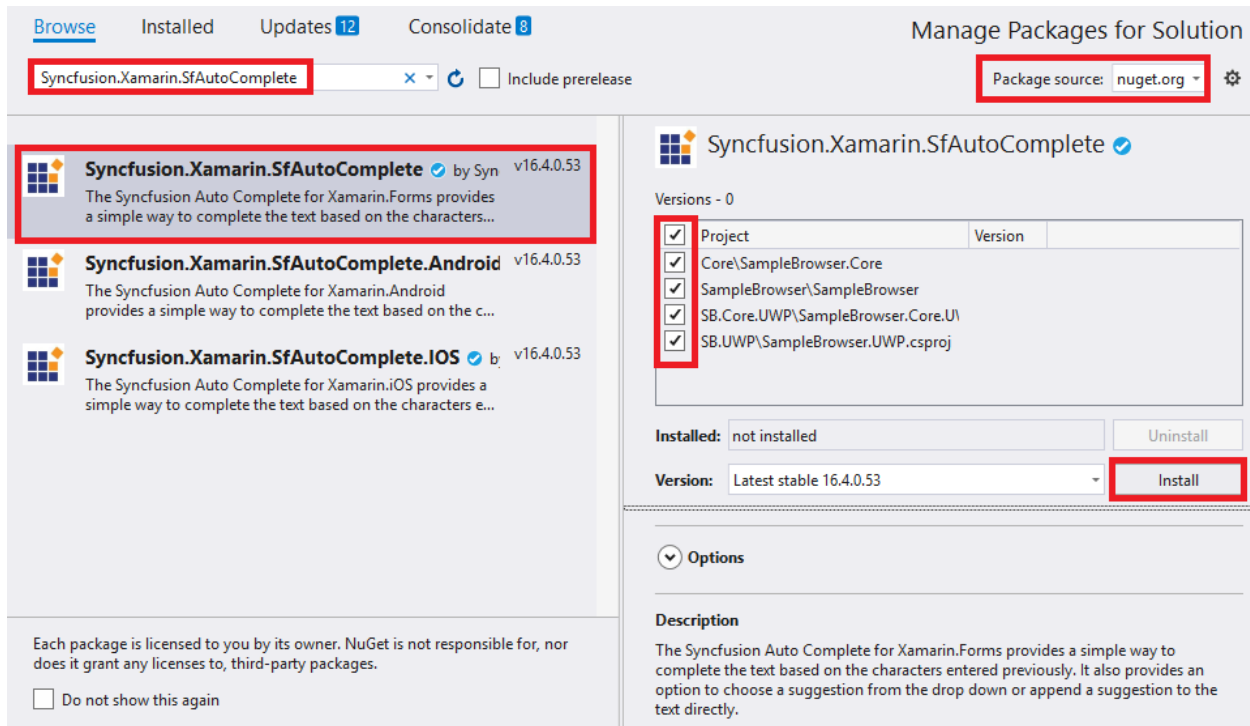
```
<style>#XamarinAutoCompleteVideoTutorial{width : 90% !important; height: 400px !important }</style>
<iframe id='XamarinAutoCompleteVideoTutorial' src='https://www.youtube.com/embed/XT5-CKZCiH8'></iframe>
```

## Adding SfAutoComplete reference

You can add SfAutoComplete reference using one of the following methods:

### Method 1: Adding SfAutoComplete reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfAutoComplete to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfAutoComplete](https://www.syncfusion.com/Products/xamarin/sfautocomplete.aspx), and then install it.



**Note:** Install the same version of SfAutoComplete NuGet in all the projects.

### Method 2: Adding SfAutoComplete reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfAutoComplete control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfAutoComplete assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfAutoComplete.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfAutoComplete.Android.dll Syncfusion.SfAutoComplete.XForms.Android.dll Syncfusion.SfAutoComplete.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfAutoComplete.iOS.dll Syncfusion.SfAutoComplete.XForms.iOS.dll Syncfusion.SfAutoComplete.XForms.dll Syncfusion.Core.XForms.dll

	Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfInput.UWP.dll Syncfusion.SfShared.UWP.dll Syncfusion.SfAutoComplete.XForms.UWP.dll Syncfusion.SfAutoComplete.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

#### *Additional Step for iOS*

**Note:** If you are adding the references from toolbox, this step is not needed.

Create an instance of **SfAutoCompleteRenderer** in FinishedLaunching overridden method of an AppDelegate class in iOS project as shown below:

#### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    new Syncfusion.SfAutoComplete.XForms.iOS.SfAutoCompleteRenderer();
    global::Xamarin.Forms.Forms.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### *Additional Step for UWP*

This step is required only if the application is deployed in Release mode with .NET native tool chain enabled and it is for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the SfAutoComplete assembly at OnLaunched overridden method of the App class in UWP project is the suggested workaround. And the code example is shown below:

#### **C#**

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    #if DEBUG
    if (System.Diagnostics.Debugger.IsAttached)
    {
        this.DebugSettings.EnableFrameRateCounter = true;
    }
    #endif
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame == null)
```

```

{
    rootFrame = new Frame();
    rootFrame.NavigationFailed += OnNavigationFailed;
    List<System.Reflection.Assembly> assembliesToInclude = new
    List<System.Reflection.Assembly>();
    // Add all the renderer assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.SfAutoComplete.XForms.UWP.SfAutoCo
    mpleteRenderer).GetTypeInfo().Assembly);
    // Replace the Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
    {
        //TODO: Load state from previously suspended application
    }
    // Place the frame in the current Window
    Window.Current.Content = rootFrame;
}
if (rootFrame.Content == null)
{
    // When the navigation stack isn't restored navigate to the first page,
    // configuring the new page by passing required information as a navigation
    // parameter
    rootFrame.Navigate(typeof(MainPage), e.Arguments);
}
// Ensure the current window is active
Window.Current.Activate();
}

```

### Initializing AutoComplete

Import the SfAutoComplete namespace in respective Page as shown below:

#### XML

```

xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"

```

#### C#

```

using Syncfusion.SfAutoComplete.XForms;

```

Then initialize an empty autocomplete as shown below,

#### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout

```

```

VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"/>
</StackLayout>
</ContentPage>

```

**C#**

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}

```

## Populating AutoComplete with Data

Now, let us create a simple list of country names and set it as the **AutoCompleteSource** of AutoComplete.

**XML**

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout
VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">

```

```
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout()
{
VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
SfAutoComplete autoComplete = new SfAutoComplete()
{
HeightRequest = 40,
AutoCompleteSource = new List<string>()
{
"India",
"Uganda",
"Ukraine",
"Canada",
"United Arab Emirates",
"France",
"United Kingdom",
"China",
"United States",
"Japan",
"Angola"
}
}
```



```
};  
stackLayout.Children.Add(autoComplete);  
this.Content = stackLayout;  
}  
}  
}
```

Refer [this](#) link to learn more about the options available in SfAutoComplete to populate data.

### Configuring filter options

By default, items are filtered in “StartsWith” case insensitive mode and the suggestions are displayed in a drop down popup. Autocomplete can now filter suggestions.

Here in this example, let us configure it to “Contains” case sensitive filter mode. This can be achieved by setting `SuggestionMode` property.

### XML

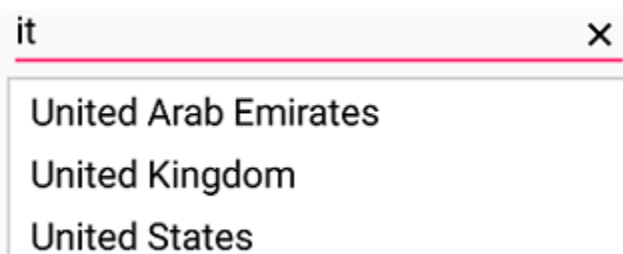
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:autocomplete="clr-  
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple  
te.XForms"  
xmlns:ListCollection="clr-  
namespace:System.Collections.Generic;assembly=netstandard"  
xmlns:local="clr-namespace:AutocompleteSample"  
x:Class="AutocompleteSample.MainPage">  
  <StackLayout  
    VerticalOptions="Start"  
    HorizontalOptions="Start"  
    Padding="30">  
    <autocomplete:SfAutoComplete x:Name="autoComplete"  
      HeightRequest="40"  
      SuggestionMode="ContainsWithCaseSensitive">  
      <autocomplete:SfAutoComplete.AutoCompleteSource>  
        <ListCollection:List x:TypeArguments="x:String">  
          <x:String>India</x:String>  
          <x:String>Uganda</x:String>  
          <x:String>Ukraine</x:String>  
          <x:String>Canada</x:String>  
          <x:String>United Arab Emirates</x:String>  
          <x:String>France</x:String>  
          <x:String>United Kingdom</x:String>  
          <x:String>China</x:String>  
          <x:String>United States</x:String>  
          <x:String>Japan</x:String>  
          <x:String>Angola</x:String>  
        </ListCollection:List>  
      </autocomplete:SfAutoComplete.AutoCompleteSource>  
    </autocomplete:SfAutoComplete>  
  </StackLayout>  
</ContentPage>
```

### C#

```
using Syncfusion.SfAutoComplete.XForms;
```

```
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                SuggestionMode = SuggestionMode.ContainsWithCaseSensitive,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "France",
                    "United Kingdom",
                    "China",
                    "United States",
                    "Japan",
                    "Angola"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```

Refer [this](#) link to learn more about the options available in SfAutoComplete to filter suggestions.



The complete Getting Started sample is available in [this](#) link.

## Populating Data

SfAutoComplete control can be populated with a list of string or business objects, which assists the user while typing. Users can choose one item from the filtered suggestion list.

[DataSource](#) property is used to populate data in SfAutoComplete control. This section explains populating AutoComplete with list of string and list of Employee details separately.

### Populating String Data

Create an instance of string list and populate items as shown below:

#### XML

```
<StackLayout
VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
```

#### C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
```

```
HeightRequest = 40,
DataSource = new List<string>()
{
    "India",
    "Uganda",
    "Ukraine",
    "Canada",
    "United Arab Emirates",
    "France",
    "United Kingdom",
    "China",
    "United States",
    "Japan",
    "Angola"
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
```



### Populating Business Objects

Apart from string data, SfAutoComplete can deal with business object data also. Now let us create Model and ViewModel classes to populate AutoComplete with Employee details.

#### Create and Initialize Business Models

Define a simple model class Employee with fields ID, Name and populate employee data in ViewModel.

#### C#

```
namespace AutocompleteSample
{
    public class Employee
    {
        private int id;
        public int ID
        {
            get { return id; }
            set { id = value; }
        }
        private string name;
        public string Name
```

```

{
    get { return name; }
    set { name = value; }
}
}

public class EmployeeViewModel
{
    private ObservableCollection<Employee> employeeCollection;
    public ObservableCollection<Employee> EmployeeCollection
    {
        get { return employeeCollection; }
        set { employeeCollection = value; }
    }
    public EmployeeViewModel()
    {
        employeeCollection = new ObservableCollection<Employee>();
        employeeCollection.Add(new Employee() { ID = 1, Name = "Eric" });
        employeeCollection.Add(new Employee() { ID = 2, Name = "James" });
        employeeCollection.Add(new Employee() { ID = 3, Name = "Jacob" });
        employeeCollection.Add(new Employee() { ID = 4, Name = "Lucas" });
        employeeCollection.Add(new Employee() { ID = 5, Name = "Mark" });
        employeeCollection.Add(new Employee() { ID = 6, Name = "Aldan" });
        employeeCollection.Add(new Employee() { ID = 7, Name = "Aldrin" });
        employeeCollection.Add(new Employee() { ID = 8, Name = "Alan" });
        employeeCollection.Add(new Employee() { ID = 9, Name = "Aaron" });
    }
}

```

#### *Populate data in AutoComplete and Setting DisplayMemberPath*

Now populate this EmployeeViewModel data in SfAutoComplete control by binding with [DataSource](#) property. At this point, the control is populated with the list of employees. But the Employee model contains two properties ID and Name so it is necessary to intimate by which property it should filter suggestions. [DisplayMemberPath](#) property specifies the property path with which filtering is done on business objects.

#### **XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
    <ContentPage.BindingContext>
    <local:EmployeeViewModel/>
    </ContentPage.BindingContext>
    <StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
    <autocomplete:SfAutoComplete x:Name="autoComplete"

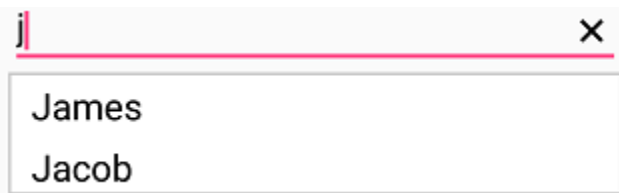
```

```
HeightRequest="40"
DisplayMemberPath="Name"
DataSource="{Binding EmployeeCollection}" />
</StackLayout>
</ContentPage>
```

**C#**

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            EmployeeViewModel employee = new EmployeeViewModel();
            StackLayout layout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                BindingContext = employee,
                DataSource = employee.EmployeeCollection,
                DisplayMemberPath = "Name"
            };
            layout.Children.Add(autoComplete);
            this.Content = layout;
        }
    }
}
```

**Note:** Set the EmployeeViewModel instance as the BindingContext of your control; this is done to bind properties of EmployeeViewModel to SfAutoComplete.

*Setting ItemTemplate*

[ItemTemplate](#) property helps to decorate suggestion items with custom templates. The following code explains the steps to add an image to the suggestion list item.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
```

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<ContentPage.BindingContext>
<local:EmployeeViewModel/>
</ContentPage.BindingContext>
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
DisplayMemberPath="Name"
DataSource="{Binding EmployeeCollection}">
<autocomplete:SfAutoComplete.ItemTemplate>
<DataTemplate>
<StackLayout Orientation="Horizontal"
Padding="2,0,0,0">
<Image Source="User.png"
WidthRequest="12"/>
<Label Text="{Binding Name}"
VerticalOptions="Center"
WidthRequest="500"/>
</StackLayout>
</DataTemplate>
</autocomplete:SfAutoComplete.ItemTemplate>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

The ItemTemplate in above XAML code is translated to C# and given below:

### C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            EmployeeViewModel employee = new EmployeeViewModel();
            StackLayout layout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
        }
    }
}

```

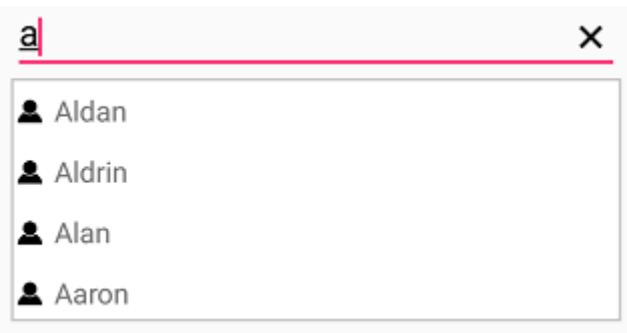
```

SfAutoComplete autoComplete = new SfAutoComplete()
{
    HeightRequest = 40,
    BindingContext = employee,
    DataSource = employee.EmployeeCollection,
    DisplayMemberPath = "Name"
};
DataTemplate itemTemplate = new DataTemplate(() =>
{
    StackLayout stack;
    Image image;
    Label label;
    stack = new StackLayout();
    stack.Orientation = StackOrientation.Horizontal;
    image = new Image();
    image.Source = (FileImageSource)ImageSource.FromFile("User.png");
    label = new Label();
    label.SetBinding(Label.TextProperty, "Name");
    label.WidthRequest = 500;
    stack.Children.Add(image);
    stack.Children.Add(label);
    return new ViewCell { View = stack };
});
autoComplete.ItemTemplate = itemTemplate;
layout.Children.Add(autoComplete);
this.Content = layout;
}
}
}

```

Refer [this](#) link to learn more about the customizing options available in SfAutoComplete control.

**Note:** Add the required image in drawable folder(Android), Resources folder(iOS) and at project location for UWP.



### DataTemplateSelector

SfAutoComplete supports DataTemplateSelector, which is used to choose a DataTemplate based on data object.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

```



```

xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<ContentPage.BindingContext>
<local:MobileDetailViewModel/>
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="default">
<ViewCell>
<Grid Padding="5">
<Label Text="{Binding Mobile}" TextColor="Green"/>
</Grid>
</ViewCell>
</DataTemplate>
<DataTemplate x:Key="specific">
<ViewCell>
<Grid Padding="5">
<Label Text="{Binding Mobile}" BackgroundColor="LightGray" TextColor="Red"/>
</Grid>
</ViewCell>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<autocomplete:SfAutoComplete DataSource="{Binding MobileCollection}"
DisplayMemberPath="Mobile">
<autocomplete:SfAutoComplete.ItemTemplate>
<local:DataTemplateSelectorViewModel DefaultTemplate="{StaticResource
default}" SpecificDataTemplate="{StaticResource specific}" />
</autocomplete:SfAutoComplete.ItemTemplate>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage
{
DataTemplate defaultTemplate;
DataTemplate specifictempalte;
public MainPage()
{
InitializeComponent();
MobileDetailViewModel mobileDetailViewModel = new MobileDetailViewModel();
defaultTemplate = new DataTemplate(() =>
{

```

```

Grid grid = new Grid();
grid.Padding = new Thickness(5);
Label label = new Label();
label.SetBinding(Label.TextProperty, "Mobile");
label.TextColor = Color.Green;
grid.Children.Add(label);
return new ViewCell { View = grid };
});
specifictempalte = new DataTemplate(() =>
{
    Grid grid = new Grid();
    grid.Padding = new Thickness(5);
    Label label = new Label();
    label.SetBinding(Label.TextProperty, "Mobile");
    label.BackgroundColor = Color.LightGray;
    label.TextColor = Color.Red;
    grid.Children.Add(label);
    return new ViewCell { View = grid };
});
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
SfAutoComplete autoComplete = new SfAutoComplete();
autoComplete.HeightRequest = 40;
autoComplete.DataSource = mobileDetailViewModel.MobileCollection;
this.BindingContext = mobileDetailViewModel;
autoComplete.DisplayMemberPath = "Mobile";
autoComplete.ItemTemplate = new DataTemplateSelectorViewModel {
    DefaultTemplate = defaultTemplate, SpecificDataTemplate = specifictempalte
};
layout.Children.Add(autoComplete);
Content = layout;
}
}
}

```

### Create and Initialize Business Models

Define a simple model class MobileDetail with fields IsAvailableInStock, Mobile and populate mobile detail in ViewModel.

### C#

```

namespace AutocompleteSample
{
    public class MobileDetailViewModel
    {
        public ObservableCollection<MobileDetail> MobileCollection { get; set; }
        public MobileDetailViewModel()
        {
            this.MobileCollection = new ObservableCollection<MobileDetail>()
            {
                new MobileDetail () { Mobile="Samasung S8", IsAvailableInStock=false },
                new MobileDetail () { Mobile="Samasung S9", IsAvailableInStock=true },
            }
        }
    }
}

```

```
new MobileDetail () { Mobile="Samsung S10", IsAvailableInStock=true },
new MobileDetail () { Mobile="Samsung S10 plus", IsAvailableInStock=true },
new MobileDetail () { Mobile="iPhone 7", IsAvailableInStock=true },
new MobileDetail () { Mobile="iPhone 8", IsAvailableInStock=true },
new MobileDetail () { Mobile="iPhone X", IsAvailableInStock=false },
new MobileDetail () { Mobile="iPhone XR", IsAvailableInStock=false },
new MobileDetail () { Mobile="iPhone XS", IsAvailableInStock=true },
};
}
}

public class MobileDetail
{
public string Mobile { get; set; }
public bool IsAvailableInStock { get; set; }
}
}
```

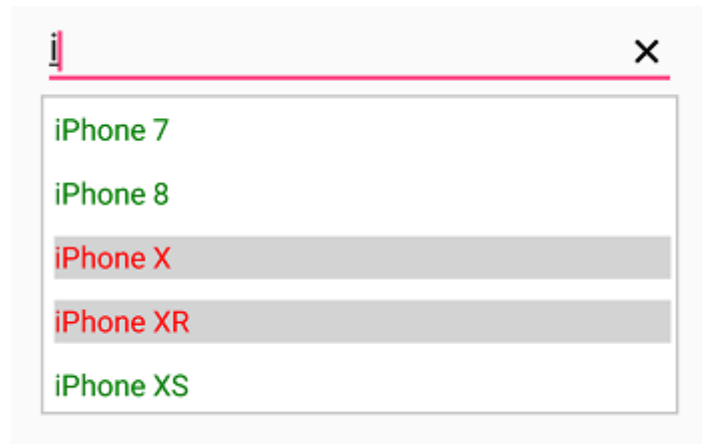
### OnSelectTemplate

The OnSelectTemplate is an overridden method to return a particular DataTemplate. The following code sample demonstrates how to use the OnSelectTemplate method.

#### C#

```
namespace AutocompleteSample
{
public class DataTemplateSelectorViewModel : DataTemplateSelector
{
public DataTemplate DefaultTemplate { get; set; }
public DataTemplate SpecificDataTemplate { get; set; }
protected override DataTemplate OnSelectTemplate(object item, BindableObject container)
{
var message = item as MobileDetail;
if (message.IsAvailableInStock == null)
return null;
return message.IsAvailableInStock == false ? SpecificDataTemplate :
DefaultTemplate;
}
}
}
```

The following screenshot illustrates the output of above code.



We have attached sample for reference. You can download the sample from the following link.

Sample Link: [SfAutoComplete DataTemplateSelector](#)

### AutoComplete Modes

AutoComplete provides three different ways to display the filtered suggestions. They are

- Suggest - displaying suggestion in drop down list
- Append - appending the first suggestion to text
- SuggestAppend - Both of these

**AutoCompleteMode** property is used to choose the suggestion display mode in SfAutoComplete control. The default value is Suggest.

### Suggesting Choices in List

The filtered suggestions are displayed in a drop down list. User can pick an item from the list.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
AutoCompleteMode="Suggest">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
```

```

<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

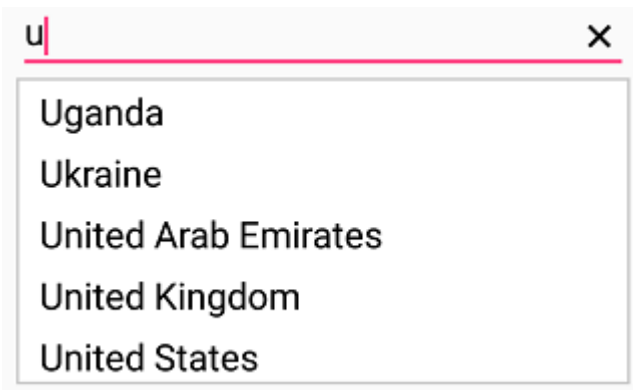
```

## C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                AutoCompleteMode = AutoCompleteMode.Suggest,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "France",
                    "United Kingdom",
                    "China",
                    "United States",
                    "Japan",
                    "Angola"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}

```



### Appending Suggestion to Text

The first item in filtered suggestions is appended to SfAutoComplete text. In this mode, drop down remains closed.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
AutoCompleteMode="Append">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

### C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                AutoCompleteMode = AutoCompleteMode.Append,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "France",
                    "United Kingdom",
                    "China",
                    "United States",
                    "Japan",
                    "Angola"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}

```

Uganda ✕

### Suggesting Choices and Appending Suggestions to Text

The text is appended to the first matched item in the suggestions collection and filtered suggestions are displayed in a drop down list. The user can pick from a list directly or use up and down keys for browsing the list.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

```

```

xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
AutoCompleteMode="SuggestAppend">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

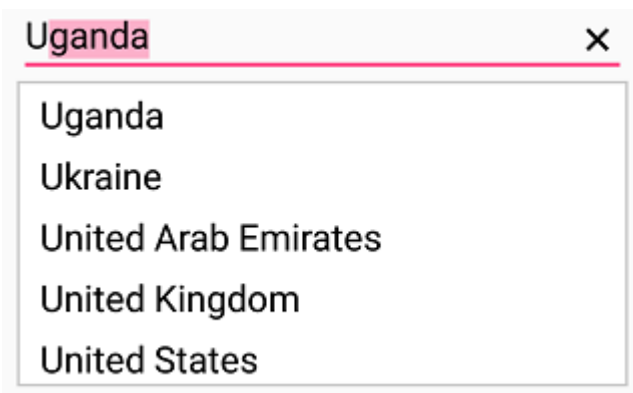
```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                AutoCompleteMode = AutoCompleteMode.SuggestAppend,
                AutoCompleteSource = new List<string>()
            {

```



```
"India",  
"Uganda",  
"Ukraine",  
"Canada",  
"United Arab Emirates",  
"France",  
"United Kingdom",  
"China",  
"United States",  
"Japan",  
"Angola"  
}  
};  
stackLayout.Children.Add(autoComplete);  
this.Content = stackLayout;  
}  
}  
}
```



## Multiple Selection

Select multiple items from a suggestion list. There are two ways to perform multi selection in autocomplete.

- Token Representation
- Delimiter

### Token Representation

Selected items will be displayed with a customizable token representation and the users can remove each tokenized item with the close button.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:autocomplete="clr-  
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple  
te.XForms"  
xmlns:ListCollection="clr-  
namespace:System.Collections.Generic;assembly=netstandard"  
xmlns:local="clr-namespace:AutocompleteSample"
```

```

x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
MultiSelectMode="Token"
TokensWrapMode="Wrap"
IsSelectedItemsVisibleInDropDown="false">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

**C#**

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                MultiSelectMode = MultiSelectMode.Token,
                TokensWrapMode = TokensWrapMode.Wrap,
                IsSelectedItemsVisibleInDropDown = false,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates"
                }
            };
            stackLayout.Children.Add(autoComplete);
        }
    }
}

```

```

this.Content = stackLayout;
}
}
}

```

### Wrap Mode of Token

The selected item can be displayed as token inside SfAutoComplete in two ways. They are

- **Wrap** - When **TokensWrapMode** is set to **Wrap** the selected items will be wrap to the next line of the SfAutoComplete.
- **None** - When **TokensWrapMode** is set to **None** the selected item will be wrap in horizontal orientation.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<ContentPage.BindingContext>
<local:EmployeeViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
DropDownItemHeight="50"
DisplayMemberPath="Name"
ImageMemberPath="Image"
MultiSelectMode="Token"
TokensWrapMode="Wrap"
DataSource="{Binding EmployeeCollection}">
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

### C#

```

using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage

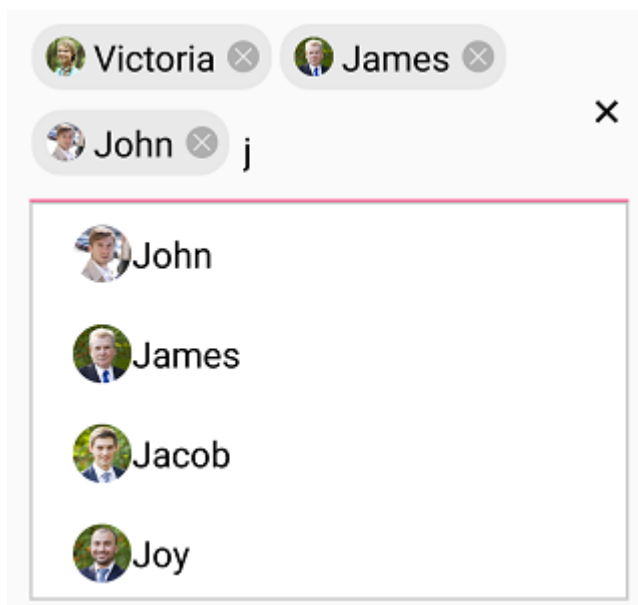
```

```
{
public MainPage()
{
InitializeComponent();
}
}
public class Employee
{
private string image;
public string Image
{
get { return image; }
set { image = value; }
}
private string name;
public string Name
{
get { return name; }
set { name = value; }
}
}
// Create EmployeeViewModel class holds the collection of employee data.
public class EmployeeViewModel : INotifyPropertyChanged
{
private ObservableCollection<Employee> employeeCollection;
public ObservableCollection<Employee> EmployeeCollection
{
get { return employeeCollection; }
set { employeeCollection = value; }
}
public EmployeeViewModel()
{
employeeCollection = new ObservableCollection<Employee>();
employeeCollection.Add(new Employee() { Image = "John.png", Name = "John"
});
employeeCollection.Add(new Employee() { Image = "James.png", Name = "James"
});
employeeCollection.Add(new Employee() { Image = "Jacob.png", Name = "Jacob"
});
employeeCollection.Add(new Employee() { Image = "Joy.png", Name = "Joy" });
employeeCollection.Add(new Employee() { Image = "Justin.png", Name =
"Justin" });
employeeCollection.Add(new Employee() { Image = "Jerome.png", Name =
"Jerome" });
employeeCollection.Add(new Employee() { Image = "Jessica.png", Name =
"Jessica" });
employeeCollection.Add(new Employee() { Image = "Victoria.png", Name =
"Victoria" });
}
public int GetHeight(bool value)
{
if (value)
{
return 80;
}
return 40;
}
}
```

```

private int toHeight = 40;
public int ToHeight
{
    get { return toHeight; }
    set
    {
        toHeight = value;
        RaisePropertyChanged("ToHeight");
    }
}
private bool isToFocused = false;
public bool IsToFocused
{
    get { return isToFocused; }
    set
    {
        isToFocused = value;
        ToHeight = GetHeight(value);
        RaisePropertyChanged("IsToFocused");
    }
}
public event PropertyChangedEventHandler PropertyChanged;
private void RaisePropertyChanged(String name)
{
    if (PropertyChanged != null)
        this.PropertyChanged(this, new PropertyChangedEventArgs(name));
}
}

```



#### Token Customization

Customization can be done for Token. There are various ways to customize the tokens. They are as follows.

- **TextColor** - sets the color of the text inside the token.
- **FontSize** - sets the size of the Font inside the token.
- **FontFamily** - sets the Font family for the text inside the token.
- **BackgroundColor** - sets the background color of the token.
- **SelectedBackgroundColor** - sets the background color of the token when it is selected.
- **IsCloseButtonVisible** - Enables and disables the close button inside SfAutoComplete.
- **DeleteButtonColor** - sets the color of the close button inside SfAutoComplete.
- **CornerRadius** - sets the corner radius for the token.
- **DeleteButtonPlacement** - sets the placement of delete button. **Left** and **Right** are the placement options. By default, it is set placed at right side of the token.

**Note:** SelectedBackgroundColor and CornerRadius support has enhanced only on iOS and Android platform.

### XML

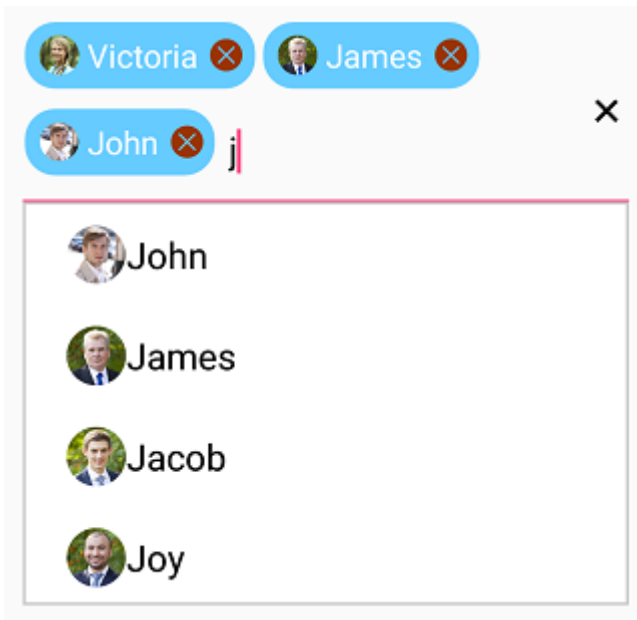
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
  <ContentPage.BindingContext>
    <local:EmployeeViewModel/>
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <StackLayout>
      <autocomplete:SfAutoComplete
        DisplayMemberPath="Name"
        MultiSelectMode="Token"
        ImageMemberPath="Image"
        TokensWrapMode="Wrap"
        DataSource="{Binding EmployeeCollection}">
        <autocomplete:SfAutoComplete.TokenSettings>
          <autocomplete:TokenSettings
            FontSize="16"
            BackgroundColor="#66ccff"
            TextColor="White"
            SelectedBackgroundColor="#ffffe0"
            DeleteButtonColor="#993300"
            FontFamily="Times New Roman"
            DeleteButtonPlacement="Right"
            IsCloseButtonVisible="true"
            CornerRadius="15"/>
        </autocomplete:SfAutoComplete.TokenSettings>
      </autocomplete:SfAutoComplete>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

**C#**

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = new EmployeeViewModel();
            StackLayout stackLayout = new StackLayout();
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                MultiSelectMode = MultiSelectMode.Token,
                DisplayMemberPath = "Name",
                ImageMemberPath = "Image",
                TokensWrapMode = TokensWrapMode.Wrap,
            };
            autoComplete.SetBinding(SfAutoComplete.DataSourceProperty,
                "EmployeeCollection");
            autoComplete.TokenSettings = new TokenSettings()
            {
                FontSize = 16,
                BackgroundColor = Color.FromHex("#66ccff"),
                TextColor = Color.White,
                SelectedBackgroundColor = Color.FromHex("#ffffe0"),
                DeleteButtonColor = Color.FromHex("#993300"),
                FontFamily = "Times New Roman",
                IsCloseButtonVisible = true,
                DeleteButtonPlacement = DeleteButtonPlacement.Right,
                CornerRadius = 15
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
    public class Employee
    {
        private string image;
        private string name;
        public string Image
        {
            get
            {
                return image;
            }
            set
            {
                image = value;
            }
        }
        public string Name
        {
            get
```

```
{
    return name;
}
set
{
    name = value;
}
}
}
public class EmployeeViewModel
{
    private ObservableCollection<Employee> employeeCollection;
    public ObservableCollection<Employee> EmployeeCollection
    {
        get
        {
            return employeeCollection;
        }
        set
        {
            employeeCollection = value;
        }
    }
    public EmployeeViewModel()
    {
        employeeCollection.Add(new Employee() { Image = "John.png", Name = "John"
        });
        employeeCollection.Add(new Employee() { Image = "James.png", Name = "James"
        });
        employeeCollection.Add(new Employee() { Image = "Jacob.png", Name = "Jacob"
        });
        employeeCollection.Add(new Employee() { Image = "Joy.png", Name = "Joy" });
        employeeCollection.Add(new Employee() { Image = "Justin.png", Name =
        "Justin" });
        employeeCollection.Add(new Employee() { Image = "Jerome.png", Name =
        "Jerome" });
        employeeCollection.Add(new Employee() { Image = "Jessica.png", Name =
        "Jessica" });
        employeeCollection.Add(new Employee() { Image = "Victoria.png", Name =
        "Victoria" });
    }
}
}
```





### Delimiter

When selecting the multiple items, the selected items can be divided with a desired character given for a delimiter. We can set delimiter character with the `Delimiter` property.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
MultiSelectMode="Delimiter"
Delimiter=", ">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

N> The optimal value for Delimiter property is any single character.

**C#**

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                MultiSelectMode = MultiSelectMode.Delimiter,
                Delimiter = ",",
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```



*Selection indicator*

The autocomplete enables the user to indicate the selected item from the datasource when selecting multiple items from the dropdown. It can be performed by enabling `EnableSelectionIndicator` property.

**Note:** Selection Indicator support has enhanced only on iOS and Android platform.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String"
x:Key="customfontfamily"
iOS="Sample"
Android="sample.ttf"
WinPhone="sample.ttf#Sample" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.BindingContext>
<local:EmployeeViewModel/>
</ContentPage.BindingContext>
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
ShowSuggestionsOnFocus="true"
IsSelectedItemsVisibleInDropDown="true"
IndicatorText="A"
IndicatorTextSize="15"
IndicatorFontFamily="{StaticResource customfontfamily}"
IndicatorTextColor="Red"
EnableSelectionIndicator="true"
MultiSelectMode="Token"
DataSource="{Binding EmployeeCollection}"/>
</StackLayout>
</ContentPage>
```

**C#**

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage
```

```
{
public MainPage()
{
InitializeComponent();
EmployeeViewModel viewModel = new EmployeeViewModel();
StackLayout stackLayout = new StackLayout()
{
VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
SfAutoComplete autoComplete = new SfAutoComplete()
{
HeightRequest = 40,
ShowSuggestionsOnFocus = true,
IsSelectedItemVisibleInDropDown = true,
IndicatorText = "A",
IndicatorTextSize = 15,
IndicatorFontFamily = Device.RuntimePlatform == "iOS" ? "Sample" :
Device.RuntimePlatform == "Android" ? "sample.ttf" : "sample.ttf#Sample",
IndicatorTextColor = Color.Red,
EnableSelectionIndicator = true,
MultiSelectMode = MultiSelectMode.Token,
DataSource = viewModel.EmployeeCollection,
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
public class Employee
{
private string image;
private string name;
public string Image
{
get
{
return image;
}
set
{
image = value;
}
}
public string Name
{
get
{
return name;
}
set
{
name = value;
}
}
}
public class EmployeeViewModel
```

```

{
    private ObservableCollection<Employee> employeeCollection;
    public ObservableCollection<Employee> EmployeeCollection
    {
        get
        {
            return employeeCollection;
        }
        set
        {
            employeeCollection = value;
        }
    }
    public EmployeeViewModel()
    {
        employeeCollection.Add(new Employee() { Image = "John.png", Name = "John" });
        employeeCollection.Add(new Employee() { Image = "James.png", Name = "James" });
        employeeCollection.Add(new Employee() { Image = "Jacob.png", Name = "Jacob" });
        employeeCollection.Add(new Employee() { Image = "Joy.png", Name = "Joy" });
        employeeCollection.Add(new Employee() { Image = "Justin.png", Name = "Justin" });
        employeeCollection.Add(new Employee() { Image = "Jerome.png", Name = "Jerome" });
        employeeCollection.Add(new Employee() { Image = "Jessica.png", Name = "Jessica" });
        employeeCollection.Add(new Employee() { Image = "Victoria.png", Name = "Victoria" });
    }
}

```

### Item padding

The autocomplete enables the user to provide padding for the items inside dropdown using `ItemPadding` property.

---

**Note:** `ItemPadding` property is available only on iOS and Android platform.

---

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComplete.XForms"
xmlns:ListCollection="clr-namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
    <StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
        <autocomplete:SfAutoComplete x:Name="autoComplete"

```

```

ShowSuggestionsOnFocus="true"
ItemPadding="20,10,0,0"
MultiSelectMode="Token">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

**C#**

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                MultiSelectMode = MultiSelectMode.Token,
                ShowSuggestionsOnFocus = true,
                ItemPadding = new Thickness(20, 10, 0, 0),
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}

```

## AutoComplete 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. The available filtering modes are

- StartsWith
- StartsWithCaseSensitive
- Contains
- ContainsWithCaseSensitive
- Equals
- EqualsWithCaseSensitive
- EndsWith
- EndsWithCaseSensitive
- Custom

### Filtering Words that Starts with Input Text

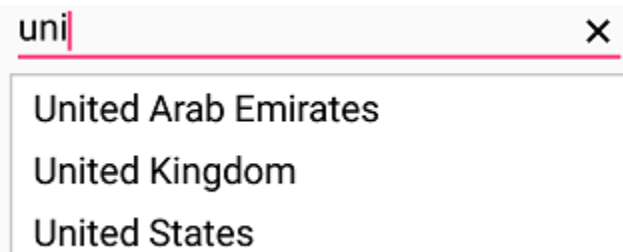
Displays all the matches that starts with the typed characters in control. This strategy is case in-sensitive.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
SuggestionMode="StartsWith">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

**C#**

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                SuggestionMode = SuggestionMode.StartsWith,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "France",
                    "United Kingdom",
                    "China",
                    "United States",
                    "Japan",
                    "Angola"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```

*Filtering Words that Starts with Input Text - CaseSensitive*

Displays all the matches that starts with the typed characters in control. This strategy is case sensitive.

**XML**



```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
SuggestionMode="StartsWithCaseSensitive">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

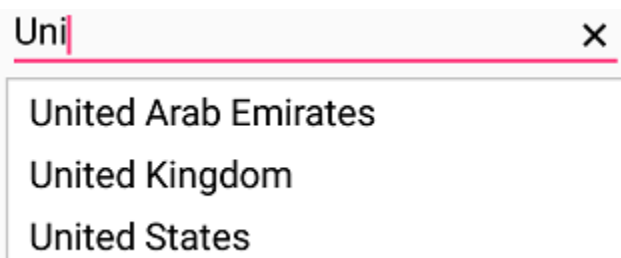
## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout()
{
VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
SfAutoComplete autoComplete = new SfAutoComplete()
{
HeightRequest = 40,
SuggestionMode = SuggestionMode.StartsWithCaseSensitive,
```

```

AutoCompleteSource = new List<string>()
{
    "India",
    "Uganda",
    "Ukraine",
    "Canada",
    "United Arab Emirates",
    "France",
    "United Kingdom",
    "China",
    "United States",
    "Japan",
    "Angola"
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```



### Filtering Words that Contains the Input Text

Displays all the matches that contains the typed characters in control. This strategy is case in-sensitive.

### XML

```

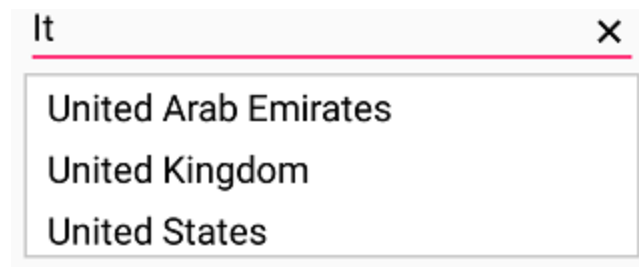
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
SuggestionMode="Contains">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>

```

```
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                SuggestionMode = SuggestionMode.Contains,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "France",
                    "United Kingdom",
                    "China",
                    "United States",
                    "Japan",
                    "Angola"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```



#### *Filtering Words that Contains the Input Text - CaseSensitive*

Displays all the matches that contains the typed characters in control. This strategy is case sensitive.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
SuggestionMode="ContainsWithCaseSensitive">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

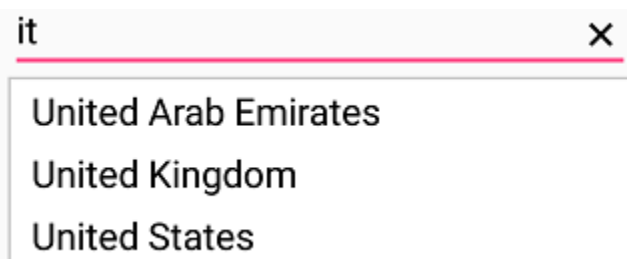
#### C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage
```

```

{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout()
{
VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
SfAutoComplete autoComplete = new SfAutoComplete()
{
HeightRequest = 40,
SuggestionMode = SuggestionMode.ContainsWithCaseSensitive,
AutoCompleteSource = new List<string>()
{
"India",
"Uganda",
"Ukraine",
"Canada",
"United Arab Emirates",
"France",
"United Kingdom",
"China",
"United States",
"Japan",
"Angola"
}
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```



### Filtering Words that Equals the Input Text

Displays all the words that completely matches with the typed characters in control. This strategy is case in-sensitive.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"

```

```

xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
SuggestionMode="Equals">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                SuggestionMode = SuggestionMode.Equals,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",

```

```

"United Arab Emirates",
"France",
"United Kingdom",
"China",
"United States",
"Japan",
"Angola"
}
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```

### *Filtering Words that Equals the Input Text - CaseSensitive*

Displays all the words that completely matches with the typed characters in control. This strategy is case sensitive.

### **XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
SuggestionMode="EqualsWithCaseSensitive">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

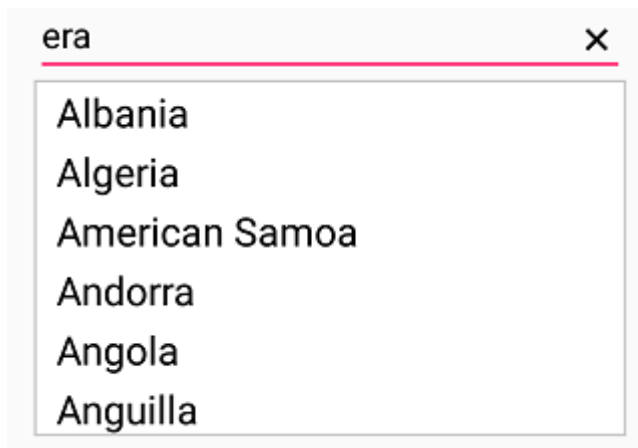
**C#**

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                SuggestionMode = SuggestionMode.EqualsWithCaseSensitive,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "France",
                    "United Kingdom",
                    "China",
                    "United States",
                    "Japan",
                    "Angola"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```

*Custom*

Filter items in the suggestion list based on a custom search by the user. This will help to apply our typo toleration functionality to the control.





### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
DropDownTextSize="20"
AutoCompleteMode="Suggest"
MaximumDropDownHeight="200"
SuggestionMode="Custom"
>
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>American Samoa</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
<x:String>Anguilla</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

### C#

```
using Syncfusion.SfAutoComplete.XForms;
using System;
using System.Collections.Generic;
```

```
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                DropDownTextSize = 20,
                AutoCompleteMode = AutoCompleteMode.Suggest,
                MaximumDropDownHeight = 200,
                SuggestionMode = SuggestionMode.Custom,
                Filter = ContainingSpaceFilter,
                AutoCompleteSource = new List<string>()
                {
                    "Albania",
                    "Algeria",
                    "American Samoa",
                    "Andorra",
                    "Angola",
                    "Anguilla"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
        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;
                    }
                }
                catch (Exception)
                {
                    return text.Contains(search);
                }
            }
        }
    }
}
```

```

}
}
else
return false;
}
}
}

```

### Filtering Words that Ends with the Input Text

Displays all the matches that ends with the typed characters in control. This strategy is case in-sensitive.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
SuggestionMode="EndsWith">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

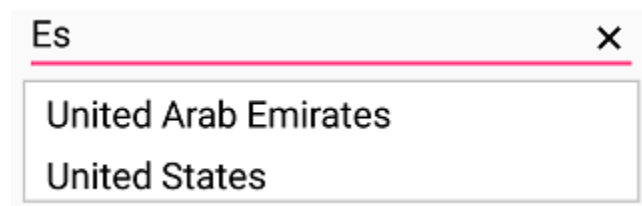
### C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage
{

```

```
public MainPage()
{
    InitializeComponent();
    StackLayout stackLayout = new StackLayout()
    {
        VerticalOptions = LayoutOptions.Start,
        HorizontalOptions = LayoutOptions.Start,
        Padding = new Thickness(30)
    };
    SfAutoComplete autoComplete = new SfAutoComplete()
    {
        HeightRequest = 40,
        SuggestionMode = SuggestionMode.EndsWith,
        AutoCompleteSource = new List<string>()
        {
            "India",
            "Uganda",
            "Ukraine",
            "Canada",
            "United Arab Emirates",
            "France",
            "United Kingdom",
            "China",
            "United States",
            "Japan",
            "Angola"
        }
    };
    stackLayout.Children.Add(autoComplete);
    this.Content = stackLayout;
}
}
```



#### *Filtering Words that Ends with the Input Text - CaseSensitive*

Displays all the matches that ends with the typed characters in control. This strategy is case sensitive.

#### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
```

```

<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
SuggestionMode="EndsWithCaseSensitive">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

```

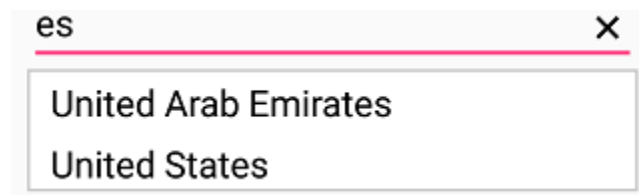
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                SuggestionMode = SuggestionMode.EndsWithCaseSensitive,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "France",
                    "United Kingdom",
                    "China",

```

```

"United States",
"Japan",
"Angola"
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```



### Diacritic Sensitivity

The control does not stick with one type of keyboard, so you can populate items from a language with letters containing diacritics, and search for them with English characters from an en-US keyboard. Users can enable or disable the diacritic sensitivity with the `IgnoreDiacritic` property. In the below code example we have illustrate how to enables the diacritic sensitivity so that items in the suggestion list get populated by entering any diacritic character of that alphabet.

### XML

```

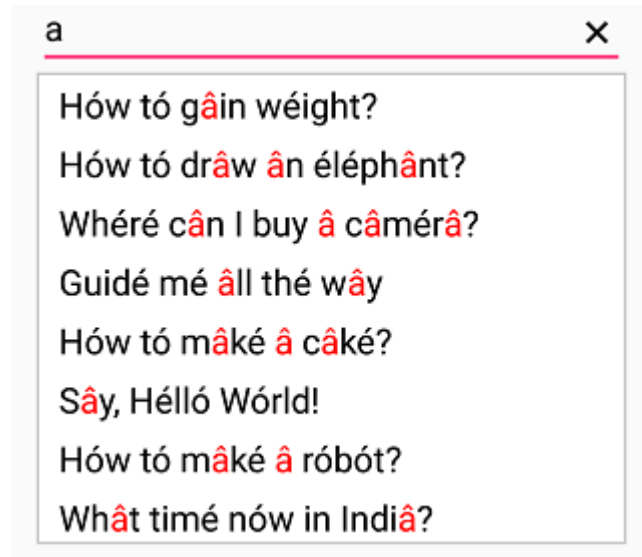
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
TextHighlightMode="MultipleOccurrence"
SuggestionMode="Contains"
HighlightedTextColor="Red"
IgnoreDiacritic="false">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Hów tó gâin wéight?</x:String>
<x:String>Hów tó drâw ân éléphânt?</x:String>
<x:String>Whéré cân I buy â câmérâ?</x:String>
<x:String>Guidé mé âll thé wây</x:String>
<x:String>Hów tó mâké â câké?</x:String>
<x:String>Sây, Hélló Wórlđ!</x:String>
<x:String>Hów tó mâké â róbót?</x:String>
<x:String>Whât tímé nów in Indiâ?</x:String>

```

```
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                TextHighlightMode = OccurrenceMode.MultipleOccurrence,
                SuggestionMode = SuggestionMode.Contains,
                HighlightedTextColor = Color.Red,
                IgnoreDiacritic = false,
                AutoCompleteSource = new List<string>()
                {
                    "Hów tó gâin wéight?",
                    "Hów tó drâw ân éléphânt?",
                    "Whéré cân I buy â câmérâ?",
                    "Guidé mé âll thé wây",
                    "Hów tó mâké â câké?",
                    "Sây, Hélló Wórlđ!",
                    "Hów tó mâké â róbót?",
                    "Whât timé nów in Indiâ?"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```



### Header and Footer

We can provide Header and Footer view in the suggestion list in SfAutoComplete by enabling `ShowDropDownHeaderView` and `ShowDropDownFooterView`.

### Header Content

We can provide Header Content at the top of the AutoComplete's Suggestion box.

`DropDownHeaderView` property is used to set the content of the header. The height of the Header in the SfAutoComplete can be adjusted by the property `DropDownHeaderViewHeight`.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
ShowDropDownHeaderView ="True"
DropDownHeaderViewHeight="50"
ValueChanged="SfAutoComplete_ValueChanged">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
```



```

<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
<autocomplete:SfAutoComplete.DropDownHeaderView>
<StackLayout BackgroundColor="#f0f0f0" >
<Label x:Name="SearchLabel"
FontSize="20"
VerticalTextAlignment="Center"
HorizontalOptions="Center"
VerticalOptions="Center"
TextColor="#006bcd" />
</StackLayout>
</autocomplete:SfAutoComplete.DropDownHeaderView>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

**C#**

```

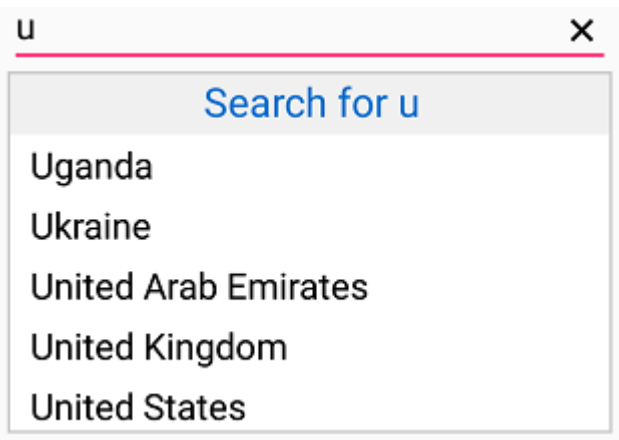
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        Label SearchLabel;
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                ShowDropDownHeaderView = true,
                DropDownHeaderViewHeight = 50,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "France",
                    "United Kingdom",
                    "China",
                    "United States",
                    "Japan",

```

```

"Angola"
}
};
StackLayout layout = new StackLayout()
{
    BackgroundColor = Color.FromHex("#f0f0f0")
};
SearchLabel = new Label()
{
    FontSize = 20,
    VerticalTextAlignment = TextAlignment.Center,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    TextColor = Color.FromHex("#006bcd")
};
layout.Children.Add(SearchLabel);
autoComplete.DropDownHeaderView = layout;
autoComplete.ValueChanged += SfAutoComplete_ValueChanged;
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
private void SfAutoComplete_ValueChanged(object sender,
Syncfusion.SfAutoComplete.XForms.ValueChangedEventArgs e)
{
    SearchLabel.Text = "Search for " + e.Value;
}
}
}

```



### Footer Content

We can provide Footer Content at the bottom of the AutoComplete's Suggestion box.

**DropDownFooterView** property is used to set the content of the footer. The height of the Header in the SfAutoComplete can be adjusted by the property **DropDownFooterViewHeight**.

The following code example illustrates how to set Footer content in SfAutoComplete.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

```

```

xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
ShowDropDownFooterView ="True"
DropDownFooterViewHeight="50">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
<autocomplete:SfAutoComplete.DropDownFooterView>
<StackLayout BackgroundColor="#f0f0f0" >
<Label Text="Add New"
FontSize="20"
VerticalTextAlignment="Center"
HorizontalOptions="Center"
VerticalOptions="Center"
TextColor="#006bcd" />
</StackLayout>
</autocomplete:SfAutoComplete.DropDownFooterView>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

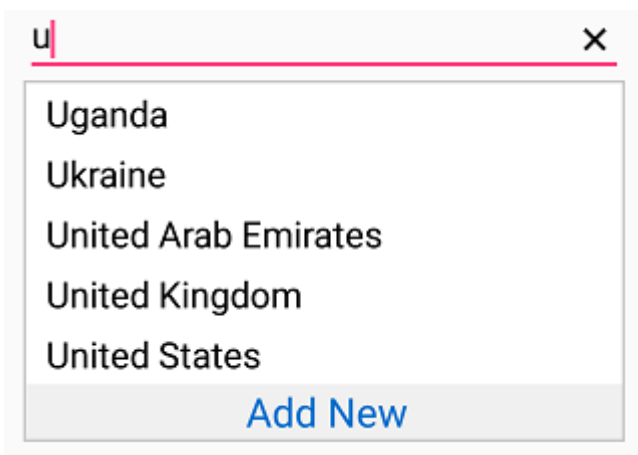
## C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {

```

```
VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
SfAutoComplete autoComplete = new SfAutoComplete()
{
    HeightRequest = 40,
    ShowDropDownFooterView = true,
    DropDownFooterViewHeight = 50,
    AutoCompleteSource = new List<string>()
    {
        "India",
        "Uganda",
        "Ukraine",
        "Canada",
        "United Arab Emirates",
        "France",
        "United Kingdom",
        "China",
        "United States",
        "Japan",
        "Angola"
    }
};
StackLayout layout = new StackLayout()
{
    BackgroundColor = Color.FromHex("#f0f0f0")
};
Label SearchLabel = new Label()
{
    FontSize = 20,
    Text = "Add New",
    VerticalTextAlignment = TextAlignment.Center,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    TextColor = Color.FromHex("#006bcd")
};
layout.Children.Add(SearchLabel);
autoComplete.DropDownFooterView = layout;
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
```



### Highlighting matched text

Highlight matching characters in a suggestion list to pick an item with more clarity. There are two ways to highlight the matching text:

- First Occurrence
- Multiple Occurrence

The text highlight can be indicated with various customizing styles by enabling the below properties. They are

- HighlightedTextColor - sets the color of the highlighted text for differentiating the highlighted characters.
- HighlightedTextFontAttributes - sets the FontAttributes of the highlighted text.

### First Occurrence

It highlights the first position of the matching characters in the suggestion list.

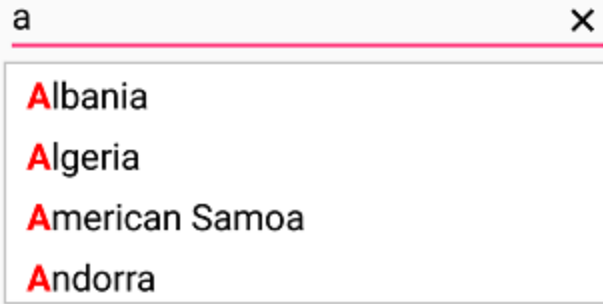
### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
TextHighlightMode="FirstOccurrence"
HighlightedTextColor="Red"
HighlightedTextFontAttributes="Bold"
SuggestionMode="StartsWith">
<autocomplete:SfAutoComplete.AutoCompleteSource>
```

```
<ListCollection:List x:TypeArguments="x:String">
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>American Samoa</x:String>
<x:String>Andorra</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                SuggestionMode = SuggestionMode.StartsWith,
                TextHighlightMode = OccurrenceMode.FirstOccurrence,
                HighlightedTextColor = Color.Red,
                HighlightedTextFontAttributes = FontAttributes.Bold,
                AutoCompleteSource = new List<string>()
                {
                    "Albania",
                    "Algeria",
                    "American Samoa",
                    "Andorra"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```



### Multiple Occurrence

It highlights the matching character that are present everywhere in the suggestion list for Contains case in SuggestionMode.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
TextHighlightMode="MultipleOccurrence"
HighlightedTextColor="Red"
HighlightedTextFontAttributes="Bold"
SuggestionMode="Contains">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>American Samoa</x:String>
<x:String>Andorra</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

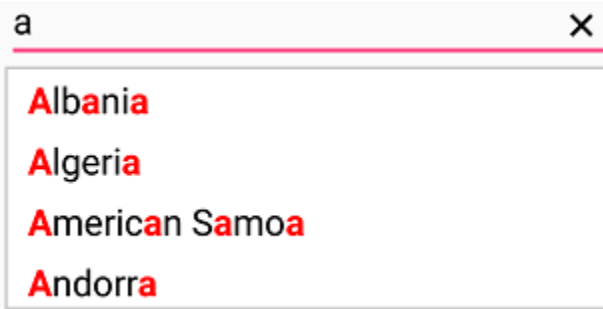
### C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
```

```

public MainPage()
{
    InitializeComponent();
    StackLayout stackLayout = new StackLayout()
    {
        VerticalOptions = LayoutOptions.Start,
        HorizontalOptions = LayoutOptions.Start,
        Padding = new Thickness(30)
    };
    SfAutoComplete autoComplete = new SfAutoComplete()
    {
        HeightRequest = 40,
        SuggestionMode = SuggestionMode.Contains,
        TextHighlightMode = OccurrenceMode.MultipleOccurrence,
        HighlightedTextColor = Color.Red,
        HighlightedTextFontAttributes = FontAttributes.Bold,
        AutoCompleteSource = new List<string>()
        {
            "Albania",
            "Algeria",
            "American Samoa",
            "Andorra"
        }
    };
    stackLayout.Children.Add(autoComplete);
    this.Content = stackLayout;
}
}

```



### Maximum Display Item with Expander

Restrict the number of suggestions displayed and have the remaining items loaded by selecting LoadMore. We can restrict maximum suggestion to be displayed with the `MaximumSuggestion` property. We can set the desired text for displaying the Load more text with the property `LoadMoreText`.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"

```



```

xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
MultiSelectMode="Delimiter"
Delimiter=", "
LoadMoreText="LOAD MORE"
MaximumSuggestion="2">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>American Samoa</x:String>
<x:String>Andorra</x:String>
<x:String>Aruba</x:String>
<x:String>Angola</x:String>
<x:String>Argentina</x:String>
<x:String>Armenia</x:String>
<x:String>America</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

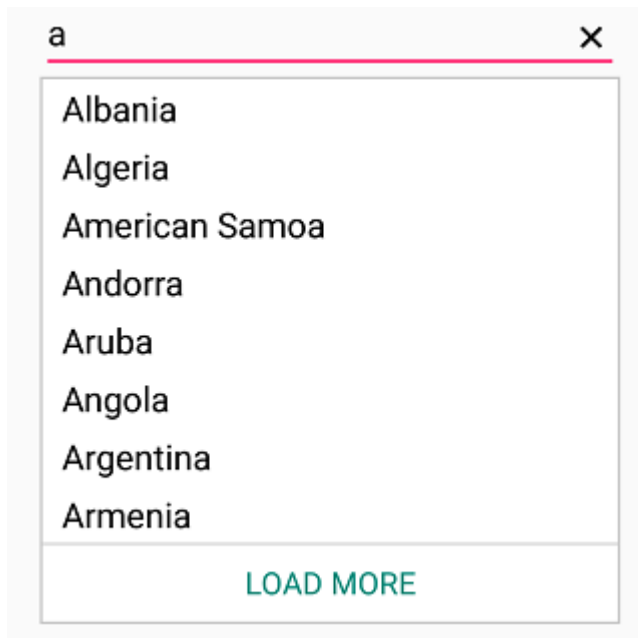
**C#**

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                MultiSelectMode = MultiSelectMode.Delimiter,
                Delimiter = ", ",
                LoadMoreText = "LOAD MORE",
                MaximumSuggestion = 2,
                AutoCompleteSource = new List<string>()
            {

```

```
"Albania",  
"Algeria",  
"American Samoa",  
"Andorra",  
"Aruba",  
"Angola",  
"Argentina",  
"Armenia",  
"America"  
}  
};  
stackLayout.Children.Add(autoComplete);  
this.Content = stackLayout;  
}  
}  
}
```



#### *Restricting the maximum display of item dynamically*

We can restrict the maximum display of items dynamically by calling `LoadMore` method. The user can dynamically change the maximum suggestion count by calling `LoadMore` method by giving the maximum suggestion as the argument inside.

**Note:** `LoadMore` method has enhanced only on iOS and Android platform.

#### **C#**

```
// without passing argument  
autocomplete.LoadMore();  
// with passing argument  
autocomplete.LoadMore(5);
```

## No Results Found

When the entered item is not in the suggestion list, SfAutoComplete displays a text indicating there is no search results found. We can set the desire text to be displayed for indicating no results found with the `NoResultsFoundText` property.

### XML

```
?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
NoResultsFoundText="No Results Found">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

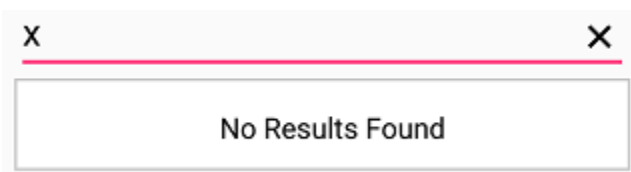
### C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
```

```

NoResultsFoundText = "No Results Found",
AutoCompleteSource = new List<string>()
{
    "India",
    "Uganda",
    "Ukraine",
    "Canada",
    "United Arab Emirates"
}
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```



### Customizing NoResultsFoundText

The `NoResultsFoundTextColor`, `NoResultsFoundFontSize`, `NoResultsFoundFontAttributes`, and `NoResultsFoundFontFamily` properties are used to customize the foreground color, font size, font attribute, and font family of `NoResultsFoundText`.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String"
x:Key="fonts"
iOS="Pacifico"
Android="Pacifico.ttf"
UWP="Pacifico.ttf#Pacifc"/>
</ResourceDictionary>
</ContentPage.Resources>
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
NoResultsFoundText="No Results Found"
NoResultsFoundTextColor="DarkGreen"
NoResultsFoundFontSize="20"

```

```

NoResultsFoundFontAttributes="Bold"
NoResultsFoundFontFamily="{StaticResource fonts}">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

**C#**

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                NoResultsFoundText = "No Results Found",
                NoResultsFoundFontAttributes = FontAttributes.Bold,
                NoResultsFoundFontSize = 20,
                NoResultsFoundTextColor = Color.DarkGreen,
                NoResultsFoundFontFamily = Device.RuntimePlatform == "iOS" ? "Pacifico" :
                Device.RuntimePlatform == "Android" ? "Pacifico.ttf" :
                "Pacifico.ttf#Pacifico",
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}

```

```
}

```

**Note:** <https://xamarinhelp.com/custom-fonts-xamarin-forms/> provides how to add the ttf file in each platform



## Handling Selected Items

SfAutoComplete provides a way to handle the selected item using the following properties:

- SelectedIndex
- SelectedIndices
- SelectedItem

### SelectedIndex

You can get or set the index of the selected item using the [SelectedIndex](#) property. It can be applicable only when [MultiSelectMode](#) is None.

*How to set the index of item to be selected*

The [SelectedIndex](#) property holds the index of selected item in suggestion list.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
MultiSelectMode="None"
SelectedIndex="1">
```

```

<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

**C#**

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                SelectedIndex = 1,
                MultiSelectMode = MultiSelectMode.None,
                AutoCompleteSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",
                    "Argentina",
                    "Anguilla",
                    "Albania",
                    "Algeria",
                    "Andorra",
                    "Angola"
                }
            };
            stackLayout.Children.Add(autoComplete);

```

```

this.Content = stackLayout;
}
}
}

```

### *Retrieving the index of selected item*

When an item is selected from suggestion list, its index can be retrieved using the [SelectedIndex](#) property.

The following code snippet demonstrates the way to retrieve [SelectedIndex](#) and display it in an alert.

### **XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
MultiSelectMode="None"
SelectionChanged="autoComplete_SelectionChanged">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

### **C#**

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{

```



```

public partial class MainPage : ContentPage
{
    SfAutoComplete autoComplete;
    public MainPage()
    {
        InitializeComponent();
        StackLayout stackLayout = new StackLayout()
        {
            VerticalOptions = LayoutOptions.Start,
            HorizontalOptions = LayoutOptions.Start,
            Padding = 30
        };
        autoComplete = new SfAutoComplete()
        {
            HeightRequest = 40,
            MultiSelectMode = MultiSelectMode.None,
            AutoCompleteSource = new List<string>()
            {
                "Antigua and Barbuda",
                "American Samoa",
                "Afghanistan",
                "Antarctica",
                "Argentina",
                "Anguilla",
                "Albania",
                "Algeria",
                "Andorra",
                "Angola"
            }
        };
        autoComplete.SelectionChanged += autoComplete_SelectionChanged;
        stackLayout.Children.Add(autoComplete);
        this.Content = stackLayout;
    }
    private void autoComplete_SelectionChanged(object sender,
        Syncfusion.SfAutoComplete.XForms.SelectionChangedEventArgs e)
    {
        DisplayAlert("Selection Changed", "SelectedIndex: " +
            autoComplete.SelectedIndex, "OK");
    }
}

```

### SelectedIndices

You can get or set the indices of the selected items using the [SelectedIndices](#) property. It can be applicable when [MultiSelectMode](#) is in either Token or Delimiter.

#### *How to set the indices of items*

[SelectedIndices](#) property holds the Indices of selected items in suggestion list.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

```

```

xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
MultiSelectMode="Token"
SelectedIndices="{Binding SelectedIndices}">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

**C#**

```

using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        private object selectedIndices;
        public object SelectedIndices
        {
            get { return selectedIndices; }
            set { selectedIndices = value; }
        }
        public MainPage()
        {
            InitializeComponent();
            SelectedIndices = new List<int>() { 2, 4, 7 };
            this.BindingContext = this;
        }
    }
}

```

*Retrieving the indices of selected item*

When an item is selected from suggestion list, its index can be retrieved using the [SelectedIndices](#) property.

The following code snippet demonstrates the way to retrieve [SelectedIndices](#) and display in the ListView.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
MultiSelectMode="Token">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
<ListView
x:Name="MainListView"
RowHeight="30"
ItemsSource="{Binding Source={x:Reference
autoComplete}, Path=SelectedIndices}">
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout Orientation="Horizontal">
<Label Text="SelectedIndex:"/>
<Label Text="{Binding}" />
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
```

```
</StackLayout>
</ContentPage>
```

## C#

```
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage, INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void NotifyPropertyChanged([CallerMemberName] string
        propertyName = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
        private object selectedIndices;
        public object SelectedIndices
        {
            get { return selectedIndices; }
            set
            {
                selectedIndices = value;
                NotifyPropertyChanged("SelectedIndices");
            }
        }
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = this;
        }
    }
}
```

## SelectedItem

The **SelectedItem** property is used to select the particular item from the suggestion list. You can either get or set the SelectedItem.

### How to set the SelectedItem

The following code snippet demonstrates the way to set **SelectedItem**.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
te.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
```

```

<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
SelectedItem="Angola">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                SelectedItem = "Angola",
                MultiSelectMode = MultiSelectMode.None,
                AutoCompleteSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",
                    "Argentina",
                    "Anguilla",

```

```

"Albania",
"Algeria",
"Andorra",
"Angola"
}
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```

### Retrieving the selected item

When an item is selected from suggestion list, it can be retrieved using the [SelectedItem](#) property.

The following code snippet demonstrates the way to retrieve [SelectedItem](#) and display it in an alert.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
MultiSelectMode="None"
SelectionChanged="autoComplete_SelectionChanged">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

### C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        SfAutoComplete autoComplete;
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                MultiSelectMode = MultiSelectMode.None,
                AutoCompleteSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",
                    "Argentina",
                    "Anguilla",
                    "Albania",
                    "Algeria",
                    "Andorra",
                    "Angola"
                }
            };
            autoComplete.SelectionChanged += autoComplete_SelectionChanged;
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
        private void autoComplete_SelectionChanged(object sender,
            Syncfusion.SfAutoComplete.XForms.SelectionChangedEventArgs e)
        {
            DisplayAlert("Selection Changed", "SelectedItem: " +
                autoComplete.SelectedItem, "OK");
        }
    }
}

```

## Dealing with Suggestion Box

Suggestion box is the drop-down list box, which displays the filtered suggestions inside a popup. This section explains the properties that deals with drop-down list in the SfAutoComplete control.

### Suggestion box placement mode

Suggestion box can be placed either at the top or bottom using the `SuggestionBoxPlacement` property. By default, it is placed at the bottom.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Center"
HorizontalOptions="Center"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
SuggestionBoxPlacement="Top">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

## C#

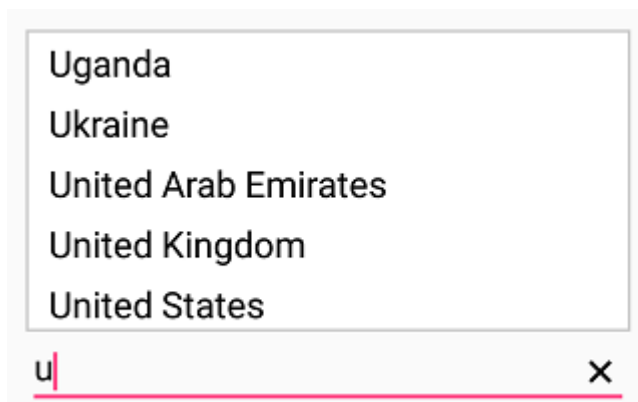
```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Center,
                HorizontalOptions = LayoutOptions.Center,
                Padding = 30
            };
        }
    }
}
```



```

SfAutoComplete autoComplete = new SfAutoComplete()
{
    HeightRequest = 40,
    SuggestionBoxPlacement = SuggestionBoxPlacement.Top,
    AutoCompleteSource = new List<string>()
    {
        "India",
        "Uganda",
        "Ukraine",
        "Canada",
        "United Arab Emirates",
        "France",
        "United Kingdom",
        "China",
        "United States",
        "Japan",
        "Angola"
    }
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```



### Maximum suggestion box height

The maximum height of the suggestion box in the SfAutocomplete control can be varied using the `MaximumDropDownHeight` property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"

```

```

HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
MaximumDropDownHeight="100">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

```

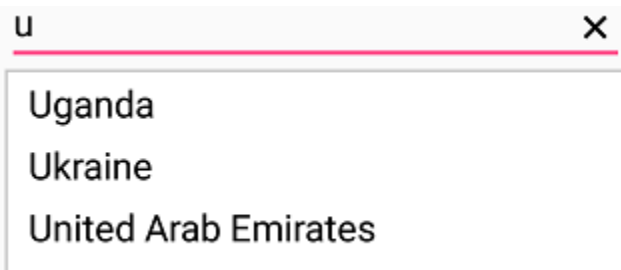
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                MaximumDropDownHeight = 100,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "France",
                    "United Kingdom",

```

```

"China",
"United States",
"Japan",
"Angola"
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```



Opening suggestion box on focus

Suggestion box can be shown whenever the control receives focus using the `ShowSuggestionsOnFocus` property. At this time, suggestion list is the complete list of data source.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
ShowSuggestionsOnFocus="True">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Great Britain</x:String>
<x:String>Canada</x:String>
<x:String>France</x:String>
<x:String>China</x:String>
<x:String>Japan</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>

```

```
</ContentPage>
```

## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                ShowSuggestionsOnFocus = true,
                AutoCompleteSource = new List<string>()
                {
                    "Great Britain",
                    "Canada",
                    "France",
                    "China",
                    "Japan"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```



### Delay opening suggestion box

The `PopupDelay` property is used to delay the suggestion box opening process. It gets milliseconds as input in integer data type.

In this example, a time duration of 3 seconds is set to popup delay.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
PopupDelay="3000">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

#### C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

```

StackLayout stackLayout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = 30
};
SfAutoComplete autoComplete = new SfAutoComplete()
{
    HeightRequest = 40,
    PopupDelay = 3000,
    AutoCompleteSource = new List<string>()
    {
        "India",
        "Uganda",
        "Ukraine",
        "Canada",
        "United Arab Emirates",
        "France",
        "United Kingdom",
        "China",
        "United States",
        "Japan",
        "Angola"
    }
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```

### Delay before searching algorithm starts

The **SearchDelay** property is used to delay the searching algorithm process. It gets milliseconds as input in integer data type.

In this example, a time duration of 3 seconds is set to search delay.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
SearchDelay="3000">

```

```

<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

**C#**

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout()
{
VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = 30
};
SfAutoComplete autoComplete = new SfAutoComplete()
{
HeightRequest = 40,
SearchDelay = 3000,
AutoCompleteSource = new List<string>()
{
"India",
"Uganda",
"Ukraine",
"Canada",
"United Arab Emirates",
"France",
"United Kingdom",
"China",
"United States",
"Japan",
"Angola"
}
};
}
}

```

```

stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```

### Avoid opening suggestion box

APIs are available to avoid pop-ups and retrieve filtered suggestion items that help you arrange lists or items control.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
SuggestionBoxPlacement="None">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>France</x:String>
<x:String>United Kingdom</x:String>
<x:String>China</x:String>
<x:String>United States</x:String>
<x:String>Japan</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

### C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{

```



```

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        StackLayout stackLayout = new StackLayout()
        {
            VerticalOptions = LayoutOptions.Start,
            HorizontalOptions = LayoutOptions.Start,
            Padding = 30
        };
        SfAutoComplete autoComplete = new SfAutoComplete()
        {
            HeightRequest = 40,
            SuggestionBoxPlacement = SuggestionBoxPlacement.None,
            AutoCompleteSource = new List<string>()
            {
                "India",
                "Uganda",
                "Ukraine",
                "Canada",
                "United Arab Emirates",
                "France",
                "United Kingdom",
                "China",
                "United States",
                "Japan",
                "Angola"
            }
        };
        stackLayout.Children.Add(autoComplete);
        this.Content = stackLayout;
    }
}

```

## 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 by `[MinimumPrefixCharacters]` property and its default value is 1.

## XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">

```

```
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
MinimumPrefixCharacters="3">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                MinimumPrefixCharacters = 3,
                AutoCompleteSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",
                    "Argentina",
                    "Anguilla",
                    "Albania",
                    "Algeria",
                    "Andorra",
                    "Angola"
                }
            };
        }
    }
}
```

```

    }
    };
    stackLayout.Children.Add(autoComplete);
    this.Content = stackLayout;
}
}
}

```

## Watermark

Watermark provides a short note about the type of input to enter in the editor control. Watermarks are visible only if the text is empty. Also it will reappear if the text is cleared.

The following example, explains the usability of watermark which hints user to start with the character "U".

## XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
Watermark="Enter 'U' to filter suggestions">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

```

using Syncfusion.SfAutoComplete.XForms;

```

```

using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                Watermark = "Enter 'A' to filter suggestions",
                AutoCompleteSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",
                    "Argentina",
                    "Anguilla",
                    "Albania",
                    "Algeria",
                    "Andorra",
                    "Angola"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}

```

## Enter 'U' to filter suggestions

### Changing Watermark Text Color

Text color of watermark can be customized using `WatermarkColor` property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"

```

```

x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
Watermark="Enter some text"
WatermarkColor="#1976d2">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                Watermark = "Enter some text",
                WatermarkColor = Color.FromHex("1976d2"),
                AutoCompleteSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",

```

```

"Argentina",
"Anguilla",
"Albania",
"Algeria",
"Andorra",
"Angola"
}
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```

## Enter some text

Focus the control

The autocomplete sets the user to focus the autocomplete textbox initially after the control gets rendered using `IsFocused` property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete
x:Name="autoComplete"
HeightRequest="40"
Watermark="Enter 'A' to filter suggestions"
WatermarkColor="#1976d2"
IsFocused="True">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>

```

```
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                Watermark = "Enter 'A' to filter suggestions",
                WatermarkColor = Color.FromHex("#1976d2"),
                IsFocused = true,
                AutoCompleteSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",
                    "Argentina",
                    "Anguilla",
                    "Albania",
                    "Algeria",
                    "Andorra",
                    "Angola"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```

## Customizing AutoComplete

AutoComplete provides user friendly customizing options for both entry part and drop down part. In this section, customizing entire AutoComplete control is explained.

### Customizing the Entry

[TextColor](#), [TextSize](#), [FontAttributes](#), [FontFamily](#) and [BorderColor](#) are the properties used to customize the foreground color, font size, font attribute, font family and border color of the entry part.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout
VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="45"
Text="Sample text"
TextColor="#1976d2"
TextSize="20"
BorderColor="#1976d2"/>
</StackLayout>
</ContentPage>
```

### C#

```
using Syncfusion.SfAutoComplete.XForms;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 45,
                Text = "Sample text",
                TextColor = Color.FromHex("1976d2"),
                TextSize = 20,
                BorderColor = Color.FromHex("1976d2")
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```



```
}
}
```

## Sample text



### Custom Template for Suggestion Items

[ItemTemplate](#) property helps to decorate suggestion items with custom templates. The following code explains the steps to add an image to the suggestion list item.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<ContentPage.BindingContext>
<local:PersonViewModel/>
</ContentPage.BindingContext>
<StackLayout
VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete HeightRequest="40"
DisplayMemberPath="Name"
DataSource="{Binding PersonCollection}">
<autocomplete:SfAutoComplete.ItemTemplate>
<DataTemplate>
<StackLayout Orientation="Horizontal"
Padding="2,0,0,0">
<Image Source="User.png"
WidthRequest="12"/>
<Label Text="{Binding Name}"
VerticalOptions="Center"/>
</StackLayout>
</DataTemplate>
</autocomplete:SfAutoComplete.ItemTemplate>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

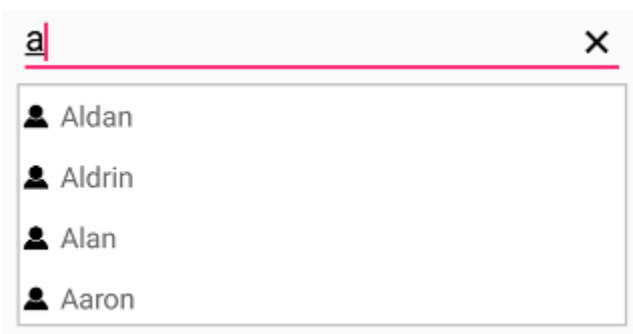
#### C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace AutocompleteSample
{
```

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
}

public class Person
{
    private int age;
    public int Age
    {
        get { return age; }
        set { age = value; }
    }
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}

public class PersonViewModel
{
    private ObservableCollection<Person> personCollection;
    public ObservableCollection<Person> PersonCollection
    {
        get { return personCollection; }
        set { personCollection = value; }
    }
    public PersonViewModel()
    {
        personCollection = new ObservableCollection<Person>();
        personCollection.Add(new Person() { Age = 21, Name = "Aldan" });
        personCollection.Add(new Person() { Age = 25, Name = "Clara" });
        personCollection.Add(new Person() { Age = 23, Name = "Aldrin" });
        personCollection.Add(new Person() { Age = 25, Name = "Mark" });
        personCollection.Add(new Person() { Age = 25, Name = "Lucas" });
        personCollection.Add(new Person() { Age = 24, Name = "Alan" });
        personCollection.Add(new Person() { Age = 25, Name = "James" });
        personCollection.Add(new Person() { Age = 22, Name = "Aaron" });
    }
}
}
```



## Customizing the Suggestion Box

### *Changing suggestion item height*

[DropDownItemHeight](#) property is used to modify the height of suggestion items in drop down list. The code example is given below:

#### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
  <StackLayout
    VerticalOptions="Start"
    HorizontalOptions="Start"
    Padding="30">
    <autocomplete:SfAutoComplete x:Name="autoComplete"
      HeightRequest="40"
      DropDownItemHeight="45">
      <autocomplete:SfAutoComplete.AutoCompleteSource>
      <ListCollection:List x:TypeArguments="x:String">
      <x:String>India</x:String>
      <x:String>Uganda</x:String>
      <x:String>Ukraine</x:String>
      <x:String>Canada</x:String>
      <x:String>United Arab Emirates</x:String>
      <x:String>United Kingdom</x:String>
      </ListCollection:List>
      </autocomplete:SfAutoComplete.AutoCompleteSource>
    </autocomplete:SfAutoComplete>
  </StackLayout>
</ContentPage>
```

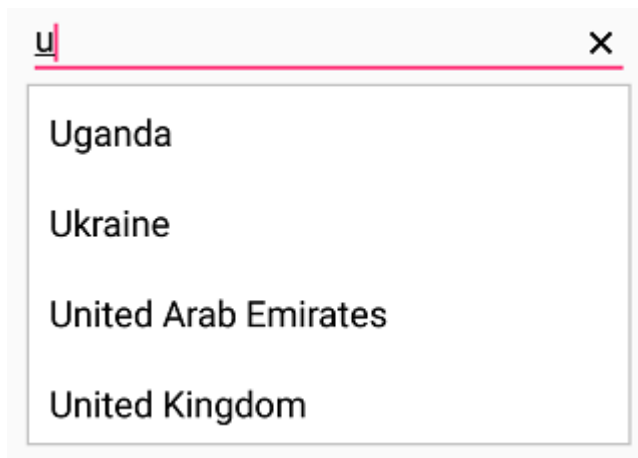
#### **C#**

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
        }
    }
}
```

```

SfAutoComplete autoComplete = new SfAutoComplete()
{
    HeightRequest = 40,
    DropDownItemHeight = 45,
    AutoCompleteSource = new List<string>()
    {
        "India",
        "Uganda",
        "Ukraine",
        "Canada",
        "United Arab Emirates",
        "United Kingdom"
    }
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```



#### Changing suggestion box corner radius

The [DropDownCornerRadius](#) property is used to modify the corner radius of suggestion box. The following code example demonstrates how to change the suggestion box corner radius.

#### XML

```

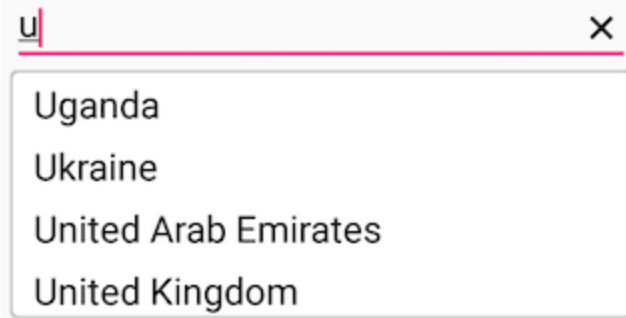
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout
VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">

```

```
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
DropDownCornerRadius="3">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>United Kingdom</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout()
{
VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
SfAutoComplete autoComplete = new SfAutoComplete()
{
HeightRequest = 40,
DropDownCornerRadius = 3,
AutoCompleteSource = new List<string>()
{
"India",
"Uganda",
"Ukraine",
"Canada",
"United Arab Emirates",
"United Kingdom"
}
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}
```



#### Changing suggestion box background color

The [DropDownBackgroundColor](#) property is used to modify the background color of suggestion box. The following code example demonstrates how to change the suggestion box background color.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
  <StackLayout
    VerticalOptions="Start"
    HorizontalOptions="Start"
    Padding="30">
    <autocomplete:SfAutoComplete x:Name="autoComplete"
      HeightRequest="40"
      DropDownBackgroundColor="Red">
      <autocomplete:SfAutoComplete.AutoCompleteSource>
      <ListCollection:List x:TypeArguments="x:String">
      <x:String>India</x:String>
      <x:String>Uganda</x:String>
      <x:String>Ukraine</x:String>
      <x:String>Canada</x:String>
      <x:String>United Arab Emirates</x:String>
      <x:String>United Kingdom</x:String>
      </ListCollection:List>
      </autocomplete:SfAutoComplete.AutoCompleteSource>
      </autocomplete:SfAutoComplete>
    </StackLayout>
  </ContentPage>
```

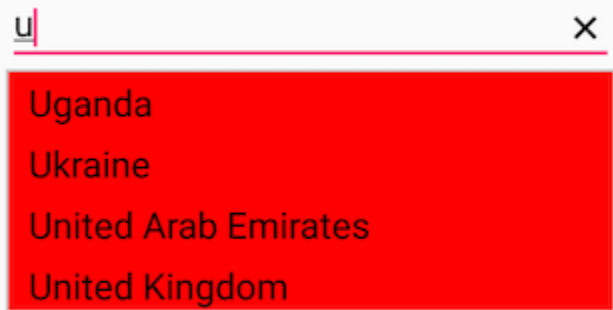
#### C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
```

```

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        StackLayout stackLayout = new StackLayout()
        {
            VerticalOptions = LayoutOptions.Start,
            HorizontalOptions = LayoutOptions.Start,
            Padding = new Thickness(30)
        };
        SfAutoComplete autoComplete = new SfAutoComplete()
        {
            HeightRequest = 40,
            DropDownBackgroundColor = Color.Red,
            AutoCompleteSource = new List<string>()
            {
                "India",
                "Uganda",
                "Ukraine",
                "Canada",
                "United Arab Emirates",
                "United Kingdom"
            }
        };
        stackLayout.Children.Add(autoComplete);
        this.Content = stackLayout;
    }
}

```



#### *Changing the border color of suggestion box*

The `DropDownBorderColor` property is used to change the border color of suggestion box. The following code example demonstrates how to change the border color of suggestion box.

#### **XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"

```

```

xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
<StackLayout>
<autoComplete:SfAutoComplete HeightRequest="40"
DropDownBorderColor="Blue">
<autoComplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>1920 x 1080</x:String>
<x:String>1680 x 1050</x:String>
<x:String>1600 x 900</x:String>
<x:String>1440 x 900</x:String>
<x:String>1400 x 1050</x:String>
<x:String>1366 x 768</x:String>
<x:String>1360 x 768</x:String>
<x:String>1280 x 1024</x:String>
<x:String>1280 x 960</x:String>
<x:String>1280 x 720</x:String>
<x:String>854 x 480</x:String>
<x:String>800 x 480</x:String>
<x:String>480 x 640</x:String>
<x:String>480 x 320</x:String>
<x:String>432 x 240</x:String>
<x:String>360 x 640</x:String>
<x:String>320 x 240</x:String>
</ListCollection:List>
</autoComplete:SfAutoComplete.AutoCompleteSource>
</autoComplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout layout = new StackLayout()
{
VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
List<String> resolutionList = new List<String>();
resolutionList.Add("1920 x 1080");
resolutionList.Add("1680 x 1050");
resolutionList.Add("1600 x 900");
resolutionList.Add("1440 x 900");
resolutionList.Add("1400 x 1050");
}
}

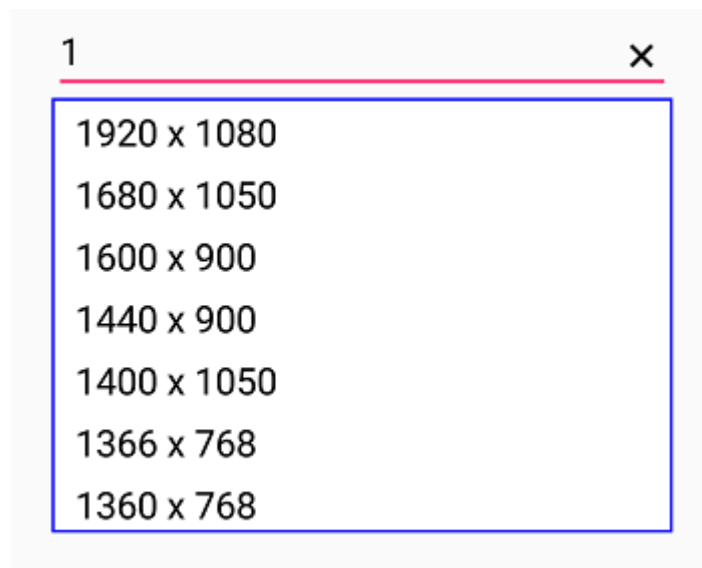
```



```

resolutionList.Add("1366 x 768");
resolutionList.Add("1360 x 768");
resolutionList.Add("1280 x 1024");
resolutionList.Add("1280 x 960");
resolutionList.Add("1280 x 720");
resolutionList.Add("854 x 480");
resolutionList.Add("800 x 480");
resolutionList.Add("480 x 640");
resolutionList.Add("480 x 320");
resolutionList.Add("432 x 240");
resolutionList.Add("360 x 640");
resolutionList.Add("320 x 240");
SfAutoComplete autoComplete = new SfAutoComplete();
autoComplete.HeightRequest = 40;
autoComplete.AutoCompleteSource = resolutionList;
autoComplete.DropDownBorderColor = Color.Blue;
layout.Children.Add(autoComplete);
Content = layout;
}
}
}

```



#### Customizing suggestion items

Suggestion box items can be customized using [DropDownItemFontAttributes](#), [DropDownItemFontFamily](#), [DropDownTextSize](#) and [DropDownTextColor](#) properties.

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"

```

```

x:Class="AutocompleteSample.MainPage">
<StackLayout
VerticalOptions="Start"
HorizontalOptions="Start"
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
DropDownTextColor="#1976d2"
DropDownTextSize="12">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>United Kingdom</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>

```

## C#

```

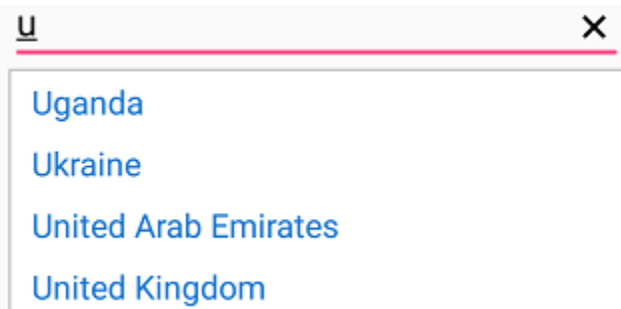
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                DropDownTextColor = Color.FromHex("#1976d2"),
                DropDownTextSize = 16,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "United Kingdom"
                }
            };
        }
    }
}

```

```

stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
}

```



#### Show clear button

The autocomplete provided the user to show or hide the clear button using [ShowClearButton](#) property.

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
  <StackLayout
    VerticalOptions="Start"
    HorizontalOptions="Start"
    Padding="30">
    <autocomplete:SfAutoComplete x:Name="autoComplete"
      HeightRequest="40"
      ShowClearButton="True">
      <autocomplete:SfAutoComplete.AutoCompleteSource>
      <ListCollection:List x:TypeArguments="x:String">
      <x:String>India</x:String>
      <x:String>Uganda</x:String>
      <x:String>Ukraine</x:String>
      <x:String>Canada</x:String>
      <x:String>United Arab Emirates</x:String>
      <x:String>United Kingdom</x:String>
      </ListCollection:List>
      </autocomplete:SfAutoComplete.AutoCompleteSource>
      </autocomplete:SfAutoComplete>
    </StackLayout>
  </ContentPage>

```

#### C#

```

using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                ShowClearButton = true,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "United Kingdom"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}

```

### Customizing clear button

The user can customize the clear button color in the autocomplete using [ClearButtonColor](#) Property.

---

**Note:** ClearButtonColor property is available only on iOS and Android platform.

---

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
    <StackLayout
        VerticalOptions="Start"
        HorizontalOptions="Start"

```

```
Padding="30">
<autocomplete:SfAutoComplete x:Name="autoComplete"
HeightRequest="40"
ClearButtonColor="Red">
<autocomplete:SfAutoComplete.AutoCompleteSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>India</x:String>
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>Canada</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>United Kingdom</x:String>
</ListCollection:List>
</autocomplete:SfAutoComplete.AutoCompleteSource>
</autocomplete:SfAutoComplete>
</StackLayout>
</ContentPage>
```

## C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
            {
                HeightRequest = 40,
                ClearButtonColor = Color.Red,
                AutoCompleteSource = new List<string>()
                {
                    "India",
                    "Uganda",
                    "Ukraine",
                    "Canada",
                    "United Arab Emirates",
                    "United Kingdom"
                }
            };
            stackLayout.Children.Add(autoComplete);
            this.Content = stackLayout;
        }
    }
}
```

### Changing border visibility

The [ShowBorder](#) property is used to modify the visibility of border. The following code example demonstrates how to change the border visibility.

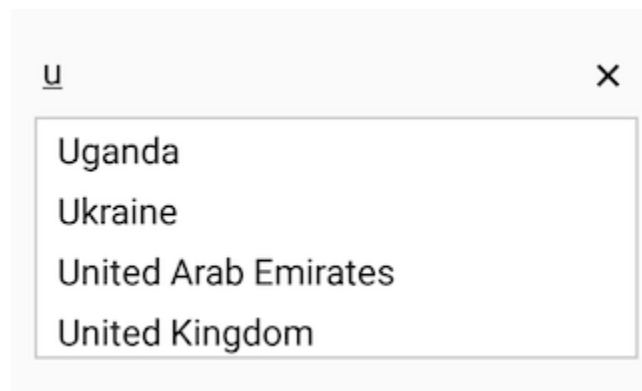
#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:autocomplete="clr-
namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
e.XForms"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:local="clr-namespace:AutocompleteSample"
x:Class="AutocompleteSample.MainPage">
  <StackLayout
    VerticalOptions="Start"
    HorizontalOptions="Start"
    Padding="30">
    <autocomplete:SfAutoComplete x:Name="autoComplete"
      HeightRequest="40"
      ShowBorder="False">
      <autocomplete:SfAutoComplete.AutoCompleteSource>
        <ListCollection:List x:TypeArguments="x:String">
          <x:String>India</x:String>
          <x:String>Uganda</x:String>
          <x:String>Ukraine</x:String>
          <x:String>Canada</x:String>
          <x:String>United Arab Emirates</x:String>
          <x:String>United Kingdom</x:String>
        </ListCollection:List>
      </autocomplete:SfAutoComplete.AutoCompleteSource>
    </autocomplete:SfAutoComplete>
  </StackLayout>
</ContentPage>
```

#### C#

```
using Syncfusion.SfAutoComplete.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace AutocompleteSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = new Thickness(30)
            };
            SfAutoComplete autoComplete = new SfAutoComplete()
```

```
{
    HeightRequest = 40,
    ShowBorder = false,
    AutoCompleteSource = new List<string>()
    {
        "India",
        "Uganda",
        "Ukraine",
        "Canada",
        "United Arab Emirates",
        "United Kingdom"
    }
};
stackLayout.Children.Add(autoComplete);
this.Content = stackLayout;
}
}
```



### How to perform an operation when selecting an item from drop-down list?

You can perform an operation when selecting an item among the filtered suggestions using the **SelectionChanged** event. The SelectionChanged event returns the following arguments:

Members	Description
AddedItems	Shows recently added item in AutoComplete.
RemovedItems	Shows recently removed items in AutoComplete.
Value	Holds all selected items in AutoComplete.

### XML

```
[XAML]
<?xml version="1.0" encoding="utf-8"?>
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:autocomplete="clr-
        namespace:Syncfusion.SfAutoComplete.XForms;assembly=Syncfusion.SfAutoComple
        e.XForms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
xmlns:local="clr-namespace:Events"
x:Class="Events.MainPage">
<ContentPage.BindingContext>
<local:EmployeeViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout>
<autocomplete:SfAutoComplete
DisplayMemberPath="Name"
DataSource="{Binding EmployeeCollection}"
SelectionChanged="Handle_SelectionChanged"/>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

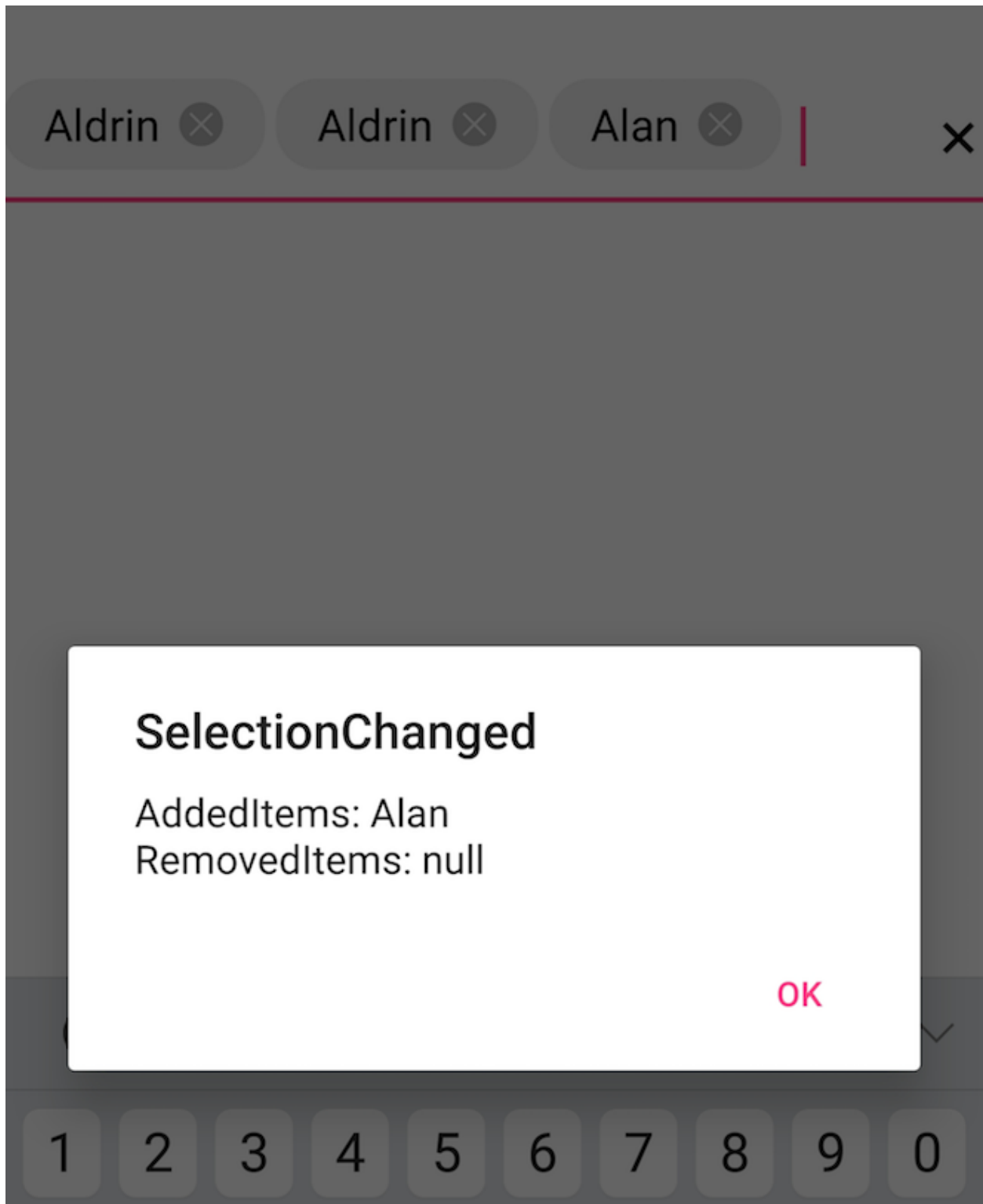
## C#

```
[C#]
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace Events
{
    public class Employee
    {
        private int id;
        private string name;
        public int ID
        {
            get
            {
                return this.id;
            }
            set
            {
                this.id = value;
            }
        }
        public string Name
        {
            get
            {
                return this.name;
            }
            set
            {
                this.name = value;
            }
        }
    }
    public class EmployeeViewModel
```

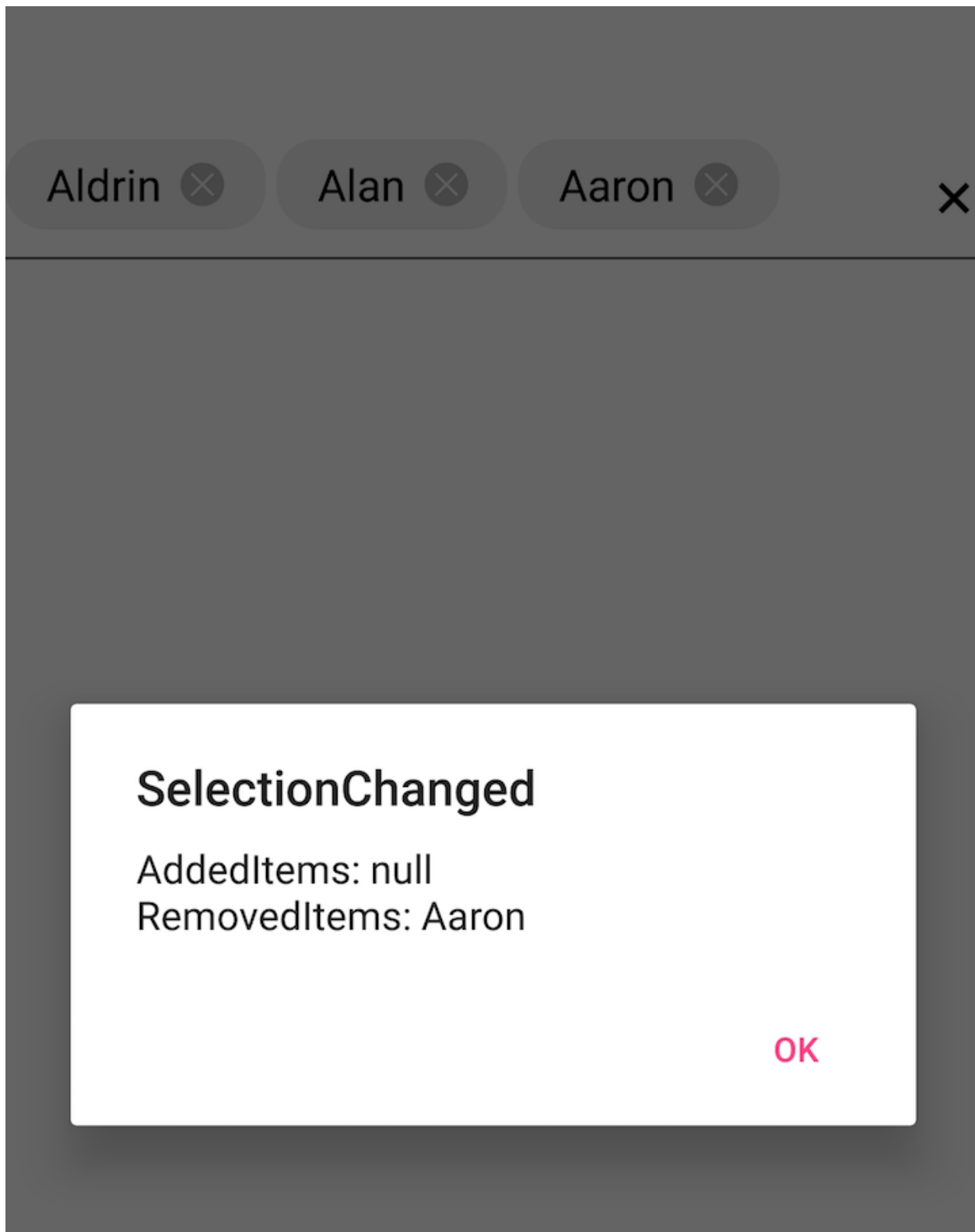


```
{
private ObservableCollection<Employee> employeeCollection;
public ObservableCollection<Employee> EmployeeCollection
{
get
{
return this.employeeCollection;
}
set
{
this.employeeCollection = value;
}
}
public EmployeeViewModel()
{
employeeCollection = new ObservableCollection<Employee>();
employeeCollection.Add(new Employee() { ID = 1, Name = "Eric" });
employeeCollection.Add(new Employee() { ID = 2, Name = "James" });
employeeCollection.Add(new Employee() { ID = 3, Name = "Jacob" });
employeeCollection.Add(new Employee() { ID = 4, Name = "Lucas" });
employeeCollection.Add(new Employee() { ID = 5, Name = "Mark" });
employeeCollection.Add(new Employee() { ID = 6, Name = "Aldan" });
employeeCollection.Add(new Employee() { ID = 7, Name = "Aldrin" });
employeeCollection.Add(new Employee() { ID = 8, Name = "Alan" });
employeeCollection.Add(new Employee() { ID = 9, Name = "Aaron" });
}
}
public partial class MainPage : ContentPage
{
void Handle_SelectionChanged(object sender,
Syncfusion.SfAutoComplete.XForms.SelectionChangedEventArgs e)
{
var addedEmployee = e.AddedItems as Employee;
string addedItems = addedEmployee != null ? addedEmployee.Name : "null";
var removedEmployee = e.RemovedItems as Employee;
string removedItems = removedEmployee != null ? removedEmployee.Name :
"null";
DisplayAlert("SelectionChanged", "AddedItems: " + addedItems + "\n" +
"RemovedItems: " + removedItems, "Ok");
}
public MainPage()
{
InitializeComponent();
}
}
}
```

The following screenshot illustrates the result of adding the item.



The following screenshot illustrates the result of removing the item.



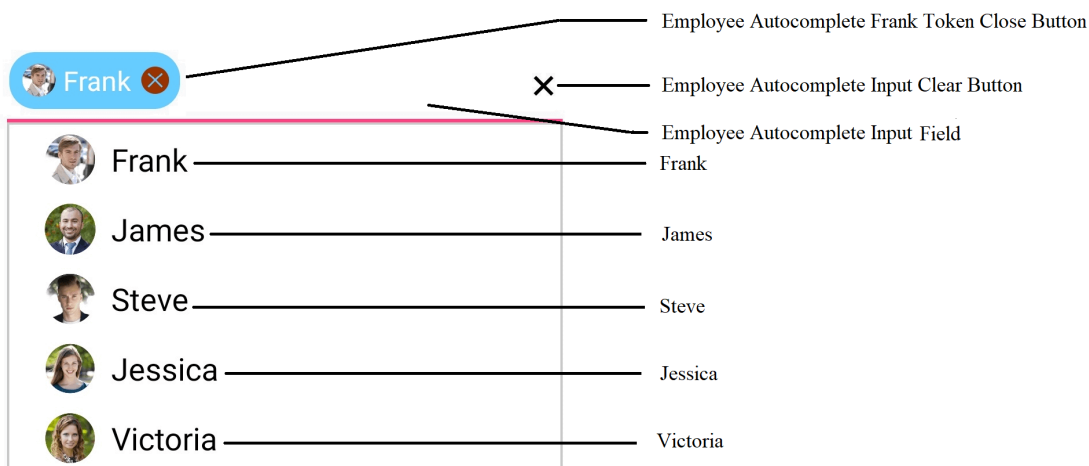
#### AutomationId

The SfAutoComplete control has built-in `AutomationId` for inner elements. The `AutomationId` API allows the automation framework to find and interact with the inner elements of the SfAutoComplete

control. To keep unique AutomationId, these inner elements' AutomationIds are updated based on the control's AutomationId.

For example, If you set SfAutoComplete's AutomationId as "Employee Autocomplete", then the Automation framework will interact with the Token Close Button as "Employee Autocomplete Frank Token Close Button".

The following screenshot illustrates the AutomationIds of inner elements. The Automation framework will interact with the dropdown for scrolling the items as "Employee Autocomplete Dropdown". You can also interact with the elements inside the HeaderView and FooterView with the element's AutomationId. The Automation framework will not interact with the Input Clear Button when the MultiSelectMode is None and Delimiter mode.



## SfAvatarView

### Overview

The SfAvatarView control for Xamarin.Forms provides a graphical representation of user image that allows you to customize the view by adding image, background color, icon, text, etc.



## Key features

- Supports for adding image and initials.
- Customizes the height, width, BorderColor, BackgroundColor, and CornerRadius of the view.
- **AvatarCharacter**: Adds the default vector images.
- **GroupView**: Supports to add maximum three custom images or initials in a single view.
- Supports different types of visual styles.
- Supports **BadgeView** and **GradientBrush**.

## Getting Started

This section explains the steps required to work with the SfAvatarView control for Xamarin.Forms.

### Adding SfAvatarView reference

You can add SfAvatarView reference using one of the following methods:

#### Method 1: Adding SfAvatarView reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfAvatarView to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Core](https://www.syncfusion.com/nuget-packages), and then install it.

![Xamarin Forms SfAvatarView Nuget reference](images/Adding SfAvatarView reference.png)

**Note:** Install the same version of SfAvatarView NuGet in all the projects.

### Method 2: Adding SfAvatarView reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfAvatarView control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfAvatarView assemblies manually from the installed location

If you prefer to manually reference the assemblies instead of referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching an application on each platform with SfAvatarView

To use the SfAvatarView control inside an application, each platform application must initialize the SfAvatarView renderer. This initialization step varies from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, the following steps are not needed.

#### Android and UWP

The Android and UWP launch the SfAvatarView without any initialization, and it is enough to only initialize the Xamarin.Forms Framework to launch the application.

#### iOS

To launch the SfAvatarView in iOS, call the `SfAvatarView.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework has been initialized and before the `LoadApplication` is called as demonstrated in the following code example.

**C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
...
global::Xamarin.Forms.Forms.Init ();
Syncfusion.XForms.iOS.Core.SfAvatarViewRenderer.Init();
LoadApplication (new App ());
...
}
```

*Release mode issue in UWP platform*

The known Framework issue in UWP platform is that the custom controls will not be rendered when deploying an application in **Release Mode**. It can be resolved by initializing the SfBorder assemblies in the **App.xaml.cs** file in the UWP project as demonstrated in the following code example.

**C#**

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
...
rootFrame.NavigationFailed += OnNavigationFailed;
// You will need to add `using System.Reflection;`
List<Assembly> assembliesToInclude = new List<Assembly>();
//Now, add all the assemblies your app uses
assembliesToInclude.Add(typeof(SfBorderRenderer).GetTypeInfo().Assembly);
// Replaces Xamarin.Forms.Forms.Init(e);
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
...
}
```

*Creating an SfAvatarView control*

The **SfAvatarView** control is configured entirely in C# or in XAML. The following steps explain how to create an **SfAvatarView** control and configure its elements.

*Adding namespace for referred assemblies***XML**

```
xmlns:sfavatar="clr-
namespace:Syncfusion.XForms.AvatarView;assembly=Syncfusion.Core.XForms"
```

**C#**

```
using Syncfusion.XForms.AvatarView;
```

*Adding the SfAvatarView control as the content of ContentPage*

You can add a custom image for displaying in SfAvatarView using the **ImageSource** property.

**XML**

```
<ContentPage.Content>
<Grid>
```

```
<sfavatar:SfAvatarView ContentType="Custom"
ImageSource="alex.png"
VerticalOptions="Center"
HorizontalOptions="Center"
HeightRequest="50"
CornerRadius="25"
WidthRequest="50" />
</Grid>
</ContentPage.Content>
```

## C#

```
using System;
using Syncfusion.XForms.AvatarView;
using Xamarin.Forms;
namespace AvatarViewGettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            //main grid
            Grid mainGrid = new Grid();
            // Create an SfAvatarView control.
            SfAvatarView avatarview = new SfAvatarView();
            avatarview.VerticalOptions = LayoutOptions.Center;
            avatarview.HorizontalOptions = LayoutOptions.Center;
            avatarview.BackgroundColor = Color.FromHex("#ffb6c1");
            avatarview.ContentType = ContentType.Custom;
            avatarview.ImageSource = "alex.png";
            avatarview.WidthRequest = 50;
            avatarview.HeightRequest = 50;
            avatarview.CornerRadius = 25;
            mainGrid.Children.Add(avatarview);
            this.Content = mainGrid;
        }
    }
}
```



The Getting Started sample is available in this following link: [Getting Started](#).



## Avatar Types

The **SfAvatarView** control provides the following five different ways to display the view:

- **Default** - Adds the default image when initializing without any other source such as image and group.
- **AvatarName** - Set the initial value in SfAvatarView.
- **Custom** - Adds the user custom image in SfAvatarView.
- **AvatarCharacter** - Sets the default image in SfAvatarView.
- **Group** - Adds maximum three images or initials in a single **SfAvatarView**.

### Default

Automatic type avatar view is used for displaying the default vector image when initializing without the initials, custom, or group view types.

### XML

```
<sfavatar:SfAvatarView ContentType="Default"
VerticalOptions="Center"
HorizontalOptions="Center"
WidthRequest="50"
HeightRequest="50"
CornerRadius="25">
</sfavatar:SfAvatarView>
```

### C#

```
Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.WidthRequest = 50;
avatarview.HeightRequest = 50;
avatarview.CornerRadius = 25;
avatarview.ContentType = ContentType.Default;
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
```



## Initials

When using the `SfAvatarType` as initials, you need to set the initial character using the following properties:

- `InitialsType` - Defines the type of characters to be displayed.
- `AvatarName` - Gets or sets the value for the initials type, which displays the text in the avatar view.
- `InitialsColor` - Gets or sets the color of the initial color value that defines color for the initial string.

## InitialsType

The `InitialsType` contains the following two types:

- `SingleCharacter`
- `DoubleCharacter`

You must set the `AvatarName` string property for displaying the initial value in the `AvatarView`.

## SingleCharacter

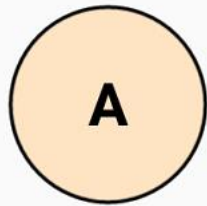
The `SingleCharacter` is used for displaying the first character in the string you have set in the `Initials` property.

## XML

```
<sfavatar:SfAvatarView ContentType="Initials"
HorizontalOptions="Center"
VerticalOptions="Center"
InitialsType="SingleCharacter"
AvatarName="Alex"
InitialsColor="Black"
WidthRequest="50"
FontAttributes="Bold"
HeightRequest="50"
CornerRadius="25"
BackgroundColor="Bisque">
</sfavatar:SfAvatarView>
```

## C#

```
Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.BackgroundColor = Color.Bisque;
avatarview.WidthRequest = 50;
avatarview.HeightRequest = 50;
avatarview.CornerRadius = 25;
avatarview.ContentType = ContentType.Initials;
avatarview.AvatarName = "A";
avatarview.InitialsType = InitialsType.SingleCharacter;
avatarview.InitialsColor = Color.Black;
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
```



### DoubleCharacter

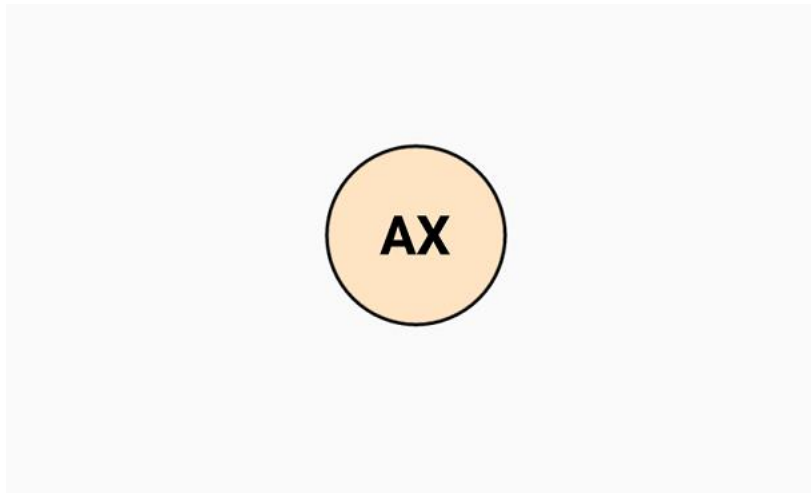
The **DoubleCharacter** is used for displaying a two-characters text you have set in the **Initials** property. If the initials contain one word, it shows the first and last letters of the single string. If it contains two or more words, it displays the first letter of the first string and last letter of the second or last string.

### XML

```
<sfavatar:SfAvatarView ContentType="Initials"
InitialsType="DoubleCharacter"
AvatarName="Alex"
InitialsColor="Black"
WidthRequest="50"
FontAttributes="Bold"
HeightRequest="50"
CornerRadius="25"
BackgroundColor="Bisque">
</sfavatar:SfAvatarView>
```

### C#

```
Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.BackgroundColor = Color.Bisque;
avatarview.WidthRequest = 50;
avatarview.HeightRequest = 50;
avatarview.CornerRadius = 25;
avatarview.ContentType = ContentType.Initials;
avatarview.AvatarName = "AA";
avatarview.InitialsType = InitialsType.DoubleCharacter;
avatarview.InitialsColor = Color.Black;
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
```



#### Custom image

You can add a custom user image by setting the `ImageSource` property. Refer to this [documentation](#).

#### Character avatars

You can set the default vector images that already present in avatar view by setting the `AvatarCharacter` property.

#### XML

```
<sfavatar:SfAvatarView VerticalOptions="Center"
HorizontalOptions="Center"
ContentType="AvatarCharacter"
AvatarCharacter="Avatar10"
WidthRequest="50"
HeightRequest="50"
CornerRadius="25">
</sfavatar:SfAvatarView>
```

#### C#

```
Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.WidthRequest = 50;
avatarview.HeightRequest = 50;
avatarview.CornerRadius = 35;
avatarview.ContentType = ContentType.AvatarCharacter;
avatarview.AvatarCharacter = AvatarCharacter.Avatar10;
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
```



### GroupView

You can add maximum three images or initials in the same view using a GroupView type.

Set the `InitialsMemberPath` for displaying the initials in the group view. For image, set the `ImageSourcePath`. The following code sample demonstrates how to add images using the `GroupView` property.

### C#

```
public class Employee
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    private string imagesource;
    public string ImageSource
    {
        get { return imagesource; }
        set { imagesource = value; }
    }
    private Color colors;
    public Color Colors
    {
        get { return colors; }
        set { colors = value; }
    }
}

public class EmployeeViewMdoel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    private void OnPropertyRaised(string propertyname)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyname));
        }
    }
    private ObservableCollection<Employee> collectionimage;
    public ObservableCollection<Employee> CollectionImage
    {

```

```

get { return collectionimage; }
set
{
collectionimage = value;
OnPropertyRaised("CollectionImage");
}
}
public EmployeeViewMdoel()
{
CollectionImage = new ObservableCollection<Employee>();
CollectionImage.Add(new Employee { Name="Mike" , ImageSource =
"mike.png",Colors=Color.Gray });
CollectionImage.Add(new Employee { Name="Alex",ImageSource= "alex.png",
Colors = Color.Bisque });
CollectionImage.Add(new Employee { Name="Ellanaa", ImageSource=
"ellanaa.png",Colors=Color.LightCoral })
}
}

```

**XML**

```

<ContentPage.BindingContext>
<local:EmployeeViewMdoel/>
</ContentPage.BindingContext>
<sfavatar:SfAvatarView ContentType="Group"
VerticalOptions="Center"
HorizontalOptions="Center"
GroupSource="{Binding CollectionImage}"
InitialsMemberPath="Name"
BackgroundColorMemberPath="Colors"
ImageSourceMemberPath="ImageSource"
WidthRequest="50"
HeightRequest="50"
CornerRadius="25">
</sfavatar:SfAvatarView>

```

**C#**

```

public partial class MainPage : ContentPage, INotifyPropertyChanged
{
EmployeeViewMdoel emp;
public MainPage()
{
InitializeComponent();
Grid mainGrid = new Grid();
emp = new EmployeeViewMdoel();
SfAvatarView avatarview = new SfAvatarView();
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.WidthRequest = 50;
avatarview.HeightRequest = 50;
avatarview.CornerRadius = 25;
avatarview.ContentType = ContentType.Group;
avatarview.GroupSource = emp.CollectionImage;
avatarview.InitialsMemberPath = "Name";
avatarview.ImageSourceMemberPath = "ImageSource";
}
}

```

```
avatarview.BackgroundColorMemberPath = "Colors";
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
}
}
```



#### Add initials only in GroupView

You can set the initials only in the group view by setting the `InitialsMemberPath` alone. It is demonstrated in the following code sample.

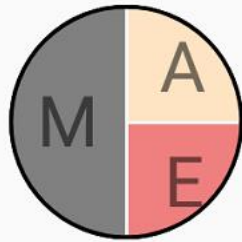
#### XML

```
<sfavatar:SfAvatarView ContentType="Group"
VerticalOptions="Center"
HorizontalOptions="Center"
GroupSource="{Binding CollectionImage}"
InitialsMemberPath="Name"
BackgroundColorMemberPath="Colors"
WidthRequest="50"
HeightRequest="50"
CornerRadius="25">
</sfavatar:SfAvatarView>
```

#### C#

```
public partial class MainPage : ContentPage, INotifyPropertyChanged
{
    EmployeeViewMdoel emp;
    public MainPage()
    {
        Grid mainGrid = new Grid();
        emp = new EmployeeViewMdoel();
        SfAvatarView avatarview = new SfAvatarView();
        avatarview.VerticalOptions = LayoutOptions.Center;
        avatarview.HorizontalOptions = LayoutOptions.Center;
        avatarview.WidthRequest = 50;
        avatarview.HeightRequest = 50;
        avatarview.CornerRadius = 25;
        avatarview.ContentType = ContentType.Group;
        avatarview.GroupSource = emp.CollectionImage;
        avatarview.InitialsMemberPath = "Name";
        avatarview.BackgroundColorMemberPath = "Colors";
```

```
mainGrid.Children.Add(avatarview);  
this.Content = mainGrid;  
}  
}
```



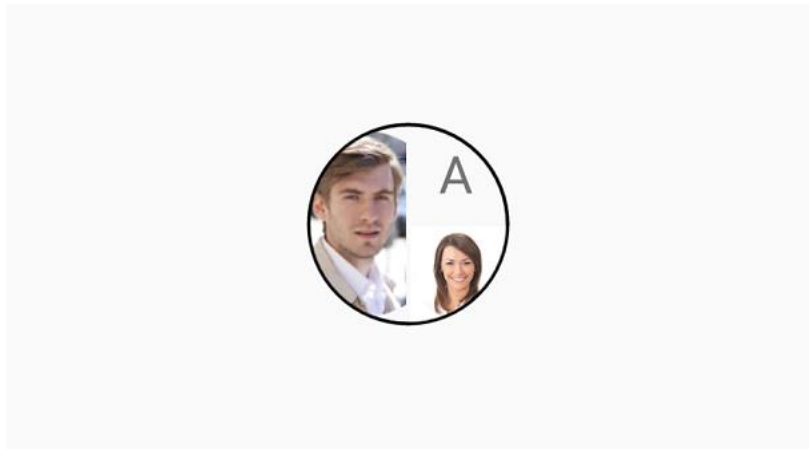
#### Add both image and initials in a GroupView

You have to set both the image and initials based on what should be added in the collection. If the image needs to be added, set `ImageSourcePath`, or if initials need to be added, set `InitialsMemberPath`. This is demonstrated in the following code snippet.

#### C#

```
public class EmployeeViewMdoel : INotifyPropertyChanged  
{  
    public EmployeeViewMdoel()  
    {  
        CollectionImage = new ObservableCollection<Employee>();  
        CollectionImage.Add(new Employee { ImageSource="mike.png" });  
        CollectionImage.Add(new Employee { Name= "alex", Colors=Color.White });  
        CollectionImage.Add(new Employee { ImageSource= "ellanaa.png" });  
    }  
}
```





### Add GroupView using MVVM

Create model and view model for initializing and assigning a value for adding image or initials in the SfAvatarView. The following code sample demonstrates how to use the GroupView property using MVVM pattern.

#### C#

```
//Model.cs
public class Model
{
    public string Name { get; set; }
    public string Images { get; set; }
}
```

#### C#

```
//ViewModel.cs
public class AvatarViewModel
{
    /// <summary>
    /// Gets or sets the value for PeopleCollection field.
    /// </summary>
    public ObservableCollection<Model> PeopleCollection { get; set; }
    /// <summary>
    /// Gets or sets the value for the group view collections.
    /// </summary>
    public ObservableCollection<Model> GroupViewCollection { get; set; }
    /// <summary>
    /// AvatarViewModel
    /// </summary>
    /// <param name="count">Count value</param>
    public AvatarViewModel(int count)
    {
        PopulateModel();
        PopulatePeopleCollection(count);
    }
    /// <summary>
    /// Static count value.
    /// </summary>
    static int staticcount = 0;
    /// <summary>
```

```

/// Change the collections for each group.
/// </summary>
/// <param name="count">count value</param>
private void PopulatePeopleCollection(int count)
{
    PeopleCollection = new ObservableCollection<Model>();
    for (int i = 0; i < count; i++)
    {
        while (true)
        {
            if (GroupViewCollection.Count <= staticcount)
                staticcount = 0;
            var person = GroupViewCollection[staticcount++];
            if (!PeopleCollection.Contains(person))
            {
                PeopleCollection.Add(person);
                break;
            }
        }
    }
}

/// <summary>
/// To add the name and images.
/// </summary>
private void PopulateModel()
{
    GroupViewCollection = new ObservableCollection<Model>();
    GroupViewCollection.Add(new Model() { Name = "Adriana", Images =
    "Adriana.png" });
    GroupViewCollection.Add(new Model() { Name = "Aiden", Images = "Aiden.png"
    });
    GroupViewCollection.Add(new Model() { Name = "Alexander" });
    GroupViewCollection.Add(new Model() { Name = "Arianna", Images =
    "Arianna.png" });
    GroupViewCollection.Add(new Model() { Name = "Clara", Images = "Clara.png"
    });
    GroupViewCollection.Add(new Model() { Name = "Daleyza", Images =
    "Daleyza.png" });
    GroupViewCollection.Add(new Model() { Name = "Ellie", Images = "Ellie.png"
    });
    GroupViewCollection.Add(new Model() { Name = "Finley" });
    GroupViewCollection.Add(new Model() { Name = "Jackson", Images =
    "Jackson.png" });
    GroupViewCollection.Add(new Model() { Name = "Jayden" });
    GroupViewCollection.Add(new Model() { Name = "Kaylee", Images = "Kaylee.png"
    });
    GroupViewCollection.Add(new Model() { Name = "Lucy" });
}

```

## XML

```

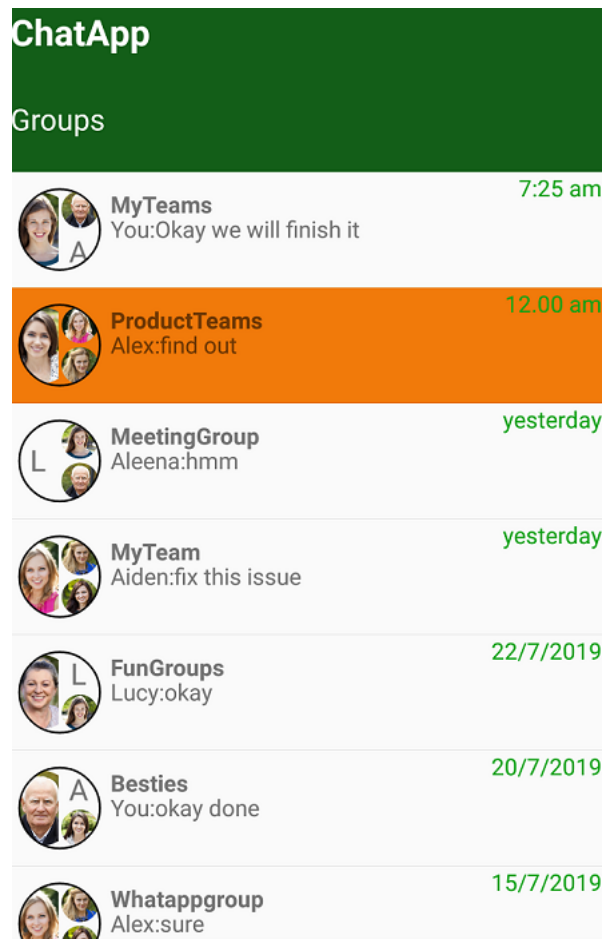
<!--MainPage.Xaml-->
<sfavatarview:SfAvatarView ContentType="Group"
Margin="5,0,0,0"
HorizontalOptions="Center"

```

```
HeightRequest="50"  
WidthRequest="50"  
CornerRadius="25"  
VerticalOptions="Center"  
GroupSource="{Binding PeopleCollection}"  
ImageSourceMemberPath="Images"  
InitialsMemberPath="Name">  
</sfavatarview:SfAvatarView>
```

## C#

```
//MainPage.Xaml.cs  
emp = new EmployeeViewMdoel();  
SfAvatarView avatarview = new SfAvatarView();  
avatarview.VerticalOptions = LayoutOptions.Center;  
avatarview.HorizontalOptions = LayoutOptions.Center;  
avatarview.WidthRequest = 50;  
avatarview.HeightRequest = 50;  
avatarview.CornerRadius = 25;  
avatarview.ContentType = ContentType.Group;  
avatarview.GroupSource = emp.CollectionImage;  
avatarview.InitialsMemberPath = "Name";  
avatarview.BackgroundColorMemberPath = "Colors";  
mainGrid.Children.Add(avatarview);
```



The complete SfAvatarView of GroupView type using MVVM is available in this [sample](#).

## Visual Style

The **SfAvatarView** control supports customization using the following built-in visual styles:

- Custom
- Circle
- Square

### Custom

Custom type allows you to customize the control, where you can handle the size, colors, images, etc. of the control. Refer to this [documentation](#).

---

**Note:** The default visual type is custom.

---

### Circle

You can directly set value to the circle in the **SfAvatarView** using the following styles:

- ExtraLargeCircle
- LargeCircle
- MediumCircle
- SmallCircle

- ExtraSmallCircle

The following code sample demonstrates how to define visual style of circle AvatarView.

#### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<Style x:Key="AvatarViewStyle" TargetType="sfavatar:SfAvatarView">
<Setter Property="VerticalOptions" Value="Center"/>
<Setter Property="HorizontalOptions" Value="Center"/>
<Setter Property="ContentType" Value="Custom"/>
<Setter Property="ImageSource" Value="ellanaa.png"/>
</Style>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<StackLayout Orientation="Vertical" HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand">
<Grid >
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<sfavatar:SfAvatarView AvatarShape="Circle" AvatarSize="ExtraLarge"
Grid.Row="0" Grid.Column="4" Style="{StaticResource AvatarViewStyle}" />
<Label Text="ExtraLargeCircle" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="4" FontSize="10" />
<sfavatar:SfAvatarView AvatarShape="Circle" AvatarSize="Large" Grid.Row="0"
Grid.Column="3" Style="{StaticResource AvatarViewStyle}" />
<Label Text="LargeCircle" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="3" FontSize="10" />
<sfavatar:SfAvatarView AvatarShape="Circle" AvatarSize="Medium" Grid.Row="0"
Grid.Column="2" Style="{StaticResource AvatarViewStyle}" />
<Label Text="MediumCircle" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="2" FontSize="10" />
<sfavatar:SfAvatarView AvatarShape="Circle" AvatarSize="Small" Grid.Row="0"
Grid.Column="1" Style="{StaticResource AvatarViewStyle}" />
<Label Text="SmallCircle" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="1" FontSize="10" />
<sfavatar:SfAvatarView AvatarShape="Circle" AvatarSize="ExtraSmall"
Grid.Row="0" Grid.Column="0" Style="{StaticResource AvatarViewStyle}" />
```

```
<Label Text="ExtraSmallCircle" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="0" FontSize="10"/>
</Grid>
</StackLayout>
</ContentPage.Content>
```

**C#**

```
StackLayout stack = new StackLayout();
stack.Orientation = StackOrientation.Vertical;
stack.HorizontalOptions = LayoutOptions.CenterAndExpand;
stack.VerticalOptions = LayoutOptions.CenterAndExpand;
Grid mainGrid = new Grid();
mainGrid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
GridUnitType.Star) });
mainGrid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
GridUnitType.Star) });
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
Label XLargeCirclelabel = new Label();
XLargeCirclelabel.Text = "ExtraLargeCircle";
XLargeCirclelabel.FontSize = 12;
XLargeCirclelabel.FontAttributes = FontAttributes.Bold;
XLargeCirclelabel.HorizontalOptions = LayoutOptions.Center;
XLargeCirclelabel.VerticalOptions = LayoutOptions.Center;
XLargeCirclelabel.HorizontalTextAlignment = TextAlignment.Center;
XLargeCirclelabel.VerticalTextAlignment = TextAlignment.Center;
XLargeCirclelabel.FontSize = 10;
SfAvatarView avatarview1 = new SfAvatarView();
avatarview1.VerticalOptions = LayoutOptions.Center;
avatarview1.HorizontalOptions = LayoutOptions.Center;
avatarview1.AvatarShape = AvatarShape.Circle;
avatarview1.AvatarSize = AvatarSize.ExtraLarge;
avatarview1.ContentType = ContentType.Default;
avatarview1.ImageSource = "ellanaa.png";
Label LargeCirclelabel = new Label();
LargeCirclelabel.Text = "LargeCircle";
LargeCirclelabel.FontSize = 12;
LargeCirclelabel.FontAttributes = FontAttributes.Bold;
LargeCirclelabel.HorizontalOptions = LayoutOptions.Center;
LargeCirclelabel.VerticalOptions = LayoutOptions.Center;
LargeCirclelabel.HorizontalTextAlignment = TextAlignment.Center;
LargeCirclelabel.VerticalTextAlignment = TextAlignment.Center;
LargeCirclelabel.FontSize = 10;
SfAvatarView avatarview2 = new SfAvatarView();
avatarview2.VerticalOptions = LayoutOptions.Center;
```

```
avatarview2.HorizontalOptions = LayoutOptions.Center;
avatarview2.AvatarShape = AvatarShape.Circle;
avatarview2.AvatarSize = AvatarSize.Large;
avatarview2.ContentType = ContentType.Default;
avatarview2.ImageSource = "ellanaa.png";
Label MediumCirclelabel = new Label();
MediumCirclelabel.Text = "MediumCircle";
MediumCirclelabel.FontSize = 12;
MediumCirclelabel.FontAttributes = FontAttributes.Bold;
MediumCirclelabel.HorizontalOptions = LayoutOptions.Center;
MediumCirclelabel.VerticalOptions = LayoutOptions.Center;
MediumCirclelabel.HorizontalTextAlignment = TextAlignment.Center;
MediumCirclelabel.VerticalTextAlignment = TextAlignment.Center;
MediumCirclelabel.FontSize = 10;
SfAvatarView avatarview3 = new SfAvatarView();
avatarview3.VerticalOptions = LayoutOptions.Center;
avatarview3.HorizontalOptions = LayoutOptions.Center;
avatarview3.AvatarShape = AvatarShape.Circle;
avatarview3.AvatarSize = AvatarSize.Medium;
avatarview3.ContentType = ContentType.Default;
avatarview3.ImageSource = "ellanaa.png";
Label SmallCirclelabel = new Label();
SmallCirclelabel.Text = "SmallCircle";
SmallCirclelabel.FontSize = 12;
SmallCirclelabel.FontAttributes = FontAttributes.Bold;
SmallCirclelabel.HorizontalOptions = LayoutOptions.Center;
SmallCirclelabel.VerticalOptions = LayoutOptions.Center;
SmallCirclelabel.HorizontalTextAlignment = TextAlignment.Center;
SmallCirclelabel.VerticalTextAlignment = TextAlignment.Center;
SmallCirclelabel.FontSize = 10;
SfAvatarView avatarview4 = new SfAvatarView();
avatarview4.VerticalOptions = LayoutOptions.Center;
avatarview4.HorizontalOptions = LayoutOptions.Center;
avatarview4.AvatarShape = AvatarShape.Circle;
avatarview4.AvatarSize = AvatarSize.Small;
avatarview4.ContentType = ContentType.Default;
avatarview4.ImageSource = "ellanaa.png";
Label XSmallCirclelabel = new Label();
XSmallCirclelabel.Text = "ExtraSmallCircle";
XSmallCirclelabel.FontSize = 12;
XSmallCirclelabel.FontAttributes = FontAttributes.Bold;
XSmallCirclelabel.HorizontalOptions = LayoutOptions.Center;
XSmallCirclelabel.VerticalOptions = LayoutOptions.Center;
XSmallCirclelabel.HorizontalTextAlignment = TextAlignment.Center;
XSmallCirclelabel.VerticalTextAlignment = TextAlignment.Center;
XSmallCirclelabel.FontSize = 10;
SfAvatarView avatarview5 = new SfAvatarView();
avatarview5.VerticalOptions = LayoutOptions.Center;
avatarview5.HorizontalOptions = LayoutOptions.Center;
avatarview5.AvatarShape = AvatarShape.Circle;
avatarview5.ImageSource = "ellanaa.png";
avatarview5.AvatarSize = AvatarSize.ExtraSmall;
avatarview5.ContentType = ContentType.Default;
mainGrid.Children.Add(XLargeCirclelabel, 4, 1);
mainGrid.Children.Add(avatarview1, 4, 0);
mainGrid.Children.Add(LargeCirclelabel, 3, 1);
mainGrid.Children.Add(avatarview2, 3, 0);
```

```
mainGrid.Children.Add(MediumCircleLabel, 2, 1);
mainGrid.Children.Add(avatarview3, 2, 0);
mainGrid.Children.Add(SmallCircleLabel, 1, 1);
mainGrid.Children.Add(avatarview4, 1, 0);
mainGrid.Children.Add(XSmallCircleLabel, 0, 1);
mainGrid.Children.Add(avatarview5, 0, 0);
stack.Children.Add(mainGrid);
this.Content = stack;
```



## Square

You can directly set value to the square in the **SfAvatarView** using the following styles:

- ExtraLargeSquare
- LargeSquare
- MediumSquare
- SmallSquare
- ExtraSmallSquare

The following code sample demonstrates how to define visual style of square AvatarView.

## XML

```
<ContentPage.Resources>
  <ResourceDictionary>
    <Style x:Key="AvatarViewStyle" TargetType="sfavatar:SfAvatarView">
      <Setter Property="VerticalOptions" Value="Center"/>
      <Setter Property="HorizontalOptions" Value="Center"/>
      <Setter Property="ContentType" Value="Custom"/>
      <Setter Property="ImageSource" Value="ellanaa.png"/>
    </Style>
  </ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
  <StackLayout Orientation="Vertical" HorizontalOptions="CenterAndExpand"
    VerticalOptions="CenterAndExpand">
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
    </Grid>
  </StackLayout>
</ContentPage.Content>
```



```

<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<sfavatar:SfAvatarView AvatarShape="Square" AvatarSize="ExtraLarge"
Grid.Row="0" Grid.Column="4" Style="{StaticResource AvatarViewStyle}" />
<Label Text="ExtraLargeSquare" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="4" FontSize="10" />
<sfavatar:SfAvatarView AvatarShape="Square" AvatarSize="Large" Grid.Row="0"
Grid.Column="3" Style="{StaticResource AvatarViewStyle}" />
<Label Text="LargeSquare" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="3" FontSize="10" />
<sfavatar:SfAvatarView AvatarShape="Square" AvatarSize="Medium"
Grid.Row="0" Grid.Column="2" Style="{StaticResource AvatarViewStyle}" />
<Label Text="MediumSquare" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="2" FontSize="10" />
<sfavatar:SfAvatarView AvatarShape="Square" AvatarSize="Small" Grid.Row="0"
Grid.Column="1" Style="{StaticResource AvatarViewStyle}" />
<Label Text="SmallSquare" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="1" FontSize="10" />
<sfavatar:SfAvatarView AvatarShape="Square" AvatarSize="ExtraSmall"
Grid.Row="0" Grid.Column="0" Style="{StaticResource AvatarViewStyle}" />
<Label Text="ExtraSmallSquare" FontAttributes="Bold" Grid.Row="1"
HorizontalOptions="Center" VerticalOptions="Center"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="0" FontSize="10" />
</Grid>
</StackLayout>
</ContentPage.Content>

```

## C#

```

StackLayout stack = new StackLayout();
stack.Orientation = StackOrientation.Vertical;
stack.HorizontalOptions = LayoutOptions.CenterAndExpand;
stack.VerticalOptions = LayoutOptions.CenterAndExpand;
Grid mainGrid = new Grid();
mainGrid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
GridUnitType.Star) });
mainGrid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
GridUnitType.Star) });
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });

```

```
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
mainGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
Label XLargeSquarelabel = new Label();
XLargeSquarelabel.Text = "ExtraLargeSquare";
XLargeSquarelabel.FontSize = 12;
XLargeSquarelabel.FontAttributes = FontAttributes.Bold;
XLargeSquarelabel.HorizontalOptions = LayoutOptions.Center;
XLargeSquarelabel.VerticalOptions = LayoutOptions.Center;
XLargeSquarelabel.HorizontalTextAlignment = TextAlignment.Center;
XLargeSquarelabel.VerticalTextAlignment = TextAlignment.Center;
XLargeSquarelabel.FontSize = 10;
SfAvatarView avatarviewsquare1 = new SfAvatarView();
avatarviewsquare1.VerticalOptions = LayoutOptions.Center;
avatarviewsquare1.HorizontalOptions = LayoutOptions.Center;
avatarviewsquare1.AvatarShape = AvatarShape.Square;
avatarviewsquare1.AvatarSize = AvatarSize.ExtraLarge;
avatarviewsquare1.ContentType = ContentType.Default;
avatarviewsquare1.ImageSource = "ellanaa.png";
Label LargeSquarelabel = new Label();
LargeSquarelabel.Text = "LargeSquare";
LargeSquarelabel.FontSize = 12;
LargeSquarelabel.FontAttributes = FontAttributes.Bold;
LargeSquarelabel.HorizontalOptions = LayoutOptions.Center;
LargeSquarelabel.VerticalOptions = LayoutOptions.Center;
LargeSquarelabel.HorizontalTextAlignment = TextAlignment.Center;
LargeSquarelabel.VerticalTextAlignment = TextAlignment.Center;
LargeSquarelabel.FontSize = 10;
SfAvatarView avatarviewsquare2 = new SfAvatarView();
avatarviewsquare2.VerticalOptions = LayoutOptions.Center;
avatarviewsquare2.HorizontalOptions = LayoutOptions.Center;
avatarviewsquare2.AvatarShape = AvatarShape.Square;
avatarviewsquare2.AvatarSize = AvatarSize.Large;
avatarviewsquare2.ContentType = ContentType.Default;
avatarviewsquare2.ImageSource = "ellanaa.png";
Label MediumSquarelabel = new Label();
MediumSquarelabel.Text = "MediumSquare";
MediumSquarelabel.FontSize = 12;
MediumSquarelabel.FontAttributes = FontAttributes.Bold;
MediumSquarelabel.HorizontalOptions = LayoutOptions.Center;
MediumSquarelabel.VerticalOptions = LayoutOptions.Center;
MediumSquarelabel.HorizontalTextAlignment = TextAlignment.Center;
MediumSquarelabel.VerticalTextAlignment = TextAlignment.Center;
MediumSquarelabel.FontSize = 10;
SfAvatarView avatarviewsquare3 = new SfAvatarView();
avatarviewsquare3.VerticalOptions = LayoutOptions.Center;
avatarviewsquare3.HorizontalOptions = LayoutOptions.Center;
avatarviewsquare3.AvatarShape = AvatarShape.Square;
avatarviewsquare3.AvatarSize = AvatarSize.Medium;
avatarviewsquare3.ContentType = ContentType.Default;
avatarviewsquare3.ImageSource = "ellanaa.png";
Label SmallSquarelabel = new Label();
SmallSquarelabel.Text = "SmallSquare";
SmallSquarelabel.FontSize = 12;
SmallSquarelabel.FontAttributes = FontAttributes.Bold;
SmallSquarelabel.HorizontalOptions = LayoutOptions.Center;
```

```

SmallSquarelabel.VerticalOptions = LayoutOptions.Center;
SmallSquarelabel.HorizontalTextAlignment = TextAlignment.Center;
SmallSquarelabel.VerticalTextAlignment = TextAlignment.Center;
SmallSquarelabel.FontSize = 10;
SfAvatarView avatarviewsquare4 = new SfAvatarView();
avatarviewsquare4.VerticalOptions = LayoutOptions.Center;
avatarviewsquare4.HorizontalOptions = LayoutOptions.Center;
avatarviewsquare4.AvatarShape = AvatarShape.Square;
avatarviewsquare4.AvatarSize = AvatarSize.Small;
avatarviewsquare4.ContentType = ContentType.Default;
avatarviewsquare4.ImageSource = "ellanaa.png";
Label XSmallSquarelabel = new Label();
XSmallSquarelabel.Text = "ExtraSmallSquare";
XSmallSquarelabel.FontSize = 12;
XSmallSquarelabel.FontAttributes = FontAttributes.Bold;
XSmallSquarelabel.HorizontalOptions = LayoutOptions.Center;
XSmallSquarelabel.VerticalOptions = LayoutOptions.Center;
XSmallSquarelabel.HorizontalTextAlignment = TextAlignment.Center;
XSmallSquarelabel.VerticalTextAlignment = TextAlignment.Center;
XSmallSquarelabel.FontSize = 10;
SfAvatarView avatarviewsquare5 = new SfAvatarView();
avatarviewsquare5.VerticalOptions = LayoutOptions.Center;
avatarviewsquare5.HorizontalOptions = LayoutOptions.Center;
avatarviewsquare5.AvatarShape = AvatarShape.Square;
avatarviewsquare5.AvatarSize = AvatarSize.ExtraSmall;
avatarviewsquare5.ContentType = ContentType.Default;
avatarviewsquare5.ImageSource = "ellanaa.png";
mainGrid.Children.Add(XLargeSquarelabel, 4, 1);
mainGrid.Children.Add(avatarviewsquare1, 4, 0);
mainGrid.Children.Add(LargeSquarelabel, 3, 1);
mainGrid.Children.Add(avatarviewsquare2, 3, 0);
mainGrid.Children.Add(MediumSquarelabel, 2, 1);
mainGrid.Children.Add(avatarviewsquare3, 2, 0);
mainGrid.Children.Add(SmallSquarelabel, 1, 1);
mainGrid.Children.Add(avatarviewsquare4, 1, 0);
mainGrid.Children.Add(XSmallSquarelabel, 0, 1);
mainGrid.Children.Add(avatarviewsquare5, 0, 0);
stack.Children.Add(mainGrid);
this.Content = stack;

```



The visual style sample is available in the following link: [Sample](#).

## Customization

The SfAvatarView control provides options to customize the color and size. The control can be customized using the following properties:

### Colors

Color in the SfAvatarView can be customized by the border color, the default background color, and automatic background color.

### Border color

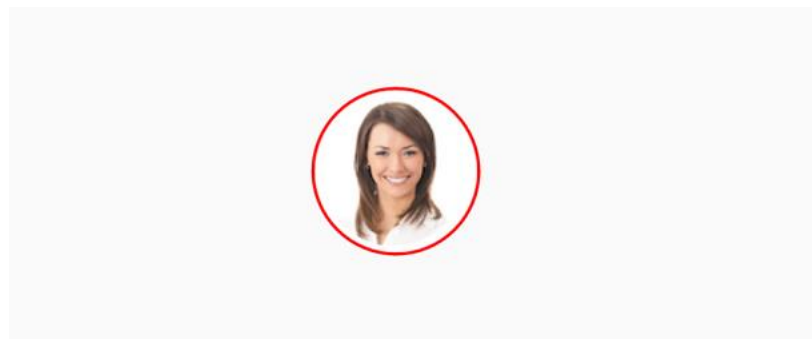
The border color is used for setting color to the border of SfAvatarView using the **BorderColor** property.

### XML

```
<sfavatar:SfAvatarView ContentType="Default"
AvatarShape="Circle"
AvatarSize="Large"
HorizontalOptions="Center"
VerticalOptions="Center"
ImageSource="ellanaa.png"
BorderColor="Red">
</sfavatar:SfAvatarView>
```

### C#

```
Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.ContentType = ContentType.Default;
avatarview.AvatarShape = AvatarShape.Circle;
avatarview.AvatarSize = AvatarSize.Large;
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.ImageSource = "ellanaa.png";
avatarview.BorderColor = Color.Red;
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
```



### Default background color

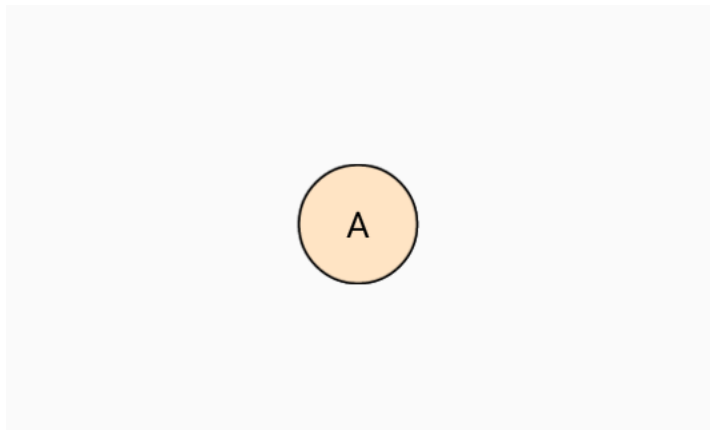
The background color for the SfAvatarView control can be set using the **AvatarColorMode** property. When the **AvatarColorMode** property is set to default, it displays the background color set in the **BackgroundColor** property.

### XML

```
<sfavatar:SfAvatarView ContentType="Initials"
AvatarShape="Circle"
AvatarSize="Large"
HorizontalOptions="Center"
VerticalOptions="Center"
AvatarName="Alex"
BackgroundColor="Bisque"
AvatarColorMode="Default"
BorderColor="Black">
</sfavatar:SfAvatarView>
```

## C#

```
Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.ContentType = ContentType.Initials;
avatarview.AvatarShape = AvatarShape.Circle;
avatarview.AvatarSize = AvatarSize.Large;
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.AvatarName = "Alex";
avatarview.BorderColor = Color.Black;
avatarview.AvatarColorMode = AvatarColorMode.Default;
avatarview.BackgroundColor = Color.Bisque;
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
```



### Automatic background color

The **SfAvatarView** control allows automatic background color while initializing. It contains the following properties:

- **DarkBackground**: Shows a dark color for the initials and background color.
- **LightBackground**: Shows a light color for the initials and background color.

### Dark color

The dark background color can be set using **DarkBackground** in the **AvatarColorMode** property.

## XML

```
<sfavatar:SfAvatarView ContentType="Initials"  
InitialsType="DoubleCharacter"  
AvatarShape="Circle"  
AvatarSize="Large"  
HorizontalOptions="Center"  
VerticalOptions="Center"  
AvatarName="Alex"  
AvatarColorMode="DarkBackground"  
BorderColor="Black">  
</sfavatar:SfAvatarView>
```

## C#

```
Grid mainGrid = new Grid();  
SfAvatarView avatarview = new SfAvatarView();  
avatarview.VerticalOptions = LayoutOptions.Center;  
avatarview.HorizontalOptions = LayoutOptions.Center;  
avatarview.AvatarShape = AvatarShape.Circle;  
avatarview.AvatarSize = AvatarSize.Large;  
avatarview.ContentType = ContentType.Initials;  
avatarview.InitialsType = InitialsType.DoubleCharacter;  
avatarview.AvatarName = "Alex";  
avatarview.BorderColor = Color.Black;  
avatarview.AvatarColorMode = AvatarColorMode.DarkBackground;  
mainGrid.Children.Add(avatarview);  
this.Content = mainGrid;
```



## Light color

The light background color can be set using `LightBackground` in the `AvatarColorMode` property.

## XML

```
<sfavatar:SfAvatarView ContentType="Initials"  
InitialsType="DoubleCharacter"  
VerticalOptions="Center"  
AvatarName="Alex"  
AvatarShape="Circle"  
AvatarSize="Large"  
AvatarColorMode="LightBackground">
```

```

BorderColor="Black"
HorizontalOptions="Center" >
</sfavatar:SfAvatarView>

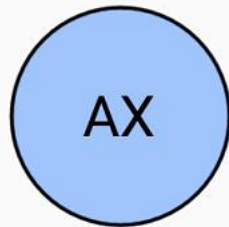
```

## C#

```

Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.AvatarShape = AvatarShape.Circle;
avatarview.AvatarSize = AvatarSize.Large;
avatarview.ContentType = ContentType.Initials;
avatarview.InitialsType = InitialsType.DoubleCharacter;
avatarview.AvatarName = "Alex";
avatarview.BorderColor = Color.Black;
avatarview.AvatarColorMode = AvatarColorMode.LightBackground;
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;

```



## Gradients

You can also specify a range of colors using **BackgroundGradient** as demonstrated in the following code example.

## XML

```

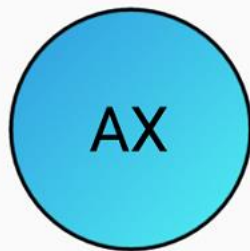
xmlns:gradient="clr-
namespace:Syncfusion.XForms.Graphics;assembly=Syncfusion.Core.XForms"
.....
<sfavatar:SfAvatarView ContentType="Initials"
AvatarName="Alex"
AvatarShape="Circle"
AvatarSize="Large"
HorizontalOptions="Center"
VerticalOptions="Center"
InitialsType="DoubleCharacter">
<sfavatar:SfAvatarView.BackgroundGradient>
<gradient:SfLinearGradientBrush>
<gradient:SfLinearGradientBrush.GradientStops>
<gradient:SfGradientStop Color="#2F9BDF" Offset="0"/>
<gradient:SfGradientStop Color="#51F1F2" Offset="1"/>

```

```
</gradient:SfLinearGradientBrush.GradientStops>
</gradient:SfLinearGradientBrush>
</sfavatar:SfAvatarView.BackgroundGradient>
</sfavatar:SfAvatarView>
```

## C#

```
using Syncfusion.XForms.Graphics;
.....
SfLinearGradientBrush gradientBrush = new SfLinearGradientBrush();
gradientBrush.GradientStops = new GradientStopCollection()
{
    new SfGradientStop() { Color = Color.FromHex("#2F9BDF"), Offset = 0 },
    new SfGradientStop() { Color = Color.FromHex("#51F1F2"), Offset = 1 }
};
Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.AvatarShape = AvatarShape.Circle;
avatarview.AvatarSize = AvatarSize.Large;
avatarview.ContentType = ContentType.Initials;
avatarview.InitialsType = InitialsType.DoubleCharacter;
avatarview.AvatarName = "Alex";
avatarview.BorderColor = Color.Black;
avatarview.BackgroundGradient = gradientBrush;
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
```



**Note:** Here, the BackgroundGradient is a type of SfGradientBrush, so you can apply SfLinearGradientBrush on it. This SfLinearGradientBrush is available in [Syncfusion.Xamarin.Core](#) from [nuget.org](#). To know more about gradient view, refer to this [documentation](#).

## Sizing

In the SfAvatarView control, size of the view can be controlled using width, height, border width, and corner radius.



### Width

You can customize the width of the avatar view using the `WidthRequest` property.

### Height

You can customize the height of the avatar view using the `HeightRequest` property.

### Border width

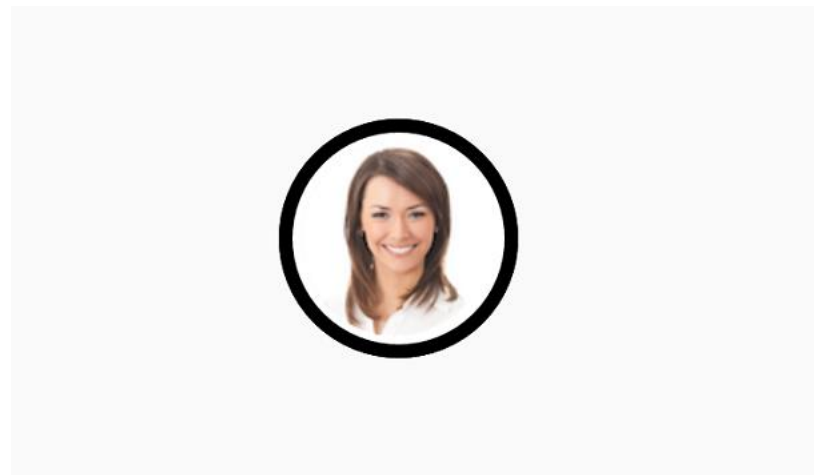
You can customize the thickness of the avatar view using the `BorderWidth` property.

### XML

```
<sfavatar:SfAvatarView ContentType="Default"
AvatarShape="Circle"
AvatarSize="Large"
ImageSource="ellanaa.png"
BorderColor="Red"
VerticalOptions="Center"
BorderWidth="4"
HorizontalOptions="Center" >
</sfavatar:SfAvatarView>
```

### C#

```
Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.AvatarShape = AvatarShape.Circle;
avatarview.AvatarSize = AvatarSize.Large;
avatarview.BorderWidth = 4;
avatarview.ContentType = ContentType.Default;
avatarview.ImageSource = "ellanaa.png";
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
```



### Corner radius

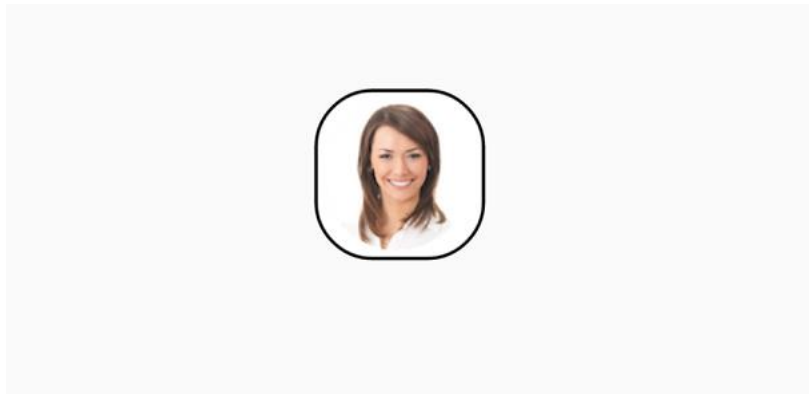
You can customize the corner radius of the avatar view using the `CornerRadius` property.

### XML

```
<sfavatar:SfAvatarView ContentType="Default"
ImageSource="ellanaa.png"
HorizontalOptions="Center"
VerticalOptions="Center"
WidthRequest="60"
HeightRequest="60"
CornerRadius="20">
</sfavatar:SfAvatarView>
```

**C#**

```
Grid mainGrid = new Grid();
SfAvatarView avatarview = new SfAvatarView();
avatarview.HorizontalOptions = LayoutOptions.Center;
avatarview.VerticalOptions = LayoutOptions.Center;
avatarview.WidthRequest = 60;
avatarview.HeightRequest = 60;
avatarview.CornerRadius = 20;
avatarview.ImageSource = "ellanaa.png";
avatarview.ContentType = ContentType.Default;
mainGrid.Children.Add(avatarview);
this.Content = mainGrid;
```

**How to****Set badge view to avatar**

The **SfAvatarView** control provides support for **BadgeView** to notify users of new or unread messages, notifications, or the status of something.

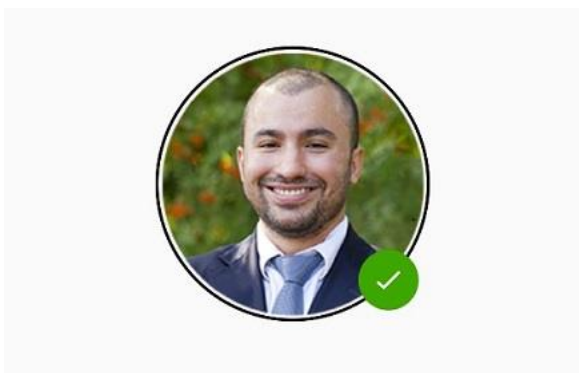
**XML**

```
xmlns:badge="clr-
namespace:Syncfusion.XForms.BadgeView;assembly=Syncfusion.SfBadgeView.XForms
"
.....
<badge:SfBadgeView VerticalOptions="Center"
HorizontalOptions="Center">
<badge:SfBadgeView.Content>
<sfavatar:SfAvatarView HorizontalOptions="Center"
ContentType="Custom"
ImageSource="alex.png"
VerticalOptions="Center"
```

```
WidthRequest="60"  
HeightRequest="60"  
CornerRadius="30">  
</sfavatar:SfAvatarView>  
</badge:SfBadgeView.Content>  
<badge:SfBadgeView.BadgeSettings>  
<badge:BadgeSetting Offset="-10,-10"  
BadgeAnimation="Scale"  
BadgePosition="BottomRight"  
BadgeType="Success"  
BadgeIcon="Away"/>  
</badge:SfBadgeView.BadgeSettings>  
</badge:SfBadgeView>
```

## C#

```
using Syncfusion.XForms.BadgeView;  
.....  
SfBadgeView badge = new SfBadgeView();  
badge.HorizontalOptions = LayoutOptions.Center;  
badge.VerticalOptions = LayoutOptions.Center;  
badge.BadgeSettings = new BadgeSetting();  
BadgeSetting badgeSetting = new BadgeSetting();  
badgeSetting.BadgeType = BadgeType.Success;  
badgeSetting.BadgeIcon = BadgeIcon.Available;  
badgeSetting.BadgePosition = BadgePosition.BottomRight;  
badgeSetting.BadgeAnimation = BadgeAnimation.Scale;  
badgeSetting.Offset = new Point(-10, -10);  
badge.BadgeSettings = badgeSetting;  
Grid mainGrid = new Grid();  
SfAvatarView avatarview = new SfAvatarView();  
avatarview.HorizontalOptions = LayoutOptions.Center;  
avatarview.VerticalOptions = LayoutOptions.Center;  
avatarview.WidthRequest = 60;  
avatarview.HeightRequest = 60;  
avatarview.CornerRadius = 30;  
avatarview.ImageSource = "alex.png";  
avatarview.ContentType = ContentType.Custom;  
badge.Content = avatarview;  
mainGrid.Children.Add(badge);  
this.Content = mainGrid;
```



---

**Note:** The SfBadgeView is available in [Syncfusion.Xamarin.SfBadgeView](#) from [nuget.org](#). To know more about SfBadgeView view, refer to this [documentation](#).

---

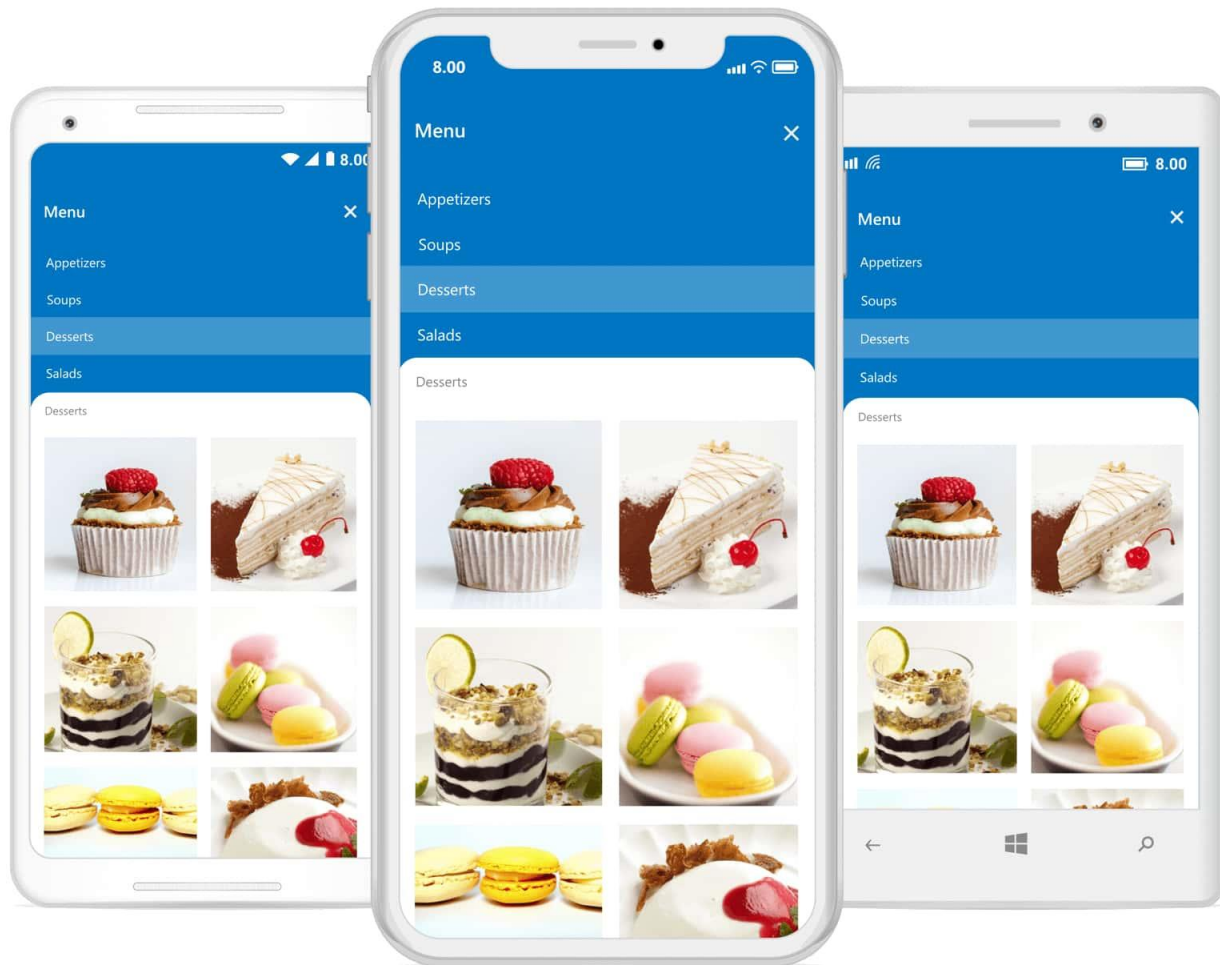
## SfBackdropPage

### Overview

The backdrop page is composed of two surfaces, a back layer and a front layer. The back layer holds actionable content (like navigation or filtration), which is relevant to the front layer.

### Key features

- Allows to add with [NavigationPage](#) and supports seamless navigation and toolbar customizations.
- Adjusts the height of back layer automatically based on its content and provides an option to expand the content to fit the screen.
- Supports curved and flat edge shapes for the front layer with corner radius customization options.
- Provides smooth animations for revealing and concealing the back layer content.
- Allows users to customize the icons of navigation header in the revealed and closed states separately.



## Getting Started

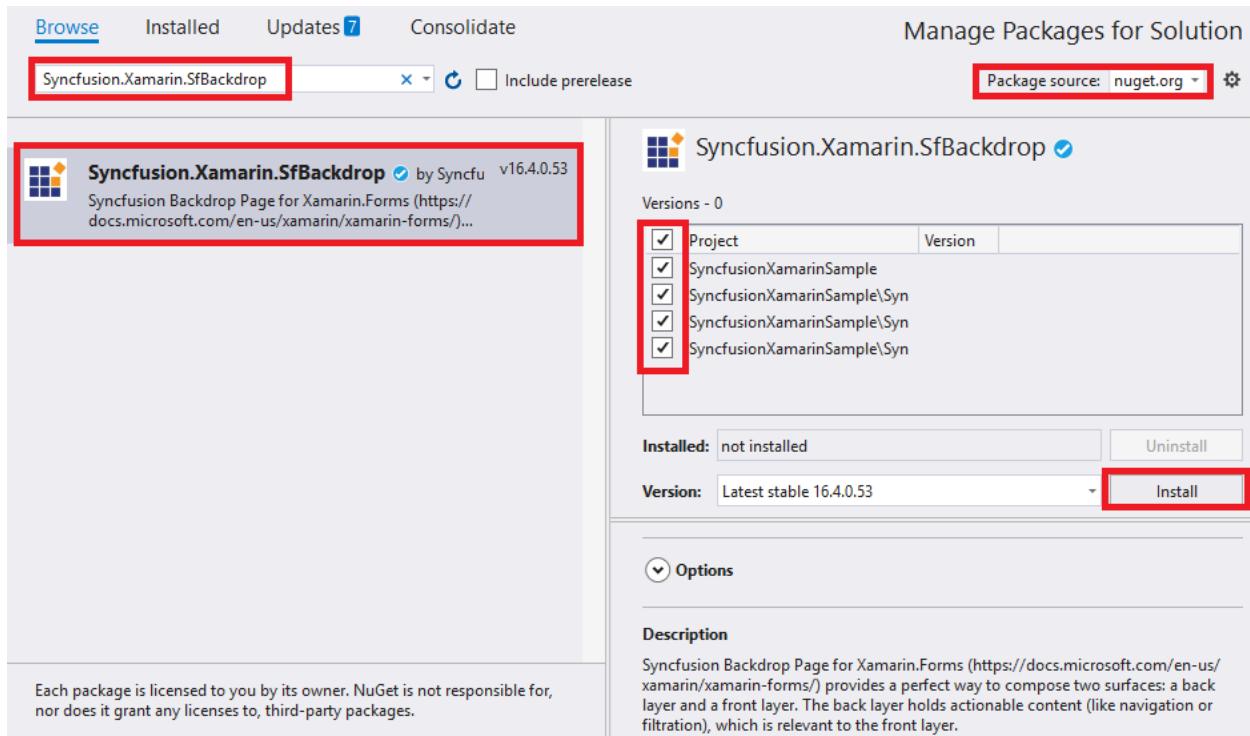
This section explains the steps required to configure the backdrop page.

### Adding SfBackdropPage reference

You can add SfBackdropPage reference using one of the following methods:

#### Method 1: Adding SfBackdropPage reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfBackdropPage to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfBackdrop](https://nuget.org/packages/Syncfusion.Xamarin.SfBackdrop), and then install it.



**Note:** Install the same version of SfBackdropPage NuGet in all the projects.

### Method 2: Adding SfBackdropPage reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfBackdropPage control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfBackdropPage assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location : {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfBackdrop.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfBackdrop.XForms.Android.dll Syncfusion.SfBackdrop.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfBackdrop.XForms.iOS.dll Syncfusion.SfBackdrop.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll

UWP	Syncfusion.SfBackdrop.XForms.UWP.dll Syncfusion.SfBackdrop.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
-----	--

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launch an application on each platform with backdrop page

To use the backdrop page inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and are discussed in the following sections:

**Note:** If you are adding the references from toolbox, below steps are not needed.

#### iOS

To launch the backdrop page in iOS, call the `SfBackdropPageRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    SfBackdropPageRenderer.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### Universal Windows Platform (UWP)

To deploy the backdrop page in **Release** mode, initialize the backdrop page assemblies in the `App.xaml.cs` file in the UWP project as demonstrated in the following code sample.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    if (rootFrame == null)
    {
        List<Assembly> assembliesToInclude = new List<Assembly>();
        assembliesToInclude.Add(typeof(SfBackdropPageRenderer).GetTypeInfo().Assembly);
        Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    }
}
```

```
}
...
}
```

### Android

Android platform does not require any additional configuration to render the backdrop page.

### Initialize backdrop page

Create a page and import the SfBackdropPage namespace along with [XAML namespaces](#) in Xamarin.Forms.

### XML

```
<?xml version="1.0" encoding="UTF-8"?>
<backdrop:SfBackdropPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:backdrop="clr-
namespace:Syncfusion.XForms.Backdrop;assembly=Syncfusion.SfBackdrop.XForms"
x:Class="BackdropGettingStarted.BackdropSamplePage"
Title="Menu">
</backdrop:SfBackdropPage>
```

### C#

```
using Syncfusion.XForms.Backdrop;
namespace BackdropGettingStarted
{
    public partial class BackdropSamplePage : SfBackdropPage
    {
        public BackdropSamplePage ()
        {
            InitializeComponent();
            this.Title = "Menu";
        }
    }
}
```

### NOTE

[Title](#) and [ToolBarItems](#) properties of the [Page](#) can be used to customize the appearance of header.

### Configure header

Page header for the backdrop appears only when adding backdrop as a child of NavigationPage. To know more about it, refer to [header configuration](#).

### Add back layer content

The back layer holds actionable content (navigation or filtration), which is relevant to the front layer. The back layer will either fill the entire background or occupy the background based on the content height.

### XML

```
<backdrop:SfBackdropPage.BackLayer>
<backdrop:BackdropBackLayer>
```



```

<StackLayout>
  <ListView>
    <ListView.ItemsSource>
      <x:Array Type="{x:Type x:String}">
        <x:String>Appetizers</x:String>
        <x:String>Soups</x:String>
        <x:String>Desserts</x:String>
        <x:String>Salads</x:String>
      </x:Array>
    </ListView.ItemsSource>
  </ListView>
</StackLayout>
</backdrop:BackdropBackLayer>
</backdrop:SfBackdropPage.BackLayer>

```

**C#**

```

this.BackLayer = new BackdropBackLayer
{
    Content = new StackLayout
    {
        Children =
        {
            new ListView
            {
                ItemsSource = new string[] { "Appetizers", "Soups", "Desserts", "Salads" }
            }
        }
    };

```

**Add front layer content**

The front layer always appears in front of the back layer. It is displayed to the full width and holds primary content.

**XML**

```

<backdrop:SfBackdropPage.FrontLayer>
  <backdrop:BackdropFrontLayer>
    <Grid BackgroundColor="WhiteSmoke" VerticalOptions="FillAndExpand" />
  </backdrop:BackdropFrontLayer>
</backdrop:SfBackdropPage.FrontLayer>

```

**C#**

```

this.FrontLayer = new BackdropFrontLayer()
{
    Content = new Grid
    {
        BackgroundColor = Color.WhiteSmoke,
        VerticalOptions = LayoutOptions.FillAndExpand
    }
};

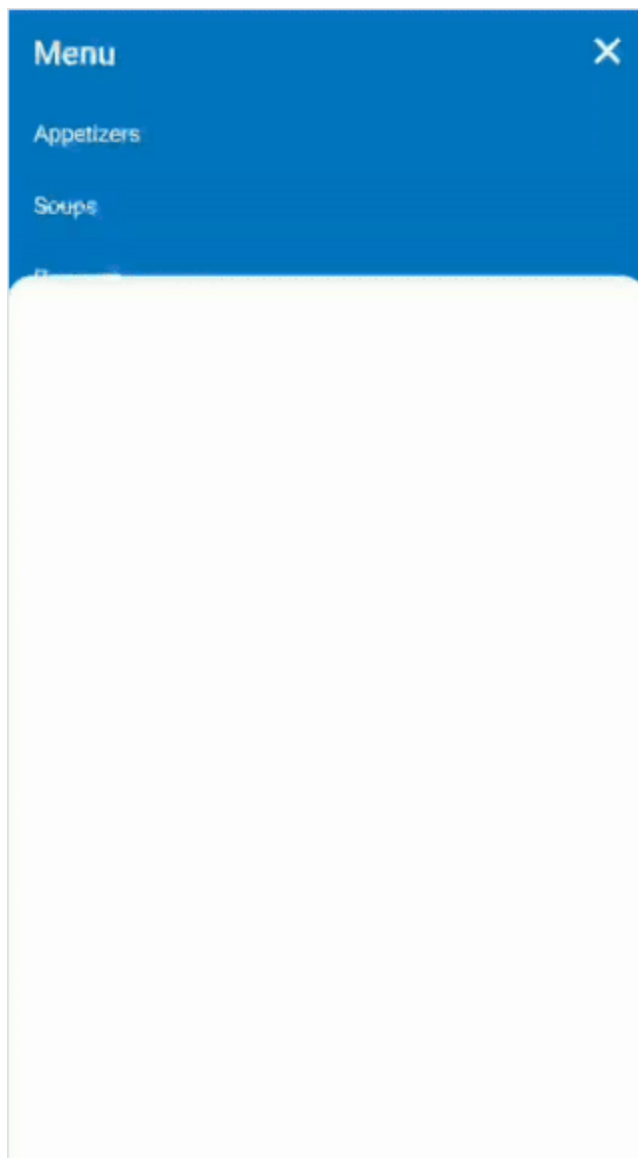
```

### Reveal and conceal the back layer

The following options are provided in backdrop to reveal and conceal the back layer:

- **Programmatically** - Reveals back layer by setting the [IsBackLayerRevealed](#) property to true. By default, it is set to false.
- **Touch interaction** - Reveals back layer by clicking the tool bar icon at the top-right corner of navigation bar header. The Hamburger (or menu ) icon reveals and the Close icon conceals the back layer. When adding the backdrop as a child of MasterDetailPage, the Hamburger and Close icons will be replaced by expand (or down arrow) and collapse (or up arrow) icons, respectively.
- **Swipe or fling action** - Reveals back layer by swipe/fling action on the front layer to the required direction. Swipe downwards to reveal, and swipe upwards to conceal the back layer. The swipe/fling action will be handled only on the top of the front layer to the [RevealedHeight](#).

To know more information about reveal height customization ,refer this [link](#).



**XML**

```
<backdrop:SfBackdropPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:backdrop="clr-
namespace:Syncfusion.XForms.Backdrop;assembly=Syncfusion.SfBackdrop.XForms"
x:Class="BackdropGettingStarted.BackdropSamplePage"
IsBackLayerRevealed="True">
</backdrop:SfBackdropPage>
```

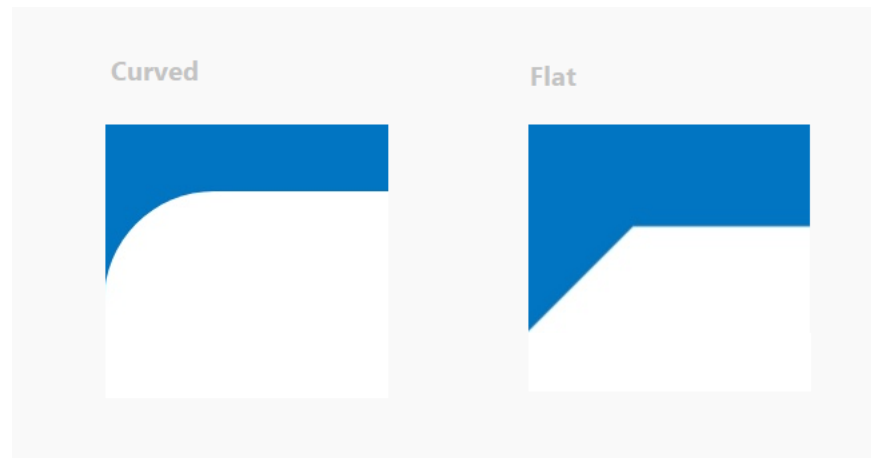
**C#**

```
#region Constructor
public BackdropSamplePage ()
{
    this.IsBackLayerRevealed = true;
}
#endregion
```

You can find the complete getting started sample from this [link](#).

**Corner Shape Customization**

Backdrop allows to customize the shapes on the top corners of the front layer. Curved and cut (flat) shape options are available , it can be switched by setting [EdgeShape](#) property of the front layer.

**NOTE**

The backdrop can only be shaped on the top left and top right corners.

Both the side of the corner radius can be customized separately by setting [LeftCornerRadius](#) and [RightCornerRadius](#) properties of [BackdropFrontLayer](#).

**XML**

```
<backdrop:SfBackdropPage.FrontLayer>
<backdrop:BackdropFrontLayer LeftCornerRadius="30" RightCornerRadius="0"
EdgeShape="Flat">
<Grid />
</backdrop:BackdropFrontLayer>
</backdrop:SfBackdropPage.FrontLayer>
```

**C#**

```

this.FrontLayer = new BackdropFrontLayer()
{
    Content = new Grid(),
    LeftCornerRadius = 30,
    RightCornerRadius = 0,
    EdgeShape = EdgeShape.Flat
};

```

## Reveal Height Customization

When revealing the [back layer](#), the front layer will be moved downwards. By setting the [BackLayerRevealOption](#) property of backdrop, you can customize how far the front layer can be moved from the header when revealing the back layer.

The following options are provided to move the front layer:

[Fill](#) – Moves the front layer downwards to the entire height of the page excluding the [RevealedHeight](#).

[Auto](#) – Moves the front layer downwards to the height of the back layer content.

**XML**

```

<?xml version="1.0" encoding="UTF-8"?>
<backdrop:SfBackdropPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:backdrop="clr-
namespace:Syncfusion.XForms.Backdrop;assembly=Syncfusion.SfBackdrop.XForms"
x:Class="BackdropGettingStarted.BackdropSamplePage"
BackLayerRevealOption="Auto">
  <backdrop:SfBackdropPage.FrontLayer>
    <backdrop:BackdropFrontLayer>
      <Grid />
    </backdrop:BackdropFrontLayer>
  </backdrop:SfBackdropPage.FrontLayer>
</backdrop:SfBackdropPage>

```

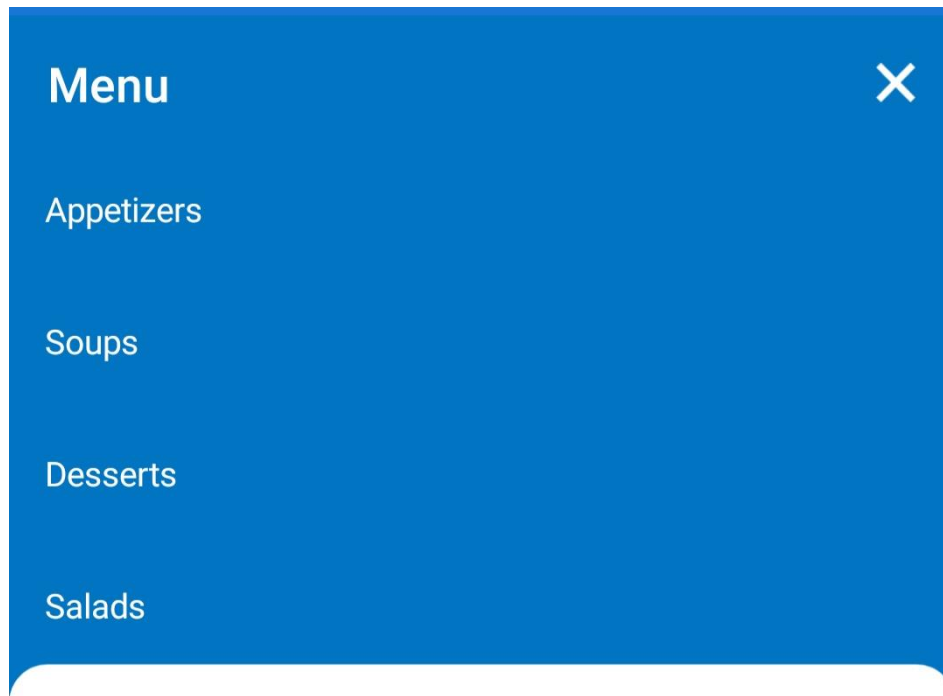
**C#**

```

using Syncfusion.XForms.Backdrop;
namespace BackdropGettingStarted
{
    public class BackdropSamplePage : SfBackdropPage
    {
        public BackdropSamplePage()
        {
            this.BackLayerRevealOption = RevealOption.Auto;
            this.FrontLayer = new BackdropFrontLayer()
            {
                Content = new Grid()
            };
        }
    }
}

```





## Events

### BackLayerStateChanged event

The **BackLayerStateChanged** event occurs when the backdrop page back layer is revealed and concealed. The event occurs in the cases mentioned in this [documentation](#).

### XML

```
<?xml version="1.0" encoding="UTF-8"?>
<backdrop:SfBackdropPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:backdrop="clr-
namespace:Syncfusion.XForms.Backdrop;assembly=Syncfusion.SfBackdrop.XForms"
x:Class="BackdropGettingStarted.MainPage"
Title="Menu"
BackLayerStateChanged="BackLayerStateChanged" >
```

### C#

```
public MainPage()
{
    InitializeComponent();
    this.BackLayerStateChanged += BackLayerStateChanged;
}
private void BackLayerStateChanged(object sender, System.EventArgs e)
{
    // handle event action.
}
```

## Header Configuration

Add backdrop page as a children of [NavigationPage](#) in App.xaml.cs class. Also, [BarBackgroundColor](#), [BarTextColor](#) and other properties of [NavigationPage](#) can be set to customize the default appearance of header.

### C#

```
// In App.xaml.cs
#region Constructor
public App()
{
    ...
    MainPage = new NavigationPage(new BackdropSamplePage());
    ...
}
#endregion
```

## NOTE

Page header for the backdrop will appear only when adding backdrop as a children of [NavigationPage](#).

### Icon customization

The default icons in the navigation header can be customized using the following ways:

*Default icons in NavigationPage*

When the backdrop page contained within the [NavigationPage](#), hamburger icon and close icon (X mark) will be used by default.

**Default icon in closed state****Default icon in revealed state***Default icons in MasterDetailsPage*

When the backdrop page placed in the [MasterDetailPage](#), down arrow icon and up arrow icon will be used by default.

**Default icon in closed state****Default icon in revealed state***Custom icons*

You can customize the default icons in the navigation header by setting the `OpenIcon` and `CloseIcon` properties in `SfBackdropPage`.

**XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<backdrop:SfBackdropPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:backdrop="clr-
namespace:Syncfusion.XForms.Backdrop;assembly=Syncfusion.SfBackdrop.XForms"
x:Class="BackdropGettingStarted.BackdropSamplePage"
OpenIcon="open.png"
CloseIcon="close.png">
</backdrop:SfBackdropPage>
```

**C#**

```
using Syncfusion.XForms.Backdrop;
namespace BackdropGettingStarted
{
    public partial class BackdropSamplePage : SfBackdropPage
    {
        public BackdropSamplePage()
        {
            InitializeComponent();
            this.OpenIcon = "open.png";
            this.CloseIcon = "close.png";
        }
    }
}
```



```
}
```

### Custom icon in closed state



### Custom icon in revealed state



## SfBadgeView

### Overview

Badges are used to notify users of new or unread messages, notifications, or the status of something.

### Key features

**Position** : Position the badge text around the badge content.

**Predefined styles**: Customize the badge background with predefined colors.

**Animation**: Use animations for badge text.

**Predefined symbols**: The Badge View control provides predefined notification symbols such as available, busy, away, and delete.



## Getting Started

This section explains the steps required to configure the [SfBadgeView](#) control and customize its elements.

### Adding SfBadgeView reference

You can add SfBadgeView reference using one of the following methods:

#### Method 1: Adding SfBadgeView reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfBadgeView to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfBadgeView](#), and then install it.

![Adding SfBadgeView reference from nuget](getting-started\_images/Adding SfBadgeView reference.png)

---

**Note:** Install the same version of SfBadgeView NuGet in all the projects.

---

#### Method 2: Adding SfBadgeView reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfBadgeView control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfBadgeView assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfBadgeView.XForms.dll Syncfusion.Core.XForms.dll
Android	Syncfusion.SfBadgeView.XForms.dll Syncfusion.SfBadgeView.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll
iOS	Syncfusion.SfBadgeView.XForms.dll Syncfusion.SfBadgeView.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll
UWP	Syncfusion.SfBadgeView.XForms.dll Syncfusion.SfBadgeView.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the application on each platform with badge view

To use the badge view in an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch the badge view in iOS, call the SfBadgeViewRenderer.Init() in the FinishedLaunching overridden method of the AppDelegate class after the Xamarin.Forms framework has been initialized and before the LoadApplication is called as demonstrated in the following code sample.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init();
}
```

```
// Add the below line if you are using SfBadgeView.
Syncfusion.XForms.iOS.BadgeView.SfBadgeViewRenderer.Init();
LoadApplication(new App());
...
}
```

### Universal Windows Platform (UWP)

You need to initialize the badge view assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with badge view in **Release** mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    if (rootFrame == null)
    {
        List<Assembly> assembliesToInclude = new List<Assembly>();
        // Add the below line if you are using SfBadgeView.
        assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.BadgeView.SfBadgeViewRe
        nderer).GetTypeInfo().Assembly);
        Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    }
    ...
}
```

### Android

The Android platform does not require any additional configuration to render the badge view.

#### Adding namespace

Add the following namespace.

#### XML

```
xmlns:badge="clr-
namespace:Syncfusion.XForms.BadgeView;assembly=Syncfusion.SfBadgeView.XForms
"
```

#### C#

```
using Syncfusion.XForms.BadgeView;
```

#### Initializing badge view

Create an instance for the badge view control, and add it as content.

#### XML

```
<badge:SfBadgeView>
</badge:SfBadgeView>
```

#### C#

```
//Initializing the badge view.  
SfBadgeView sfBadgeView = new SfBadgeView();  
Content = sfBadgeView;
```

### Adding badge text

You can add text to badge view using the [BadgeText](#) property.

#### XML

```
<badge:SfBadgeView>  
<badge:SfBadgeView BadgeText="20" />  
</badge:SfBadgeView>
```

#### C#

```
SfBadgeView sfBadgeView = new SfBadgeView();  
//Adding text to the badge view.  
sfBadgeView.BadgeText = "20";  
Content = sfBadgeView;
```



### Adding content

You can add an image, button, or label to the badge view using the [Content](#) property.

#### XML

```
<badge:SfBadgeView HorizontalOptions="Center" VerticalOptions="Center" >  
<badge:SfBadgeView.Content>  
<Button Text="Primary" WidthRequest="120" HeightRequest="60"/>  
</badge:SfBadgeView.Content>  
</badge:SfBadgeView>
```

#### C#

```
SfBadgeView sfBadgeView = new SfBadgeView();  
sfBadgeView.HorizontalOptions = LayoutOptions.Center;  
sfBadgeView.VerticalOptions = LayoutOptions.Center;  
//Adding image to the content of the badge view.  
Button button = new Button();  
button.Text = "Primary";  
button.WidthRequest = 120;  
button.HeightRequest = 60;  
sfBadgeView.Content = button;  
Content = sfBadgeView;
```

The following code sample gives you the complete code for badge view with badge types and text.

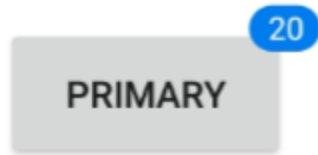
### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
xmlns:badge="clr-
namespace:Syncfusion.XForms.BadgeView;assembly=Syncfusion.SfBadgeView.XForms
">
<badge:SfBadgeView HorizontalOptions="Center" VerticalOptions="Center"
BadgeText="20">
<badge:SfBadgeView.Content>
<Button Text="Primary" WidthRequest="120" HeightRequest="60"/>
</badge:SfBadgeView.Content>
</badge:SfBadgeView>
</ContentPage>
```

### C#

```
using Xamarin.Forms;
using Syncfusion.XForms.BadgeView;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBadgeView sfBadgeView = new SfBadgeView();
            sfBadgeView.HorizontalOptions = LayoutOptions.Center;
            sfBadgeView.VerticalOptions = LayoutOptions.Center;
            sfBadgeView.BadgeText = "20";
            //Adding image to the content of the badge view.
            Button button = new Button();
            button.Text = "Primary";
            button.WidthRequest = 120;
            button.HeightRequest = 60;
            sfBadgeView.Content = button;
            Content = sfBadgeView;
        }
    }
}
```

The following screenshot illustrates the result of the above code sample.



You can find the complete getting started sample from this [link](#).

### Badge Settings

The [BadgeSettings](#) property helps you customize the basic look and feel of the badge view.

[BadgeSettings](#) contains the sub elements such as badge types, positions, icons, colors, and alignments. You can customize the background color, text color, stroke color, width, offset, and font attributes.

### Font customization

The font can be customized using the [FontSize](#), [FontAttributes](#), and [FontFamily](#) properties.

### XML

```
<badge:SfBadgeView BadgeText="48" HorizontalOptions="Center"
VerticalOptions="Center">
  <badge:SfBadgeView.Content>
    <Button Text ="Primary" WidthRequest="120" HeightRequest="60" />
  </badge:SfBadgeView.Content>
  <badge:SfBadgeView.BadgeSettings>
    <badge:BadgeSetting FontSize="15" FontAttributes="Bold">
      <badge:BadgeSetting.FontFamily>
        <OnPlatform x:TypeArguments="x:String" iOS="Chalkduster" Android="serif"
WinPhone="Chiller" />
      </badge:BadgeSetting.FontFamily>
    </badge:BadgeSetting>
  </badge:SfBadgeView.BadgeSettings>
</badge:SfBadgeView>
```

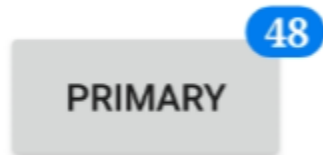
### C#

```
SfBadgeView sfBadgeView = new SfBadgeView();
sfBadgeView.HorizontalOptions = LayoutOptions.Center;
sfBadgeView.VerticalOptions = LayoutOptions.Center;
sfBadgeView.BadgeText = "48";
Button button = new Button();
button.Text = "Primary";
button.WidthRequest = 120;
button.HeightRequest = 60;
sfBadgeView.Content = button;
BadgeSetting badgeSetting = new BadgeSetting();
badgeSetting.FontAttributes = FontAttributes.Bold;
```

```

badgeSetting.FontSize = 15;
badgeSetting.FontFamily = Device.RuntimePlatform == Device.iOS ?
"Chalkduster" : Device.RuntimePlatform == Device.Android ? "serif" :
"Chiller";
sfBadgeView.BadgeSettings = badgeSetting;
Content = sfBadgeView;

```



### Stroke customization

The stroke color and stroke width of the badge view can be customized using the [Stroke](#) and [StrokeWidth](#) properties, respectively.

### XML

```

<badge:SfBadgeView BadgeText="30" HorizontalOptions="Center"
VerticalOptions="Center">
  <badge:SfBadgeView.Content>
    <Button Text ="Primary" WidthRequest="120" HeightRequest="60"/>
  </badge:SfBadgeView.Content>
  <badge:SfBadgeView.BadgeSettings>
    <badge:BadgeSetting Stroke="Orange" StrokeWidth="2" />
  </badge:SfBadgeView.BadgeSettings>
</badge:SfBadgeView>

```

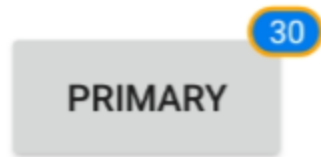
### C#

```

SfBadgeView sfBadgeView = new SfBadgeView();
sfBadgeView.BadgeText = "30";
sfBadgeView.HorizontalOptions = LayoutOptions.Center;
sfBadgeView.VerticalOptions = LayoutOptions.Center;
Button button = new Button();
button.Text = "Primary";
button.WidthRequest = 120;
button.HeightRequest = 60;
sfBadgeView.Content = button;
BadgeSetting badgeSetting = new BadgeSetting();
badgeSetting.Stroke = Color.Orange;
badgeSetting.StrokeWidth = 2;
sfBadgeView.BadgeSettings = badgeSetting;
Content = sfBadgeView;

```





### Text customization

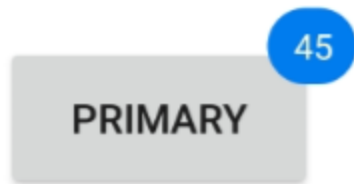
The text color and margin of badge view can be changed using the [TextColor](#) and [TextPadding](#) properties, respectively.

### XML

```
<badge:SfBadgeView BadgeText="45" HorizontalOptions="Center"
VerticalOptions="Center">
  <badge:SfBadgeView.Content>
    <Button Text="Primary" WidthRequest="120" HeightRequest="60"/>
  </badge:SfBadgeView.Content>
  <badge:SfBadgeView.BadgeSettings>
    <badge:BadgeSetting TextColor="LightYellow" TextPadding="10"/>
  </badge:SfBadgeView.BadgeSettings>
</badge:SfBadgeView>
```

### C#

```
SfBadgeView sfBadgeView = new SfBadgeView();
sfBadgeView.HorizontalOptions = LayoutOptions.Center;
sfBadgeView.VerticalOptions = LayoutOptions.Center;
sfBadgeView.BadgeText = "45";
Button button = new Button();
button.WidthRequest = 120;
button.HeightRequest = 60;
button.Text = "Primary";
sfBadgeView.Content = button;
BadgeSetting badgeSetting = new BadgeSetting();
badgeSetting.TextColor = Color.LightYellow;
badgeSetting.TextPadding = 10;
sfBadgeView.BadgeSettings = badgeSetting;
Content = sfBadgeView;
```



### Predefined styles

You can change the colors of the badge using the [BadgeType](#) property. The badge supports the following eight different essential colors for various situations:

- Dark
- Error
- Info
- Light
- Primary
- Secondary
- Success
- Warning

### XML

```
<badge:SfBadgeView BadgeText="8" HorizontalOptions="Center"
VerticalOptions="Center">
  <badge:SfBadgeView.Content>
    <Image Source="BadgeFacebook.png" HeightRequest="70" WidthRequest="60"/>
  </badge:SfBadgeView.Content>
  <badge:SfBadgeView.BadgeSettings>
    <badge:BadgeSetting BadgeType="Error" Offset="-5,10"/>
  </badge:SfBadgeView.BadgeSettings>
</badge:SfBadgeView>
```

### C#

```
SfBadgeView sfBadgeView = new SfBadgeView();
sfBadgeView.HorizontalOptions = LayoutOptions.Center;
sfBadgeView.VerticalOptions = LayoutOptions.Center;
sfBadgeView.BadgeText = "8";
Image image = new Image();
image.Source = "BadgeFacebook.png";
image.HeightRequest = 70;
image.WidthRequest = 60;
sfBadgeView.Content = image;
BadgeSetting badgeSetting = new BadgeSetting();
badgeSetting.BadgeType = BadgeType.Error;
badgeSetting.Offset = new Point(-5, 10);
```

```
sfBadgeView.BadgeSettings = badgeSetting;  
Content = sfBadgeView;
```



### Badge background customization

Set the [BadgeType](#) to **None**. You can customize the color of the badge view using the [BackgroundColor](#) property.

#### XML

```
<badge:SfBadgeView BadgeText="48" HorizontalOptions="Center"  
VerticalOptions="Center" >  
  <badge:SfBadgeView.Content>  
    <Button Text="Primary" WidthRequest="120" HeightRequest="60"/>  
  </badge:SfBadgeView.Content>  
  <badge:SfBadgeView.BadgeSettings>  
    <badge:BadgeSetting BadgeType="None" BackgroundColor="Green" />  
  </badge:SfBadgeView.BadgeSettings>  
</badge:SfBadgeView>
```

#### C#

```
SfBadgeView sfBadgeView = new SfBadgeView();  
sfBadgeView.HorizontalOptions = LayoutOptions.Center;  
sfBadgeView.VerticalOptions = LayoutOptions.Center;  
sfBadgeView.BadgeText = "48";  
Button button = new Button();  
button.Text = "Primary";  
button.HeightRequest = 60;  
button.WidthRequest = 120;  
sfBadgeView.Content = button;  
BadgeSetting badgeSetting = new BadgeSetting();  
badgeSetting.BadgeType = BadgeType.None;  
badgeSetting.BackgroundColor = Color.Green;  
sfBadgeView.BadgeSettings = badgeSetting;  
Content = sfBadgeView;
```



Setting corner radius of the badge

The [CornerRadius](#) property is used to reduce the radius of the corners.

#### **XML**

```
<badge:SfBadgeView BadgeText="100" HorizontalOptions="Center"
VerticalOptions="Center" >
  <badge:SfBadgeView.Content>
    <Button Text ="Primary" WidthRequest="120" HeightRequest="60"/>
  </badge:SfBadgeView.Content>
  <badge:SfBadgeView.BadgeSettings>
    <badge:BadgeSetting CornerRadius="5"/>
  </badge:SfBadgeView.BadgeSettings>
</badge:SfBadgeView>
```

#### **C#**

```
SfBadgeView sfBadgeView = new SfBadgeView();
sfBadgeView.HorizontalOptions = LayoutOptions.Center;
sfBadgeView.VerticalOptions = LayoutOptions.Center;
sfBadgeView.BadgeText = "100";
Button button = new Button();
button.Text = "Primary";
button.HeightRequest = 60;
button.WidthRequest = 120;
sfBadgeView.Content = button;
BadgeSetting badgeSetting = new BadgeSetting();
badgeSetting.CnerRadius = 5;
sfBadgeView.BadgeSettings = badgeSetting;
Content = sfBadgeView;
```



### Alignment of badge

You can align the badge view using the **Center**, **Start**, and **End** properties of [BadgeAlignment](#) property.

#### XML

```
<badge:SfBadgeView BadgeText="20" HorizontalOptions="Center"
VerticalOptions="Center" >
  <badge:SfBadgeView.Content>
    <Label Text="START" BackgroundColor="LightGray"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
WidthRequest="100" HeightRequest="60" TextColor="Black" />
  </badge:SfBadgeView.Content>
  <badge:SfBadgeView.BadgeSettings>
    <badge:BadgeSetting BadgeAlignment="Center" CornerRadius="0"/>
  </badge:SfBadgeView.BadgeSettings>
</badge:SfBadgeView>
```

#### C#

```
SfBadgeView sfBadgeView = new SfBadgeView();
sfBadgeView.HorizontalOptions = LayoutOptions.Center;
sfBadgeView.VerticalOptions = LayoutOptions.Center;
sfBadgeView.BadgeText = "20";
Label label = new Label();
label.Text = "START";
label.HeightRequest = 60;
label.WidthRequest = 100;
label.BackgroundColor = Color.LightGray;
label.HorizontalTextAlignment = TextAlignment.Center;
label.VerticalTextAlignment = TextAlignment.Center;
label.TextColor = Color.Black;
sfBadgeView.Content = label;
BadgeSetting badgeSetting = new BadgeSetting();
badgeSetting.BadgeAlignment = BadgeAlignment.Start;
badgeSetting.CornerRadius = 0;
sfBadgeView.BadgeSettings = badgeSetting;
Content = sfBadgeView;
```



### Position customization

The default position of notification is **TopRight**. You can change the position to the **TopLeft**, **BottomLeft**, and **BottomRight** using the [BadgePosition](#) properties.

#### XML

```
<badge:SfBadgeView BadgeText="NEW" HorizontalOptions="Center"
VerticalOptions="Center">
  <badge:SfBadgeView.Content>
    <Button Text="Primary" WidthRequest="120" HeightRequest="60"/>
  </badge:SfBadgeView.Content>
  <badge:SfBadgeView.BadgeSettings>
    <badge:BadgeSetting BadgePosition="TopRight"/>
  </badge:SfBadgeView.BadgeSettings>
</badge:SfBadgeView>
```

#### C#

```
SfBadgeView sfBadgeView = new SfBadgeView();
sfBadgeView.HorizontalOptions = LayoutOptions.Center;
sfBadgeView.VerticalOptions = LayoutOptions.Center;
sfBadgeView.BadgeText = "NEW";
Button button = new Button();
button.Text = "Primary";
button.WidthRequest = 120;
button.HeightRequest = 60;
sfBadgeView.Content = button;
BadgeSetting badgeSetting = new BadgeSetting();
badgeSetting.BadgePosition = BadgePosition.TopRight;
sfBadgeView.BadgeSettings = badgeSetting;
Content = sfBadgeView;
```



### Setting badge offset

You can adjust the badge text using the [Offset](#) property.

**XML**

```
<badge:SfBadgeView BadgeText="8" HorizontalOptions="Center"
VerticalOptions="Center">
  <badge:SfBadgeView.Content>
    <Image Source="BadgeImage9.png" HeightRequest="70" WidthRequest="60"/>
  </badge:SfBadgeView.Content>
  <badge:SfBadgeView.BadgeSettings>
    <badge:BadgeSetting BadgeType="Success" Offset="-5,-10"
BadgePosition="BottomRight"/>
  </badge:SfBadgeView.BadgeSettings>
</badge:SfBadgeView>
```

**C#**

```
SfBadgeView sfBadgeView = new SfBadgeView();
sfBadgeView.HorizontalOptions = LayoutOptions.Center;
sfBadgeView.VerticalOptions = LayoutOptions.Center;
sfBadgeView.BadgeText = "8";
Image image = new Image();
image.Source = "BadgeImage9.png";
image.HeightRequest = 70;
image.WidthRequest = 60;
sfBadgeView.Content = image;
BadgeSetting badgeSetting = new BadgeSetting();
badgeSetting.BadgeType = BadgeType.Success;
badgeSetting.BadgePosition = BadgePosition.BottomRight;
badgeSetting.Offset = new Point(-5, -10);
sfBadgeView.BadgeSettings = badgeSetting;
Content = sfBadgeView;
```

**Predefined symbols**

You can change the badge icon using the [Badgelcon](#) property. Badge icons will be visible, when you won't set the badge text. The badge supports the following types of [Badgelcon](#):

- Add
- Available
- Away
- Busy
- Delete

- Dot
- None
- Prohibit1
- Prohibit2

### XML

```
<badge:SfBadgeView HorizontalOptions="Center" VerticalOptions="Center" >
  <badge:SfBadgeView.Content>
    <Image Source="BadgeImage9.png" HeightRequest="70" WidthRequest="60"/>
  </badge:SfBadgeView.Content>
  <badge:SfBadgeView.BadgeSettings>
    <badge:BadgeSetting BadgeType="Warning" Offset="0, -10"
      BadgePosition="BottomRight" BadgeIcon="Away"/>
  </badge:SfBadgeView.BadgeSettings>
</badge:SfBadgeView>
```

### C#

```
SfBadgeView sfBadgeView = new SfBadgeView();
sfBadgeView.HorizontalOptions = LayoutOptions.Center;
sfBadgeView.VerticalOptions = LayoutOptions.Center;
Image image = new Image();
image.Source = "BadgeImage9.png";
image.HeightRequest = 70;
image.WidthRequest = 60;
sfBadgeView.Content = image;
BadgeSetting badgeSetting = new BadgeSetting();
badgeSetting.BadgeType = BadgeType.Warning;
badgeSetting.BadgeIcon = BadgeIcon.Away;
badgeSetting.BadgePosition = BadgePosition.BottomRight;
badgeSetting.Offset = new Point(0, -10);
sfBadgeView.BadgeSettings = badgeSetting;
Content = sfBadgeView;
```



Away



Available



Add



Busy



Delete



Dot



Prohibit1



Prohibit2



## Animation

You can enable or disable the animation of the badge text using `Scale` or `None` properties of [BadgeAnimation](#) property. You can see the animation when we change the badge text.

### XML

```
<badge:SfBadgeView HorizontalOptions="Center" BadgeText="6"  
VerticalOptions="Center">  
  <badge:SfBadgeView.Content>  
    <Image Source="BadgeFacebook.png" HeightRequest="70" WidthRequest="70" />  
  </badge:SfBadgeView.Content>  
  <badge:SfBadgeView.BadgeSettings>  
    <badge:BadgeSetting BadgeType="Error" BadgeAnimation="Scale" Offset="-12,6"  
      BadgePosition="TopRight" />  
  </badge:SfBadgeView.BadgeSettings>  
</badge:SfBadgeView>
```

### C#

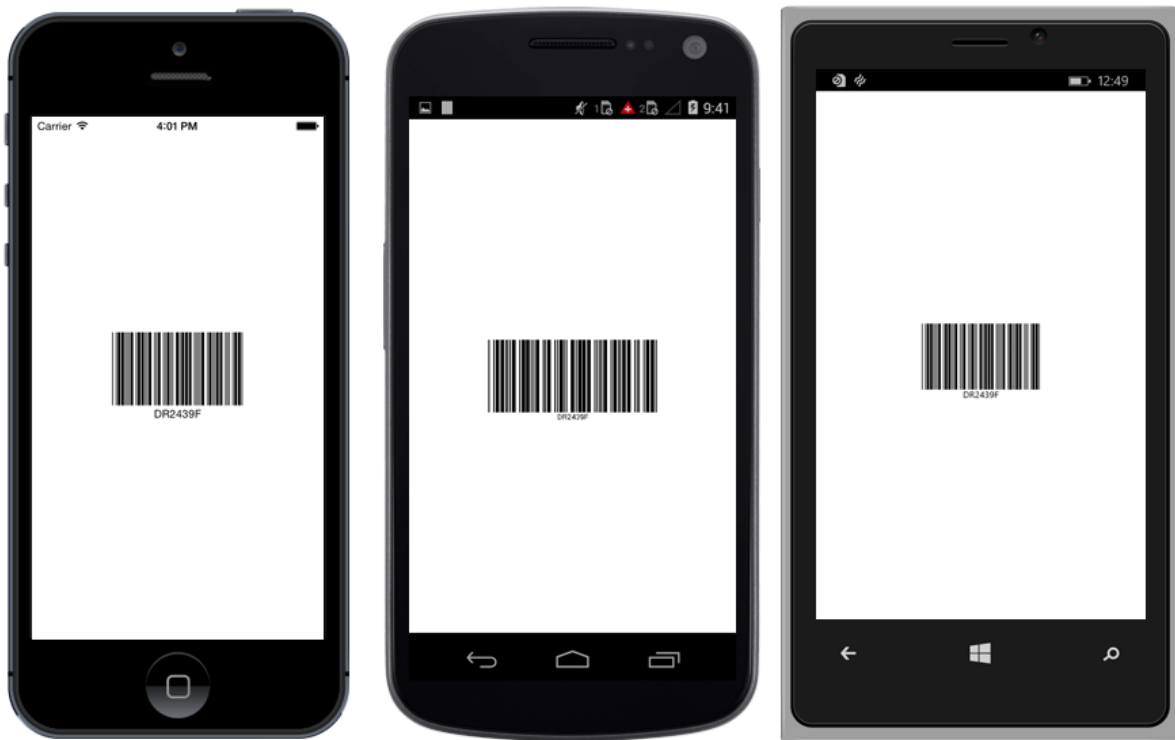
```
SfBadgeView sfBadgeView = new SfBadgeView();  
sfBadgeView.HorizontalOptions = LayoutOptions.Center;  
sfBadgeView.VerticalOptions = LayoutOptions.Center;  
sfBadgeView.BadgeText = "6";  
Image image = new Image();  
image.Source = "BadgeFacebook.png";  
image.HeightRequest = 70;  
image.WidthRequest = 70;  
sfBadgeView.Content = image;  
BadgeSetting badgeSetting = new BadgeSetting();  
badgeSetting.BadgeType = BadgeType.Error;  
badgeSetting.BadgeAnimation = BadgeAnimation.Scale;  
badgeSetting.Offset = new Point(-12, 6);  
badgeSetting.BadgePosition = BadgePosition.TopRight;  
sfBadgeView.BadgeSettings = badgeSetting;  
Content = sfBadgeView;
```



## SfBarcode

### OVERVIEW

**Essential Barcode** for Xamarin.Forms provides a perfect approach to encode texts using supported symbol types that comprises one dimensional barcodes and two dimensional barcodes. The basic structure of a barcode consists of one or more data characters, optionally one or two check characters, a start pattern as well as a stop pattern.



### Getting Started

This section explains how to configure a Barcode for Xamarin.Forms application. The following screenshot illustrates the final output of barcode on iOS, Android and Windows Phone devices.



To get started with Essential Barcode, go through the following steps.

#### Adding SfBarcode reference

You can add SfBarcode reference using one of the following methods:

##### Method 1: Adding SfBarcode reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](http://nuget.org). To add SfBarcode to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfBarcode](#), and then install it.

![Adding SfBarcode reference from nuget](Getting-Started\_images/getting-started/Adding SfBarcode reference.png)

---

**Note:** Install the same version of SfBarcode NuGet in all the projects.

---

##### Method 2: Adding SfBarcode reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfBarcode control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

##### Method 3: Adding SfBarcode assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfBarcode.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfBarcode.Android.dll Syncfusion.SfBarcode.XForms.Android.dll Syncfusion.SfBarcode.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfBarcode.iOS.dll Syncfusion.SfBarcode.XForms.iOS.dll Syncfusion.SfBarcode.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfBarcode.UWP.dll Syncfusion.SfBarcode.XForms.UWP.dll Syncfusion.SfBarcode.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** If you are adding the references from toolbox, this step is not needed.

Currently an additional step is required for Windows Phone and iOS projects. Create an instance of the Barcode custom renderer as mentioned.

Create an instance of `SfBarcodeRenderer` in MainPage constructor in of the Windows Phone project as shown

### C#

```
public MainPage ()
{
    ...
    new SfBarcodeRenderer();
    InitializeComponent();
    ...
}
```

Similarly, create an instance of `SfBarcodeRenderer` in Finished Launching overridden method of `AppDelegate` class in iOS Project as follows.

#### C#

```
public override bool Finished Launching (UIApplication app, NSDictionary options)
{
    ...
    new SfBarcodeRenderer ();
    return base.FinishedLaunching(app, options);
    ...
}
```

#### Configure the Barcode control

You can configure the Barcode control entirely in C# code or by using the XAML markup.

Here, the following steps illustrates how to create and configure a barcode.

1. Add reference to SfBarcode such as follows.

#### XML

```
xmlns:syncfusion="clr-namespace:Syncfusion.SfBarcode.XForms;assembly=Syncfusion.SfBarcode.XForms"
```

#### C#

```
using Syncfusion.SfBarcode.XForms;
```

2. Create an instance of `SfBarcode` in XAML or code-behind using the reference of `SfBarcode`.

#### XML

```
<syncfusion:SfBarcode/>
```

#### C#

```
SfBarcode barcode = new SfBarcode();
```

3. Then, you can assign the text that you want to encode.

#### XML

```
<syncfusion:SfBarcode Text="http://www.syncfusion.com"/>
```

#### C#

```
barcode.Text = " http://www.syncfusion.com ";
```

- Specify the required Symbology to encode the given text. By default, the given text is encoded using Code 39 Symbology.

**XML**

```
<syncfusion:SfBarcode Text="http://www.syncfusion.com" Symbology="QRCode"/>
```

**C#**

```
barcode.Symbology = BarcodeSymbolType.QRCode;
```

- For customizing the barcode, initialize the settings of corresponding barcode symbology.

**XML**

```
<syncfusion:SfBarcode Text="http://www.syncfusion.com" Symbology="QRCode">
  <syncfusion:SfBarcode.SymbologySettings>
    <Syncfusion:SfQRBarcodeSettings XDimension="6"/>
  </syncfusion:SfBarcode.SymbologySettings>
</syncfusion:SfBarcode>
```

**C#**

```
SfQRBarcodeSettings settings = new SfQRBarcodeSettings();
settings.XDimension = 6;
barcode.SymbologySettings = settings;
```

- Finally, the barcode is generated as displayed in the following screenshot for the following code example.

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:syncfusion="clr-namespace:Syncfusion.SfBarcode.XForms;assembly=Syncfusion.SfBarcode.XForms"
  x:Class="BarcodeGettingStarted.SamplePage">
  <syncfusion:SfBarcode BackgroundColor="Gray" Text="www.wikipedia.org"
    Symbology="QRCode">
    <syncfusion:SfBarcode.SymbologySettings>
      <syncfusion:QRBarcodeSettings XDimension="6"/>
    </syncfusion:SfBarcode.SymbologySettings>
  </syncfusion:SfBarcode> </ContentPage>
```

**C#**

```
public SamplePage()
{
  InitializeComponent();
  SfBarcode barcode = new SfBarcode();
  barcode.BackgroundColor = Color.Gray;
```

```
barcode.Text = "http://www.syncfusion.com";  
barcode.Symbology = BarcodeSymbolType.QRCode;  
QRBarcodeSettings settings = new QRBarcodeSettings();  
settings.XDimension = 6;  
barcode.SymbologySettings = settings;  
this.Content = barcode;  
}
```



## SUPPORTED SYMBOLOGIES

Essential Barcode supports 10 variants of one dimensional and 2 variants of two dimensional Barcodes that are illustrated as follows.

### One Dimensional Barcodes

One dimensional Barcode is also called as linear Barcode. The bars and spaces signified for each symbol in one dimensional Barcodes are grouped in such a way to represent a specific ASCII character.

- Codabar
- Code 11
- Code 32
- Code 39
- Code 39 Extended
- Code 93
- Code 93 Extended
- Code 128A
- Code 128B



- Code 128C

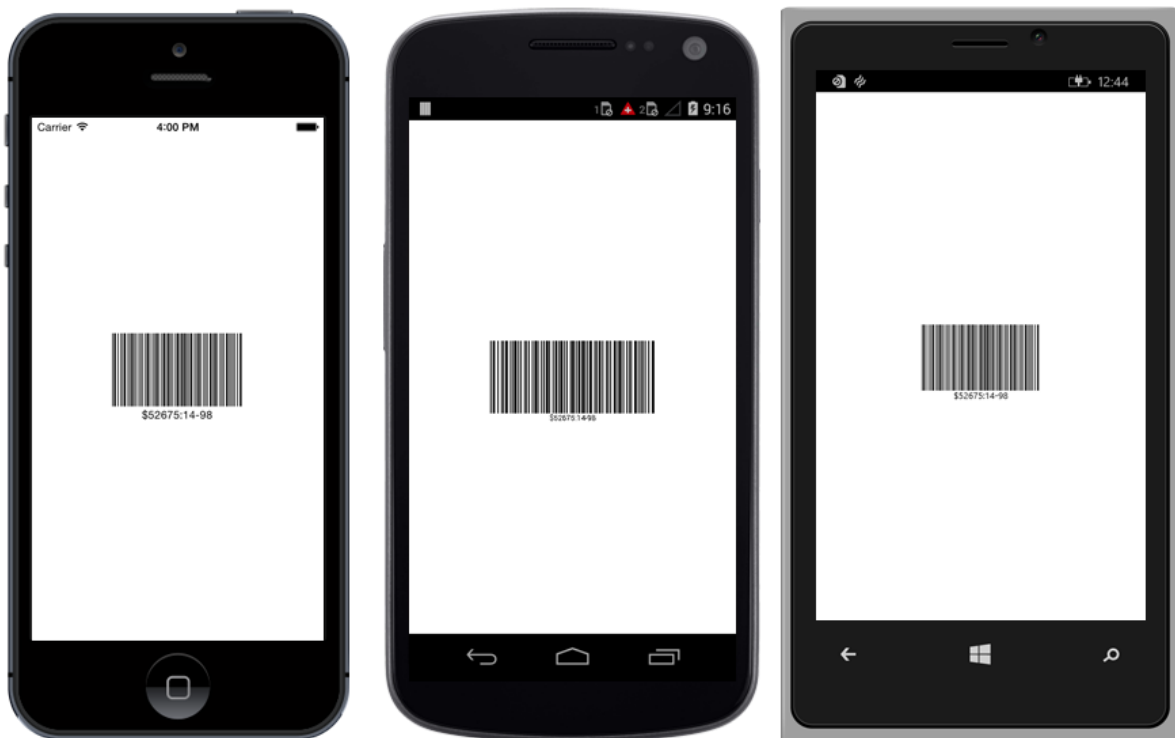
### Codabar

**Codabar** is a discrete numeric symbology that is used in libraries, blood banks and a variety of other information processing applications.

- Encodes only numeric characters and some special characters like dash (-), colon (:), slash (/), plus (+).
- Each character has three bars and four spaces.
- Uses characters of A, B, C, D as start and stop characters.

### C#

```
barcode.Symbology = BarcodeSymbolType.CodaBar;
```



Codabar barcode

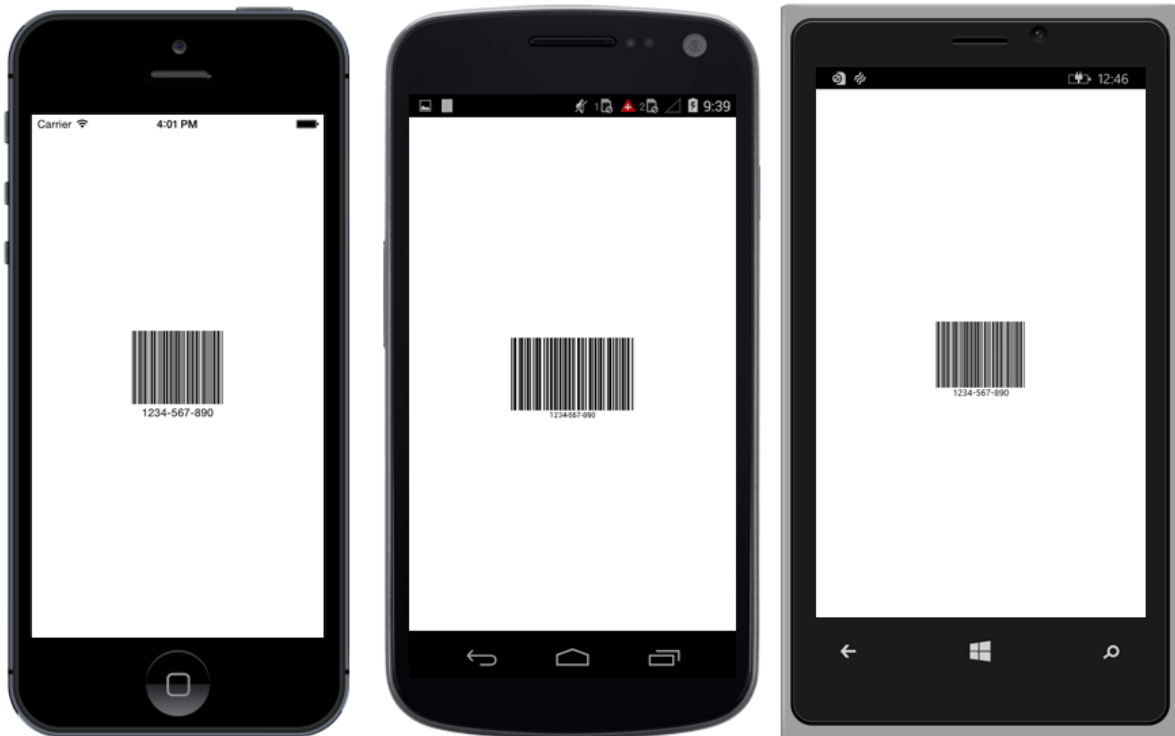
### Code 11

**Code11** Symbology is used mainly for labeling the telecommunications equipment and it has the following structure.

- Allows character set of digits (0-9), dash (-).
- Each character is encoded with 3 bars and 2 spaces.
- Of these five elements, there may be two wide and three narrow elements or one wide and four narrow elements.

**C#**

```
barcode.Symbology = BarcodeSymbolType.Code11;
```



Code 11 barcode

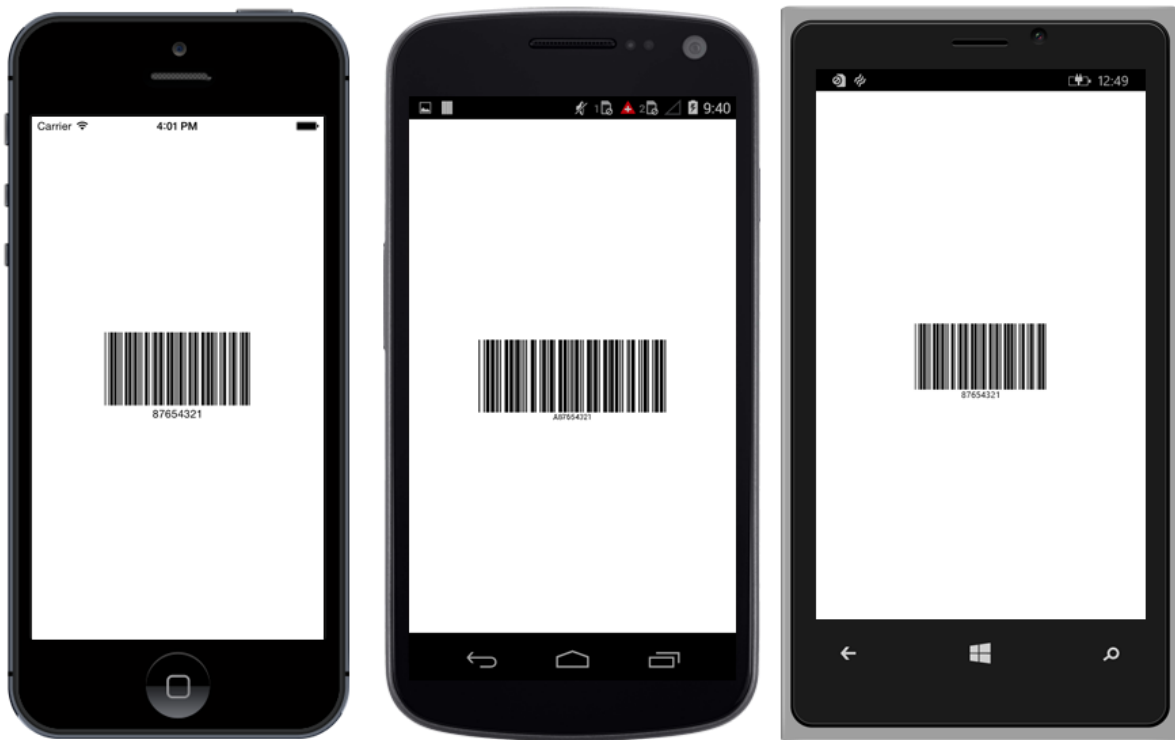
[Code 32](#)

**Code32** is mainly used for coding pharmaceuticals, cosmetics and dietetics and it contains the following structure:

- Starts with 'A' character (ASCII 65) that is not really encoded.
- Encodes only the character set of length 8.
- One digit for Checksum module 10 that is automatically calculated.

**C#**

```
barcode.Symbology = BarcodeSymbolType.Code32;
```



## Code32 barcode

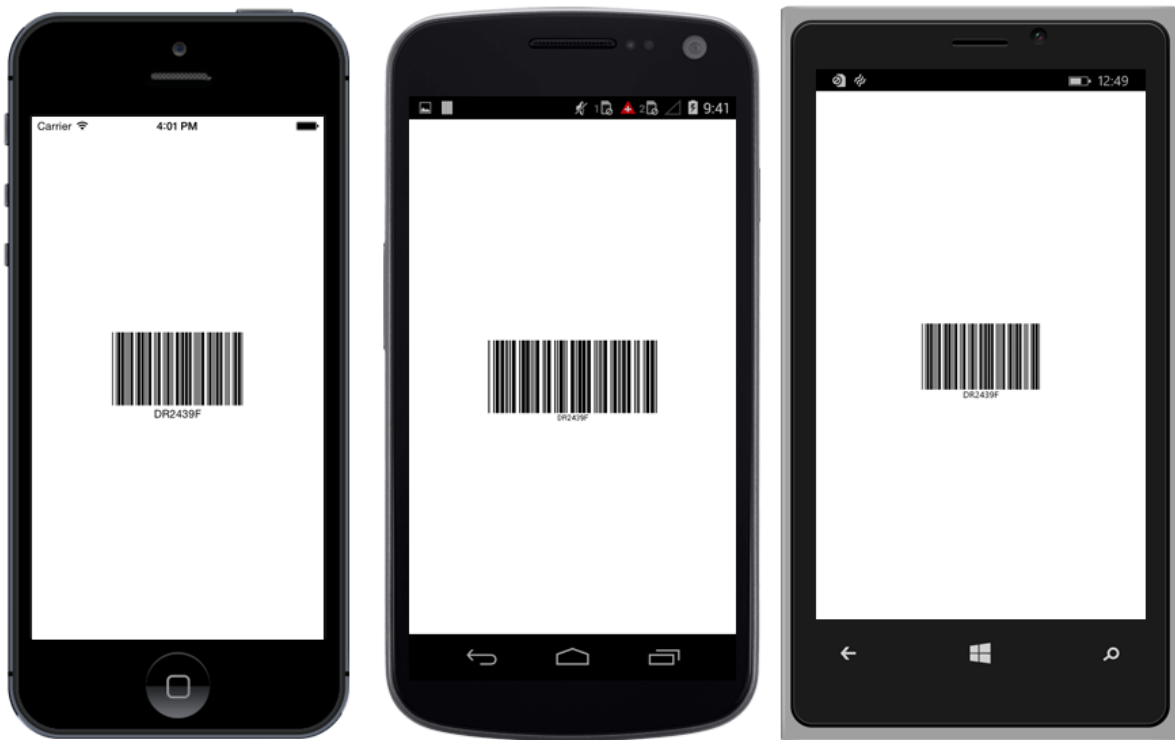
### [Code 39](#)

**Code39** is a Symbology of Barcode that encodes alphanumeric characters into a series of bars. It may be of any length, although more than 25 characters begin to push the bounds. This Symbology is the only type of the Barcode in common use that does not require a checksum.

- Allows character set of digits (0-9), upper case alphabets (A-Z), and symbols like space, minus (-), plus (+), period (.), dollar sign (\$), slash (/), and percent (%).
- Always starts and ends with an asterisk (\*) symbol, termed as start and stop character.
- Each character is encoded with 5 bars and 4 spaces where 3 are wide and 6 are narrow.

### **C#**

```
barcode.Symbology = BarcodeSymbolType.Code39;
```



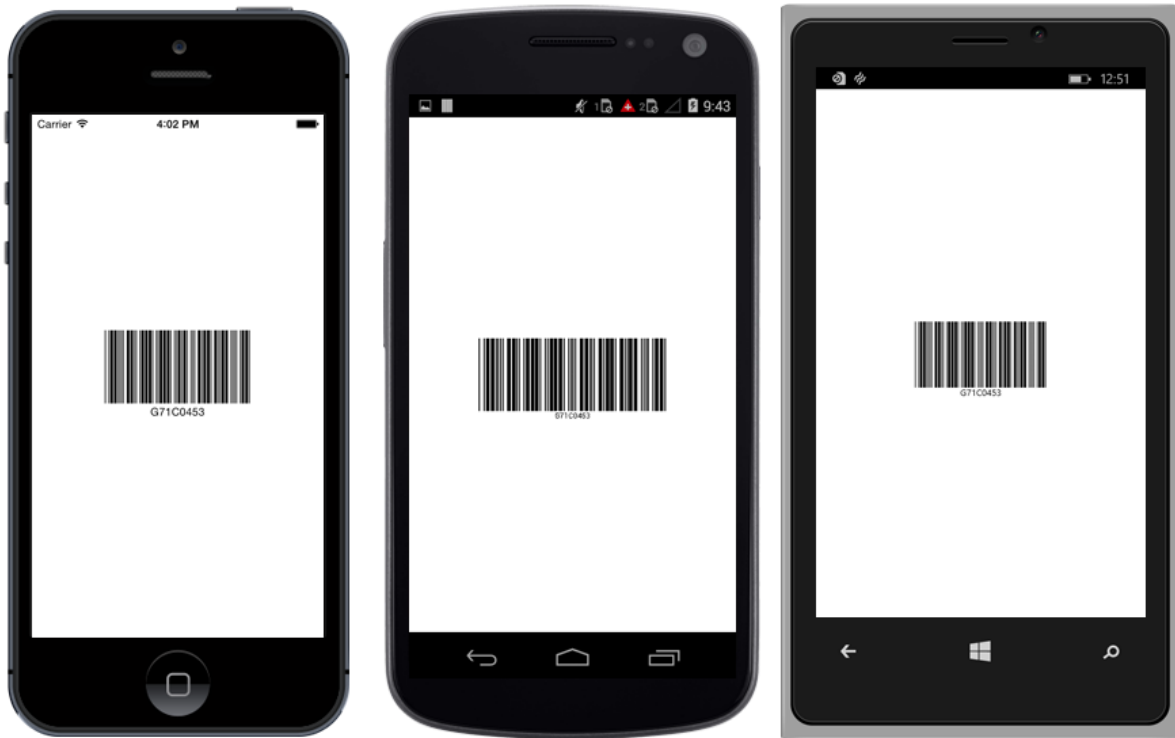
Code 39 barcode

[Code 39 Extended](#)

**Code39 Extended** Symbology is an extended version of **Code39** that supports full ASCII character set. So, it encodes lower case alphabets (a-z) as well as special characters in the keyboard.

**C#**

```
barcode.Symbology = BarcodeSymbolType.Code39Extended;
```



### Extended Code 39 barcode

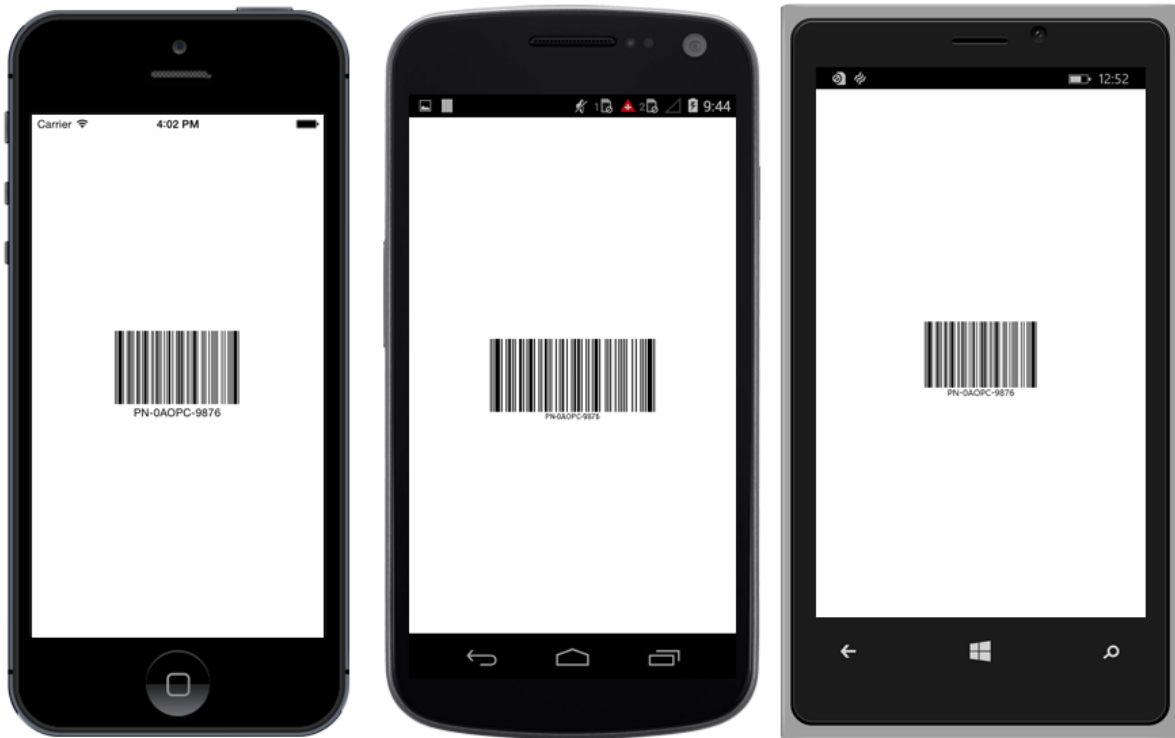
#### [Code 93](#)

**Code93** is designed to complement and improve upon **Code39**. It represents the full ASCII character set by using the combination of 2 characters. It is a continuous, variable-length symbology and it produces denser code.

- Encodes character set of uppercase alphabets (A-Z), digits (0-9), and special characters like asterisk (\*), dash (-), dollar (\$), percent (%), Space, dot (.), slash (/), and plus (+).
- The asterisk (\*) is not a true encoding character, but it is the start and stop symbol for **Code93 Symbology**.

#### **C#**

```
barcode.Symbology = BarcodeSymbolType.Code93;
```



## Code93 barcode

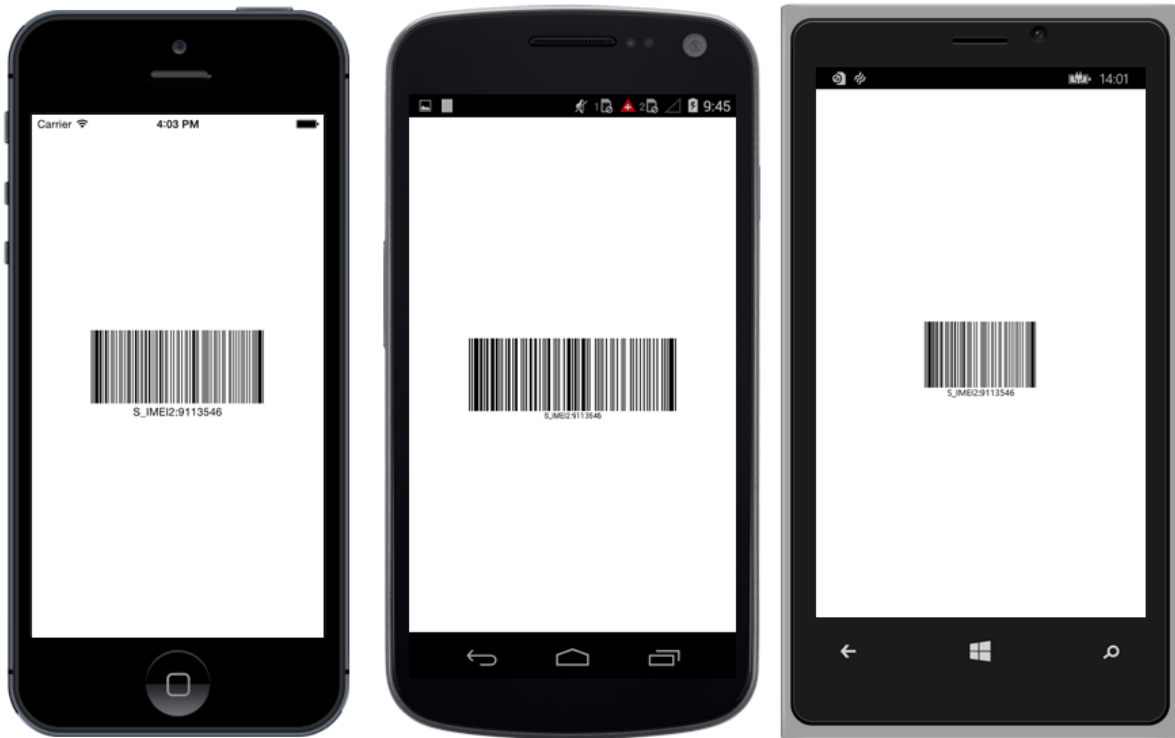
### *Code 93 Extended*

**Code93** is designed to complement and improve upon Code 39. It represents the full ASCII character set by using the combination of 2 characters. It is a continuous, variable-length Symbology and it produces denser code.

- Encodes character set of uppercase alphabets (A-Z), digits (0-9), and special characters like asterisk (\*), dash (-), dollar (\$), percent (%), Space, dot (.), slash (/), and plus (+).
- The asterisk (\*) is not a true encoding character, but it is the start and stop symbol for Code 93 symbology.

### **C#**

```
barcode.Symbology = BarcodeSymbolType.Code93Extended;
```



Code93 Extended barcode

#### [Code 128](#)

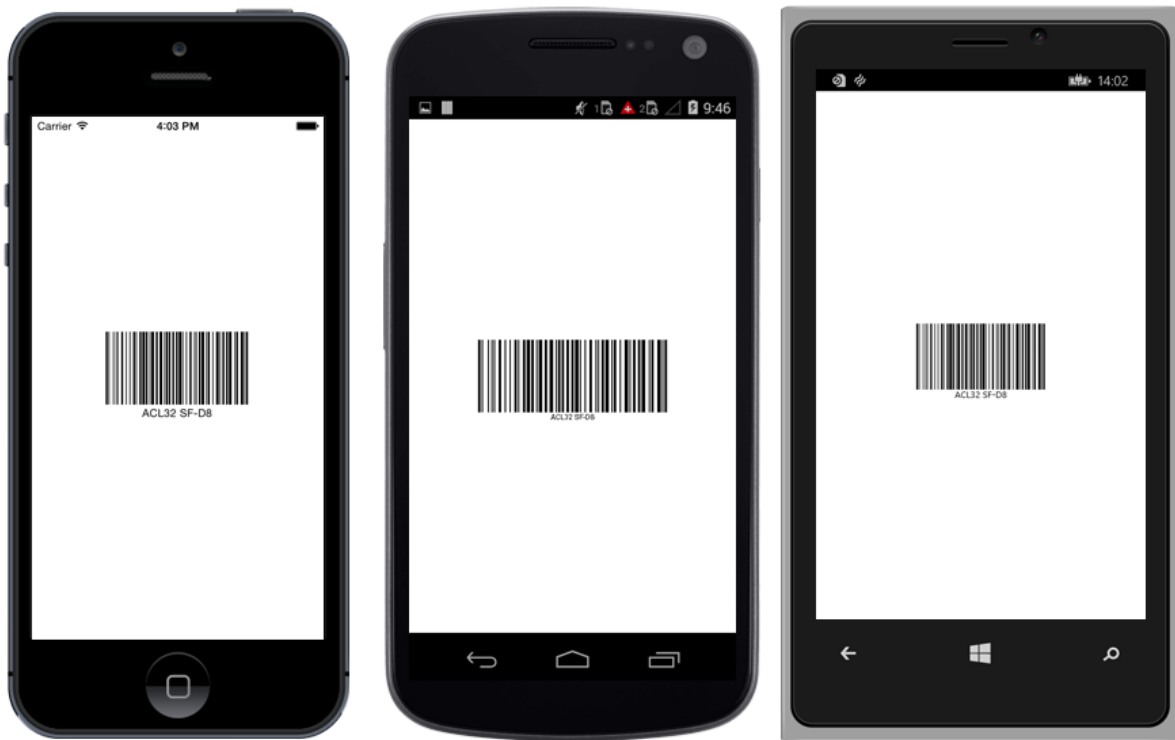
**Code128** is a variable length, high density, alphanumeric, linear Barcode Symbology. It is capable of encoding full ASCII character set and extended character sets. This symbol includes a checksum digit for verification and the Barcode may also be verified character-by-character for parity of each data byte.

#### [Code 128 A](#)

**Code128A** (or Chars Set A) includes all the standard upper case U.S. alphanumeric keyboard characters and punctuation characters together with the control characters, (characters with ASCII values from 0 to 95 inclusive), and seven special characters.

#### **C#**

```
barcode.Symbology = BarcodeSymbolType.Code128A;
```



Code128A barcode

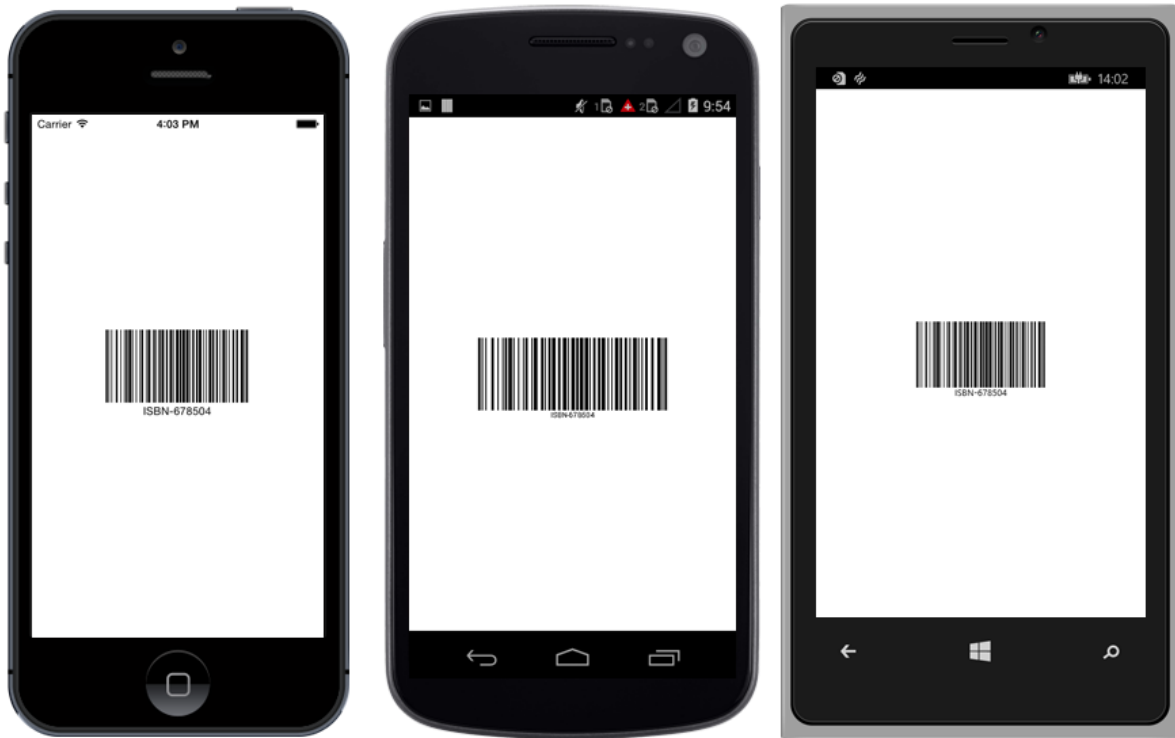
[Code128 B](#)

**Code128B** (or Chars Set B) includes all the standard upper case alphanumeric keyboard characters and punctuation characters together with the lower case alphabetic characters (characters with ASCII values from 32 to 127 inclusive), and seven special characters.

**C#**

```
barcode.Symbology = BarcodeSymbolType.Code128B;
```





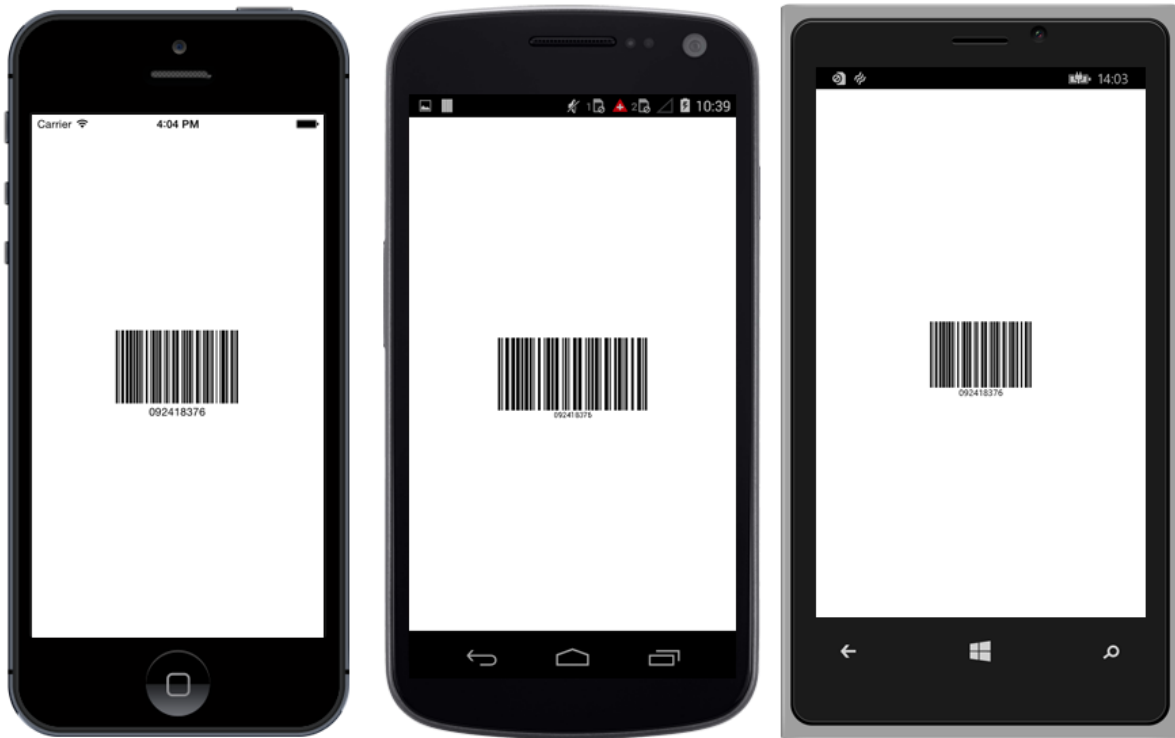
#### Code128B barcode

##### [Code128 C](#)

**Code128C** (or Chars Set C) includes a set of 100 digit pairs from 00 to 99 inclusive, as well as three special characters. This allows numeric data to be encoded as two data digits per symbol character effectively twice the density of standard data.

##### **C#**

```
barcode.Symbology = BarcodeSymbology.Code128C;
```



## Code128C barcode

### *Code 128 Special characters*

The last seven characters of Code Sets A and B (character values 96 - 102) and the last three characters of Code Set C (character values 100 - 102) are special non-data characters with no ASCII character equivalents that have particular significance to the Barcode reading device.

---

#### **NOTE:**

---

When you specify that the data must be encoded by using Char Set C, then the number of characters after it must be even.

### *Two Dimensional Barcodes*

Two dimensional Barcode is a way to represent information by using two-dimensional approach. It is similar to one dimensional Barcode, but can represent more data per unit area.

- QR Code
- Data Matrix

### *QR Code*

QR Code is a two dimensional symbology that is popularly used in automotive industry. It is known for fast readability and greater storage capacity.

#### **C#**

```
barcode.Symbology = BarcodeSymbolType.QRCode;
```



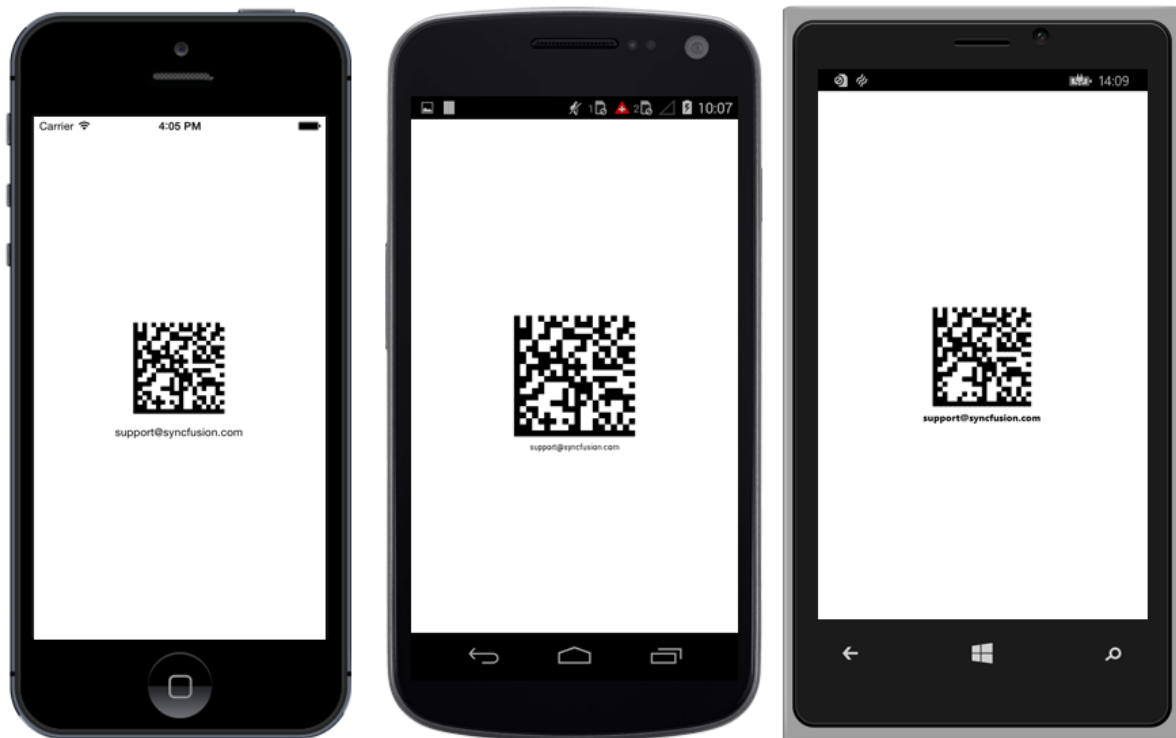
## QR bar code

### *Data Matrix*

DataMatrix symbology is widely used in printed media such as labels and letters. It can be read easily by a Barcode reader and also by mobile phones. It consists of a Grid of dark and light dots or blocks forming square or rectangular symbol. The data encoded in the Barcode can be either number or alphanumeric.

### **C#**

```
barcode.Symbology = BarcodeSymbolType.DataMatrix;
```



Data Matrix bar code

---

**NOTE:**

---

By default, the width of the quiet zone on all four sides of the Barcode is equal to the dimension of the blocks.

### SYMBOLGY SETTINGS

Each and every Barcode **Symbology** can be personalized with optional settings that may affect the appearance of specific Barcode.

#### One Dimensional Barcode settings

One dimensional Barcodes can be customized by using the following properties and they are commonly used for all the categories of supported one dimensional Barcodes.

- By setting the **BarHeight** property, the height of the linear bars can be changed.
- By setting the **NarrowBarWidth** property, the width ratio of the wide and narrow bars can be customized.

One dimensional Barcodes can also have error detection settings.

- By enabling the **EncodeStartStopSymbols** property, start and stop symbols are added to signal a Barcode reader that a Barcode has been scanned.
- The **EnableCheckDigit** property enables or disables the redundancy check by using a check digit that is the decimal equivalent of a binary parity bit.

- With the help of `ShowCheckDigit` property, the check digit can be shown or hidden.

The following code example shows how to change the settings of the `Code39` linear Barcode.

#### XML

```
// Changes the settings of Code 39 linear Barcode.
<sync:SfBarcode.SymbologySettings>
<sync:Code39Setting BarHeight="45" EnableCheckDigit="False"
EncodeStartStopSymbols="True" NarrowBarWidth="1" ShowCheckDigit="False"/>
</sync:SfBarcode.SymbologySettings>
```

Similarly, you can specify the settings of other linear Barcodes corresponding to specified symbology.

#### Two Dimensional Barcode Settings

Two dimensional Barcodes are customized by using the `XDimension` property that modifies its block size and it is commonly used for all kinds of supported two dimensional Barcodes.

#### Data Matrix Settings

The `DataMatrixSetting` can be customized with the help of the following settings.

#### XML

```
//Changes the settings of Data Matrix Barcode.
<sync:SfBarcode.SymbologySettings>
sync:DataMatrixSetting XDimension="8" Encoding="ASCIINumeric"
Size="Size104x104" />
</sync:SfBarcode.SymbologySettings>
```

- The `Encoding` property is used to specify the encoding technique from enumeration of `DataMatrixEncoding` that contains encoding techniques such as ASCII, ASCIINumeric, Auto and Base256.
- The `Size` property allows you to specify the size of the Barcode from a set of predefined sizes available in the `DataMatrixSize` enumeration.

Matrix Size	Description
Auto	Size is chosen based on the input data
Size10x10	Square matrix with 10 rows and 10 columns.
Size12x12	Square matrix with 12 rows and 12 columns.
Size14x14	Square matrix with 14 rows and 14 columns.
Size16x16	Square matrix with 16 rows and 16 columns.
Size18x18	Square matrix with 18 rows and 18 columns.
Size20x20	Square matrix with 20 rows and 20 columns.
Size22x22	Square matrix with 22 rows and 22 columns.

Size24x24	Square matrix with 24 rows and 24 columns.
Size26x26	Square matrix with 26 rows and 26 columns.
Size32x32	Square matrix with 32 rows and 32 columns.
Size36x36	Square matrix with 36 rows and 36 columns.
Size40x40	Square matrix with 40 rows and 40 columns.
Size44x44	Square matrix with 44 rows and 44 columns.
Size48x48	Square matrix with 48 rows and 48 columns.
Size52x52	Square matrix with 52 rows and 52 columns.
Size64x64	Square matrix with 64 rows and 64 columns.
Size72x72	Square matrix with 72 rows and 72 columns.
Size80x80	Square matrix with 80 rows and 80 columns.
Size88x88	Square matrix with 88 rows and 88 columns.
Size96x96	Square matrix with 96 rows and 96 columns.
Size104x104	Square matrix with 104 rows and 104 columns.
Size120x120	Square matrix with 120 rows and 120 columns.
Size132x132	Square matrix with 132 rows and 132 columns.
Size144x144	Square matrix with 144 rows and 144 columns.
Size8x18	Rectangular matrix with 8 rows and 18 columns.
Size8x32	Rectangular matrix with 8 rows and 32 columns.
Size12x26	Rectangular matrix with 12 rows and 26 columns.
Size12x36	Rectangular matrix with 12 rows and 36 columns.
Size16x36	Rectangular matrix with 16 rows and 36 columns.
Size16x48	Rectangular matrix with 16 rows and 48 columns.

### QR Code Settings

The **QRCode** symbology can be customized by using the following settings.

### XML

```
//Changes the settings of QR Barcode.
<sync:SfBarcode.SymbologySettings>
  <sync:QRBarcodeSetting XDimension="8" ErrorCorrectionLevel="High"
    InputMode="BinaryMode" Version="Auto" />
</sync:SfBarcode.SymbologySettings>
```

- The **Version** property allows you to set various types of version for QR code from **QRVersion** enumeration. By default, its value is set as Auto.

QRVersion	Description
Version01	Measures 21 x 21 modules
Version02	Measures 25 x 25 modules
Version03	Measures 29 x 29 modules
Version04	Measures 33 x 33 modules
Version05	Measures 37 x 37 module
Version06	Measures 41 x 41 modules
Version07	Measures 45 x 45 modules
Version08	Measures 49 x 49 modules
Version09	Measures 53 x 53 modules
Version10	Measures 57 x 57 modules
Version11	Measures 61 x 61 modules
Version12	Measures 65 x 65 modules
Version13	Measures 69 x 69 modules
Version14	Measures 73 x 73 modules
Version15	Measures 77 x 77 modules
Version16	Measures 81 x 81 modules
Version17	Measures 85 x 85 modules
Version18	Measures 89 x 89 modules
Version19	Measures 93 x 93 modules
Version20	Measures 97 x 97 modules
Version21	Measures 101 x 101 modules
Version22	Measures 105 x 105 modules
Version23	Measures 109 x 109 modules
Version24	Measures 113 x 113 modules
Version25	Measures 117 x 117 modules
Version26	Measures 121 x 121 modules

Version27	Measures 125 x 125 modules
Version28	Measures 129 x 129 modules
Version29	Measures 133 x 133 modules
Version30	Measures 137 x 137 modules
Version31	Measures 141 x 141 modules
Version32	Measures 145 x 145 modules
Version33	Measures 149 x 149 modules
Version34	Measures 153 x 153 modules
Version35	Measures 157 x 157 modules
Version36	Measures 161 x 161 modules
Version37	Measures 165 x 165 modules
Version38	Measures 169 x 169 modules
Version39	Measures 173 x 173 modules
Version40	Measures 177 x 177 modules

- The **ErrorCorrectionLevel** property employs error correction to generate a series of error correction code words that are added to the data code word sequence in order to enable the symbol to withstand damage without loss of data. By default, its value is set as Low.

Error Correction Level	Recovery Capacity % (approx.)
Low	7
Medium	15
Quartile	25
High	30

- The **InputMode** property allows you to select specific set of input characters. You may select the most suitable input mode. By default, its value is set as BinaryMode.

Input Mode	Possible characters
NumericMode	0,1,2,3,4,5,6,7,8,9
AlphanumericMode	0 to 9, A to Z, space, \$, %, *, +, -,., /, :



BinaryMode	Shift JIS characters
------------	----------------------

## BARCODE CUSTOMIZATION

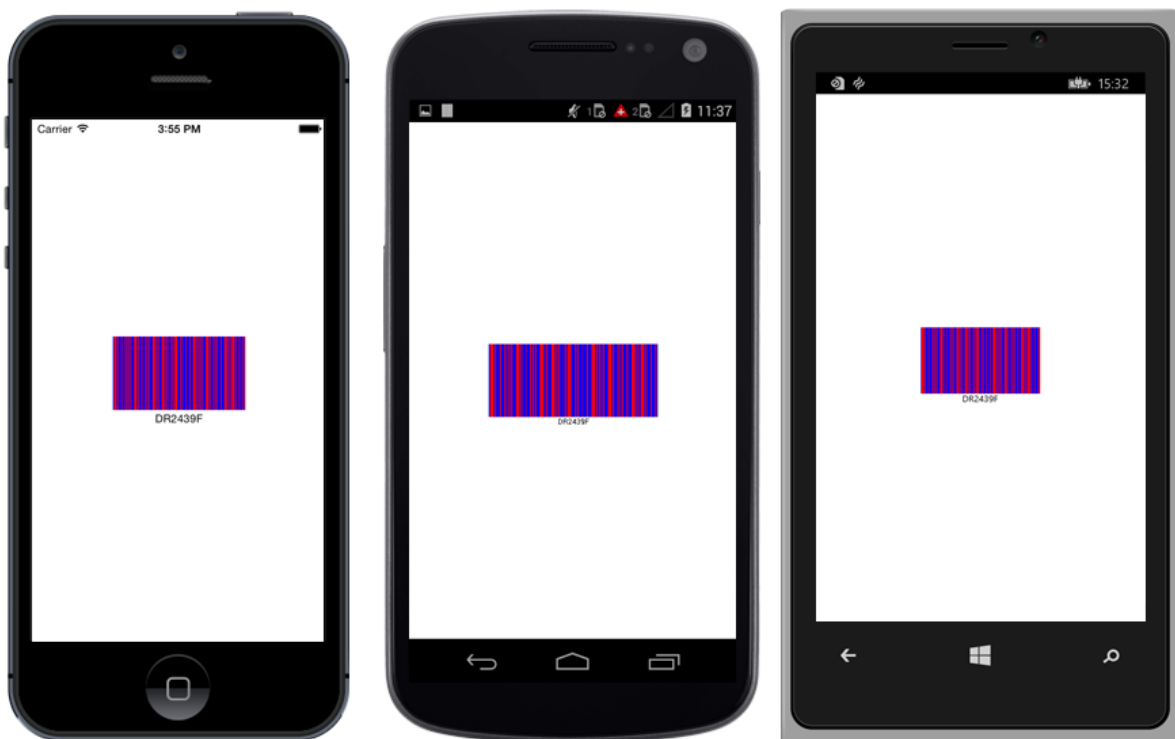
### Bar Customization

The color of the Barcode can be customized by using the properties of **DarkBarBrush** and **LightBarBrush** in the **SfBarcode**.

The **DarkBarBrush** represents the Color of the dark bar (Black color by default) and the **LightBarBrush** represents the color of the gap between two adjacent black bars (White color by default).

### C#

```
//Changes the color of darker area of Barcode.  
barcode.DarkBarColor = Color.Blue;  
//Changes the color of lighter area of Barcode.  
barcode.LightBarColor = Color.Red;
```



*Barcode with bar color customization*

### NOTE:

The **DarkBarBrush** and **LightBarBrush** customizations are applicable only for one dimensional Barcodes. In order, to recognize a Barcode symbol by a scanner, there must be an adequate contrast between the dark bars and the light spaces. All the Barcode scanners do not have support for colored Barcodes.

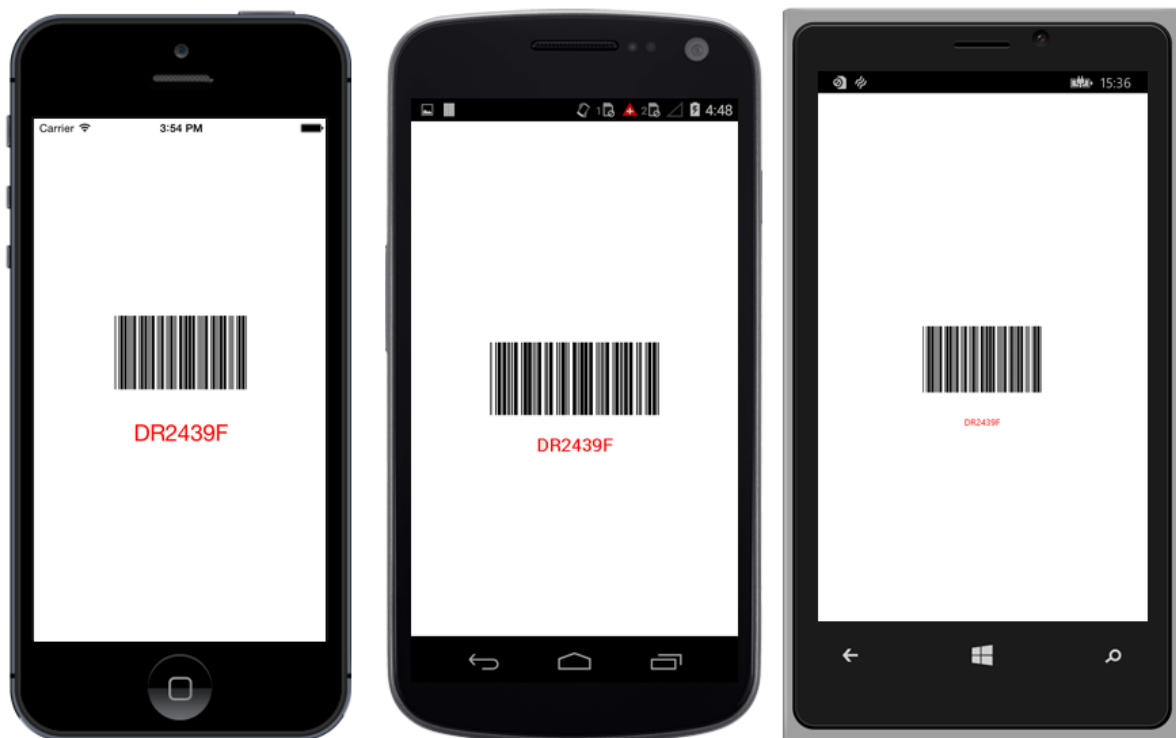
### Text Customization

The text representing the Barcode can be customized by using the following properties.

- The color of text can be altered by using the `TextColor` property.
- The size of text can be altered by using the `TextSize` property.
- The horizontal alignment of text can be customized with the help of the `TextPosition` property.
- The gap between Barcode and text can be adjusted by setting the property of `TextGapHeight`.
- To change the location of text vertically, you can make use of the `TextLocation` property with options of top and bottom location.

### C#

```
//Changes the color of text.
barcode.TextColor = Color.Red;
//Changes the size of text.
barcode.TextFont = Font.SystemFontOfSize(25);
//Changes the font style of text.
Typeface textStyle = Typeface.create("Times new roman",1);
barcode.TextFont = textStyle**;**
//Changes the height of gap between text and Barcode.
barcode.TextGapHeight = 30;
//Changes the location of text.
barcode.TextLocation = BarcodeTextLocation.Bottom;
//Changes the alignment of text.
barcode.TextAlignment = BarcodeTextAlignment.Center;
```



### Barcode with text customization

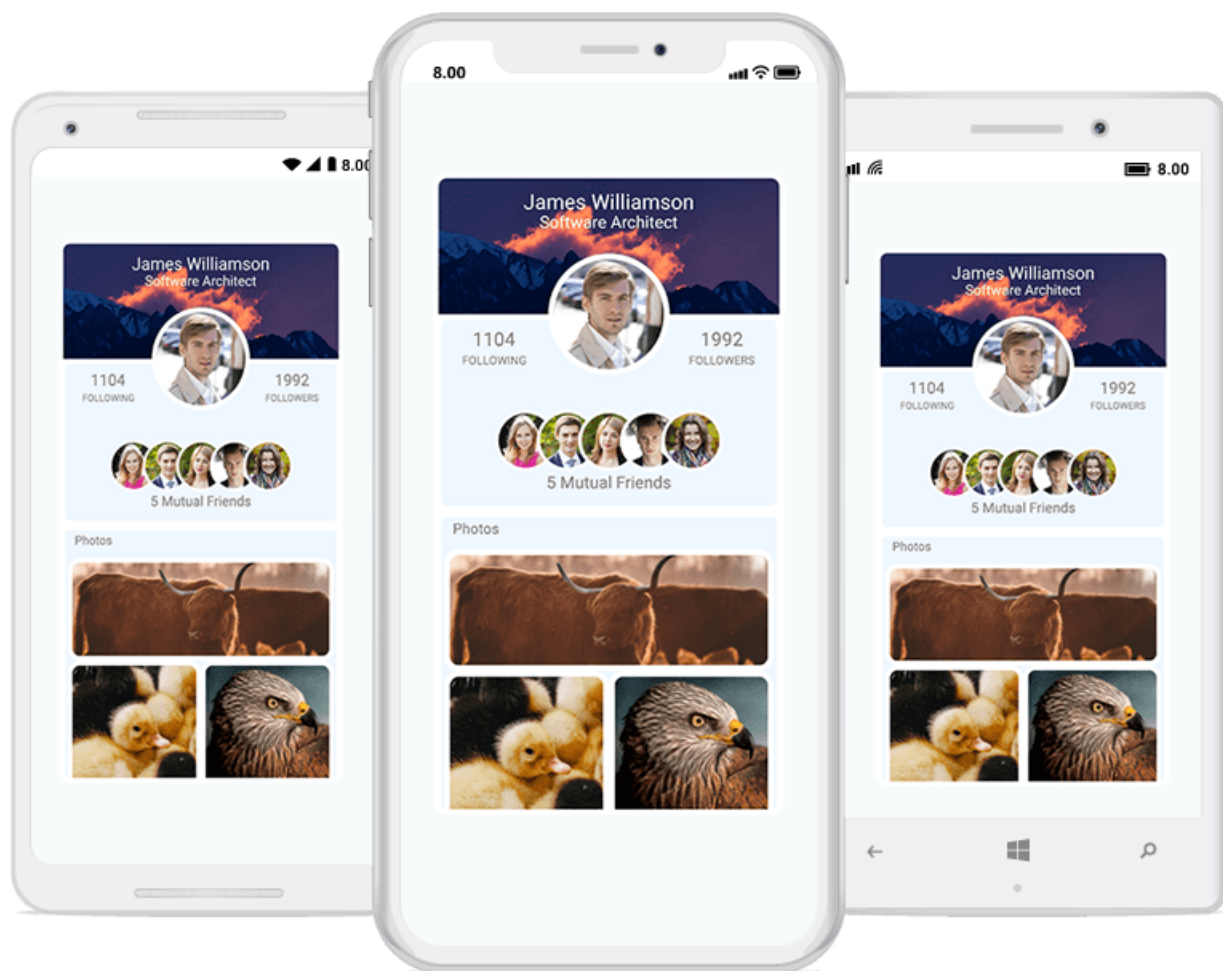
## SfBorder

### Overview

The Xamarin.Forms Border control is a container control that allows you to apply borders, backgrounds, border width, and corner radii

### Key Features

- Customize the background, border width, and border color of the Border control.
- Implement a circular image.
- Customize all the edges with different corner radii.



### Getting Started

This section provides an overview for working with the SfBorder control for Xamarin.Forms and explains the entire process of creating a real-world application.

### Adding SfBorder reference

You can add SfBorder reference using one of the following methods:

### Method 1: Adding SfBorder reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfBorder to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Core](#), and then install it.

![Xamarin Forms SfBorder Nuget reference](images/Adding SfBorder reference.png)

#### Note:

- Install the same version of SfBorder NuGet in all the projects.
- In addition, you need to install the [Syncfusion.Xamarin.Core.WPF]() package for Xamarin.Forms WPF platform only.

### Method 2: Adding SfBorder reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfBorder control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfBorder assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
WPF	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.WPF.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching an application on each platform with SfBorder.

To use the SfBorder control inside an application, each platform application must initialize the SfBorder renderer. This initialization step varies from platform to platform and is discussed in the following sections:

---

**Note:** If you are adding the references from toolbox, below steps are not needed.

---

#### *Android and UWP*

The Android and UWP launches the SfBorder without any initialization, and it is enough to only initialize the Xamarin.Forms Framework to launch the application.

#### *iOS*

To launch the SfBorder in iOS, call the `SfBorderRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework has been initialized and before the `LoadApplication` is called, as demonstrated in the following code example.

#### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.XForms.iOS.Border.SfBorderRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

#### *Release mode issue in UWP platform*

The known Framework issue in UWP platform is that the custom controls will not be rendered when deployed an application in Release Mode. It can be resolved by initializing the SfBorder assemblies in the `App.xaml.cs` file in the UWP project, as demonstrated in the following code example.

#### **C#**

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.Border.SfBorderRenderer)
        .GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

#### *Windows Presentation Foundation (WPF)*

To launch the border in WPF, call the `SfBorderRenderer.Init()` method in the `MainWindow` constructor of the `MainWindow` class after the Xamarin.Forms framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

**C#**

```
public partial class MainWindow : FormsApplicationPage
{
    public MainWindow()
    {
        InitializeComponent();
        Forms.Init();
        Syncfusion.XForms.WPF.Border.SfBorderRenderer.Init();
        LoadApplication(new App());
    }
}
```

## Creating SfBorder control

The **SfBorder** control is configured entirely in C# code or in XAML markup. The following steps explain how to create a **SfBorder** and configure its elements.

*Adding namespace for referred assemblies*

**XML**

```
xmlns:border="clr-
namespace:Syncfusion.XForms.Border;assembly=Syncfusion.Core.XForms"
```

**C#**

```
using Syncfusion.XForms.Border;
```

*Add the SfBorder control as the content of ContentPage.*

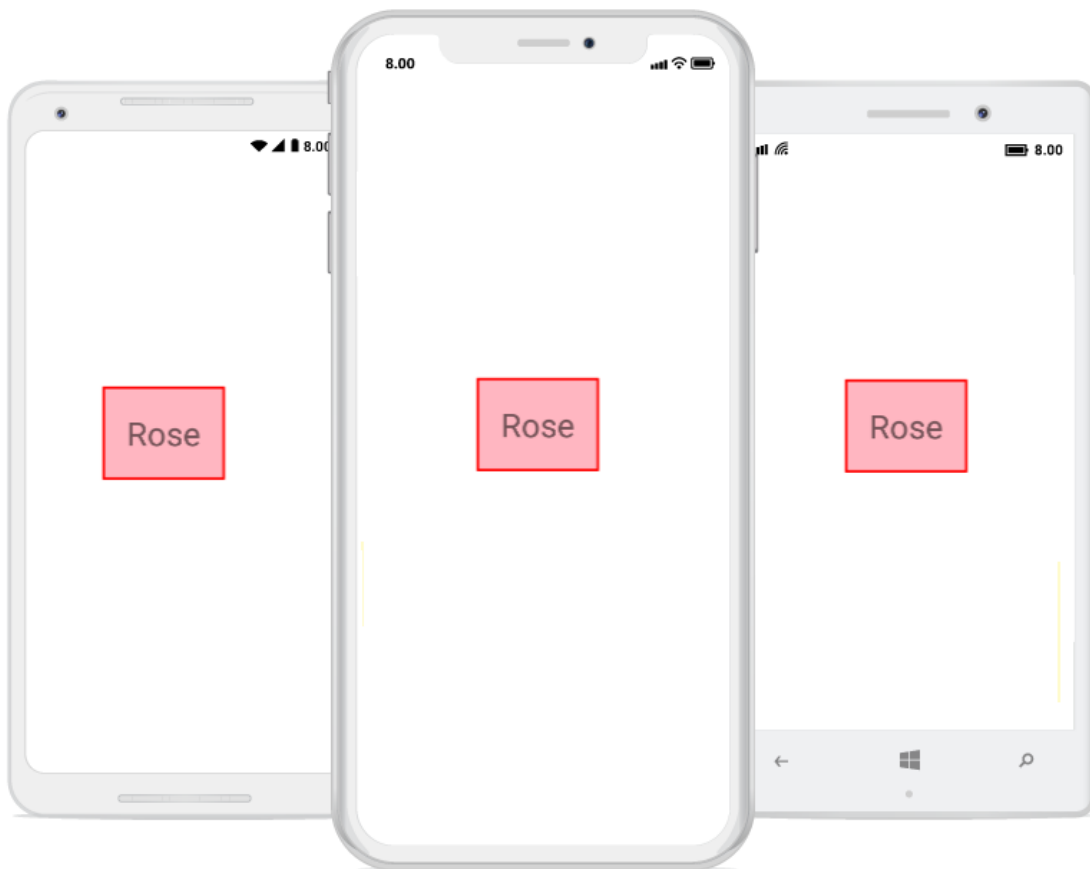
**XML**

```
<Grid>
<border:SfBorder
    BorderColor="Red"
    BackgroundColor="#ffb6c1"
    HorizontalOptions="Center"
    VerticalOptions="Center"
    BorderWidth="3">
    <Label
        Text="Rose"
        Margin="10"
        Font="15" />
    </border:SfBorder>
</Grid>
```

**C#**

```
using System;
using Syncfusion.XForms.Border;
using Xamarin.Forms;
namespace BorderGettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
    }
}
```

```
{
InitializeComponent();
Grid mainGrid = new Grid();
// Create Border control
SfBorder border = new SfBorder();
border.VerticalOptions = LayoutOptions.Center;
border.HorizontalOptions = LayoutOptions.Center;
border.BorderColor = Color.Red;
border.BackgroundColor = Color.FromHex("#ffb6c1");
//Create Label control
Label label = new Label();
label.Text = "Rose";
label.FontSize = 15;
border.Content = label;
mainGrid.Children.Add(border);
this.Content = stack;
}
}
```



The complete Getting Started sample is available in [this](#) link.

### Customization

The border control supports customizing the border color, width, corner radius, background color, and more. The border can be customized using the following properties.

### Border color

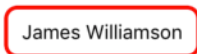
You can customize the color of the border using the **BorderColor** property.

#### XML

```
<border:SfBorder x:Name="border" BorderColor = "Red" CornerRadius="20"
BorderWidth="3">
<Label Text="James Williamson"
TextColor="Black"/>
</border:SfBorder>
```

#### C#

```
SfBorder border = new SfBorder();
border.CornerRadius = 20;
border.BorderColor = Color.Red;
border.BorderWidth = 3;
Label label = new Label();
label.Text = "James Williamson";
label.TextColor = Color.Black;
border.Content = label;
this.Content = border;
```



James Williamson

### Background color

The background color of the border control can be customized using the **BackgroundColor** property.

#### XML

```
<border:SfBorder x:Name="border" BorderColor = "Red" BackgroundColor=
"Green" CornerRadius="20" BorderWidth="3">
<Label Text="James Williamson"
TextColor="Black"/>
</border:SfBorder>
```

#### C#

```
SfBorder border = new SfBorder();
border.CornerRadius = 20;
border.BorderColor = Color.Red;
border.BorderWidth = 3;
Label label = new Label();
label.Text = "James Williamson";
label.TextColor = Color.Black;
border.Content = label;
border.BackgroundColor = Color.Green;
this.Content = border;
```





### Border width

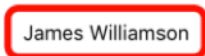
You can customize the thickness of the border using the `BorderWidth` property.

#### XML

```
<border:SfBorder x:Name="border" BorderColor = "Red" CornerRadius="20"
BorderWidth="12">
<Label Text="James Williamson"
TextColor="Black"/>
</border:SfBorder>
```

#### C#

```
SfBorder border = new SfBorder();
border.CnerRadius = 20;
border.BorderColor = Color.Red;
border.BorderWidth = 12;
Label label = new Label();
label.Text = "James Williamson";
label.TextColor = Color.Black;
border.Content = label;
this.Content = border;
```



### Corner radius

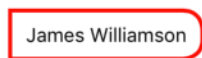
You can customize the corner radius of the border on four sides using the `CornerRadius` property with type as `Thickness`.

#### XML

```
<border:SfBorder x:Name="border" BorderColor = "Red"
CornerRadius="0,10,10,0" BorderWidth="12">
<Label Text="James Williamson"
TextColor="Black"/>
</border:SfBorder>
```

### C#

```
SfBorder border = new SfBorder();
border.CornerRadius = new Thickness (0,10,10,0);
border.BorderColor = Color.Red;
border.BorderWidth = 12;
Label label = new Label();
label.Text = "James Williamson";
label.TextColor = Color.Black;
border.Content = label;
this.Content = border;
```



### Adding circular image

You can add any view inside the border control by adding the Content property. The following code sample demonstrates how to apply border using the `CornerRadius` property for a circular image.

### XML

```
<Grid HeightRequest="100" WidthRequest="100" HorizontalOptions="Center"
VerticalOptions="Center">
  <border:SfBorder BorderColor="Black" HorizontalOptions="Center"
VerticalOptions="Center" CornerRadius="50">
    <Image Source="plus.jpeg" />
  </border:SfBorder>
</Grid>
```

### C#

```
Grid grid = new Grid();
grid.HeightRequest = 100;
grid.WidthRequest = 100;
grid.HorizontalOptions = LayoutOptions.Center;
grid.VerticalOptions = LayoutOptions.Center;
SfBorder border = new SfBorder();
border.BorderColor = Color.Black;
border.HorizontalOptions = LayoutOptions.Center;
border.VerticalOptions = LayoutOptions.Center;
border.CornerRadius = 50;
Image image = new Image();
image.Source = "plus.jpeg";
border.Content = image;
grid.Children.Add(border);
this.Content = grid;
```



### Shadow Effect

The border control provides shadow effect support. To enable shadow effect, set the `HasShadow` property to true.

You can customize the color of shadow using the `ShadowColor` property.

### XML

```
<border:SfBorder
  HorizontalOptions="Center"
  VerticalOptions="Center"
  BackgroundColor="Green"
  HeightRequest="50"
  WidthRequest="200"
  CornerRadius="20"
  BorderWidth="0"
  HasShadow="True"
  ShadowColor="Gray">
  <Label
    Text="James Williamson"
    HorizontalTextAlignment="Center"
    VerticalTextAlignment="Center"
    TextColor="White"/>
</border:SfBorder>
```

### C#

```
SfBorder border = new SfBorder()
{
  BackgroundColor = Color.Green,
  CornerRadius = 20,
  BorderWidth = 0,
  HeightRequest = 50,
  WidthRequest = 200,
  HorizontalOptions = LayoutOptions.Center,
  VerticalOptions = LayoutOptions.Center,
  HasShadow = true,
  ShadowColor = Color.Gray
};
Label label = new Label()
{
  Text = "James Williamson",
  HorizontalTextAlignment = TextAlignment.Center,
  VerticalTextAlignment = TextAlignment.Center,
  TextColor = Color.White
};
border.Content = label;
this.Content = border;
```

---

**Note:** Shadow support has not been provided for UWP Platform.

---

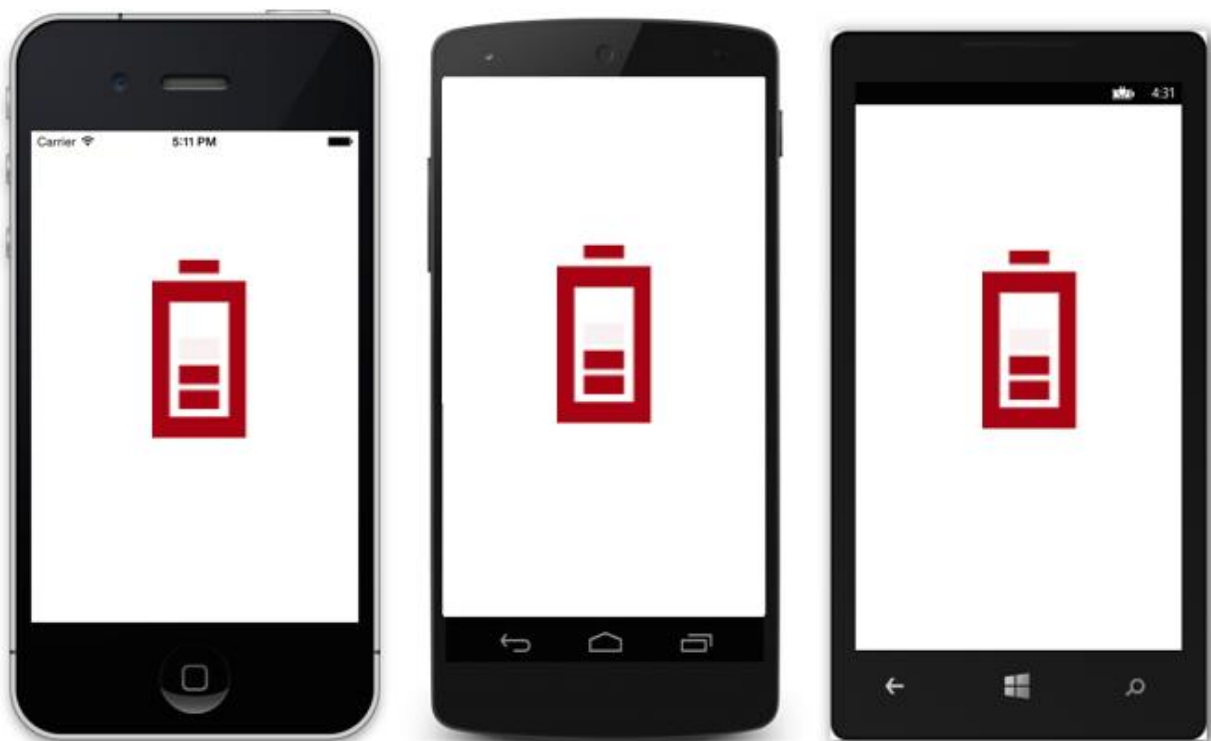


James Williamson

## SfBusyIndicator

### Overview

The Essential Xamarin BusyIndicator control includes over 10 built-in animations that can be displayed within your applications. It can be used to indicate busy status during app loading, data processing, etc.



## Key Features

- Provides 10 built-in busy indicator animations
- Supports to show/hide busy indicator.
- Displays busy text with animation.
- Provides simple and easy options to size the control.
- Provides option to adjust the animation speed.

## Getting Started

This section explains you the steps to configure a SfBusyIndicator control in a real-time scenario and also provides a walk-through on some of the customization features available in SfBusyIndicator control.

### Adding SfBusyIndicator reference

You can add SfBusyIndicator reference using one of the following methods:

#### Method 1: Adding SfBusyIndicator reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfBusyIndicator to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfBusyIndicator](#), and then install it.

![Adding SfBusyIndicator reference](images/Adding SfBusyIndicator reference.png)

---

**Note:** Install the same version of SfBusyIndicator NuGet in all the projects.

---

#### Method 2: Adding SfBusyIndicator reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfBusyIndicator control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfBusyIndicator assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfBusyIndicator.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfBusyIndicator.Android.dll Syncfusion.SfBusyIndicator.XForms.Android.dll Syncfusion.SfBusyIndicator.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfBusyIndicator.iOS.dll Syncfusion.SfBusyIndicator.XForms.iOS.dll Syncfusion.SfBusyIndicator.XForms.dll Syncfusion.Core.XForms.dll

	Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfBusyIndicator.UWP.dll Syncfusion.SfBusyIndicator.XForms.UWP.dll Syncfusion.SfBusyIndicator.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the SfBusyIndicator on each platform

To use SfBusyIndicator inside an application, each platform application must initialize the SfBusyIndicator renderer. This initialization step varies from platform to platform and is discussed in the following sections.

#### Android and UWP

The Android and UWP launches the SfBusyIndicator without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch SfBusyIndicator in iOS, need to create an instance of SfBusyIndicatorRenderer in FinishedLaunching overridden method of AppDelegate class in iOS Project as shown below.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    new SfBusyIndicatorRenderer();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### ReleaseMode issue in UWP platform

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the SfBusyIndicator assemblies in **App.xaml.cs** in UWP project as like in below code snippet.

#### C#

```
// In App.xaml.cs
```

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfBusyIndicatorRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Create a Simple SfBusyIndicator

The SfBusyIndicator control is configured entirely in C# code or by using XAML markup. The following steps explain on how to create a SfBusyIndicator and configure its elements,

- Adding namespace for the added assemblies.

#### XML

```
xmlns:busyindicator="clr-
namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndica
tor.XForms"
```

#### C#

```
using Syncfusion.SfBusyIndicator.XForms;
```

- Now add the SfBusyIndicator control with a required optimal name by using the included namespace.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-
namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndica
tor.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<busyindicator:SfBusyIndicator x:Name="busyindicator" />
</ContentPage.Content>
</ContentPage>
```

#### C#

```
using Syncfusion.SfBusyIndicator.XForms;
using Xamarin.Forms;
```

```

namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBusyIndicator busyIndicator = new SfBusyIndicator();
            this.Content = busyIndicator;
        }
    }
}

```

### Setting Animation Type

SfBusyIndicator provides 15 predefined animation types like Ball, Battery, Globe and so on. User can select any one of the animation types using `AnimationType` property.

Following example depicts the battery type animation for SfBusyIndicator.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"
x:Class="GettingStarted.MainPage">
    <ContentPage.Content>
        <busyindicator:SfBusyIndicator x:Name="busyindicator"
AnimationType="Battery"
ViewBoxWidth = "150"
ViewBoxHeight="150"
TextColor="Maroon" />
    </ContentPage.Content>
</ContentPage>

```

### C#

```

using Syncfusion.SfBusyIndicator.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBusyIndicator busyIndicator = new SfBusyIndicator()
            {
                AnimationType = AnimationTypes.Battery,
                ViewBoxHeight = 150,
                ViewBoxWidth = 150,
                TextColor = Color.Maroon
            };
        }
    }
}

```



```
this.Content = busyIndicator;  
}  
}
```

### EnableAnimation

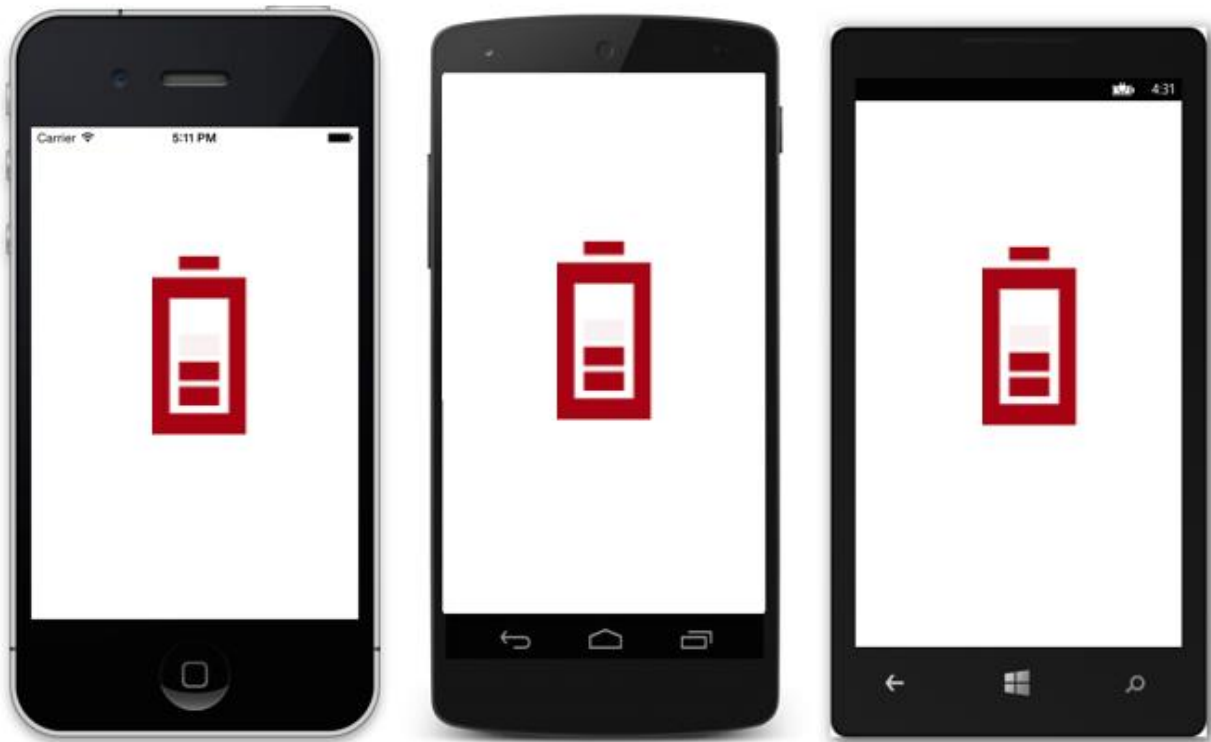
SfBusyIndicator provides options to enable or disable animation of the Busy Indicator. Animation can be enabled or disabled using the [EnableAnimation](#) property.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:local="clr-namespace:GettingStarted"  
xmlns:busyindicator="clr-  
namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndica  
tor.XForms"  
x:Class="GettingStarted.MainPage">  
<ContentPage.Content>  
<busyindicator:SfBusyIndicator x:Name="busyindicator"  
AnimationType="Battery"  
ViewBoxWidth = "150"  
ViewBoxHeight="150"  
EnableAnimation="True"/>  
</ContentPage.Content>  
</ContentPage>
```

### C#

```
using Syncfusion.SfBusyIndicator.XForms;  
using Xamarin.Forms;  
namespace GettingStarted  
{  
    public partial class MainPage : ContentPage  
    {  
        public MainPage()  
        {  
            InitializeComponent();  
            SfBusyIndicator busyIndicator = new SfBusyIndicator()  
            {  
                AnimationType = AnimationTypes.Ball,  
                ViewBoxHeight = 150,  
                ViewBoxWidth = 150,  
                EnableAnimation = true  
            };  
            this.Content = busyIndicator;  
        }  
    }  
}
```



You can find the complete getting started sample from this [link](#).

### Make Busy Animation Idle

SfBusyIndicator control provides support to determine whether an animation needs to be executed or not. Setting the `IsBusy` property to false will stop the animation and removes the control from view.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"
x:Class="GettingStarted.MainPage">
  <ContentPage.Content>
    <busyindicator:SfBusyIndicator x:Name="busyindicator"
    AnimationType="Ball"
    ViewBoxWidth = "150"
    ViewBoxHeight="150"
    TextColor="Maroon"
    IsBusy="False"/>
  </ContentPage.Content>
</ContentPage>
```

#### C#

```
using Syncfusion.SfBusyIndicator.XForms;
```

```

using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBusyIndicator busyIndicator = new SfBusyIndicator()
            {
                AnimationType = AnimationTypes.Ball,
                ViewBoxHeight = 150,
                ViewBoxWidth = 150,
                TextColor = Color.Maroon,
                IsBusy = false
            };
            this.Content = busyIndicator;
        }
    }
}

```

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

## Set Header

### Title

SfBusyIndicator provides option to set the text that indicates the information related to loading. This can be done using **Title** property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<busyindicator:SfBusyIndicator x:Name="busyindicator"
AnimationType="Ball"
ViewBoxWidth = "150"
Title="Loading..."
ViewBoxHeight="150"
TextColor="Maroon"/>
</ContentPage.Content>
</ContentPage>

```

### C#

```

using Syncfusion.SfBusyIndicator.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage

```

```

{
    public MainPage()
    {
        InitializeComponent();
        SfBusyIndicator busyIndicator = new SfBusyIndicator()
        {
            AnimationType = AnimationTypes.Ball,
            ViewBoxHeight = 150,
            ViewBoxWidth = 150,
            TextColor = Color.Maroon,
            Title = "Loading..."
        };
        this.Content = busyIndicator;
    }
}

```

### TextColor

SfBusyIndicator provides options to change the color of the text. The color of the text can be changed using the [TextColor](#) property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"
x:Class="GettingStarted.MainPage">
    <ContentPage.Content>
        <busyindicator:SfBusyIndicator x:Name="busyindicator"
AnimationType="Ball"
ViewBoxWidth = "150"
Title="Loading..."
ViewBoxHeight="150"
TextColor="Maroon"/>
    </ContentPage.Content>
</ContentPage>

```

### C#

```

using Syncfusion.SfBusyIndicator.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBusyIndicator busyIndicator = new SfBusyIndicator()
            {
                AnimationType = AnimationTypes.Ball,

```

```

ViewBoxHeight = 150,
ViewBoxWidth = 150,
TextColor = Color.Maroon,
Title = "Loading..."
};
this.Content = busyIndicator;
}
}
}

```

### TextSize

SfBusyIndicator provides options to change the size of the text. The size of the text can be changed using the [TextSize](#) property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-
namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndica
tor.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<busyindicator:SfBusyIndicator x:Name="busyindicator"
AnimationType="Battery"
ViewBoxWidth = "150"
Title="Loading..."
TextSize="10"
ViewBoxHeight="150"
TextColor="Maroon"/>
</ContentPage.Content>
</ContentPage>

```

### C#

```

using Syncfusion.SfBusyIndicator.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBusyIndicator busyIndicator = new SfBusyIndicator()
            {
                AnimationType = AnimationTypes.Ball,
                ViewBoxHeight = 150,
                ViewBoxWidth = 150,
                TextColor = Color.Maroon,
                Title = "Loading...",
                TextSize = 10
            };
        }
    }
}

```

```

this.Content = busyIndicator;
}
}
}

```

### TitlePlacement

SfBusyIndicator provides options to set the **Title** at the top or bottom of the Busy Indicator. The **Title** can be set using the [TitlePlacement](#) property. When the **Title** is not needed, set the **TitlePlacement** property of SfBusyIndicator to None.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-
namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndica
tor.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<busyindicator:SfBusyIndicator x:Name="busyindicator"
AnimationType="Battery"
ViewBoxWidth = "150"
Title="Loading..."
TitlePlacement="None"
ViewBoxHeight="150"
TextColor="Maroon"/>
</ContentPage.Content>
</ContentPage>

```

### C#

```

using Syncfusion.SfBusyIndicator.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBusyIndicator busyIndicator = new SfBusyIndicator()
            {
                AnimationType = AnimationTypes.Ball,
                ViewBoxHeight = 150,
                ViewBoxWidth = 150,
                TextColor = Color.Maroon,
                Title = "Loading...",
                TitlePlacement= TitlePlacement.Top
            };
            this.Content = busyIndicator;
        }
    }
}

```



## Sizing

Drawing size can be customized in SfBusyIndicator. `ViewBoxHeight` and `ViewBoxWidth` properties can be used to set height and width of the Indicator.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-
namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndica
tor.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<busyindicator:SfBusyIndicator x:Name="busyindicator"
AnimationType="SlicedCircle"
ViewBoxWidth = "20"
ViewBoxHeight="20"
TextColor="Maroon"/>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using Syncfusion.SfBusyIndicator.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBusyIndicator busyIndicator = new SfBusyIndicator()
            {
                AnimationType = AnimationTypes.SlicedCircle,
                ViewBoxHeight = 20,
                ViewBoxWidth = 20,
                TextColor = Color.Maroon
            };
            this.Content = busyIndicator;
        }
    }
}
```

![Height and width](images/heightand width.png)

## Animation Type

The **AnimationType** property for the SfBusyIndicator allows the user to set one of the 15 animations from the built-in animations.

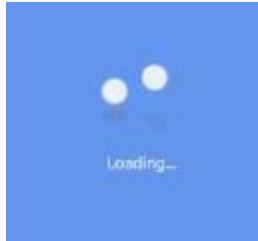
### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-
namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndica
tor.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<busyindicator:SfBusyIndicator x:Name="busyindicator"
Title="Loading..."
AnimationType="Ball"
ViewBoxHeight="100"
ViewBoxWidth="100"
BackgroundColor="Blue"
TextColor="White"/>
</ContentPage.Content>
</ContentPage>
```

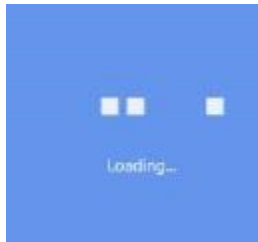
### C#

```
using Syncfusion.SfBusyIndicator.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBusyIndicator busyIndicator = new SfBusyIndicator()
            {
                AnimationType = AnimationTypes.Ball,
                ViewBoxWidth = 100,
                ViewBoxHeight = 100,
                Title = "Loading...",
                BackgroundColor="Blue",
                TextColor = Color.White
            };
            this.Content = busyIndicator;
        }
    }
}
```

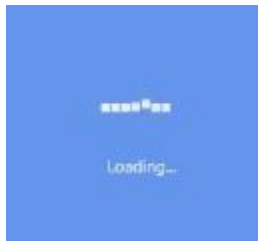




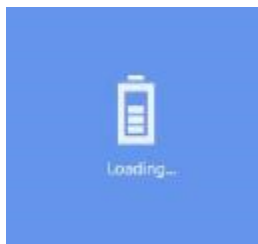
Busy Indicator with Ball type animation



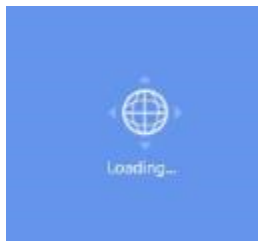
Busy Indicator with HorizontalPulsingBox type animation



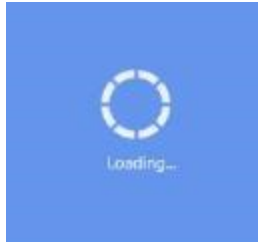
Busy Indicator with Rectangle type animation



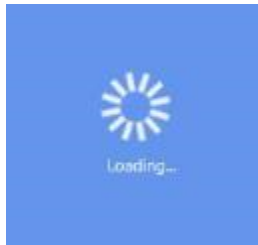
Busy Indicator with Battery type animation



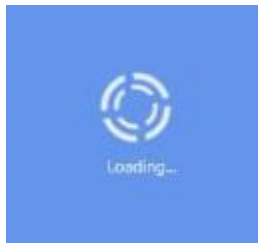
Busy Indicator with Globe type animation



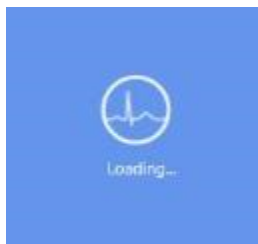
Busy Indicator with SingleCircle type animation



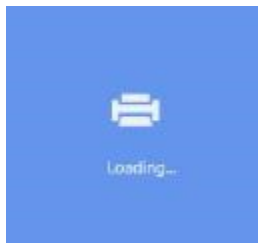
Busy Indicator with SlicedCircle type animation



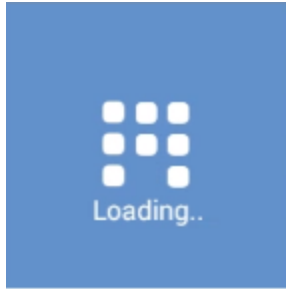
Busy Indicator with DoubleCircle type animation



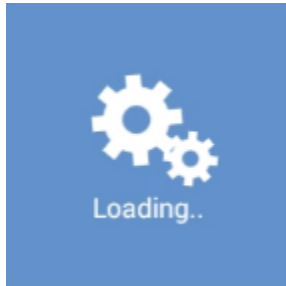
Busy Indicator with ECG type animation



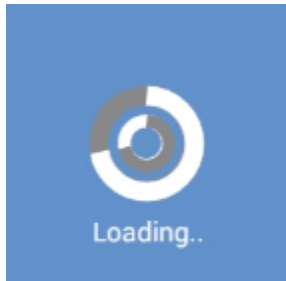
Busy Indicator with Print type animation



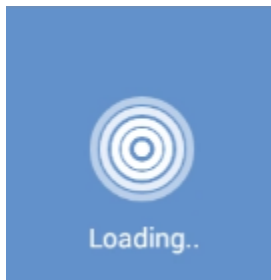
Busy Indicator with Box type animation



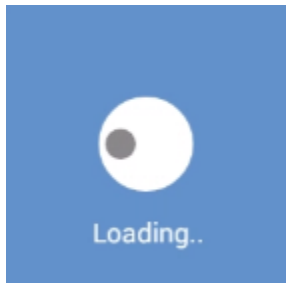
Busy Indicator with Gear type animation



Busy Indicator with MovieTimer type animation



Busy Indicator with ZoomingTarget type animation



Busy Indicator with RollingBall type animation

### Animation duration

The **Duration** property of SfBusyIndicator indicates timeline for completing one animation cycle. Setting smaller duration value accelerates animation speed.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"
x:Class="GettingStarted.MainPage">
  <ContentPage.Content>
    <busyindicator:SfBusyIndicator x:Name="busyindicator"
AnimationType="Battery"
ViewBoxHeight="100"
ViewBoxWidth="100"
Duration="0.5"/>
  </ContentPage.Content>
</ContentPage>
```

### C#

```
using Syncfusion.SfBusyIndicator.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfBusyIndicator busyIndicator = new SfBusyIndicator()
            {
                AnimationType = AnimationTypes.Battery,
                ViewBoxWidth = 100,
                ViewBoxHeight = 100,
                Duration = 0.5f
            };
            this.Content = busyIndicator;
        }
    }
}
```



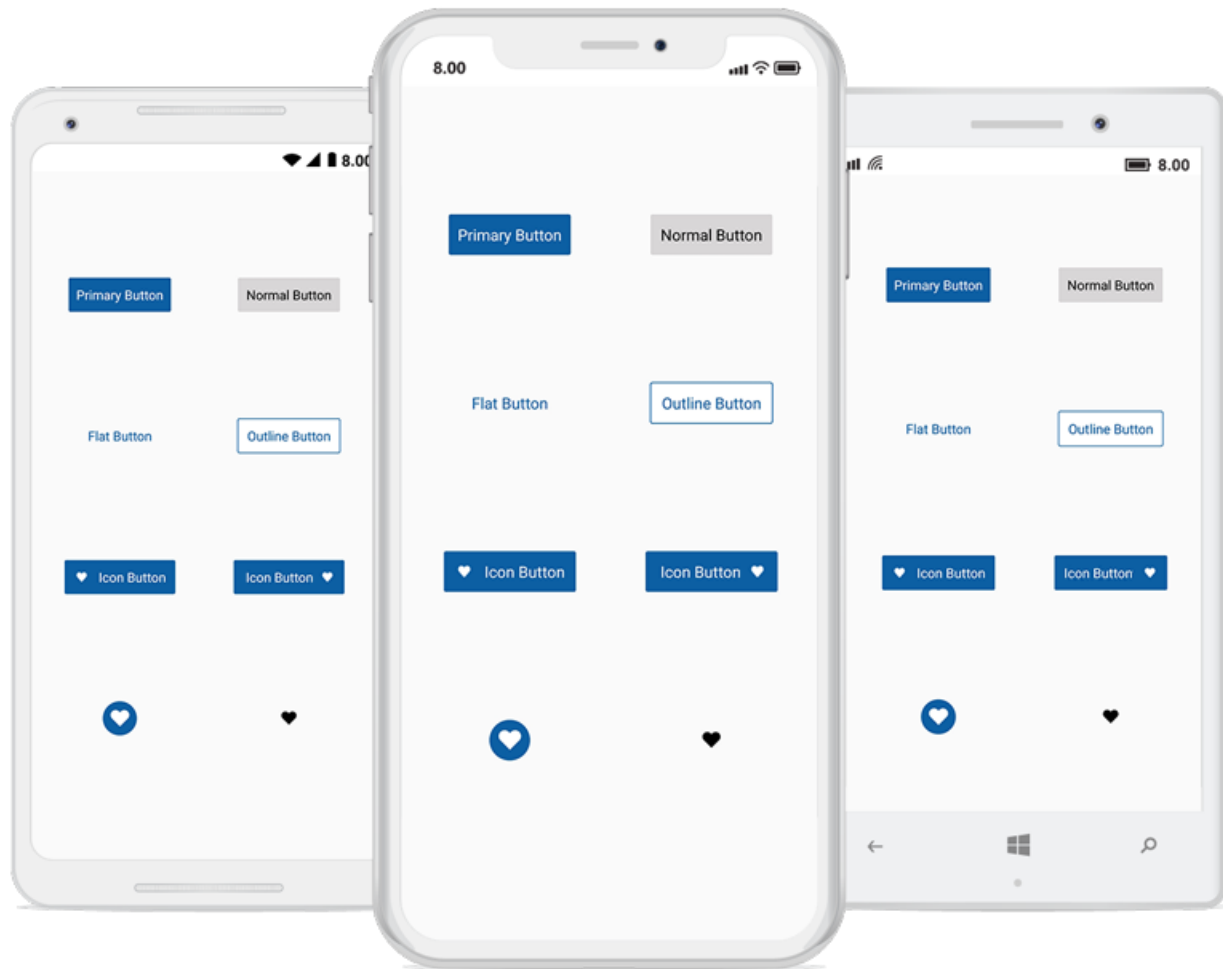
## SfButton

### Overview

The Xamarin.Forms Button is a custom button control with UI customization, toggle states, and theme support. You can set icons, background images, and corner edge radii and customize the appearance for different visual states using the visual state manager.

### Key features

- Easily customize a button as an outline, flat, circle, or icon button.
- Use visual states such as pressed, normal, checked, and unchecked.
- Display an image as the background for the control.
- Display custom content in the button control.
- Use a toggle button.



## Getting Started

This section explains the steps required to work with the button control for Xamarin.Forms.

### Adding SfButton reference

You can add SfButton reference using one of the following methods:

#### Method 1: Adding SfButton reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Buttons). To add SfButton to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Buttons](https://www.nuget.org/packages/Syncfusion.Xamarin.Buttons), and then install it.

![Adding SfButton reference from NuGet](Images/Adding SfButton reference.png)

#### Note:

- Install the same version of SfButton NuGet in all the projects.
- In addition, you need to install the [Syncfusion.Xamarin.Buttons.WPF]() package for Xamarin.Forms WPF platform only.

#### Method 2: Adding SfButton reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfButton control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfButton assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Buttons.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll <tr&gt; <="" td=""></tr&gt;>
WPF	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.WPF.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.WPF.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** After adding the reference, an additional step is required for iOS and UWP projects. If you are adding the references from toolbox, this step is not needed.

*Additional step for iOS*

To launch [SfButton](#) in iOS, call `SfButtonRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class in iOS Project as demonstrated in the following code example.

**C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    global::Xamarin.Forms.Forms.Init();
    LoadApplication(new App());
    SfBorderRenderer.Init();
    SfButtonRenderer.Init();
    return base.FinishedLaunching(app, options);
}
```

*Additional step for UWP*

This step is required only if the application is deployed in Release mode with .NET native tool chain enabled. It is needed for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the [SfButton](#) assembly at the `OnLaunched` overridden method of the `App` class in UWP project is the suggested work around. The following code example demonstrates initializing the [SfButton](#) assembly.

**C#**

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    .....
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies that your app uses
    assembliesToInclude.Add(typeof(SfButtonRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfBorderRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    .....
}
```

*Additional step for WPF*

To launch the button in WPF, call the `SfButtonRenderer.Init()` method in the `MainWindow` constructor of the `MainWindow` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

**C#**

```
public partial class MainWindow : FormsApplicationPage
{
    public MainWindow()
    {
        InitializeComponent();
        Forms.Init();
        Syncfusion.XForms.WPF.Border.SfBorderRenderer.Init();
        Syncfusion.XForms.WPF.Buttons.SfButtonRenderer.Init();
    }
}
```



```
LoadApplication(new App());
}
}
```

### Creating a simple SfButton

The [SfButton](#) control is configured entirely in C# code or in XAML markup. The following steps explain how to create a [SfButton](#) and configure its elements.

#### *Adding namespace for referred assemblies*

##### **XML**

```
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
```

##### **C#**

```
using Syncfusion.XForms.Buttons;
```

#### *Referring SfButton control with declared suffix name for namespace*

##### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SfButton"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="SfButton.MainPage">
<StackLayout HorizontalOptions="Center" VerticalOptions="Center">
<buttons:SfButton x:Name="SfButton"/>
</StackLayout>
</ContentPage>
```

##### **C#**

```
namespace SfButton
{
    using Syncfusion.XForms.Buttons;
    using Xamarin.Forms;
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            stackLayout.HorizontalOptions = LayoutOptions.Center;
            stackLayout.VerticalOptions = LayoutOptions.Center;
            SfButton button = new SfButton();
            stackLayout.Children.Add(button);
            this.Content = stackLayout;
        }
    }
}
```

### Setting caption

The button caption can be defined using the [Text](#) property of [SfButton](#). This caption normally describes the meaning of the button and is displayed in button.

#### XML

```
<buttons:SfButton x:Name="SfButton" Text="Button"/>
```

#### C#

```
SfButton button = new SfButton();  
button.Text = "Button";
```



### Toggle button

The button behavior can be changed as toggle button by defining the [IsCheckable](#) property of [SfButton](#).

#### XML

```
<buttons:SfButton x:Name="SfButton" Text="Button" IsCheckable="True"/>
```

#### C#

```
SfButton button = new SfButton();  
button.Text = "Button";  
button.IsCheckable = true;
```

Checked state:



Unchecked state:



### Button icon

The button icon can be defined using the [ImageSource](#) and [ShowIcon](#) properties of [SfButton](#).

#### XML

```
<buttons:SfButton x:Name="SfButton" Text="Button" ShowIcon="True"  
ImageSource="button_Heart.png"/>
```

#### C#

```
SfButton button = new SfButton();  
button.Text = "Button";  
button.ImageSource = "button_Heart.png";
```

```
button.ShowIcon = true;
```



Button background image

The button background icon can be defined using the [BackgroundImage](#) property of [SfButton](#).

#### XML

```
<buttons:SfButton x:Name="SfButton" Text="Button"
BackgroundImage="button_background.png" CornerRadius="20"
WidthRequest="100"/>
```

#### C#

```
SfButton button = new SfButton();
button.Text = "Button";
button.WidthRequest = 100;
button.BackgroundImage = "button_background.png";
button.CornerRadius = new Thickness(20);
```



You can find the complete getting started sample here: [Getting started](#).

#### Visual States

The button visual can be customized through **VisualStates**. The [SfButton](#) control have the following four visual states:

- Normal
- Pressed
- Checked
- Unchecked

**Note:** In addition, **MouseOver** VisualState is available only in the UWP platform.

#### XML

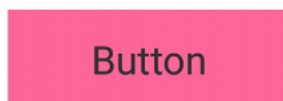
```
<StackLayout HorizontalOptions="Center" VerticalOptions="Center">
<buttons:SfButton x:Name="SfButton" WidthRequest="100" Text="Button">
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="CommonStates">
<VisualState x:Name="Normal">
<VisualState.Setters>
<Setter Property="TextColor" Value="White" />
</VisualState.Setters>
</VisualState>
<VisualState x:Name="Pressed">
<VisualState.Setters>
```

```
<Setter Property="TextColor" Value="Black" />
</VisualState.Setters>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</buttons:SfButton>
</StackLayout>
```

## C#

```
StackLayout stackLayout = new StackLayout
{
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center
};
SfButton button = new SfButton
{
    Text = "Button",
    WidthRequest = 100
};
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState normalState = new VisualState
{
    Name = "Normal"
};
normalState.Setters.Add(new Setter { Property = SfButton.TextColorProperty,
    Value = Color.White });
VisualState pressedState = new VisualState
{
    Name = "Pressed"
};
pressedState.Setters.Add(new Setter { Property = SfButton.TextColorProperty,
    Value = Color.Black });
commonStateGroup.States.Add(normalState);
commonStateGroup.States.Add(pressedState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(button, visualStateGroupList);
stackLayout.Children.Add(button);
this.Content = stackLayout;
```

### Pressed visual state:



### Normal visual state:



## Customization

The button control supports to customize the border color, image width, corner radius, background color, and more. The button control can be customized using the following properties:

### Text Customization

The text inside the button can be customized by its text color, font size, font attributes, font family and text alignment.

#### TextColor

The [TextColor](#) property is used to customize the color of text in SfButton.

#### XML

```
<button:SfButton x:Name="button" Text="Button" TextColor = "White">
</button:SfButton>
```

#### C#

```
SfButton button = new SfButton();
button.Text = "Button";
button.TextColor = Color.White;
```



#### FontSize

The [FontSize](#) property is used to customize the size of text in SfButton.

#### XML

```
<button:SfButton x:Name="button" Text="Button" FontSize = "18">
</button:SfButton>
```

#### C#

```
SfButton button = new SfButton();
button.Text = "Button";
button.FontSize = 18;
```



#### FontAttributes

The [FontAttributes](#) property is used to customize the font style of text in SfButton.

#### XML

```
<button:SfButton x:Name="button" Text="Button" FontAttributes = "Italic">
</button:SfButton>
```

#### C#

```
SfButton button = new SfButton();  
button.Text = "Button";  
button.FontAttributes = FontAttributes.Italic;
```



#### FontFamily

The [FontFamily](#) property is used to customize the font family of text in SfButton.

#### XML

```
<button:SfButton x:Name="button" Text="Button" FontFamily = "Arial">  
</button:SfButton>
```

#### C#

```
SfButton button = new SfButton();  
button.Text = "Button";  
button.FontFamily = "Arial";
```



#### TextAlignment

The [HorizontalTextAlignment](#) and [VerticalTextAlignment](#) properties are used to customize the alignment of text in SfButton.

#### XML

```
<button:SfButton x:Name="button" Text="Button"  
HorizontalTextAlignment="Center" VerticalTextAlignment="Center">  
</button:SfButton>
```

#### C#

```
SfButton button = new SfButton();  
button.Text = "Button";  
button.HorizontalTextAlignment = TextAlignment.Center;  
button.VerticalTextAlignment = TextAlignment.Center;
```

#### Background Customization

The background of the button can be customized by its background color, border color, border width and corner radius.

#### BackgroundColor

The [BackgroundColor](#) property is used to customize the background color of SfButton.

#### XML

```
<button:SfButton x:Name="button" Text="Button" BackgroundColor =  
"DeepSkyBlue">
```

```
</button:SfButton>
```

### C#

```
SfButton button = new SfButton();  
button.Text = "Button";  
button.BackgroundColor = Color.DeepSkyBlue;
```



#### *BorderColor*

The [BorderColor](#) property is used to customize the color of border in SfButton.

### XML

```
<button:SfButton x:Name="button" Text="Button" BorderColor = "Red"  
BorderWidth="4">  
</button:SfButton>
```

### C#

```
SfButton button = new SfButton();  
button.Text = "Button";  
button.BorderWidth = 4;  
button.BorderColor = Color.Red;
```



#### *BorderWidth*

The [BorderWidth](#) property is used to customize the thickness of border in SfButton.

### XML

```
<button:SfButton x:Name="button" Text="Button" BorderColor = "Red"  
BorderWidth="4">  
</button:SfButton>
```

### C#

```
SfButton button = new SfButton();  
button.Text = "Button";  
button.BorderWidth = 4;  
button.BorderColor = Color.Red;
```

#### *CornerRadius*

The [CornerRadius](#) property is used to customize the rounded edges in SfButton as demonstrated in the following code sample.

### XML

```
<button:SfButton x:Name="button" Text="Button" CornerRadius="3">
</button:SfButton>
```

### C#

```
SfButton button = new SfButton();
button.Text = "Button";
button.CornerRadius = 3;
```



### Image Customization

The image can be customized by its show icon, image source, image width and image alignment.

#### ShowIcon

You can enable the Icon image using the [ShowIcon](#) property to know whether any image appears to the SfButton.

### XML

```
<button:SfButton x:Name="button" Text="Button" ImageSource="Heart.png"
ShowIcon="True">
</button:SfButton>
```

### C#

```
SfButton button = new SfButton();
button.Text = "Button";
button.ImageSource = "Heart.png";
button.ShowIcon = True;
```

#### ImageSource

The [ImageSource](#) property is used to customize the icon image of SfButton by adding a custom image.

**Note:** Enable the [ShowIcon](#) property to enable the [ImageSource](#) property.

### XML

```
<button:SfButton x:Name="button" Text="Button" ImageSource="Heart.png"
ShowIcon="True">
</button:SfButton>
```

### C#

```
SfButton button = new SfButton();
button.Text = "Button";
button.ImageSource = "Heart.png";
button.ShowIcon = True;
```





#### ImageWidth

The [ImageWidth](#) property is used to customize the width of icon image in SfButton.

**Note:** Enable the [ShowIcon](#) property to enable the [ImageSource](#) property.

#### XML

```
<button:SfButton x:Name="button" Text="Button" ImageSource="Heart.png"
ShowIcon="True" ImageWidth="50">
</button:SfButton>
```

#### C#

```
SfButton button = new SfButton();
button.Text = "Button";
button.ImageSource = "Heart.png";
button.ShowIcon = true;
button.ImageWidth = 50;
```

#### ImageAlignment

The [ImageAlignment](#) property is used to customize the alignment of icon image in SfButton.

**Note:** Enable the [ShowIcon](#) property to enable the [ImageSource](#) property.

#### XML

```
<button:SfButton x:Name="button" Text="Button" ImageSource="Heart.png"
ShowIcon="True" ImageAlignment="End">
</button:SfButton>
```

#### C#

```
SfButton button = new SfButton();
button.Text = "Button";
button.ImageSource = "Heart.png";
button.ShowIcon = true;
button.ImageAlignment = Alignment.End;
```



#### Gradient background

You can set the gradient as background of SfButton using the [BackgroundGradient](#) property. It supports the following types of gradients:

- Linear gradient
- Radial gradient

**Note:** UWP platform does not support radial gradient.

Refer to this [documentation](#) to learn more details about gradient.

### XML

```
xmlns:gradient="clr-
namespace:Syncfusion.XForms.Graphics;assembly=Syncfusion.Core.XForms"
. . .
<button:SfButton Text="Linear Gradient" CornerRadius="20">
<button:SfButton.BackgroundGradient>
<gradient:SfLinearGradientBrush>
<gradient:SfLinearGradientBrush.GradientStops>
<gradient:SfGradientStop Color="#2F9BDF" Offset="0"/>
<gradient:SfGradientStop Color="#51F1F2" Offset="1"/>
</gradient:SfLinearGradientBrush.GradientStops>
</gradient:SfLinearGradientBrush>
</button:SfButton.BackgroundGradient>
</button:SfButton>
<button:SfButton Text="Radial Gradient" CornerRadius="20">
<button:SfButton.BackgroundGradient>
<gradient:SfRadialGradientBrush Radius="1.5">
<gradient:SfRadialGradientBrush.GradientStops>
<gradient:SfGradientStop Color="#FFB57B" Offset="0"/>
<gradient:SfGradientStop Color="#FF5361" Offset="1"/>
</gradient:SfRadialGradientBrush.GradientStops>
</gradient:SfRadialGradientBrush>
</button:SfButton.BackgroundGradient>
</button:SfButton>
```

### C#

```
using Syncfusion.XForms.Graphics;
. . .
SfButton linearButton = new SfButton();
linearButton.Text = "Linear Gradient";
linearButton.CnerRadius = 20;
SfLinearGradientBrush linearGradientBrush = new SfLinearGradientBrush();
linearGradientBrush.GradientStops = new GradientStopCollection()
{
    new SfGradientStop() { Color = Color.FromHex("#2F9BDF"), Offset = 0 },
    new SfGradientStop() { Color = Color.FromHex("#51F1F2"), Offset = 1 }
};
linearButton.BackgroundGradient = linearGradientBrush;
SfButton radialButton = new SfButton();
radialButton.Text = "Radial Gradient";
radialButton.CnerRadius = 20;
SfRadialGradientBrush radialGradientBrush = new SfRadialGradientBrush();
radialGradientBrush.Radius = 1.5;
radialGradientBrush.GradientStops = new GradientStopCollection()
{
    new SfGradientStop() { Color = Color.FromHex("#FFB57B"), Offset = 0 },
    new SfGradientStop() { Color = Color.FromHex("#FF5361"), Offset = 1 }
};
radialButton.BackgroundGradient = radialGradientBrush;
```



### Command

The [Command](#) property is used to associate a command with an instance of SfButton. This property is most often set with MVVM pattern to bind callbacks back into the ViewModel.

**Note:** Default value is [null].

### XML

```
<ContentPage.BindingContext>
<local:CommandDemoViewModel />
</ContentPage.BindingContext>
<button:SfButton x:Name="button" Text="Button" BackgroundColor="{Binding
Background}" Command="{Binding ButtonCommand}">
</button:SfButton>
```

### C#

```
// ViewModel
public class CommandDemoViewModel : INotifyPropertyChanged
{
    private Color _background = Color.Accent;
    public Color Background
    {
        get { return _background; }
        set
        {
            _background = value;
            NotifyPropertyChanged();
        }
    }
    private void NotifyPropertyChanged([CallerMemberName] String propertyName =
    "")
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
    public event PropertyChangedEventHandler PropertyChanged;
    public CommandDemoViewModel()
    {
        BackgroundColor();
        this.Background=Color.Accent;
    }
    private void BackgroundColor()
    {
        this.Background = this.Background == Color.DeepSkyBlue ? Color.Accent :
        Color.DeepSkyBlue;
    }
}
```

```
}  
public ICommand ButtonCommand => new Command(BackgroundColor);  
}
```

### Shadow Effect

The button control provides shadow effect support. To enable shadow effect, set the **HasShadow** property to true.

You can customize the color of shadow using the **ShadowColor** property.

### XML

```
<SyncfusionButton:SfButton  
HeightRequest="50"  
WidthRequest="200"  
VerticalOptions="Center"  
HorizontalOptions="Center"  
CornerRadius="25"  
HasShadow="True"  
BorderWidth="1"  
BorderColor="Gray"  
BackgroundColor="#538EEC"  
ImageSource="Basket.png"  
ShowIcon="True"  
ImageAlignment="End"  
Text="ADD To CART"/>
```

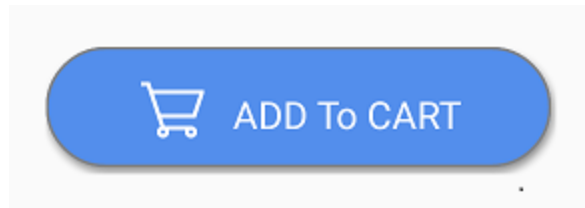
### C#

```
SfButton button = new SfButton()  
{  
    HeightRequest = 50,  
    WidthRequest = 200,  
    HorizontalOptions = LayoutOptions.Center,  
    VerticalOptions = LayoutOptions.Center,  
    CornerRadius = new Thickness(25),  
    HasShadow = true,  
    BorderWidth = 1,  
    BorderColor = Color.Gray,  
    BackgroundColor = Color.FromHex("#538EEC"),  
    ImageSource = "Basket.png",  
    ShowIcon = true,  
    ImageAlignment = Alignment.Start,  
    Text = "ADD To CART"  
};  
this.Content = button;
```

---

**Note:** Shadow support has not been provided for UWP Platform.

---



The complete customization sample: [Customization](#)

## How to

Add the custom view for button

You can customize the appearance of the button by adding your custom view in the [Content](#) property. The following code sample demonstrates how to apply the busy indicator control as custom view for a button.

## XML

```
xmlns:busyindicator="clr-
namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndica
tor.XForms"
. . .
<buttons:SfButton HeightRequest="60" CornerRadius="20"
HorizontalOptions="Center" VerticalOptions="Center">
<buttons:SfButton.Content>
<StackLayout Orientation="Horizontal">
<busyindicator:SfBusyIndicator AnimationType="SingleCircle" IsBusy="True"
TextColor="White" WidthRequest="50"/>
<Label Text="Loading..." FontSize="20" VerticalTextAlignment="Center"
TextColor="White" />
</StackLayout>
</buttons:SfButton.Content>
</buttons:SfButton>
```

## C#

```
using Syncfusion.SfBusyIndicator.XForms;
. . .
SfButton button = new SfButton();
button.HeightRequest = 60;
button.CornerRadius = 20;
button.HorizontalOptions = LayoutOptions.Center;
button.VerticalOptions = LayoutOptions.Center;
StackLayout stackLayout = new StackLayout();
stackLayout.Orientation = StackOrientation.Horizontal;
SfBusyIndicator busyindicator = new SfBusyIndicator()
{
    AnimationType = AnimationTypes.SingleCircle,
    IsBusy = true,
    TextColor = Color.White,
    WidthRequest = 50
};
Label label = new Label()
{
    Text = "Loading...",
    FontSize = 20,
```

```
VerticalTextAlignment = TextAlignment.Center,  
TextColor = Color.White  
};  
stackLayout.Children.Add(busyindicator);  
stackLayout.Children.Add(label);  
button.Content = stackLayout;
```



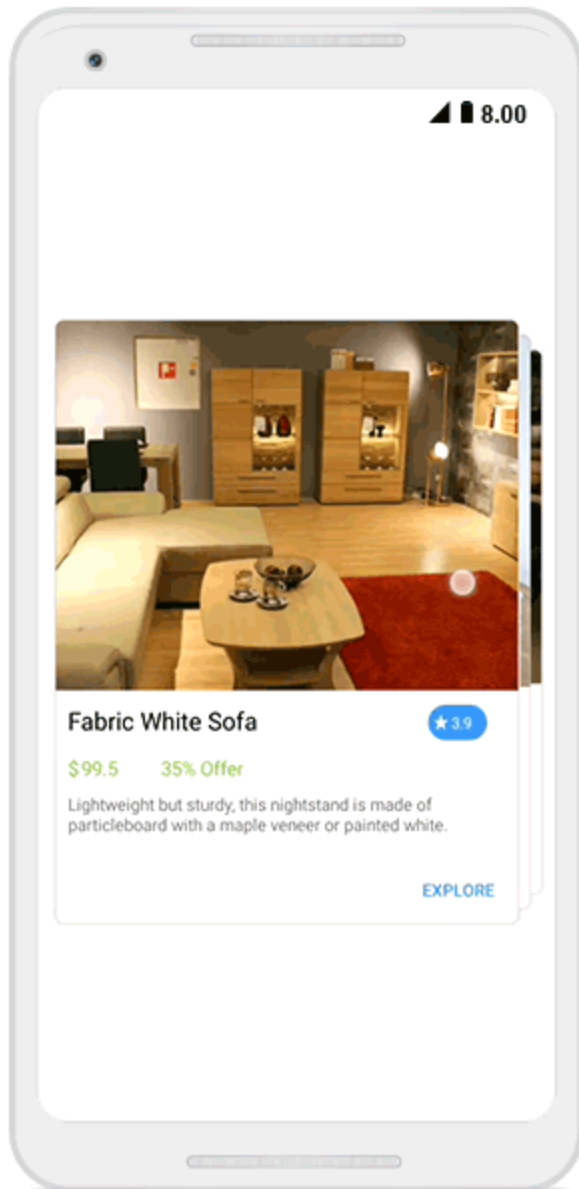
## Cards

### Overview

The new Cards control allows you to create a dismissible card or a stack of cards where one card is visible at a time until you swipe to see the next card.

### Key features

- Allows you swipe to the left or right to move the card smoothly to view the next card.
- Allows you swipe to the left or right to dismiss the card.
- Customizes the background, border width, corner radii, border color, and indicator.
- Provides elevation support in iOS and Android platforms.



## Getting Started

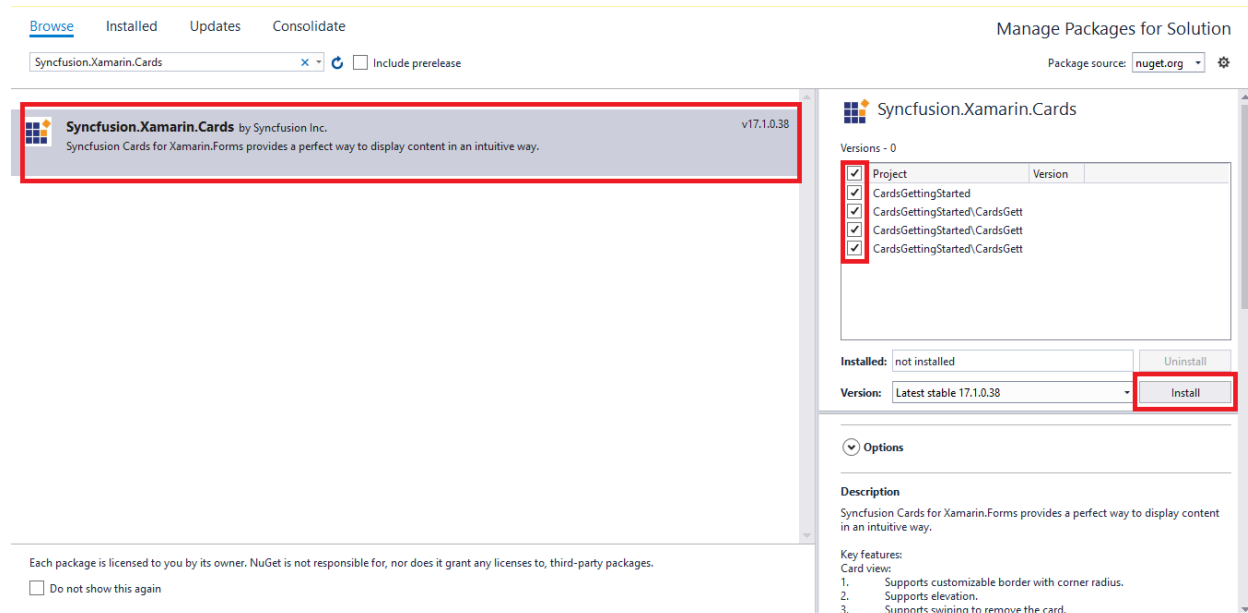
This section explains you the steps required to add content to SfCardView with indicator and add SfCardView to SCardLayout. This section covers only the minimal features needed to get started with the cards.

### Adding cards reference

You can add cards reference using one of the following methods:

#### Method 1: Adding cards reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Cards). To add cards to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Cards](https://www.nuget.org/packages/Syncfusion.Xamarin.Cards), and then install it.



**Note:** Install the same version of the cards NuGet in all the projects.

### Method 2: Adding cards reference from toolbox

Syncfusion provides Xamarin Toolbox. Using this toolbox, you can drag the SfCardView and SfCardLayout to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding Cards assemblies manually from the installed location

If you prefer to manually reference the assemblies instead of referencing from NuGet, add the following assemblies in respective projects.

Location : {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Cards.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Cards.XForms.Android.dll Syncfusion.Cards.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Cards.XForms.iOS.dll Syncfusion.Cards.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Cards.XForms.UWP.dll Syncfusion.Cards.XForms.dll Syncfusion.Core.XForms.dll



	Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
--	--

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the application on each platform with cards

To use the cards inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and are discussed in the following sections:

**Note:** If you are adding the references from toolbox, below steps are not needed.

#### iOS

To launch the cards in iOS, call the `SfCardViewRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    ...
    global::Xamarin.Forms.Forms.Init();
    Syncfusion.XForms.iOS.Cards.SfCardViewRenderer.Init();
    LoadApplication(new App());
    ...
}
```

#### Universal Windows Platform (UWP)

To deploy the cards in **Release** mode, you need to initialize the cards assemblies in `App.xaml.cs` in the UWP project as shown in the below code snippets.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    if (rootFrame == null)
    {
        List<Assembly> assembliesToInclude = new List<Assembly>();
        assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.Cards.SfCardViewRender
er).GetTypeInfo().Assembly);
        Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    }
    ...
}
```

### Android

The Android platform does not require any additional configuration to render the cards.

### Initialize cards

Import the [Cards](#) namespace as shown in the following code in your respective page.

#### XML

```
xmlns:cards="clr-
namespace:Syncfusion.XForms.Cards;assembly=Syncfusion.Cards.XForms"
```

#### C#

```
using Syncfusion.Cards.XForms;
```

### SfCardView

Initialize a card view with [Content](#) as shown in the following code.

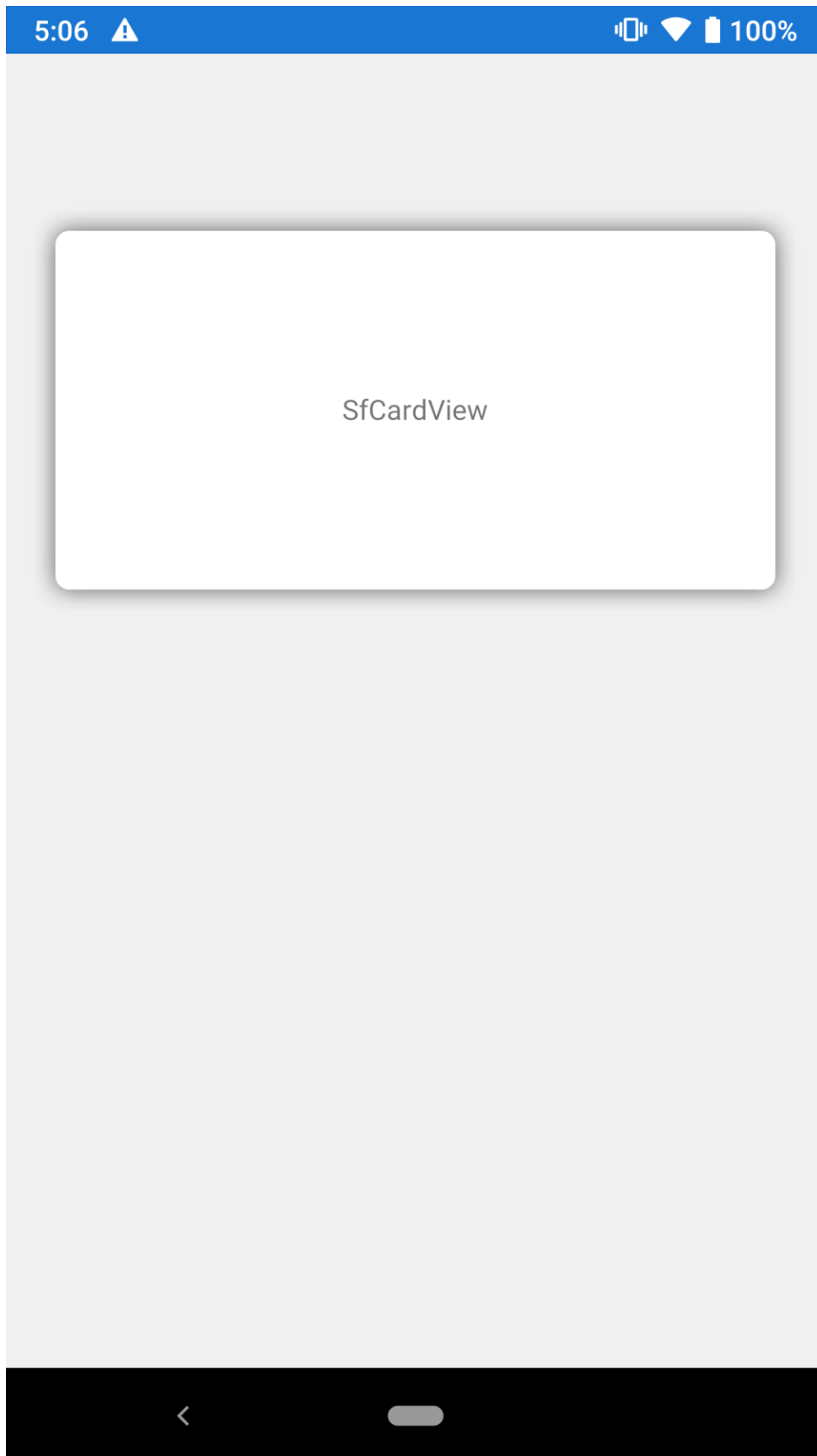
#### XML

```
<cards:SfCardView>
<Label Text="SfCardView"/>
</cards:SfCardView>
```

#### C#

```
SfCardView cardView = new SfCardView();
//set Content for card view
cardView.Content = new Label() { Text="SfCardView" };
this.Content = cardView;
```

Run the project and check if you get following output to make sure that you have configured your project properly to add [Cards](#).



### SwipeToDismiss

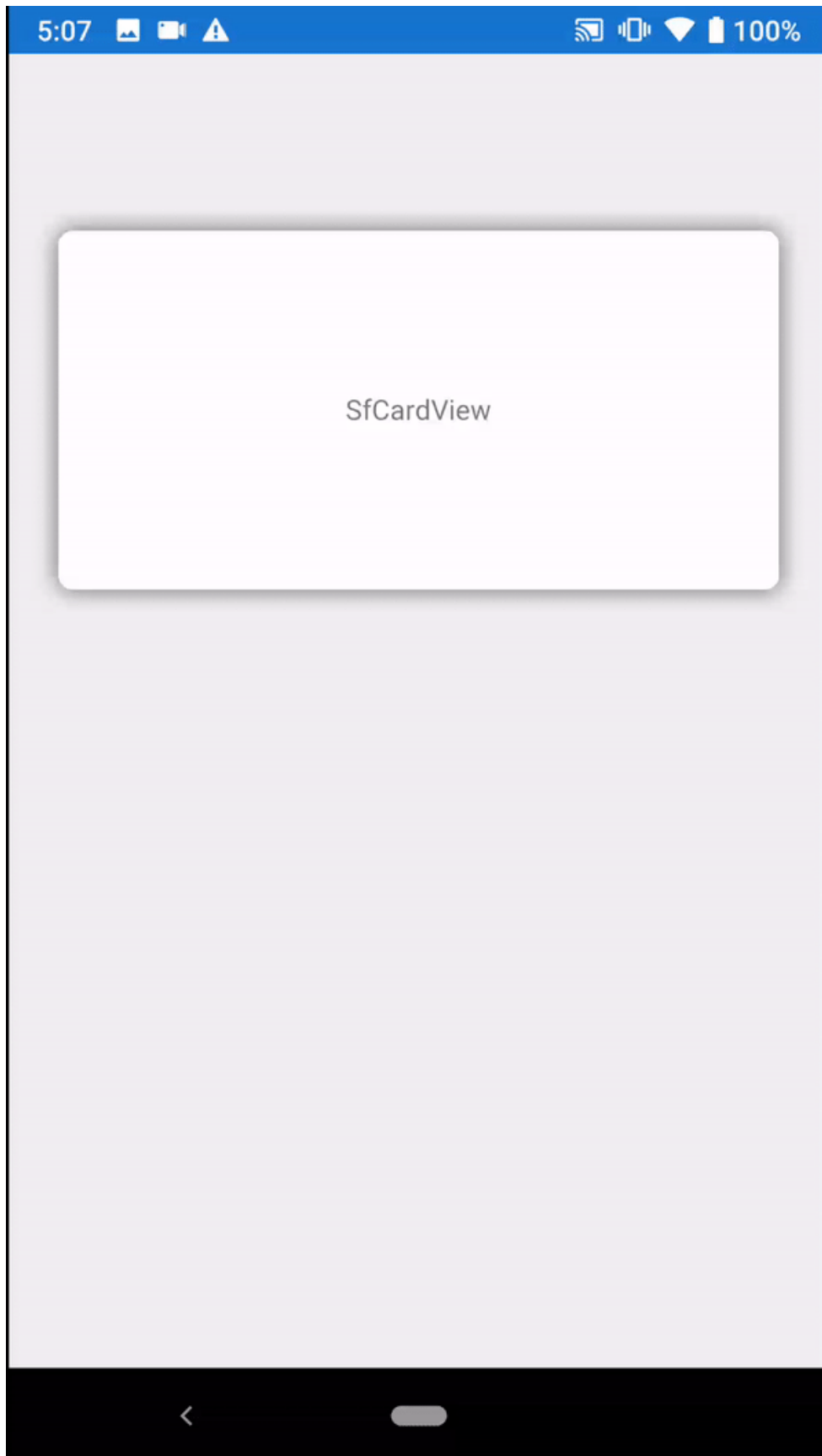
The [SwipeToDismiss](#) property is used to enable or disable swiping in SfCardView.

#### XML

```
<cards:SfCardView SwipeToDismiss="true">
  <Label Text="SfCardView"/>
</cards:SfCardView>
```

#### C#

```
SfCardView cardView = new SfCardView();
cardView.SwipeToDismiss = true;
cardView.Content = new Label() { Text="SfCardView" };
this.Content = cardView;
```



### *SfCardLayout*

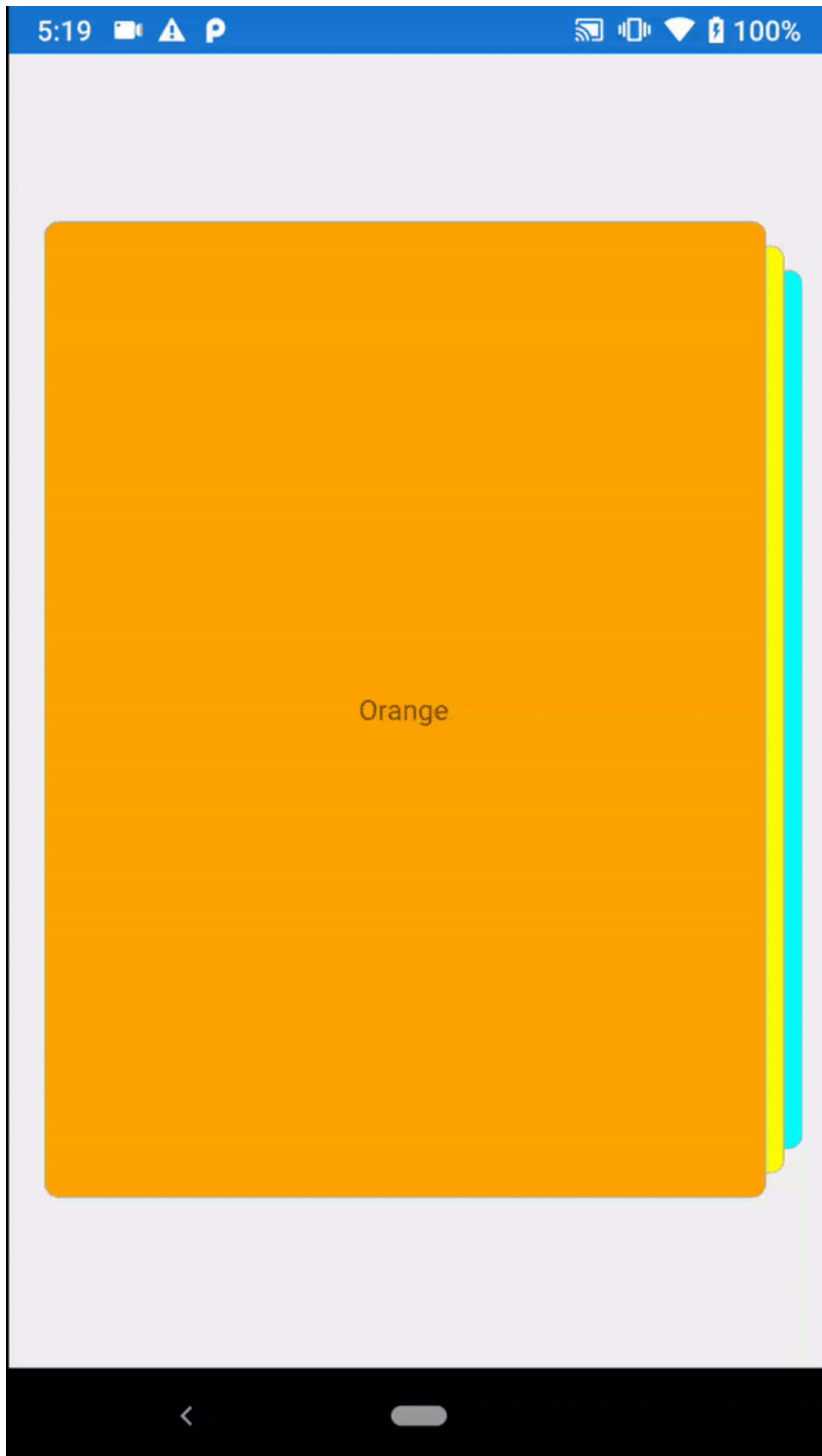
Initialize a card layout with card view as shown in the following code.

#### **XML**

```
<cards:SfCardLayout SwipeDirection="Left" HeightRequest="500"
BackgroundColor="#F0F0F0">
<cards:SfCardView>
<Label Text="Cyan" BackgroundColor="Cyan"/>
</cards:SfCardView>
<cards:SfCardView>
<Label Text="Yellow" BackgroundColor="Yellow"/>
</cards:SfCardView>
<cards:SfCardView>
<Label Text="Orange" BackgroundColor="Orange"/>
</cards:SfCardView>
</cards:SfCardLayout>
```

#### **C#**

```
SfCardLayout cardLayout = new SfCardLayout();
//Add children for card layout
cardLayout.Children.Add(new SfCardView(){Content = new Label(){ Text="Cyan",
BackgroundColor=Color.Cyan }});
cardLayout.Children.Add(new SfCardView(){Content = new Label(){
Text="Yellow", BackgroundColor=Color.Yellow }});
cardLayout.Children.Add(new SfCardView(){Content = new Label(){
Text="Orange", BackgroundColor=Color.Orange }});
this.Content = cardLayout;
```



## Customization in CardLayout

### *ShowSwipedCard*

ShowSwipedCard can be enabled to show the swiped cards at the edge of card layout.

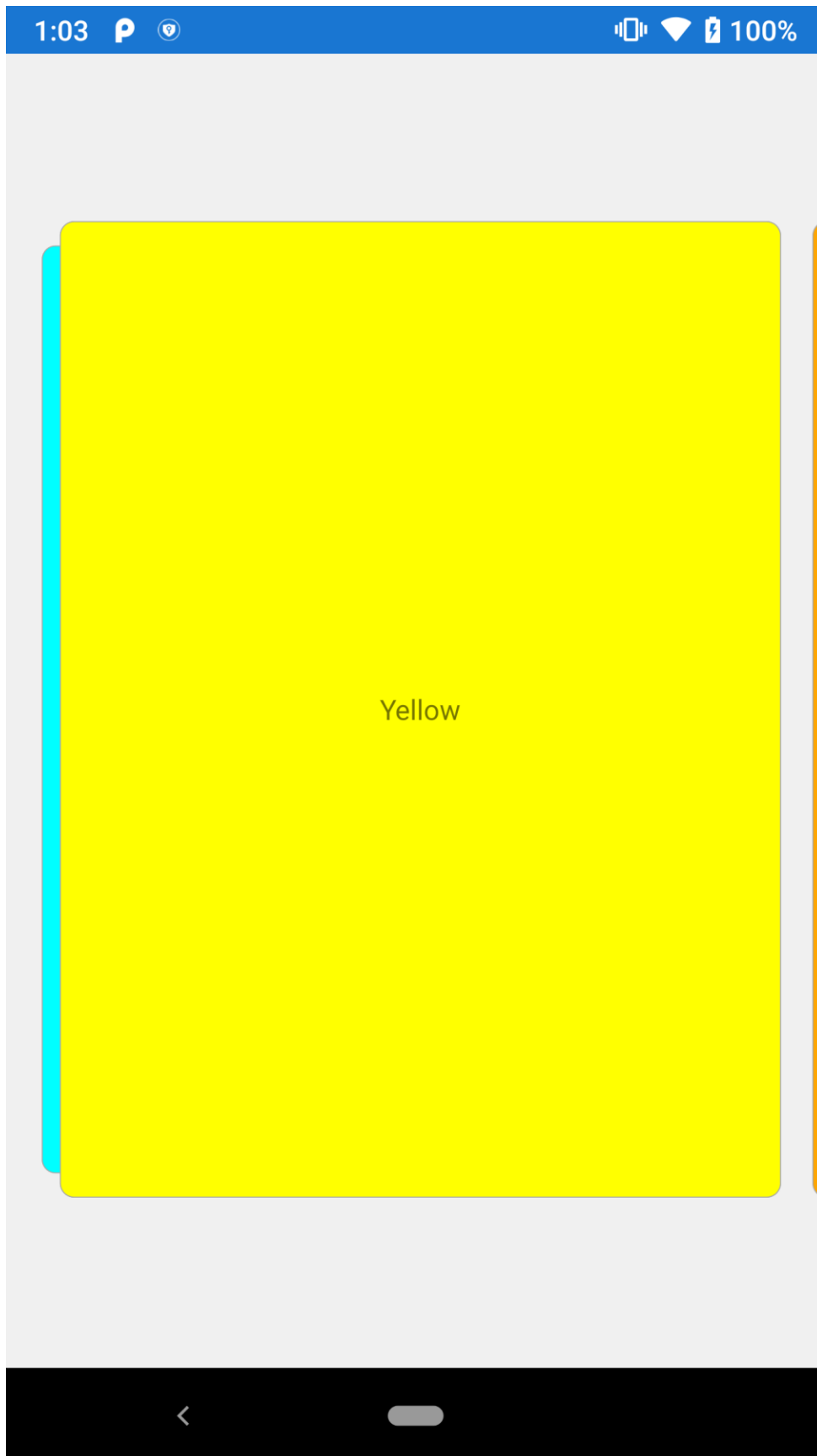
#### **XML**

```
<cards:SfCardLayout ShowSwipedCard="true">  
</cards:SfCardLayout>
```

#### **C#**

```
SfCardLayout cardLayout = new SfCardLayout();  
cardLayout.ShowSwipedCard = true;  
this.Content = cardLayout;
```





### *VisibleCardIndex*

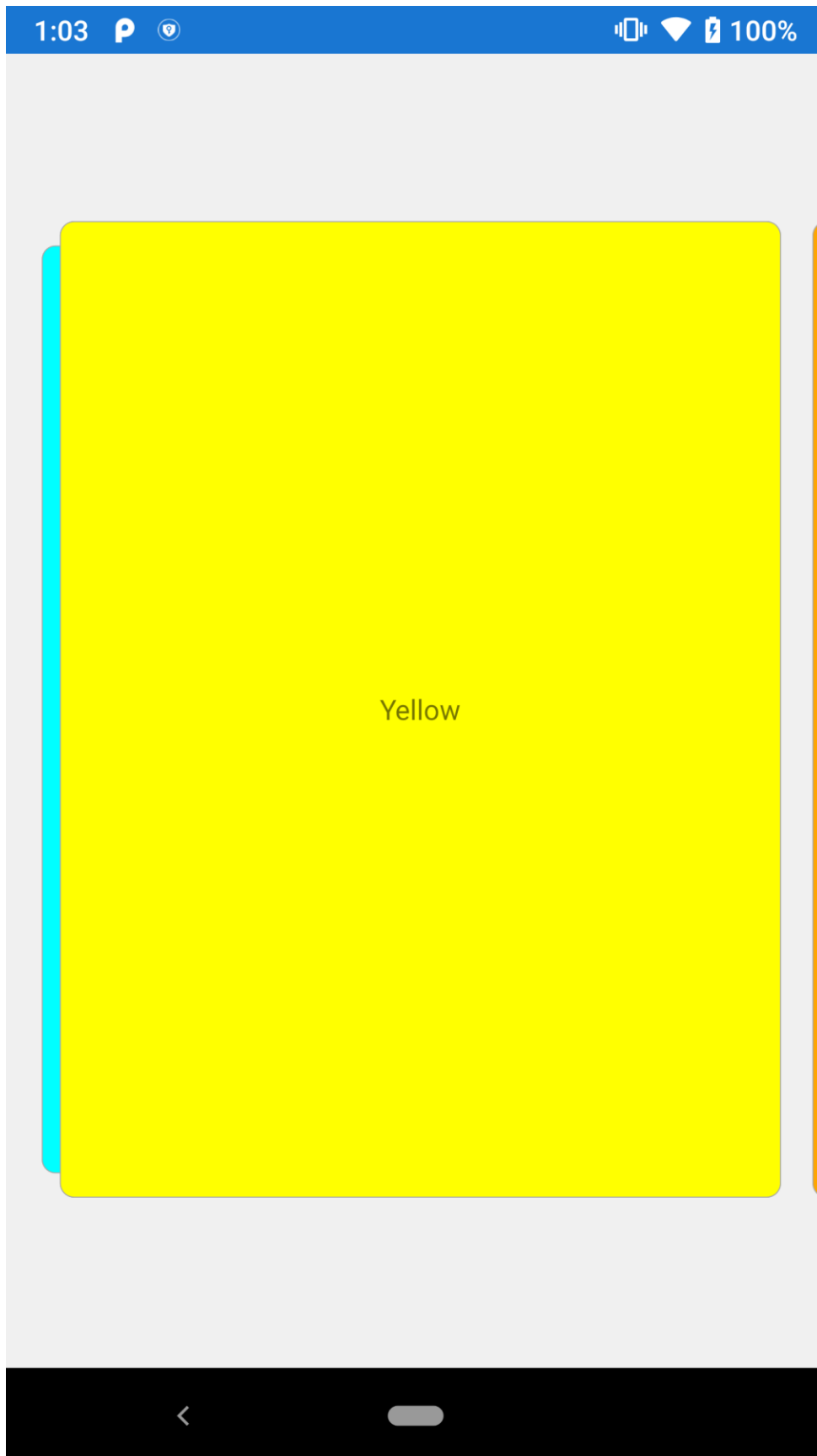
VisibleCardIndex is used when given index of the card to be displayed in front of the card layout.

#### **XML**

```
<cards:SfCardLayout VisibleCardIndex="1">  
</cards:SfCardLayout>
```

#### **C#**

```
SfCardLayout cardLayout = new SfCardLayout();  
cardLayout.VisibleCardIndex = 1;  
this.Content = cardLayout;
```



### *SwipeDirection*

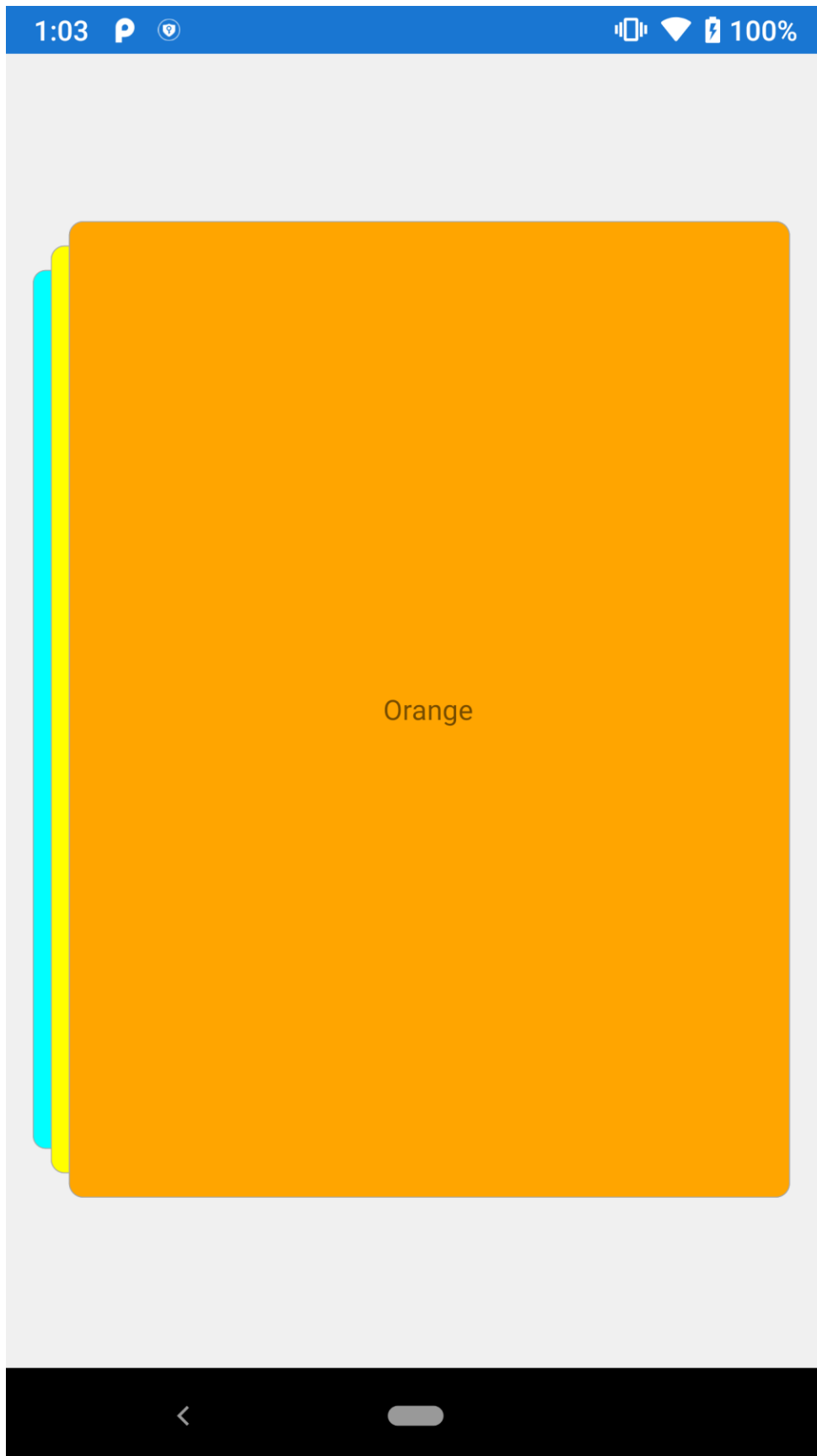
The `SwipeDirection` property indicates the swiping direction (left or right).

#### **XML**

```
<cards:SfCardLayout SwipeDirection="Right">  
</cards:SfCardLayout>
```

#### **C#**

```
SfCardLayout cardLayout = new SfCardLayout();  
cardLayout.SwipeDirection = CardSwipeDirection.Right;  
this.Content = cardLayout;
```



You can find the complete getting started sample from this [link](#).

## Customization in CardView

### *Indicator customization*

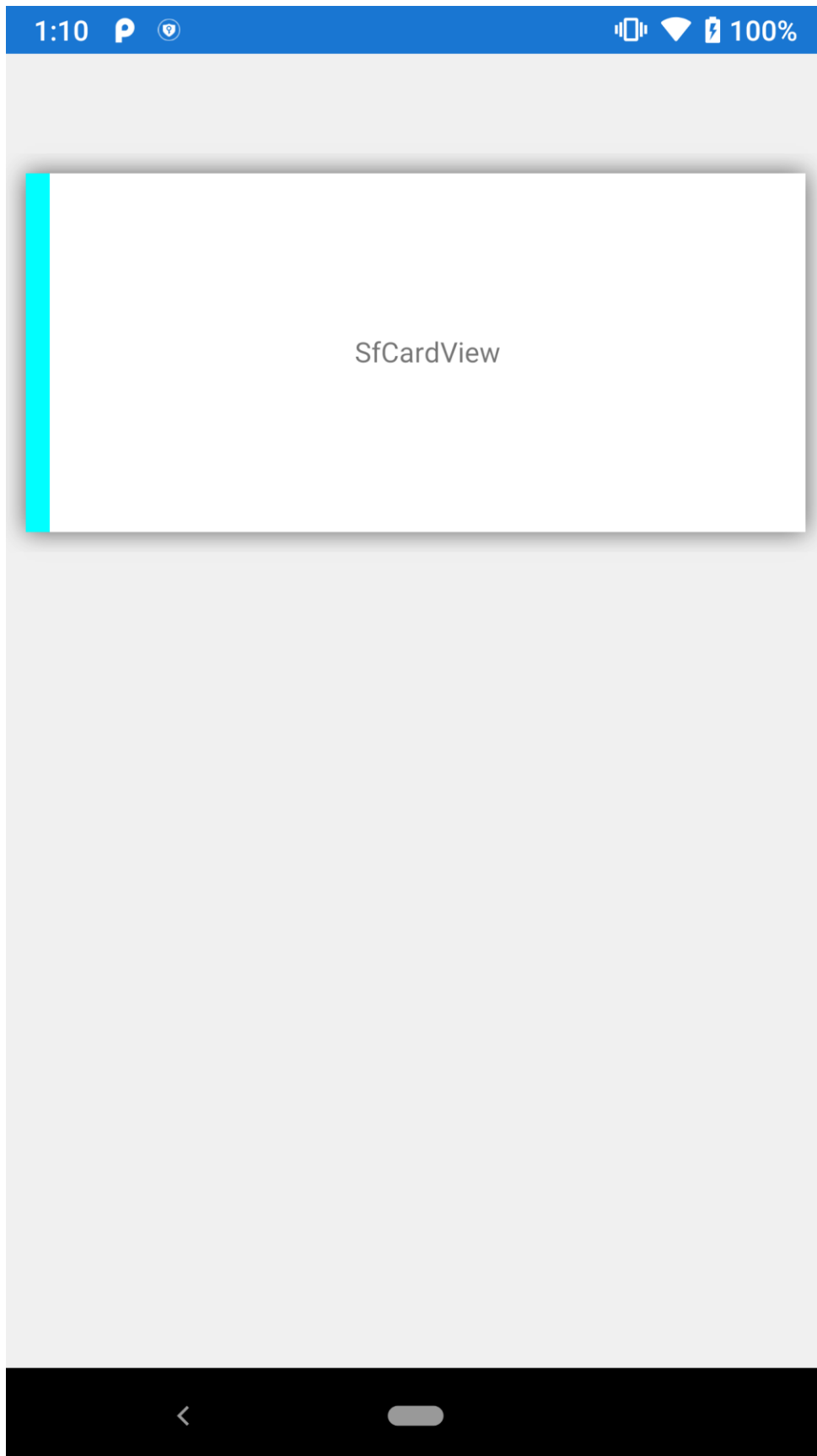
Indicators are used to indicate the state or level of something.

#### **XML**

```
<cards:SfCardView IndicatorColor="Cyan" IndicatorThickness="12"  
IndicatorPosition="Left" />
```

#### **C#**

```
SfCardView cardView = new SfCardView();  
cardView.IndicatorThickness = 12;  
cardView.IndicatorPosition = IndicatorPosition.Left;  
cardView.IndicatorColor = Color.Cyan;
```



### *FadeOutOnSwiping*

FadeOutOnSwiping can be enabled when the card view needs to be faded with respect to swiping.

#### **XML**

```
<cards:SfCardView FadeOutOnSwiping="true"/>
```

#### **C#**

```
SfCardView cardView = new SfCardView();  
cardView.FadeOutOnSwiping = true;
```

You can find the complete getting started sample from this [link](#).

From version 3.5, Xamarin.Forms has introduced a new approach, called BindableLayout, which works with all the layouts that are derived from Layout<T>. By simply setting ItemTemplate and ItemsSource, BindableLayout creates a group of UI (for the given ItemTemplate) for every data in the ItemsSource and add them as children.

Since [SfCardLayout](#) is an extended class of Layout<T>, this approach is also possible for SfCardLayout.

#### Initialize view model

Define a simple data model that represents data to be populated for [SfCardLayout](#).

#### **C#**

```
public class Model  
{  
    public IEnumerable<string> Colors { get; set; }  
}
```

Next, create a view model class and initialize a model object as demonstrated in the following code sample.

#### **C#**

```
public class ViewModel  
{  
    public Model Model { get; set; }  
    public ViewModel()  
    {  
        Model = new Model  
        {  
            Colors = new string[]  
            {  
                "Cyan", "Yellow", "Orange"  
            }  
        };  
    }  
}
```

Set the ViewModel instance as BindingContext of your page to bind properties of ViewModel to [SfCardLayout](#).



**Note:** Add namespace of ViewModel class in your XAML page if you prefer to set BindingContext in XAML.

### XML

```
<ContentPage.BindingContext>
<local:ViewModel></local:ViewModel>
</ContentPage.BindingContext>
```

### C#

```
this.BindingContext = new ViewModel();
```

Populate CardLayout with data

[SfCardLayout](#) can be populated with data by setting the ItemSource property of BindableLayout to a collection of items that can be used in [SfCardView](#).

### XML

```
<cards:SfCardLayout BindableLayout.ItemsSource="{Binding Model.Colors}">
...
</cards:SfCardLayout>
```

### C#

```
SfCardLayout cardLayout = new SfCardLayout();
BindableLayout.SetItemsSource(cardLayout, viewModel.Model.Colors);
```

Define the appearance of SfCardView

[SfCardLayout](#) accepts only [SfCardView](#) as its child. The appearance of each [SfCardView](#) can be defined by setting the BindableLayout.ItemTemplate property.

### XML

```
<cards:SfCardLayout BindableLayout.ItemsSource="{Binding Model.Colors}"
SwipeDirection="Left" VerticalOptions="Center" HeightRequest="300"
WidthRequest="300" BackgroundColor="#F0F0F0">
<BindableLayout.ItemTemplate>
<DataTemplate>
<cards:SfCardView BackgroundColor="{Binding}">
<Label Text="{Binding}" HorizontalOptions="CenterAndExpand"
VerticalTextAlignment="Center"/>
</cards:SfCardView>
</DataTemplate>
</BindableLayout.ItemTemplate>
</cards:SfCardLayout>
```

### Events

#### CardTapped

The [CardTapped](#) event occurs when any card view is tapped. The argument contains the following information:

- [CardView](#) - Gets the details of a particular card view.

### Command

The `CardTappedCommand` property is used to associate a command with an instance of `SfCardLayout`. This property is most often set with MVVM pattern to bind callbacks back into the ViewModel.

### CommandParameter

The `CardTappedCommandParameter` property is used to set the parameter reference, based on which the event argument is shown.

### NOTE

The default value of the `CardTappedCommandParameter` is `null`.

### XML

```
<cards:SfCardLayout CardTappedCommand="{Binding CardTappedCommand}"
CardTappedCommandParameter= "1">
<!--Add children for card layout-->
</cards:SfCardLayout>
```

### C#

```
public class ViewModel
{
    public ViewModel()
    {
        ItemTappedCommand = new Command<object>(CardTapped);
    }
    public ICommand CardTappedCommand { get; set; }
    private void CardTapped(object obj)
    {
        // handle event action.
    }
}
```

### VisibleCardIndexChanging

The `VisibleCardIndexChanging` event occurs when the visible card index is changing. The argument contains the following information:

- `OldCard` - Gets the details of the previous index card.
- `NewCard` - Gets the details of the next possible index card.

### XML

```
<cards:SfCardLayout VisibleCardIndexChanging="VisibleCardIndexChanging" >
<cards:SfCardView>
<Label Text="Cyan" BackgroundColor="Cyan" />
</cards:SfCardView>
<cards:SfCardView>
<Label Text="Yellow" BackgroundColor="Yellow" />
</cards:SfCardView>
<cards:SfCardView>
<Label Text="Orange" BackgroundColor="Orange" />
</cards:SfCardView>
```

```
</cards:SfCardView>
</cards:SfCardLayout>
```

**C#**

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        SfCardLayout cardLayout = new SfCardLayout();
        cardLayout.VisibleCardIndexChanging += VisibleCardIndexChanging;
        //Add children for card layout.
        cardLayout.Children.Add(new SfCardView() { Content = new Label() { Text =
"Cyan", BackgroundColor = Color.Cyan } });
        cardLayout.Children.Add(new SfCardView() { Content = new Label() { Text =
"Yellow", BackgroundColor = Color.Yellow } });
        cardLayout.Children.Add(new SfCardView() { Content = new Label() {
Text="Orange", BackgroundColor = Color.Orange } });
    }
    private void VisibleCardIndexChanging(object sender,
Syncfusion.XForms.Cards.VisibleCardIndexChangingEventArgs e)
    {
        // handle event action.
    }
}
```

**VisibleCardIndexChanged**

The **VisibleCardIndexChanged** event occurs when the visible card index is changed. The argument contains the following information:

- **OldCard** - Gets the details of the previous card.
- **NewCard** - Gets the details of the current card.

**XML**

```
<cards:SfCardLayout VisibleCardIndexChanged="VisibleCardIndexChanged" >
<cards:SfCardView>
<Label Text="Cyan" BackgroundColor="Cyan" />
</cards:SfCardView>
<cards:SfCardView>
<Label Text="Yellow" BackgroundColor="Yellow" />
</cards:SfCardView>
<cards:SfCardView>
<Label Text="Orange" BackgroundColor="Orange" />
</cards:SfCardView>
</cards:SfCardLayout>
```

**C#**

```
public partial class MainPage : ContentPage
{
    public MainPage()
```

```

{
InitializeComponent();
SfCardLayout cardLayout = new SfCardLayout();
cardLayout.VisibleCardIndexChanged += VisibleCardIndexChanged;
//Add children for card layout
cardLayout.Children.Add( new SfCardView() { Content = new Label() { Text =
"Cyan", BackgroundColor = Color.Cyan } });
cardLayout.Children.Add(new SfCardView() {Content = new Label() {
Text="Yellow", BackgroundColor = Color.Yellow } });
cardLayout.Children.Add(new SfCardView() {Content = new Label() {
Text="Orange", BackgroundColor = Color.Orange } });
}
private void VisibleCardIndexChanged(object sender
Syncfusion.XForms.Cards.VisibleCardIndexChangedEventArgs e)
{
// handle event action.
}
}

```

## Calculate

### Overview

Essential Calculate is a native .NET class library which enables to parse and compute the expression or formulas with 400+ predefined functions. It can be used in any .NET environment, including C#, VB.NET and managed C++ code.

It is a non-UI component which provides a full-fledged object model that facilitates formula calculation support without any dependency of Microsoft Office COM libraries & Microsoft Office. This library can be used in Windows Forms, WPF, ASP.NET, Xamarin and UWP application.

The range of calculations include simple algebraic expressions such as  $(1.2^3-1)/8$ , to formulas using intrinsic functions like  $4 \sqrt{\exp(8.4)}$ , to formulas relying on variables that are defined through controls on a form such as  $\cos([textBox1] \text{ pi}()/180)$ , to spreadsheet-like formulas

such as  $\text{Sum}(A2:B14)$ .

Following are the key features of Essential Calculate,

- Extensive calculation support can be added to the user's own business objects.
- It comes with a function library of more than 400+ entries of formulas.
- It provides support to named ranges, dynamic references, array formulas and formula dependencies.
- Supports cross sheet references (can work with multiple sheets).
- Supports custom functions with n-number of optional arguments.
- Culture-sensitive decimal separator and argument separator.
- It can be used in conjunction with Essential XlsIO, to fully load, manipulate and compute Excel spreadsheets.
- It does not depend upon Microsoft Excel and thus enables you to perform calculations independent of Excel.

### Getting Started

This section helps you to get started with Essential Calculate.

### Adding Calculate reference

You can add Calculate reference using one of the following methods:

#### Method 1: Adding Calculate reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Calculate). To add Calculate to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Calculate](https://www.nuget.org/packages/Syncfusion.Xamarin.Calculate), and then install it.

!{Adding Calculate reference from nuget}(Images/Adding Calculate reference.png)

---

**Note:** Install the same version of Calculate NuGet in all the projects.

---

#### Method 2: Adding Calculate reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the Calculate control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding Calculate assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Calculate.Portable.dll Syncfusion.Licensing.dll
-----	---

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Compute formula using CalcQuickBase

The [CalcQuickBase](#) will provide options to directly parse and compute a formula, or register variable names that can later be used in more complex formulas involving these variables.

[CalcQuickBase](#) is predefined derived class from [ICalcData](#).

Below example shows the computation of formula using [ParseAndCompute](#) method of [CalcQuickBase](#).

#### C#

```
CalcQuickBase calcQuick = new CalcQuickBase();
//Computing expressions,
string formula = "(5+25)*2";
string result = calcQuick.ParseAndCompute(formula);
//Computing in built formulas,
string formula = "SUM(5,5)";
string result = calcQuick.ParseAndCompute(formula);
```

### Compute formula using ICalcData

Essential Calculate provides calculation support to arbitrary business objects through [ICalcData](#) interface.

The methods and events used in [ICalcData](#) interface are

- [GetValueRowCol](#) - A method that is used to get the value from mentioned row and column index.
- [SetValueRowCol](#) - A method that is used to set the value to mentioned row and column index.
- [WireParentObject](#) - A method that wires the ParentObject after the [CalcEngine](#) object is created or when a [RegisterGridAsSheet](#) call is made.
- [ValueChanged](#) - An event that occurs when the value is changed.

Below example shows the computation of formula using [ICalcData](#) interface.

#### Creating a Class from ICalcData

Create CalcData class derived from [ICalcData](#) interface,

#### C#

```
public class CalcData : ICalcData
{
    public event ValueChangedEventHandler ValueChanged;
    Dictionary<string, object> values = new Dictionary<string, object>();
    public object GetValueRowCol(int row, int col)
    {
        object value = null;
        var key = RangeInfo.GetAlphaLabel(col) + row;
        this.values.TryGetValue(key, out value);
        return value;
    }
    public void SetValueRowCol(object value, int row, int col)
    {
        var key = RangeInfo.GetAlphaLabel(col) + row;
        if (!values.ContainsKey(key))
            values.Add(key, value);
        else if (values.ContainsKey(key) && values[key] != value)
            values[key] = value;
    }
    public void WireParentObject() {}
    private void OnValueChanged(int row, int col, string value)
    {
        if (ValueChanged != null)
            ValueChanged(this, new ValueChangedEventArgs(row, col, value));
    }
}
```

#### Setting Value into ICalcData

The [SetValueRowCol](#) method is used to set the value to [ICalcData](#) object.

#### C#

```
calcData.SetValueRowCol("10", 1, 1);
calcData.SetValueRowCol("20", 1, 2);
```

### Evaluation of formula

The `ICalcData` object can be integrated into `CalcEngine` by passing it through constructor. Now, you can compute the expressions or equations using `CalcEngine`.

The `ParseAndComputeFormula` method of `CalcEngine` is used to evaluate the formulas using the values from `ICalcData` object by cell references.

### C#

```
calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
string formula = "SUM (A1, B1)";
string result = engine.ParseAndComputeFormula(formula);
```

### Choosing between CalcQuickBase and ICalcData

#### CalcQuickBase

The simplest way to use Essential Calculate is through an instance of its `CalcQuickBase` class. This class provides options to directly parse and compute a formula, or register variable names that can later be used in more complex formulas involving these variables. But we cannot change the values of the variables at runtime for computation. Also by default, `CalcQuickBase` does not try to track any dependencies among the variables you set, hence to enable automatic recalculation of dependent variables, users need to set the `AutoCalc` property to `True`.

For more information regarding calculating with `CalcQuickBase`, refer [here](#)

#### ICalcData

To use Essential Calculate support, we need to derive the class from `ICalcData` interface which allows the `CalcEngine` class to communicate with arbitrary data sources.

To add calculation support to classes that represent data in a row/column format like a Data Grid, then you need to derive the classes inherited from `ICalcData` interface. Since `GetValueRowCol` and `SetValueRowCol` methods

of `ICalcData` interface is used to get and set the values of the variables at runtime for computation. `CalcEngine` listens to the `ValueChanged` event of `ICalcData` interface to tracks the dependencies and compute the formulas.

For more information regarding calculating with `ICalcData`, refer [here](#)

#### Cross Sheet Reference

`CalcEngine` provides support to perform the calculation by accessing the values from the different sheets using `RegisterGridAsSheet` method.

This method registers an `ICalcData` object so it can be referenced in a formula with another `ICalcData` object. `CreateSheetFamilyID` method of `CalcEngine` is used to create a unique family identifier for the sheet.

In `RegisterGridAsSheet` method, while registering the `ICalcData` objects, users need to pass this unique identifier to mark the `ICalcData` objects are belonging to this family. Also cross reference of `ICalcData` objects can be done within the same family.

The class which is derived from `ICalcData` can be registered as a worksheet for identifying the cell references.

Below code illustrates the registering of two `ICalcData` objects and use the cell references from two objects for computation of formula.

### C#

```
//Initialization of first ICalcData object in CalcEngine,
calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
//Initialization of second ICalcData object in CalcEngine,
calcData1 = new CalcData1();
engine = new CalcEngine(calcData1);
//Create a unique family id,
int i = CalcEngine.CreateSheetFamilyID();
//Register the first ICalcData object as "Sheet1",
engine.RegisterGridAsSheet("Sheet1", calcData, i);
//Register the second ICalcData object as "Sheet2",
engine.RegisterGridAsSheet("Sheet2", calcData1, i);
//Access the two sheet references in the formula,
string formula = "SUM(Sheet1!A1, Sheet2!A1)";
//Computation of result,
string result = engine.ParseAndComputeFormula(formula);
```

### Compute formulas in different region settings

Essential Calculate have support to compute the formulas in different region settings. The two static members such as `ParseArgumentSeparator` and `ParseDecimalSeparator` of `CalcEngine` class is used to set the separators based on local region settings. By default, the value of `ParseArgumentSeparator` is comma(,) and the value of `ParseDecimalSeparator` is dot(.).

### C#

```
//Assign the current culture's decimal separator,
CalcEngine.ParseDecimalSeparator =
System.Threading.Thread.CurrentThread.CurrentCulture.NumberFormat.NumberDecimalSeparator.ToCharArray()[0];
//Assign the current culture's argument separator,
CalcEngine.ParseArgumentSeparator =
System.Threading.Thread.CurrentThread.CurrentCulture.TextInfo.ListSeparator.ToCharArray()[0];
```

## Parse and Compute

This section describes about the parse and compute functions in Essential Calculate.

### Parsing

Essential Calculate have built in formula parser to parse the formula into well-formed version to compute internally.

#### Parse Formula

The built-in formula parser will parse the formula into Reverse Polish Notation expression using `ParseFormula` method of `CalcEngine` for computing it.



The parser uses some tokens to identify the operators, operands. It recognizes and replaces the `NameRanges` with their corresponding value. The parser also recognizes library functions and tokenizes them as well.

`ParseFormula` method accepts a string formula and checks whether it is a valid formula that `CalcEngine` can understand.

After that, it returns a string that represents a parsed version of the formula that can be more readily computed.

For example,

Formula: `2+3*1`

Parsed Formula in RPN format: `"n2n3n1ma"`

In this 'n' denotes as values and 'm' denotes as multiplication and 'a' as addition.

Using `ICalcData`,

**C#**

```
//Class derived from ICalcData,  
CalcData calcData = new CalcData();  
CalcEngine engine = new CalcEngine(calcData);  
string formula = "2+3*1";  
string parsedFormula = engine.ParseFormula(formula);
```

Using `CalcQuickBase`,

**C#**

```
CalcQuickBase calcQuick = new CalcQuickBase();  
string formula = "2+3*1";  
string parsedFormula = calcQuick.Engine.ParseFormula(formula);
```

### *Parsing Order*

The parsed formula is a Reverse Polish Notation expression using tokens to compactly represent the entered formula.

All the operands will be indexed into a stack and we need to pop out for calculation. It can be functions, references, operators or constants.

The parsing will be done from left to right and the order for parsing of operators is as follows,

1. E+ E- (handles exponential notation like 1.2e-1 or 1.2e+1 to 1.2e1)
2. ^
3. / \*
4. +(plus), -(minus)
5. < > = <= >= <>
6. &

### *Computation*

Essential Calculate provides support to calculate the formulas using various computation methods.

### *ComputeFormula*

[ComputeFormula](#) method of `CalcEngine` computes the parsed formula from `ParseFormula` method and returns the computed value.

It uses a stack oriented calculation technique to convert the parsed formula into the value that it represents.

Using `ICalcData`,

#### **C#**

```
//Class derived from ICalcData,  
CalcData calcData = new CalcData();  
CalcEngine engine = new CalcEngine(calcData);  
string formula = "2+3*1";  
string parsedFormula = engine.ParseFormula(formula);  
string result = engine.ComputeFormula(parsedFormula);
```

Using `CalcQuickBase`,

#### **C#**

```
CalcQuickBase calcQuick = new CalcQuickBase();  
string formula = "2+3*1";  
string parsedFormula = calcQuick.Engine.ParseFormula(formula);  
string result = calcQuick.Engine.ComputeFormula(parsedFormula);
```

### *ParseAndCompute*

The [ParseAndCompute](#) method of `CalcQuickBase` parses and computes the given formula string and returns the computed value.

#### **C#**

```
CalcQuickBase calcQuick = new CalcQuickBase();  
//Computing Expressions,  
string formula = "(5+25) *2";  
string result = calcQuick.ParseAndCompute(formula);  
//Computing In-Built formulas,  
string formula = "SUM (5,5)";  
string result = calcQuick.ParseAndCompute(formula);
```

### *ParseAndComputeFormula*

The [ParseAndComputeFormula](#) method of `CalcEngine` parses and computes the given formula string passed in and returns the computed value.

Using `ICalcData`,

#### **C#**

```
//Class derived from ICalcData,  
CalcData calcData = new CalcData();  
CalcEngine engine = new CalcEngine(calcData);  
//Computing Expressions,  
string formula = "(5+25) *2";  
string result = engine.ParseAndComputeFormula(formula);  
//Computing In-Built formulas,
```

```
string formula = "SUM (4,5,6)";  
string result = engine.ParseAndComputeFormula(formula);
```

Using CalcQuickBase,

### C#

```
CalcQuickBase calcQuick = new CalcQuickBase();  
//Computing Expressions,  
string formula = "(5+25) *2";  
string result = calcQuick.Engine.ParseAndComputeFormula(formula);  
//Computing In-Built formulas,  
string formula = "SUM (4,5,6)";  
string result = calcQuick.Engine.ParseAndComputeFormula(formula);
```

### Error Messages

The error messages that are displayed by Essential Calculate can be found in the string arrays such as [ErrorStrings](#) and [FormulaErrorStrings](#) of [CalcEngine](#).

After a [CalcEngine](#) object has been created, the text of error messages in this array lists can be changed by altering the array values.

- [ErrorStrings](#) of [CalcEngine](#) gets or sets the list of error strings which are recognized by Excel such as "#N/A", "#VALUE!", "#REF!", "#DIV/0!", "#NUM!", "#NAME?", "#NULL!".
- [FormulaErrorStrings](#) of [CalcEngine](#) holds the list of error strings which are used within the Essential Calculate internally. Users can make changes to this internal error strings

default settings by assigning the new strings to the corresponding position. [ReloadErrorStrings](#) should be invoked to reset or modify the internal error strings.

Below shows the list of [FormulaErrorStrings](#) which are used internally,

"binary operators cannot start an expression",

"cannot parse",

"bad library",

"invalid char in front of",

"number contains 2 decimal points",

"expression cannot end with an operator",

"invalid characters following an operator",

"invalid character in number",

"mismatched parentheses",

"unknown formula name",

"requires a single argument",

"requires 3 arguments",

"invalid Math argument",

"requires 2 arguments",  
"#NAME?",  
"too complex",  
"circular reference: ",  
"missing formula",  
"improper formula",  
"invalid expression",  
"cell empty",  
"bad formula",  
"empty expression",  
"",  
"mismatched string quotes",  
"wrong number of arguments",  
"invalid arguments",  
"iterations do not converge",  
"Control named '{0}' is already registered",  
"Calculation overflow",  
"Missing sheet"

### Formatting the Computed Results

By default, the values will be returned as an object after computation. This value can be converted as a string by using `ToString` method to format the results.

To format the result of the calculations, the result can be parsed by using any of the formatting methods.

#### For example,

`int.Parse` -> For converting to integer

`double.Parse` -> For converting to double

`decimal.Parse` -> For converting to decimal

Using `ICalcData`,

#### C#

```
//Class derived from ICalcData,  
CalcData calcData = new CalcData();  
CalcEngine engine = new CalcEngine(calcData);  
string formula = "SUM (4,5,6)";  
//Formatted as decimal value,  
string result = decimal.  
Parse(engine.ParseAndComputeFormula(formula)).ToString("0.00");  
//Formatted as double value,
```

```
string result1 =  
double.Parse(engine.ParseAndComputeFormula(formula)).ToString("0.00%");
```

Using CalcQuickBase,

### C#

```
CalcQuickBase calcQuick = new CalcQuickBase();  
string formula = "SUM (4,5,6)";  
//Formatted as decimal value,  
string result =  
decimal.Parse(calcQuick.ParseAndCompute(formula)).ToString("0.00");  
//Formatted as double value,  
string result1 =  
double.Parse(calcQuick.ParseAndCompute(formula)).ToString("0.00%");
```

## Working with CalcQuickBase

The simplest way to use Essential Calculate is through an instance of its [CalcQuickBase](#) class. This class provides options to directly parse and compute a formula, or register variable names that can later be used in more complex formulas involving these variables.

After registering the variables, it provides options to perform manual or automatic calculations.

[CalcQuickBase](#) is predefined derived class from [ICalcData](#) interface.

### Compute using values

The [ParseAndCompute](#) method of [CalcQuickBase](#) parses and computes the given formula string and returns the computed value.

### C#

```
CalcQuickBase calcQuick = new CalcQuickBase();  
//Computing Expressions,  
string formula = "(5+25)*2";  
string result = calcQuick.ParseAndCompute(formula);  
//Computing In-Built formulas,  
string formula = "SUM(5,5)";  
string result = calcQuick.ParseAndCompute(formula);
```

### Compute using Variables

Computation using variables is performed by evaluating the formulas or expressions with the [CalcQuickBase](#) registered variable key values.

#### Register variable names

To register the variables in [CalcQuickBase](#), it must be enclosed within square brackets "[ ]". Eg. [A]. These registered variable names are indexer keys.

A variable name must begin with an alphabetical character and can contain only letters and digits. It is not case-sensitive. To register a string as a variable name and

simply index the [CalcQuickBase](#) object with the name. To set its value, assign the value or formula to the registered variable name.

### C#

```
CalcQuickBase calcQuick = new CalcQuickBase();  
calcQuick["A"] = "5";  
calcQuick["B"] = "6";  
calcQuick["C"] = "11";
```

### Evaluation

#### Compute directly with variables

If the user wants the variable to hold a string which is a formula or be treated as a formula, then begin that string with [FormulaCharacter](#) of `CalcQuickBase`. The default value of this `FormulaCharacter` is `"="`.

so, that it is parsed and computed through the indexing code.

Below example shows the computation of formula or expressions directly with registered variable keys.

#### C#

```
CalcQuickBase calcQuick = new CalcQuickBase();  
calcQuick["A"] = "5";  
calcQuick["B"] = "6";  
calcQuick["C"] = "11";  
//Computing expressions,  
calcQuick["result"] = "=[A]+[B])/[C]";  
//Computing In-Built formulas,  
calcQuick["result"] = "=SUM([A],[B])"
```

#### Compute using ParseAndCompute method

The [ParseAndCompute](#) method of `CalcQuickBase` parses and computes the given formula string and returns the computed value.

Any string that is passed in `ParseAndCompute` method of `CalcQuickBase` directly will be treated as a formula, whether or not it begins with `FormulaCharacter`.

Below example shows the computation of formula or expressions using `ParseAndCompute` method of `CalcQuickBase`.

#### C#

```
CalcQuickBase calcQuick = new CalcQuickBase();  
calcQuick["A"] = "5";  
calcQuick["B"] = "6";  
calcQuick["C"] = "11";  
//Computing expressions,  
calcQuick["result"] = calcQuick.ParseAndCompute("([A]+[B])/[C]");  
//Computing in built formulas,  
calcQuick["result"] = calcQuick.ParseAndCompute("SUM([A],[B])");
```

For more information regarding `ParseAndCompute` method, refer [here](#).

#### Automatic calculations

By default, `CalcQuickBase` does not try to track any dependencies among the variables. To enable automatic recalculation of dependent variables,

set the [AutoCalc](#) property of `CalcQuickBase` to `True`. Once this is done, the `CalcQuickBase` object maintains the required dependency information.

[RefreshAllCalculations](#) method of `CalcQuickBase` forces the recalculation of all variables registered with the `CalcQuickBase` object.

This has to be done after the `AutoCalc` property has been set to `True`, so that the dependencies between variables can be monitored.

### C#

```
//Initialize,
CalcQuickBase calcQuick = new CalcQuickBase();
//Registering keys with values,
calcQuick["A"] = "5";
calcQuick["B"] = "6";
calcQuick["C"] = "11";
//Computing in built formulas,
calcQuick["result"] = calcQuick.ParseAndCompute("SUM([A],[B],[C])");
//Setting the Auto calculation mode,
calcQuick.AutoCalc = true;
//To recompute formulas stored in CalcQuickBase object,
calcQuick.RefreshAllCalculations();
//Changing the variable "C" value to "3",
calcQuick["C"] = "3";
//Output result after the change of variable "C",
var Output = calcQuick["result"];
```

### Reset keys

Registered variables or indexer keys registered with `CalcQuickBase` object can be cleared or reset by using [ResetKeys](#) method of `CalcQuickBase` class.

### C#

```
//Initialize,
CalcQuickBase calcQuick = new CalcQuickBase();
//Registering keys with values,
calcQuick["A"] = "5";
calcQuick["B"] = "6";
calcQuick["C"] = "11";
//Clears the registered keys,
calcQuick.ResetKeys();
```

## Working with ICalcData

Essential Calculate provides calculation support to arbitrary business objects through [ICalcData](#) interface. To add calculation support to classes that represent data in a row/column format like a Data Grid, then you need to derive the classes inherited from `ICalcData` interface.

### Methods and Events in ICalcData

`ICalcData` has three methods and one event. This interface allows the [CalcEngine](#) class in Essential Calculate to communicate with arbitrary data sources that implement this interface.

### SetValueRowCol

[SetValueRowCol](#) method is used to set the value to mentioned row and column index. Essential Calculate expects any indexes (rows / column integer values) to be one-based.

An example of defining the `SetValueRowCol` method in custom class(CalcData) is explained below,

#### C#

```
//Custom class,
public class CalcData : ICalcData
{
    Dictionary<string, object> values = new Dictionary<string, object>();
    //Defining SetValueRowCol method in Custom(user defined) Class,
    public void SetValueRowCol(object value, int row, int col)
    {
        var key = RangeInfo.GetAlphaLabel(col) + row;
        if (!values.ContainsKey(key))
            values.Add(key, value);
        else if (values.ContainsKey(key) && values[key] != value)
            values[key] = value;
    }
}

//Main class,
public void Main()
{
    CalcData calcData = new CalcData();
    //To set the data value of a specified row and column,
    calcData.SetValueRowCol(90, 1, 1);
    calcData.SetValueRowCol(50, 1, 2);
}
```

### GetValueRowCol

[GetValueRowCol](#) method is used to get the value from mentioned row and column index. Essential Calculate expects any indexes (rows / column integer values) to be one-based.

An example of defining the `GetValueRowCol` method in custom class(CalcData) is explained below,

#### C#

```
//Custom class,
public class CalcData : ICalcData
{
    Dictionary<string, object> values = new Dictionary<string, object>();
    //Defining GetValueRowCol method in Custom(user defined) Class,
    public object GetValueRowCol(int row, int col)
    {
        object value = null;
        var key = RangeInfo.GetAlphaLabel(col) + row;
        this.values.TryGetValue(key, out value);
        return value;
    }
}

//Main class,
public void Main()
{
    CalcData calcData = new CalcData();
    //To get the data value of a specified row and column,
```



```
var value1 = calcData.GetValueRowCol(1, 1);
var value2 = calcData.GetValueRowCol(1, 2);
}
```

### WireParentObject

[WireParentObject](#) method that wires the ParentObject after the [CalcEngine](#) object is created or when a [RegisterGridAsSheet](#) call is made. The purpose is to give the data object

a chance to do any initialization steps it may need, such as subscribe the events to handle the changes in data notifications.

### ValueChanged

[ValueChanged](#) event of [ICalcData](#) interface occurs whenever the value is changed. The [CalcEngine](#) listens to this event and accordingly reacts to data changes.

It is through this event that formulas are processed and dependencies are tracked by the [CalcEngine](#).

### C#

```
//Custom class,
public class CalcData : ICalcData
{
    public event ValueChangedEventHandler ValueChanged;
    private void OnValueChanged(int row, int col, string value)
    {
        if (ValueChanged != null)
            ValueChanged(this, new ValueChangedEventArgs(row, col, value));
    }
}
```

### Computation using ICalcData

Below example shows the computation of formula using [ICalcData](#) interface.

#### Creating a Class from ICalcData

Create CalcData class derived from [ICalcData](#) interface,

### C#

```
public class CalcData : ICalcData
{
    public event ValueChangedEventHandler ValueChanged;
    Dictionary<string, object> values = new Dictionary<string, object>();
    public object GetValueRowCol(int row, int col)
    {
        object value = null;
        var key = RangeInfo.GetAlphaLabel(col) + row;
        this.values.TryGetValue(key, out value);
        return value;
    }
    public void SetValueRowCol(object value, int row, int col)
    {
        var key = RangeInfo.GetAlphaLabel(col) + row;
        if (!values.ContainsKey(key))
            values.Add(key, value);
        else if (values.ContainsKey(key) && values[key] != value)
            values[key] = value;
    }
}
```

```
values[key] = value;
}
public void WireParentObject() {}
private void OnValueChanged(int row, int col, string value)
{
    if (ValueChanged != null)
        ValueChanged(this, new ValueChangedEventArgs(row, col, value));
}
}
```

### Setting Value into ICalcData

The `SetValueRowCol` method is used to set the value to `ICalcData` object.

#### C#

```
CalcData calcData = new CalcData();
calcData.SetValueRowCol(10, 1, 1);
calcData.SetValueRowCol(20, 1, 2);
```

### Initialization of CalcEngine

The `ICalcData` object can be integrated into `CalcEngine` by passing it through constructor. Now, you can compute the expressions or equations using `CalcEngine`.

#### C#

```
CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
```

### Evaluation of formula

The `ParseAndComputeFormula` method of `CalcEngine` is used to evaluate the formulas using the values from `ICalcData` object by cell references.

#### C#

```
CalcData calcData = new CalcData();
calcData.SetValueRowCol(10, 1, 1);
calcData.SetValueRowCol(20, 1, 2);
CalcEngine engine = new CalcEngine(calcData);
string formula = "SUM (A1, B1)";
string result = engine.ParseAndComputeFormula(formula);
```

---

**Note:** To support cross-references among several `ICalcData` objects, you must register the objects with a single instance of the `CalcEngine`.

---

For more reference, refer [here](#).

### Working with CalcEngine

`CalcEngine` encapsulates the code required to parse and compute the formulas. It manages several library functions, hash tables for Essential Calculate.

All the data's in `CalcEngine` is assumed to be part of a rectangular array reference through cell coordinates.

### Computation using CalcEngine

The [CalcEngine](#) interface allows the `CalcEngine` class to communicate with arbitrary data sources that implement this interface.

If any user defined class which implements the `ICalcData` interface needs to be integrated with `CalcEngine`, then that class's object can be passed as parameter

to the `CalcEngine` constructor.

The [ParseAndComputeFormula](#) method of `CalcEngine` parses and computes the string formula passed in and updates the computed value.

#### Using ICalcData

The `ICalcData` object can be integrated into `CalcEngine` by passing it through constructor. Now, you can compute the expressions or equations using `ParseAndComputeFormula`.

#### C#

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
//Computing Expressions,
string formula = "(5+25) *2";
string result = engine.ParseAndComputeFormula(formula);
//Computing In-Built formulas,
string formula = "SUM (4,5,6)";
string result = engine.ParseAndComputeFormula(formula);
```

#### Using CalcQuickBase

[CalcQuickBase](#) is predefined derived class from `ICalcData` interface. So, user can pass the `CalcQuickBase`'s object as parameter to the constructor of `CalcEngine` for computations.

#### C#

```
CalcQuickBase calcQuick = new CalcQuickBase();
//Computing Expressions,
string formula = "(5+25) *2";
string result = calcQuick.Engine.ParseAndComputeFormula(formula);
//Computing In-Built formulas,
string formula = "SUM (4,5,6)";
string result = calcQuick.Engine.ParseAndComputeFormula(formula);
```

### Data Objects maintained in CalcEngine

To track information used during calculations, `CalcEngine` manages several hash tables. In `CalcEngine`, the data's are referred in cell coordinates and these references are maintained as keys in the hash tables.

Below find the list of hash tables in `CalcEngine` and a description of their keys and values.

Hash Table	Key	Value	Description
------------	-----	-------	-------------

<a href="#">FormulaInfoTable</a>	Cell reference	FormulaInfo object	Tracks formula/value information for this cell.
<a href="#">DependentCells</a>	Cell reference	Hashtable object	Tracks cells that depend on this cell.
<a href="#">DependentFormulaCells</a>	Formula cell reference	Hashtable object	Tracks cells in which the formula cell depends upon.
<a href="#">NamedRanges</a>	Name string	Value string	Associates the named range with its value.
<a href="#">LibraryFunctions</a>	Function name	LibraryFunction delegate	Associates the function name with its method.

## Properties in CalcEngine

### [ExcelLikeComputations](#)

To return the result of formula computations of Essential Calculate like Microsoft Excel computational result, this bool property [ExcelLikeComputations](#) is set to true.

### C#

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
engine.ExcelLikeComputations = true;
//Computing the DATE formula similar to excel,
string formula = "DATE(2004,5,6)";
string result = engine.ParseAndComputeFormula(formula);
```

### [AllowShortCircuitIFs](#)

To specifically avoid the computing of non-used alternative in IF function calculations, set [AllowShortCircuitIFs](#) to true.

The default value is false for code legacy consistency. But when it is set as true, only the necessary alternative of an IF function is computed which

leads to increase in performance.

### [AlwaysComputeDuringRefresh](#)

If [PullUpdatedValue](#) method is exclusively used to retrieve the computed values, then setting [AlwaysComputeDuringRefresh](#) property to false

may be more efficient as it will only recompute the value once during the calculations.

### [CalculatingSuspended](#)

[CalculatingSuspended](#) property is set to true, to suspend calculations while a series of changes are made to dependent cells either by the user or programmatically.

When the changes are complete, set this property to false, and then call [RecalculateRange](#) to recalculate the affected ranges.

### C#

```
//Class derived from ICalcData,
```

```

CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
// Turn off calculations
engine.CalculatingSuspended = true;
// Makes multiple updates to cells involved in calculation
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 5; j++)
    {
        calcData.SetValueRowCol(random.Next(5) + 1, i, j);
    }
}
// Turn on calculations
engine.CalculatingSuspended = false;
// invoke RecalculateRange, so any formulas in the data can be computed.
engine.RecalculateRange(RangeInfo.Cells(4, 4, 2, 1), calcData);

```

### UseFormulaValues

This property is set to true, when a cell contains more dependency cells to return the formula value stored in `FormulaInfoTable` to avoid

repeated calculations.

### CheckDanglingStack

While computing certain formulas, Essential Calculate ignores the dangling value in the calculation stack and returns the computed value.

If you want this situation flagged as a invalid formula, then set [CheckDanglingStack](#) property to true. The default value is false for backward compatibility purpose.

### ForceRefreshCall

If a value changes, then the [Refresh](#) method is called recursively every time whenever the [ValueChanged](#) event of `ICalcData` is fired. Setting this [ForceRefreshCall](#) property to false will call the [Refresh](#) method for only the cells whose calculated value is actually modified.

### FormulaCharacter

In general, in order for Essential Calculate to recognize a string as containing a formula; then the string is required to start with the [FormulaCharacter](#).

A static property that gets or sets the character to identify the formula. The default value is "=".

### GetValueFromArgPreserveLeadingZeros

If you want to preserve the leading zeros in [GetValueFromArg](#) method, then set [GetValueFromArgPreserveLeadingZeros](#) to true.

## C#

```

//Class derived from ICalcData,
CalcData calcData = new CalcData();
//Assign a value with leading zeros to "A1" cell,
calcData.SetValueRowCol(04510, 1, 1);
CalcEngine engine = new CalcEngine(calcData);
//Setting this property to true, for preserving leading zeros,
engine.GetValueFromArgPreserveLeadingZeros = true;
//Returns the result with leading zeros,
string result = engine.GetValueFromArg("A1")

```

### *IterationMaxCount*

Gets or sets the maximum number of iterative calls that can be made on a cell to avoid circular exception. The default value is 0 indicating that iterative

calculation support is turned off. [ThrowCircularException](#) property will be automatically set to true when you set a non-zero value to [IterationMaxCount](#).

### *IterationMaxTolerance*

Gets or sets the success tolerance used by the **CalcEngine's** iterative calculation support. The iterations in calculation continues until either the iteration count

exceeds [IterationMaxCount](#), or two successive iteration return values have a relative difference less than [IterationMaxTolerance](#). The default value is 0.001.

### *LockDependencies*

A bool property that gets or sets whether a changed value should trigger dependent changes.

### *MaximumRecursiveCalls*

Specifies the maximum number of recursive calls that can be used to compute a cell value. This property comes into play when you have a calculated formula cell that depends on

another calculated formula that depends on another calculated formula and so on. If the depends on another formula exceeds [MaximumRecursiveCalls](#), you will see a too complex message displayed in the cell. The default value is 100, but you can set it higher or lower depending upon your expected needs.

The purpose of the limit is to avoid a circular

reference locking up your application.

### *MaxStackDepth*

Gets or sets the maximum calculation stack depth. The default value is 50. This is the number of recursive calls that can be made during calculations.

### *ParseArgumentSeparator*

A static property that gets or sets the character to be recognized by the parsing code as the delimiter for arguments in a named formula's argument list. The default value is ','.

### **C#**

```
//Assign the argument separator,  
CalcEngine.ParseArgumentSeparator = ',';
```

### *ParseDecimalSeparator*

A static property that gets or sets the character to be recognized by the parsing engine as decimal separator for numbers. The default value is '.'.

### **C#**

```
//Assign the decimal separator,  
CalcEngine.ParseDecimalSeparator = ',';
```

### *ParseDateTimeSeparator*

A static property that gets/sets the character to be recognized by the parsing engine as decimal separator for date. The default value is '/'.

**C#**

```
//Assign the date time separator,
CalcEngine.ParseDateTimeSeparator = '-';
```

*RethrowLibraryComputationExceptions*

Gets or sets whether the **CalcEngine** rethrow any exception raised during the computation of a library function. The default value is false.

*SupportLogicalOperators*

If you want to use any logical operators such as AND, OR, XOR,... in a formula, you can set [SupportLogicalOperators](#) property of **CalcEngine** as true .

**C#**

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
engine.SupportLogicalOperators = true;
string result = engine.ParseAndComputeFormula("=IF("C"="S "OR"
C"="C",1,2)");
```

*SupportsSheetRanges*

Gets or sets whether the sheet range notation is supported. The default value is true. For backward compatibility with earlier versions that did

not support the sheet range notation, you can set [SupportsSheetRanges](#) property to false.

**C#**

```
//Initialization of first ICalcData object in CalcEngine,
calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
//Initialization of second ICalcData object in CalcEngine,
calcData1 = new CalcData();
engine = new CalcEngine(calcData1);
//Create a unique family id,
int i = CalcEngine.CreateSheetFamilyID();
//Register the first ICalcData object as "Sheet1",
engine.RegisterGridAsSheet("Sheet1", calcData, i);
//Register the second ICalcData object as "Sheet2",
engine.RegisterGridAsSheet("Sheet2", calcData1, i);
engine.SupportsSheetRanges = true;
string result =
engine.ParseAndComputeFormula("=SUM(sheet1!B1:B4,sheet2!B1:B4)");
```

*ThrowCircularException*

Gets or sets whether the **CalcEngine** should throw an exception when a circular calculation is encountered. If [ThrowCircularException](#) property is true, then **CalcEngine** will throw an exception when it detects a circular calculation, else no exception is thrown and the calculation will loop recursively until [MaximumRecursiveCalls](#) is exceeded.

### *UseDependencies*

Gets or sets whether the `CalcEngine` should track dependencies or not. If you invoke [PullUpdatedValue](#) to access the computations, then setting [UseDependencies](#) property to false will make things more efficient as any requested computed value will be fully computed every time it is retrieved. In this situation, the `CalcEngine` does not need to track dependencies.

### *UseNoAmpersandQuotes*

Gets or sets whether the computational result of strings should be returned with double quotes or not.

### **C#**

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
engine.UseNoAmpersandQuotes = true;
string result =
engine.ParseAndComputeFormula("=CONCAT(\"abc\", \"sample\")");
```

### *UseDatesInCalculations*

Gets or sets whether dates can be used as operands in calculations. The default value is false. Setting [UseDatesInCalculations](#) property to true, you can compute DateTime related calculations in your application.

### **C#**

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
engine.UseDatesInCalculations = true;
calcData.SetValueRowCol("02/10/2017 11:15 am", 1, 1);
calcData.SetValueRowCol("02/12/2017 9:45 am", 2, 1);
//Computing the difference in hours between two date time values,
string result = engine.ParseAndComputeFormula("=(A2-A1)*24");
```

### *TreatStringsAsZero*

Setting the property [TreatStringsAsZero](#) as true means that if a nonempty string is encountered during an arithmetic

operation or computation, it will be treated as zero.

### Methods in CalcEngine

#### *GetValueFromArg*

This method takes the argument and checks whether it is a parsed formula, a raw number, or a cell reference. It returns the string

by computing the value based on the argument passed in.

### **C#**

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
string result = engine.GetValueFromArg("=SUM(4, 5, 6)");
```



*GetValueFromParentObject*

This method takes the argument and check whether it is a formula or a cell reference and returns either the computed value or the cell value.

**C#**

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
//Assign values,
calcData.SetValueRowCol(2, 1, 2);
calcData.SetValueRowCol(3, 2, 2);
calcData.SetValueRowCol(4, 3, 2);
calcData.SetValueRowCol("=SUM(B1,B2,B3)", 1, 4);
CalcEngine engine = new CalcEngine(calcData);
//Passing the cell reference,
string result = engine.GetValueFromParentObject("D1")
//Passing the row/column index,
string result = engine.GetValueFromParentObject(1, 4);
```

**Note:** [GetValueFromParentObject](#) does not accepts the argument as raw number or parsed formula, it only accepts the cell references whereas [GetValueFromArg](#) accepts parsed formula, a raw number, or a cell reference

*ParseAndComputeFormula*

A method that parses and computes the string formula passed in. For more reference, see [here](#).

*ParseFormula*

A method that parses the string formula passed in. For more reference, see [here](#).

*ComputeFormula*

A method that computes a parsed formula. For more reference, see [here](#).

*RegisterGridAsSheet*

A method that registers an `ICalcData` object so it can be referenced in a formula from another `ICalcData` object. For more reference, see [here](#).

*UpdateCalcID*

Every formula has a calculation ID level associated with it. Every time a formula is retrieved, its calculation ID level is compared with the `CalcEngine`

ID level. If they do not agree, the formula is recomputed. Invoking [UpdateCalcID](#) will force any formula to be recomputed the next time it is retrieved.

**C#**

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
//Assign values initially,
calcData.SetValueRowCol(2, 1, 2);
calcData.SetValueRowCol(3, 2, 2);
calcData.SetValueRowCol(4, 3, 2);
calcData.SetValueRowCol("=SUM(B1,B2,B3)", 1, 4);
CalcEngine engine = new CalcEngine(calcData);
//compute the formula,
string val = engine.ParseAndComputeFormula("D1");
//Changing the value of B2 cell,
```

```
calcData.SetValueRowCol(20, 2, 2);
//Invoking UpdateCalcID method to retrieve latest change,
engine.UpdateCalcID();
//Computing the formula with latest changes,
string val1 = engine.ParseAndComputeFormula("D1");
```

### PullUpdatedValue

This method retrieves the value in the requested cell reference by computing with latest changes of cells that affect the value of the requested cell.

### C#

```
//Initialization of first ICalcData object in CalcEngine,
calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
//Initialization of second ICalcData object in CalcEngine,
ICalcData calcData1 = new CalcData1();
engine = new CalcEngine(calcData1);
//Register the first ICalcData object as "Sheet1",
engine.RegisterGridAsSheet("Sheet1", calcData, 1);
//Register the second ICalcData object as "Sheet2",
engine.RegisterGridAsSheet("Sheet2", calcData1, 1);
//Assign a formula in the calcData1's A1 cell,
calcData1.SetValueRowCol("=SUM(Sheet1!A1, Sheet1!B2, Sheet1!C2,
Sheet1!D2)", 1, 1);
Random r = new Random();
engine.CalculatingSuspended = true;
//Set random values,
calcData.SetValueRowCol(r.Next(74) + 15, 1, 1);
calcData.SetValueRowCol(r.Next(2), 2, 2);
calcData.SetValueRowCol(r.Next(50), 3, 2);
calcData.SetValueRowCol(r.Next(15), 4, 2);
calcData.SetValueRowCol(r.Next(11), 5, 2);
calcData.SetValueRowCol(33 + r.Next(1972), 6, 2);
calcData.SetValueRowCol(r.Next(2), 7, 2);
calcData.SetValueRowCol(r.Next(20), 8, 5);
//Calculations are suspended so need to pull the computed value to make sure
it has been calculated with the latest changes,
engine.UpdateCalcID();
engine.PullUpdatedValue(engine.GetSheetID(calcData1), 1, 1);
//Get the Calculated value from calcData1 object,
string val = calcData1.GetValueRowCol(1, 1).ToString();
engine.CalculatingSuspended = false;
```

### RecalculateRange

Recalculates any formula cells in the specified range of **CalcEngine**.

### C#

```
// Creates some data object that implements ICalcData.
this.data = new ArrayCalcData(a);
// Creates a CalcEngine object using this ICalcData object.
CalcEngine engine = new CalcEngine(this.data);
//Turn off calculations.
engine.CalculatingSuspended = true;
```

```
// Makes multiple updates to this.data.
// Turn on calculations.
engine.CalculatingSuspended = false;
// Calls RecalculateRange so any formulas in the data can be computed.
engine.RecalculateRange(RangeInfo.Cells(1, 1, nRows + 1, nCols + 1), data);
```

### UpdateDependenciesAndCell

This method triggers a calculation for any cells depending upon the given cell.

### C#

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
//Assign values initially,
calcData.SetValueRowCol(2, 1, 2);
calcData.SetValueRowCol(3, 2, 2);
calcData.SetValueRowCol(4, 3, 2);
calcData.SetValueRowCol("=SUM(B1,B2,B3)", 1, 4);
CalcEngine engine = new CalcEngine(calcData);
//Add the formula in formula info hash table,
engine.FormulaInfoTable.Add("D1", new FormulaInfo{FormulaText =
"=SUM(B1,B2,B3)"});
//compute the formula,
string val = engine.ParseAndComputeFormula("D1");
//Changing the value of B2 cell,
calcData.SetValueRowCol(20, 2, 2);
//Invoke UpdateDependenciesAndCell method for the formula cell,
engine.UpdateDependenciesAndCell("D1");
//Computing the formula with latest changes,
string val1 = engine.ParseAndComputeFormula("D1");
```

## Working with Essential XlsIO

Essential XlsIO provides an Excel-like Automation-type support without having Microsoft Excel installed on the host system. This means that you can use this library

to read and write an XLS file and hold its contents in memory. But you cannot perform actual computations on the contents of the XLS file. Hence Essential Calculate

is integrated with Essential XlsIO, to calculate formulas entered at runtime without any additional references or packages.

### Open a Workbook using XlsIO

To open a workbook using XlsIO, instantiate the [ExcelEngine](#) to initialize the application object for creating or manipulating Excel documents. To open an existing workbook,

use the [Open](#) methods of [IWorkbook](#) interface.

### C#

```
//Creates a new instance for ExcelEngine.
ExcelEngine excelEngine = new ExcelEngine();
//Loads or open an existing workbook through Open method of IWorkbook
IWorkbook workbook =
excelEngine.Excel.Workbooks.Open(@"..\..\Data\Sample.xlsx");
```

### Enable and Disable Calculations in XlsIO

To perform calculation in an Excel workbook, it is recommended to invoke [EnableSheetCalculations](#) method of [IWorksheet](#). Enabling this method will initialize

[CalcEngine](#) objects and retrieves calculated values of formulas in a worksheet.

On completion of worksheet calculation, it is also recommended to invoke [DisableSheetCalculations](#) method of [IWorksheet](#).

This will dispose all the [CalcEngine](#) objects.

### C#

```
//Creates a new instance for ExcelEngine,
ExcelEngine excelEngine = new ExcelEngine();
//Loads or open an existing workbook through Open method of IWorkbook,
IWorkbook workbook =
excelEngine.Excel.Workbooks.Open(@"..\..\Data\Sample.xlsx");
//Accessing the worksheet,
IWorksheet sheet = workbook.Worksheets[0];
//Formula calculation is enabled for the sheet,
sheet.EnableSheetCalculations();
//Assigning values in the worksheet,
worksheet["C3"].Number = 45;
worksheet["C4"].Number = 20;
worksheet["C5"].Number = 38;
//Assigning the formula in the worksheet,
worksheet["C24"].Formula = "=SUM(C3:C4)-C5";
//Getting the calculated value,
var value = sheet.Range["C24"].CalculatedValue;
//Formula calculation is disabled for the sheet,
sheet.DisableSheetCalculations();
```

### Set and Compute the values at runtime in the Worksheet

At runtime, user can set the values in the particular [IWorksheet](#) by indexing the worksheet with the sheet name or id

and then use the appropriate row and column indexes. Invoking [UpdateCalcId](#) and [PullUpdatedValue](#) method of [CalcEngine](#)

guarantees the current/updated values in the workbook.

[CalculatingSuspended](#) method of [CalcEngine](#) is to suspend calculations while a series of changes are made to dependent cells

either by the user or programmatically. When the changes are complete, set this property to False.

### C#

```
//Creates a new instance for ExcelEngine,
ExcelEngine excelEngine = new ExcelEngine();
//Loads or open an existing workbook through Open method of IWorkbook,
IWorkbook workbook =
excelEngine.Excel.Workbooks.Open(@"..\..\Data\Sample.xlsx");
//Accessing the worksheet,
IWorksheet sheet = workbook.Worksheets["Inputs"];
Random r = new Random();
```

```

sheet.CalcEngine.CalculatingSuspended = true;
//Set random values into the "Inputs" sheet,
sheet[1, 2].Value = (r.Next(74) + 15).ToString();
sheet[2, 2].Value = r.Next(2) == 1 ? "M" : "F";
sheet[3, 2].Value = r.Next(50);
sheet[4, 2].Value = r.Next(15).ToString();
sheet[5, 2].Value = r.Next(11).ToString();
sheet[6, 2].Value = (33 + r.Next(1972)).ToString();
sheet[7, 2].Value = r.Next(2) == 1 ? "Yes" : "No";
sheet[8, 5].Value = r.Next(20);
//Calculations are suspended so need to pull the computed value to make sure
it has been calculated with the latest changes,
sheet.CalcEngine.UpdateCalcID();
sheet.CalcEngine.PullUpdatedValue(sheet.CalcEngine.GetSheetID("Outputs"), 1,
1);
//Get the Calculated value from "Outputs" sheet,
string val = workbook.Worksheets["Outputs"][1, 1].CalculatedValue;
sheet.CalcEngine.CalculatingSuspended = false;

```

To compute particular cell in the worksheet

To compute particular cell in the worksheet, use [ParseAndComputeFormula](#) method of `CalcEngine`. For more details regarding

`ParseAndComputeFormula` method, refer [here](#).

#### C#

```

//Creates a new instance for ExcelEngine,
ExcelEngine excelEngine = new ExcelEngine();
//Loads or open an existing workbook through Open method of IWorkbook,
IWorkbook workbook =
excelEngine.Excel.Workbooks.Open(@"..\..\Data\Sample.xlsx");
//Accessing the worksheet,
IWorksheet sheet = workbook.Worksheets["Sheet1"];
//To calculate particular cell,
sheet.CalcEngine.ParseAndComputeFormula(sheet["C5"].Formula);

```

#### Ambiguity Issue

If the `Calculate.Base` and `XlsIO.Base` references are added in the same application, it will throw conflict errors. Since `Calculate.Base` is already integrated with `XlsIO`. Hence, if `XlsIO.Base` reference has been included in the application,

there is no need to add `Calculate.Base` reference explicitly. The calculate references get reflected in the `XlsIO.Base` permanently.

But if you want both the references in your project, you can use `extern alias` to differentiate the namespaces.

For your reference, please find the MSDN [link](#) regarding

`extern alias`.

#### Table Formulas

A table is a collection of data about a specific topic that is stored in rows and columns. These tables are defined with a name and `CalcEngine` supports these table format.

For Example: =SUM(Table1[#All],[Column1]:[Column2])

A table needs to be defined with the following protocols,

- All table, column, and special item specifiers must be enclosed in matching brackets [ ]
- Expression cannot be used with these brackets. Column headers should be a text strings.
- The special characters such as comma ,, colon :, period ., left bracket [ , right bracket ], pound sign #, single quotation mark ', double quotation mark ",

left brace {, right brace }, dollar sign \$, caret ^, ampersand &, asterisk \*, plus sign +, equal sign =, minus sign -, greater than symbol >, less than symbol <, and division sign / can be used.

These table are converted into cell ranges and then it will be evaluated. The data from one row can also be taken with this table structure.

### C#

```
//Creates a new instance for ExcelEngine,
ExcelEngine excelEngine = new ExcelEngine();
//Loads or open an existing workbook through Open method of IWorkbook,
IWorkbook workbook =
excelEngine.Excel.Workbooks.Open(@"..\..\Data\Sample.xlsx");
//Accessing the worksheet,
IWorksheet sheet = workbook.Worksheets[0];
//Formula calculation is enabled for the sheet,
sheet.EnableSheetCalculations();
//Create Table,
IListObject table1 = sheet.ListObjects.Create("Table1", sheet["A1:F6"]);
// Fill table data
sheet[1, 1].Text = "Column1";
sheet[1, 2].Text = "Column2";
sheet[1, 3].Text = "Column3";
sheet[2, 1].Number = 3;
sheet[2, 2].Number = 2;
sheet[2, 3].Number = 16.80;
sheet[3, 1].Number = 5;
sheet[3, 2].Number = 3;
sheet[3, 3].Number = 15.60;
sheet[4, 1].Number = 8;
sheet[4, 2].Number = 2;
sheet[4, 3].Number = 20.10;
string result1 =
sheet.CalcEngine.ParseAndComputeFormula("=SUM(Table1[Column1])");
string result2 =
sheet.CalcEngine.ParseAndComputeFormula("=MIN(Table1[#All])");
```

## Supported Formulas

This section shows the list of formulas supported in Essential Calculate library.

### 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
DSTEV	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
DSUM	Adds the numbers in the field (column) of records in a list or database that match the conditions you specify
DPRODUCT	Multiplies the values in the field(column) of records in a list or database that match the conditions you specify

### 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 $\hat{\wedge}$
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
EFFECT	Returns the effective annual interest rate
ACCRINTM	Returns the accrued interest for a security that pays interest at maturity

### 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
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
IFNA	Tests if an expression returns the #N/A error and if so, returns an alternative specified value; Otherwise the function returns the value of the supplied expression
IFS	Tests a number of supplied conditions and returns a result corresponding to the first condition that evaluates to TRUE
SWITCH	Compares a number of supplied values to a supplied test expression and returns a result corresponding to the first value that matches the test expression
XOR	Returns a logical Exclusive Or of all arguments

### 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

LOOKUP	Searches for a specific value in one data vector, and returns a value from the corresponding position of a second data vector
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
CHOOSE	Returns one of a list of values, depending on the value of a supplied index number
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
ADDRESS	Returns a reference, in text format, for a supplied row and column number
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 $\geq 0$ 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
ROUNDDOWN	Rounds a number down, toward zero

ROUNDUP	Returns 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 0â€¦.1 exclusive
PERCENTILE.INC	Returns the Kth percentile of the values in a range, where K is in the range 0â€¦.1 inclusive
PERCENTRANK.EXC	Returns the rank of value in dataset as a percentage of the data set as percentage (0â€¦.1, exclusive) of the dataset
PERCENTRANK.INC	Returns the rank of value in dataset as a percentage of the data set as percentage (0â€¦.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
LOGNORMDIST	Returns the cumulative log-normal 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
COUNTIFS	Returns the number of cells (of a supplied range), that satisfy a set of given criteria
AVERAGEIF	Calculates the Average of the cells in a supplied range, that satisfy a given criteria
AVERAGEIFS	Calculates the Average of the cells in a supplied range, that satisfy multiple criteria
MODE.SNGL	Returns the Mode (the most frequently occurring value) of a list of supplied numbers
MODE.MULT	Returns a vertical array of the most frequently occurring values in an array or range of data
SLOPE	Returns the slope of the linear regression line through a supplied series of x- and y- values
MAXIFS	Returns the largest value from a subset of values in a list that are specified according to one or more criteria
MINIFS	Returns the smallest value from a subset of values in a list that are specified according to one or more criteria
BETADIST	Returns the cumulative beta probability density function
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
GROWTH	Returns numbers in a exponential growth trend, based on a set of supplied x- and y- values
LOGEST	Returns the parameters of an exponential trend for a supplied set of 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
BAHTTEXT	Converts a number, plus the suffix "Baht" into Thai text
EXACT	Tests if two supplied text strings are exactly the same and if so, returns TRUE; Otherwise, returns FALSE. (case-sensitive)
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
TEXTJOIN	Joins together two or more text strings, separated by a delimiter
TRIM	Removes duplicate spaces, and spaces at the start and end of a text string
CLEAN	Removes all non-printable characters from a supplied 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)
CHAR	Returns the character that corresponds to a supplied numeric value
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

## Operators

The calculation type of an equation will be specified by operators. These operators will be prioritized in a default order for calculating the equation.

### Arithmetic Operators

The mathematical operations such as addition, subtraction, multiplication etc., can be performed using the arithmetic operators.

Arithmetic operator	denotation	Example
+ (plus sign)	Addition	1 + 2
- (minus sign - unary)	Negation	- 5
- (minus sign - binary)	Subtraction	4 - 2
* (asterisk)	Multiplication	2 * 5
/ (forward slash)	Division	10 / 2
^ (caret)	Exponentiation	5 ^ 2

### Logical Operators

The logical or comparison operators will be used to compare two values of the formula. The return value will be either **True** or **False**.

If you use a well-formed logical expression in a larger calculation, True evaluates to numerical 1 and False evaluates to numerical 0 for use in the calculations.

Logical operator	denotation	Example
< (less than sign)	Less than	5 < A2
> (greater than sign)	Greater than	A1 > A2
= (Equal sign)	Equal to	A1 = 4
<= (Less than or equal)	Less than or equal	A3 <= (A4 + 2)
>= (Greater than or equal)	Greater than or equal	B1 >= 30
<>(Not equal)	Not equal	C2 <>10

### Text concatenation operator (Binary literal operator)

Two or more text strings can be concatenated into one text using this operator.

Logical operator	denotation	Example
& (ampersand)	Concatenation	"Hello" & "World"

### Operator precedence

All operations are subjected to the following hierarchy of operations. The level 1 operations are done first, followed by level 2, and so on. Within the same level, the operations are performed from left to right in the order where they are encountered during the parsing of the formula.

1. (Unary Minus)
2. /
3.
  - -
4. < > = <= >= <>
5. & (Concatenation)

When the default operators' precedence need to be changed, then the parentheses will be used to explicitly indicate the operation order.

#### Examples for Operator precedence

Formulas	Computed Value
= 6 / 2 + 1	4
= 6 / (2 + 1)	2
= 2 + 4 / 2	4
= (2 + 4) / 2	3

### Equal Sign, the Formula Character

To indicate that a particular string should be treated as a formula, user must start the string with a special character, `FormulaCharacter("=")`. This property is static, so you can change the formula character within your code.

It's default value is the equal sign, (=).

In general, in order for Essential Calculate to recognize a string as containing a formula; the string is required to start with the [FormulaCharacter](#).

There is one exception though, if you explicitly call a [CalcEngine](#) Parse method like `ParseFormula` or `ParseAndComputeFormula`, including the formula character as the first character in the passed string, is optional.

### Square Brackets, indexers in CalcQuickBase class

If you are using a [CalcQuickBase](#) object to add calculation support to your business object, then you must use strings as indexers on the `CalcQuickBase` instance to get and set values.

To register the strings in `CalcQuickBase`, it must be enclosed within square brackets "[ ]". Eg. [A]. These registered variable names are indexer keys.

### C#

```
CalcQuickBase calcQuick = new CalcQuickBase();
calcQuick["A"] = "5";
calcQuick["B"] = "6";
```

```
calcQuick["C"] = "11";
```

## Named Ranges Support in Essential Calculate

Defining a name for cell, range of cells, formulas, constants or tables is called Named Ranges. By using names, users can easily identify the purpose of cell references

also make the formulas much easier to understand and maintain.

For example, the name can be defined for the cell range of "A1:D1" as "SUMRANGE".

### Syntax to define a name

- The name can be started with a letter or underscore(\_). Also, it must not be a single letter.
- The name must not be equal to cell references. For example: A1
- No space must be included and the name cannot be empty string.
- A name can contain up to 255 characters.
- The names will be case-insensitive. Thus it does not distinguish between uppercase and lowercase characters.

### Add Named Ranges

A name can be added for a cell or range of cells using [AddNamedRange](#) method of [CalcEngine](#).

#### C#

```
//Class derived from ICalcData,  
CalcData calcData = new CalcData();  
CalcEngine engine = new CalcEngine(calcData);  
//Adding the name to the named range collection,  
engine.AddNamedRange("GROUPCELLS", "A1:C4");  
//Using the name for computing formulas,  
string formula = "SUM(GROUPCELLS)";  
string result = engine.ParseAndComputeFormula(formula);
```

### Remove Named Ranges

A name can be removed from a cell or range of cells using [RemoveNamedRange](#) method of [CalcEngine](#).

#### C#

```
//Class derived from ICalcData,  
CalcData calcData = new CalcData();  
CalcEngine engine = new CalcEngine(calcData);  
//Removing the range from NamedRange's collection,  
engine.RemoveNamedRange("GROUPCELLS");
```

### Manage Named Ranges

The names are maintained in a collection called [NamedRanges](#). Also, the ranges of particular name can be changed or replaced using this collection.

#### C#

```
//Class derived from ICalcData,  
CalcData calcData = new CalcData();  
CalcEngine engine = new CalcEngine(calcData);
```

```
//Total name ranges count,
int count = engine.NamedRanges.Count;
//Changing the range of particular named range,
engine.NamedRanges["GROUPCELLS"] = "A3:A8";
```

## Custom Function

Essential Calculate holds many functions from statistics, finance and mathematics, along with other general purpose functions. There are

more than 400+ entries in the library. Also, it is easy to add the own calculations with custom functions.

### LibraryFunction

A property which holds the collection of library functions in Essential Calculate is [LibraryFunctions](#). The function name in the library should only contain letters, digits or underscore. The function name serves as the hash key and the function delegate serves as the hash value.

### Add Custom Function

Adding a custom function to the Formula Library in Essential Calculate is a two step process. The first step is to write a method that actually does the calculation work for your

custom function. The second step is to register this method with the [CalcEngine](#). So, when the [CalcEngine](#) object is a member of an application, the additional

function methods can be added to the application and then these methods should be registered with the [CalcEngine](#) object after the object is created.

### Write a method

The method must have the signature specified by the delegate, [LibraryFunction](#). The method must accept a string argument and returns a string value.

For writing a custom formula method, any convention with respect to passing arguments can be used within the implementation code. Thus, arguments can be a single entry

like A1 or 153 or it can be more complex like A1:C15. The computed value will be returned as a string. The arguments can be enhanced with standard items like cell references,

numbers, other formulas, etc., using the parsing tools of [CalcEngine](#) to minimize the amount of code that is required to be written.

Below code example shows implementing the custom function of computing minimum value,

### C#

```
public string CustomMin(string args)
{
    double min = double.MaxValue;
    double d;
    var splitArgs = args.Split(new char[] { CalcEngine.ParseArgumentSeparator
});
    foreach (string s in splitArgs)
    {
        if (double.TryParse(s, NumberStyles.Number | NumberStyles.AllowExponent,
            null, out d))
        min = Math.Min(min, d);
    }
}
```

### Register the method with CalcEngine

The second step for adding the own formula is to register the custom method with the `CalcEngine` object. This is done with the help of `AddFunction` method of `CalcEngine` which accepts the string that is used when you refer the function, and the second argument is a delegate for method name. The only requirement here is that the function name should

start with an alpha character and should only contain alpha-numeric characters. Additionally, the string cannot be the name of any existing library function.

#### C#

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
//Adding custom function to the library with user defined formula name,
engine.AddFunction("CheckMin", new LibraryFunction(CustomMin));
```

### Compute the Custom Function

To compute the custom formula, you need to pass the registered custom formula in `PareAndComputeFormula` method of `CalcEngine`.

#### C#

```
//Set value 100 to variable "A1"
calcData.SetValueRowCol(100, 1, 1);
//Set value 200 to variable "B1"
calcData.SetValueRowCol(200, 1, 2);
//Set value 300 to variable "C1"
calcData.SetValueRowCol(300, 1, 2);
//compute the registered method using CalcEngine.
var r = engine.PareAndComputeFormula("=CustomMin(A1,B1,C1)");
```

### Remove Custom Function

To remove a single function from the Function Library, use `RemoveFunction` method of `CalcEngine`, passing a formula name as the string that references this function and to remove all functions, you can clear the hash table that holds them by using the `Clear` method of `LibraryFunction`.

#### C#

```
//To remove a single function from library,
engine.RemoveFunction("CheckMin");
//To remove all functions from library,
engine.LibraryFunctions.Clear();
```

### Replace Function

To replace a function with another implementation, remove the original name and add the same name again with a different method name.

#### C#

```
// Remove formula name "CheckMin" from the library,
engine.RemoveFunction("CheckMin");
```



```
//Adds formula name "Minimum" to the library,  
engine.AddFunction("Minimum", new LibraryFunction(CustomMin));
```

**Tips:** Removing unused functions from the Function Library, **reduces the memory usage and speeds up parsing** as well. Also, if you are only

using a selected few Library functions, you may want to remove the unused ones. This can be done using the `Clear` method of `LibraryFunction`.

and after clearing all functions, you can add few functions that will be used often by using `AddFunction` method of `CalcEngine`.

## Performance and Limitations

This section explains about the performance and limitations of Essential Calculate.

To improve the performance

By default, `CalcEngine` calculates the formulas quickly. But to improve the performance in Essential Calculate, user need to set the following

properties of `CalcEngine`.

### AllowShortCircuitIFs

If the application contains more nested if formulas for calculation, then setting `AllowShortCircuitIFs` property to `true` computes the formulas quickly.

### CalculatingSuspended

If there are more number of dependent cells in the formulas for calculation, then user can set `CalculatingSuspended` property to `true`,

to suspend calculations while a series of changes are made to dependent cells either by the user or programmatically.

### UseFormulaValues

To avoid the repeated calculations for a formula cell which contains more dependency cells, set `UseFormulaValues` property to `true` to

return the formula value stored in `FormulaInfoTable`.

### Parse Formula and Compute Formula

`ParseFormula` method in `CalcEngine` parses the formula in the cell and user can

reset the value of variables or dependent cell values of formula cell at runtime and get the updated result by computing the parsed formula using

`ComputeFormula` method which avoids the repeated parsing of formula.

**For Example:**

### C#

```
//Class derived from ICalcData,  
CalcData calcData = new CalcData();  
CalcEngine engine = new CalcEngine(calcData);
```

```
//Set values to the variables
calcData.SetValueRowCol(100, 1, 1);
calcData.SetValueRowCol(200, 1, 2);
calcData.SetValueRowCol(140, 2, 2);
calcData.SetValueRowCol(120, 3, 2);
calcData.SetValueRowCol(100, 4, 2);
//Parsing the formula,
var parsedFormula = engine.ParseFormula("=SUM(A1:E4)");
//Computing the value of parsed formula,
string result = engine.ComputeFormula(parsedFormula);
//Computes the nested IF formulas quickly,
engine.AllowShortCircuitIFs = true;
//To avoid the repeated calculations,
engine.UseFormulaValues = true;
// Turn off calculations,
engine.CalculatingSuspended = true;
// Makes multiple updates to cells involved in calculation
for (int i = 0; i < 5000; i++)
{
    for (int j = 0; j < 5000; j++)
    {
        calcData.SetValueRowCol(random.Next(5) + 1, i, j);
    }
}
// Turn on calculations,
engine.CalculatingSuspended = false;
//Again computing the value of parsed formula,
result = engine.ComputeFormula(parsedFormula);
```

### To avoid stack overflow exception

To avoid the stack overflow exception while computing the formulas iteratively exceeding the maximum capacity, user need to set the following properties of `CalcEngine`,

#### ThrowCircularException

Setting this property to `true` will throw an exception when the circular exception is encountered and also

avoids the recursive looping.

#### IterationMaxCount

`CalcEngine` provides an option to set iterative calculations and maximum iteration count to calculate the formula having circular references.

The default value is 0 indicating that iterative calculation support is turned off.

`ThrowCircularException` property will be automatically set

to `true` when you set a non-zero value to `IterationMaxCount`.

#### MaximumRecursiveCalls

`CalcEngine` have `MaximumRecursiveCalls` property for which value is 100 by default. So, if the recursive call during calculations exceeds more than 100,

stack overflow exception occurs which can be avoided by setting this property to required limit at the sample level.

### MaxStackDepth

CalcEngine have MaxStackDepth property for which value is 50 by default. So, if the stack size is exceeding more than 50, stack overflow exception

occurs which can be avoided by setting this property to required limit at the sample level.

#### C#

```
//Class derived from ICalcData,
CalcData calcData = new CalcData();
CalcEngine engine = new CalcEngine(calcData);
CalcEngine.MaxStackDepth = 10000;
engine.MaximumRecursiveCalls = 10000;
engine.IterationMaxCount = 10000;
```

If the *stack overflow exception* occurs even after setting these properties, user need to calculate the value in the thread by

specifying the maximum stack size (Default stack size is 1 Mb. Throws exception if exceeds).

#### C#

```
private string calValue1;
CalcEngine engine;
private void Main()
{
    //Class derived from ICalcData,
    CalcData calcData = new CalcData();
    engine = new CalcEngine(calcData);
    calValue1 = String.Empty;
    var maxSize = 10000000;
    var thread = new Thread(GetCalculatedValue, maxSize);
    thread.Start();
    thread.Join();
    MessageBox.Show("val:" + calValue1);
}
private void GetCalculatedValue()
{
    engine.UseFormulaValues = true;
    engine.MaximumRecursiveCalls = 10000;
    CalcEngine.MaxStackDepth = 10000;
    //Computing the value of formula cell "D1",
    calValue1 = engine.ParseAndComputeFormula("D1");
}
```

### Limitations

- Essential Calculate is not a UI component, but it can be added to the user's own business objects.
- Essential Calculate does not support to localize the registered formulas in the Calculate API to any specific language other

than English(en-US), which is by default. But we have support to compute the formulas in different region settings.

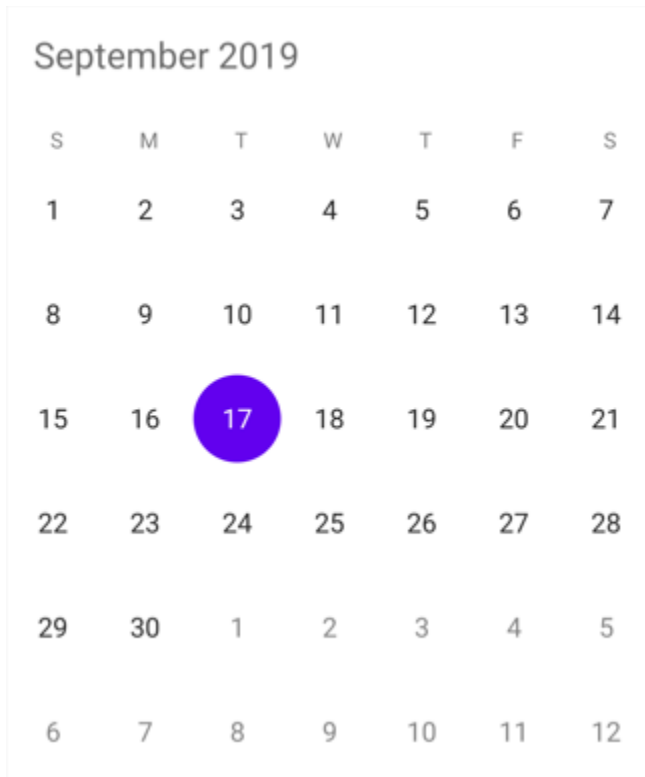
- The parameters passed in any of the function library of Essential Calculate is of type string.

## SfCalendar

### Overview

The Essential Xamarin Calendar widget provides the multi-view representation to display and select one or more dates within specified ranges. Also provides a gesture friendly UI to perform operations like navigations, events, etc.

Essential Calendar can be used in various scenarios like ticket booking, events notifying, display working days etc.



### Key Features

- **Built-in Views** – A multi-view representation to display dates in month, year, decade, century view based on the view mode.
- **Selection** – Enables users to select one or multiple dates.
- **Min Max dates** – Visible dates can be limited using the specified Minimum and Maximum dates.
- **Blackout dates** – A collection of dates with cross mark representation that cannot be interacted.

### Getting Started

This section explains how to implement simple holiday indicator application which allows user to select working days using [SfCalendar](#) control.

### Adding SfCalendar reference

You can add SfCalendar reference using one of the following methods:

#### Method 1: Adding SfCalendar reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfCalendar). To add SfCalendar to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfCalendar](https://www.nuget.org/packages/Syncfusion.Xamarin.SfCalendar), and then install it.

![Adding SfCalendar reference from NuGet](images/Adding SfCalendar reference.png)

---

**Note:** Install the same version of SfCalendar NuGet in all the projects.

---

#### Method 2: Adding SfCalendar reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfCalendar control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfCalendar assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfCalendar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfCalendar.Android.dll Syncfusion.SfCalendar.XForms.Android.dll Syncfusion.SfCalendar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfCalendar.iOS.dll Syncfusion.SfCalendar.XForms.iOS.dll Syncfusion.SfCalendar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfInput.UWP.dll Syncfusion.SfShared.UWP.dll Syncfusion.SfCalendar.XForms.UWP.dll Syncfusion.SfCalendar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

---

---

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Launching the SfCalendar on each platform

To use SfCalendar inside an application, each platform application must initialize the SfCalendar renderer. This initialization step varies from platform to platform and is discussed in the following sections.

#### Android and UWP

The Android and UWP launches the SfCalendar without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

#### iOS

To launch SfCalendar in iOS, need to create an instance of SfCalendarRenderer in FinishedLaunching overridden method of AppDelegate class in iOS Project as shown below.

#### C#

```
using Syncfusion.SfCalendar.XForms.iOS;
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    SfCalendarRenderer.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### ReleaseMode issue in UWP platform

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in Release Mode.

The above problem can be resolved by initializing the SfCalendar assemblies in App.xaml.cs in UWP project as like in below code snippet.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfCalendarRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
}
```

### Create a Simple SfCalendar

The **SfCalendar** control is configured entirely in C# code or by using XAML markup. The following steps explain on how to create a **SfCalendar** and configure its elements,

- Adding namespace for the added assemblies.

#### XML

```
xmlns:syncfusion="clr-namespace:Syncfusion.SfCalendar.XForms;assembly=Syncfusion.SfCalendar.XForms"
```

#### C#

```
using Syncfusion.SfCalendar.XForms;
```

- Now add the SfCalendar control with a required optimal name by using the included namespace.

#### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:syncfusion="clr-namespace:Syncfusion.SfCalendar.XForms;assembly=Syncfusion.SfCalendar.XForms"
x:Class="GettingStarted.CalendarPage">
<ContentPage.Content>
<syncfusion:SfCalendar x:Name="calendar" />
</ContentPage.Content>
</ContentPage>
```

#### C#

```
using Syncfusion.SfCalendar.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class CalendarPage : ContentPage
    {
        public CalendarPage()
        {
            InitializeComponent();
            SfCalendar calendar = new SfCalendar();
            this.Content = calendar;
        }
    }
}
```

### Set Blackout Dates

**SfCalendar** control provides option to black out the desired date which is in disabled state among the visible dates. Here, holidays are blacked out in the form which cannot be selected by the user. To black out the holiday, add them into [BlackoutDates](#) list.

#### C#

```
SfCalendar calendar = new SfCalendar();
List<DateTime> black_dates = new List<DateTime>();
for (int i = 0; i < 5; i++)
{
    DateTime date = new DateTime(2018, 4, 1+i);
    black_dates.Add(date);
}
calendar.BlackoutDates = black_dates;
```

### Enable Multiple Selection

**SfCalendar** control allows user to select one or more dates at a time among the non black out dates.

To enable it set **MultiSelection** option in **SelectionMode** enumeration property.

#### XML

```
<syncfusion:SfCalendar x:Name="calendar" SelectionMode="MultiSelection" />
```

#### C#

```
SfCalendar calendar = new SfCalendar();
calendar.SelectionMode=SelectionMode.MultiSelection;
```

### Restrict Dates

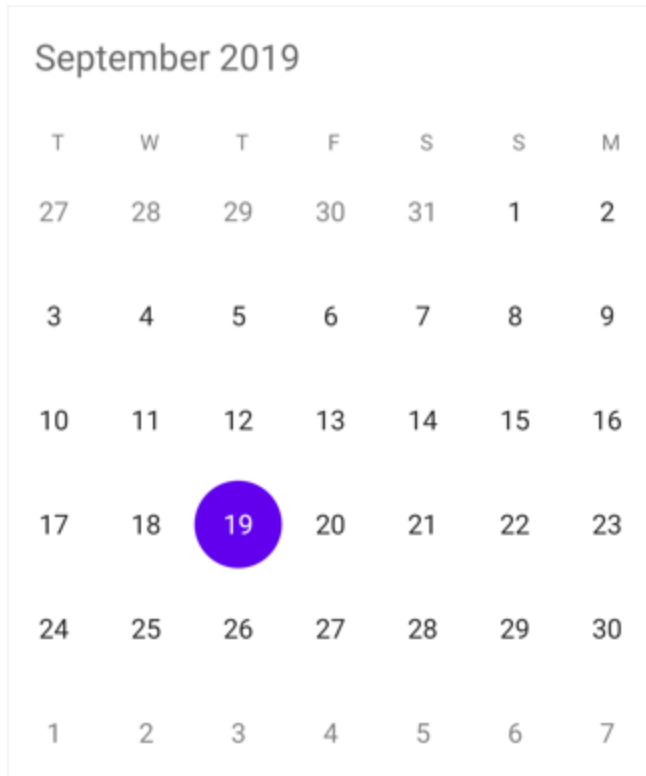
**SfCalendar** allows to select dates that falls between certain range of dates. Here, restrict user to select dates only in current year.

**Note:** To specify the range, set start date and end date to [MinDate](#) and [MaxDate](#) properties respectively.

#### C#

```
SfCalendar calendar = new SfCalendar();
calendar.MinDate = new DateTime(2019, 9, 1);
calendar.MaxDate = new DateTime(2019, 9, 30);
this.Content = calendar;
```





You can download the entire source code of this demo for Xamarin.Forms from here [CalendarGettingStarted](#)

### Calendar views

Xamarin.Forms calendar control provides 4 different types of views such month, year, decade and century. It allows users to easily select and navigate between all built-in views. This can be achieved by using [ViewMode](#) property of SfCalendar.

---

**Note:** By default calendar control is assigned with month view.

---

### Month view

This displays entire dates of a particular month, by default current month will be displayed on loading. The current date is provided with separate color different from the rest of the dates color in a month. The events availability will be denoted within the cell based on its duration.

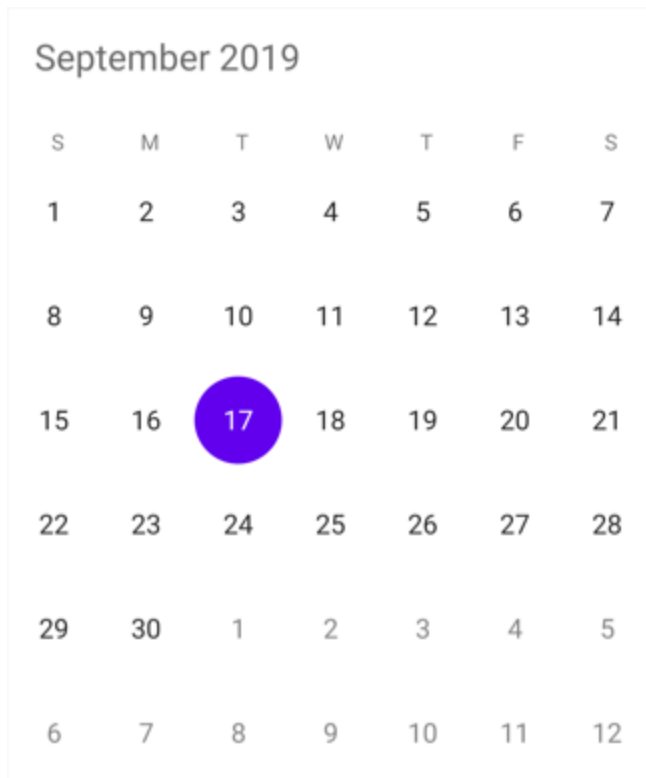
The dates in month view can be selected by four ways such as single, multiple, range and multi range selections which can be achieved using [SelectionMode](#). Refer [here](#).

### XML

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="MonthView"/>
```

### C#

```
calendar.ViewMode = ViewMode.MonthView;
```



### Trailing and leading days

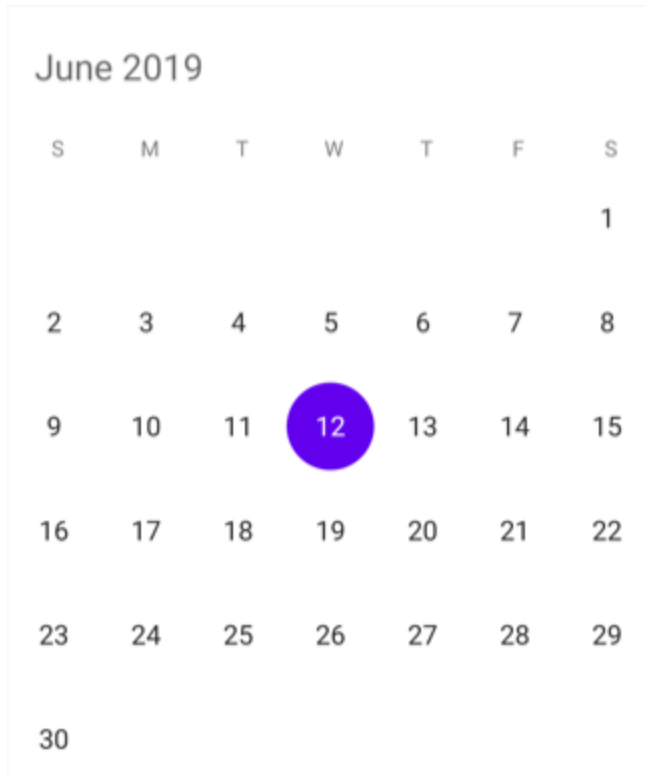
The **SfCalendar** allows you show/hide the days of the next month and previous month in calendar to enhance the appearance. This can be achieved by enabling the [ShowLeadingAndTrailingDays](#) property. The following code demonstrates how to hide the leading and trailing dates in calendar.

#### XML

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="MonthView"
ShowLeadingAndTrailingDays="False"/>
```

#### C#

```
calendar.ViewMode = ViewMode.MonthView;
calendar.ShowLeadingAndTrailingDays = False;
```

**Note:**

- The OnMonthCellLoaded event is triggered for the current month dates.
- The VisibleDates in the MonthChanged event will return the current month dates.

### Enable and disable past dates

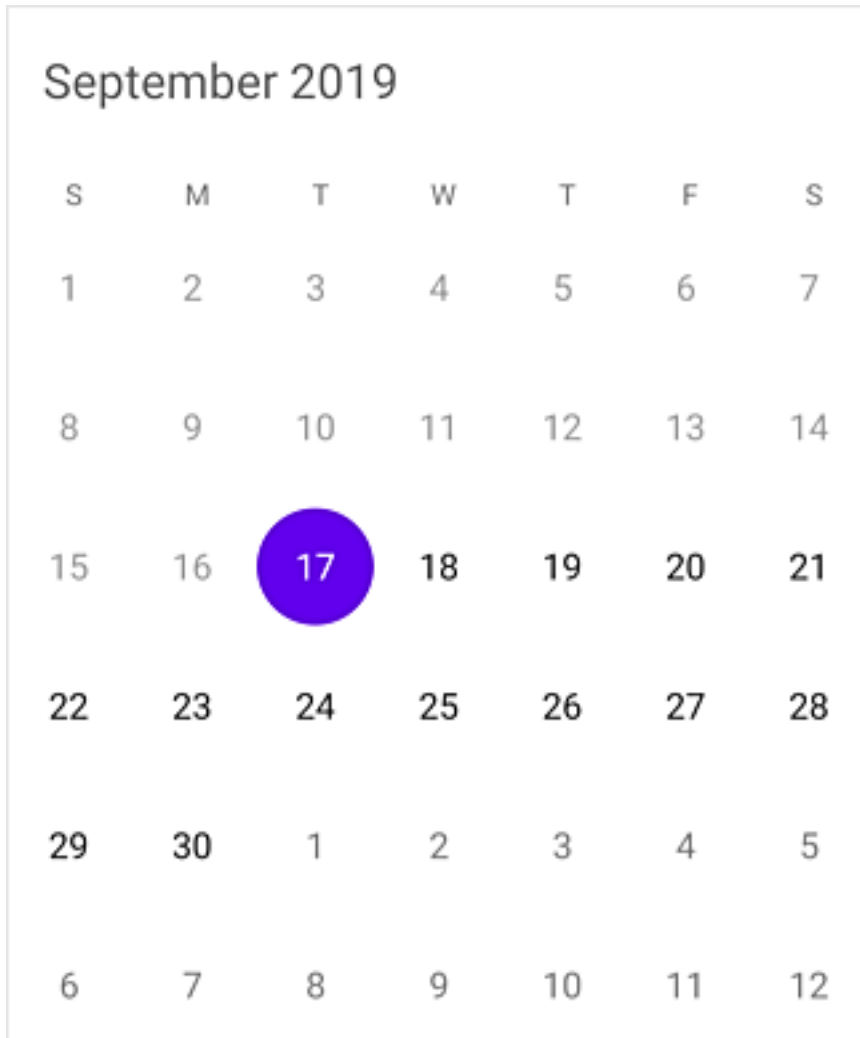
The **SfCalendar** allows you to enable/disable the past dates in MonthView. This can be achieved by changing the [EnableDatesInPast](#) property. By default, value of this property is set to true.

**XML**

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="MonthView"
    EnableDatesInPast="False"/>
```

**C#**

```
calendar.ViewMode = ViewMode.MonthView;
calendar.EnableDatesInPast = False;
```

**Note:**

The `EnableDatesInPast` is not applicable for UWP.

**Month view customization**

You can customize the calendar month view by using [MonthViewSettings](#) of `SfCalendar`.

- Current day text color can be modified using [TodayTextColor](#).
- The day header format, day font size, day header font size can be modified using [DayHeaderFormat](#), [DayFontSize](#), [DayHeaderFontSize](#)
- The background color of the inline view can be modified using [InlineBackgroundColor](#) property.
- In `AgendaView` the selected date color can be modified using [AgendaSelectedDateColor](#)

**XML**

```
<syncfusion:SfCalendar x:Name="calendar">
  <syncfusion:SfCalendar.MonthViewSettings>
    <syncfusion:MonthViewSettings
      CurrentMonthBackgroundColor="#800080"
      CurrentMonthTextColor="#ffffff"
    />
  />
</syncfusion:SfCalendar>
```

```
PreviousMonthBackgroundColor="#9895F0"  
PreviousMonthTextColor="#000000"  
DateSelectionColor="#ffffff"  
SelectedDayTextColor="#000000"  
DayHeaderFormat="EEEEEE"  
DayFontSize="12"  
DayHeaderFontSize="14"  
AgendaSelectedDateColor="Blue"  
TodaySelectionBackgroundColor="Green"/>  
</syncfusion:SfCalendar.MonthViewSettings>  
</syncfusion:SfCalendar>
```

## C#

```
SfCalendar calendar = new SfCalendar();  
MonthViewSettings monthViewSettings = new MonthViewSettings();  
monthViewSettings.CurrentMonthBackgroundColor = Color.FromHex("#800080");  
monthViewSettings.CurrentMonthTextColor = Color.FromHex("#ffffff");  
monthViewSettings.PreviousMonthBackgroundColor = Color.FromHex("#9895F0");  
monthViewSettings.PreviousMonthTextColor = Color.FromHex("#000000");  
monthViewSettings.DateSelectionColor = Color.FromHex("#ffffff");  
monthViewSettings.SelectedDayTextColor = Color.FromHex("#000000");  
monthViewSettings.DayHeaderFormat = "EEEEEE";  
monthViewSettings.DayFontSize = 12;  
monthViewSettings.DayHeaderFontSize = 20;  
monthViewSettings.SelectionRadius = 15;  
monthViewSettings.TodaySelectionTextColor= Color.Black;  
monthViewSettings.TodaySelectionBackgroundColor= Color.Green;  
monthViewSettings.AgendaSelectedDateColor = Color.Blue;  
calendar.MonthViewSettings = monthViewSettings;  
this.Content = calendar;
```



**Note:** To disable the current day selection, use `TodaySelectionBackgroundColor` color as Transparent.

#### *Month view border color customization*

You can customize the border color of calendar month view using [MonthViewSettings](#). The border color of month view can be customized using the [BorderColor](#) property and the lines of month cells can be enabled using the [CellGridOptions](#) property.

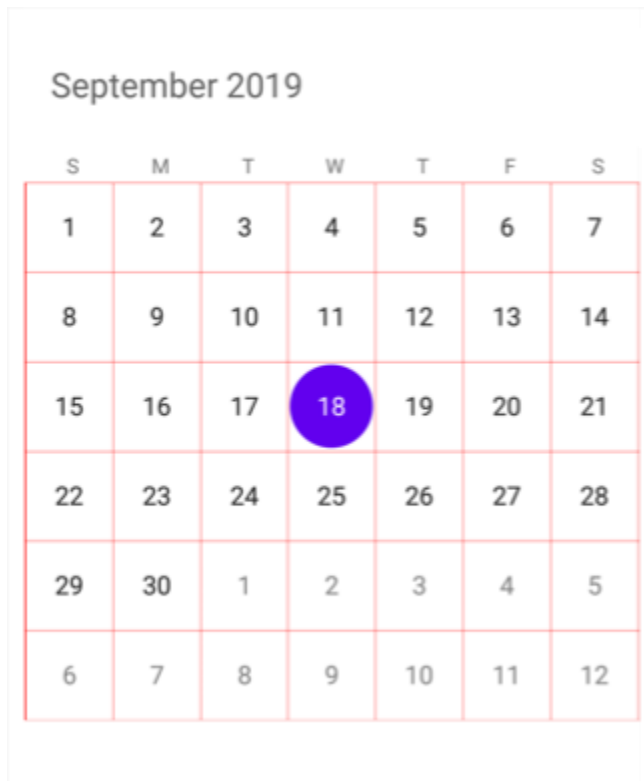
#### **XML**

```
<syncfusion:SfCalendar x:Name="calendar">
  <syncfusion:SfCalendar.MonthViewSettings>
    <syncfusion:MonthViewSettings BorderColor="#ff0000"
      CellGridOptions="Both"/>
  </syncfusion:SfCalendar.MonthViewSettings>
</syncfusion:SfCalendar>
```

#### **C#**

```
SfCalendar calendar = new SfCalendar();
MonthViewSettings monthViewSettings = new MonthViewSettings();
monthViewSettings.BorderColor = Color.FromHex("#ff0000");
monthViewSettings.CellGridOptions = CellGridOptions.Both;
```

```
calendar.MonthViewSettings = monthViewSettings;
this.Content = calendar;
```



#### Today border color customization

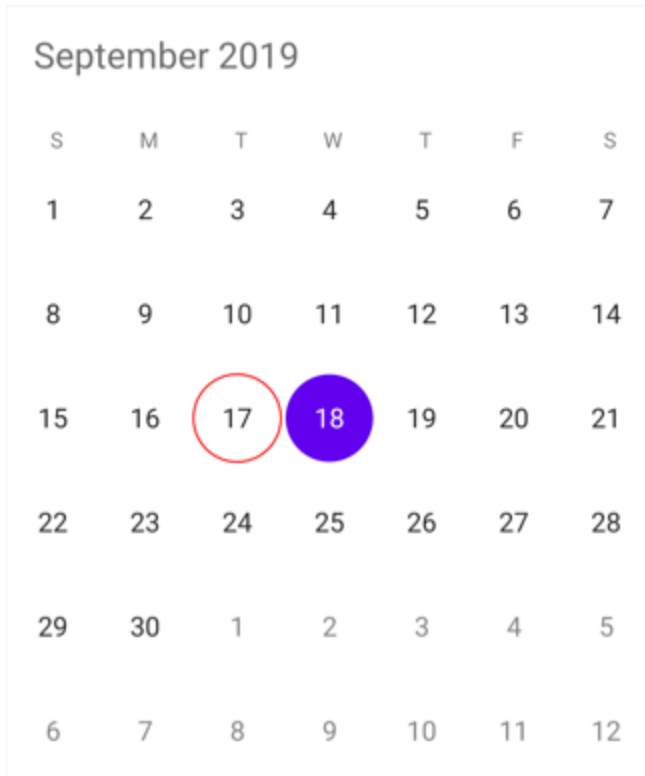
You can customize the today border color of calendar month cell using [MonthViewSettings](#). The border color of current day can be customized using the [TodayBorderColor](#) property and it is applicable for both [Fill](#) and [Circle SelectionShape](#).

#### XML

```
<syncfusion:SfCalendar x:Name="calendar">
  <syncfusion:SfCalendar.MonthViewSettings>
    <syncfusion:MonthViewSettings TodayBorderColor="#ff0000"/>
  </syncfusion:SfCalendar.MonthViewSettings>
</syncfusion:SfCalendar>
```

#### C#

```
SfCalendar calendar = new SfCalendar();
MonthViewSettings monthViewSettings = new MonthViewSettings();
monthViewSettings.TodayBorderColor = Color.FromHex("#ff0000");
calendar.MonthViewSettings = monthViewSettings;
this.Content = calendar;
```



### Week view

The number of weeks in the month view can be changed by setting the [NumberOfWeeksInView](#) property in SfCalendar. By default, `NumberOfWeeksInView` starts from current week, and this can be modified using the [MoveToDate](#) property of calendar. It also supports all existing features such as [FirstDayOfWeek](#), [MinDate](#), [MaxDate](#), and [SelectionMode](#).

### Note:

- Week number ranges from 1 to 6. If lesser or greater than these range is considered, `NumberOfWeeksInView` will be displayed as 6.
- Inline view considers `NumberOfWeeksInView` as only 6. For other count, only agenda view will be displayed in calendar.
- Dynamically changing `NumberOfWeeksInView` shows the first row of month view dates. It can be handled using the `MoveToDate` property of calendar
- [ShowLeadingAndTrailingDays](#) is not applicable if the `NumberOfWeeksInView` is lesser than 6.

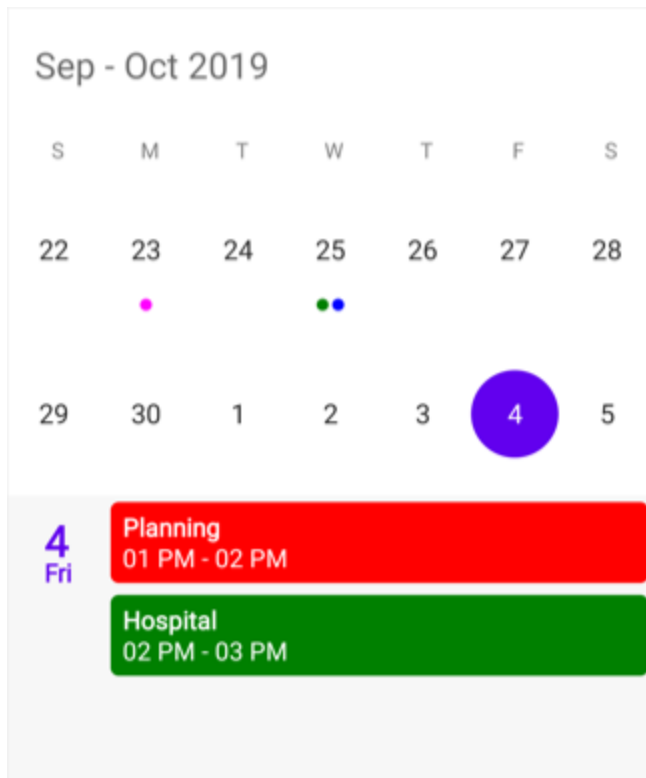
### XML

```
<syncfusion:SfCalendar x:Name="calendar" NumberOfWeeksInView="2"/>
```

### C#

```
SfCalendar calendar = new SfCalendar();
calendar.NumberOfWeeksInView = 2;
```





### Year view

This displays entire dates/month of a particular year, by default current year will be displayed on loading. The Years can be changed by swiping back and forth or [Forward](#) and [Backward](#) methods. The Months can be navigated quickly by selecting on the particular month in year view.

### XML

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="YearView"/>
```

### C#

```
SfCalendar calendar = new SfCalendar();  
calendar.ViewMode=ViewMode.YearView;  
this.Content = calendar;
```



### Year view mode

You can set the year view as either date or month using [YearViewMode](#). By default, current year and months will be displayed.

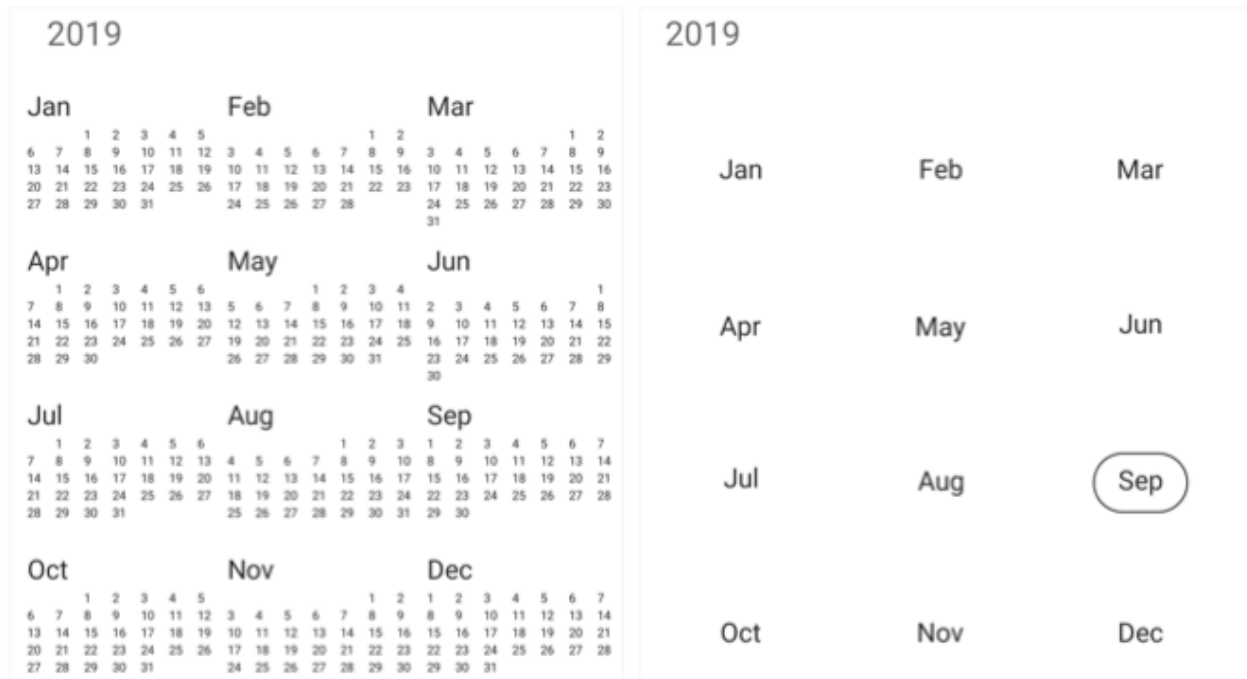
- If the `YearViewMode` is date, it will be displays all the months with dates in a particular year view.
- If the `YearViewMode` is month, it will be displays all the months in a particular year view.

### XML

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="YearView"
YearViewMode="Date"/>
```

### C#

```
SfCalendar calendar = new SfCalendar();
calendar.ViewMode=ViewMode.YearView;
calendar.YearViewMode = YearViewMode.Date;
this.Content = calendar;
```

**Note:**

- The `YearViewMode` property is only applicable for calendar in Android and iOS platforms.

**Year view customization**

We can customize the calendar view in yearView mode by using [YearViewSettings](#) property of SfCalendar. Date text color can be modified using [DateTextColor](#). You can also customize the header, month layout in year view's text color and background color by using the [HeaderBackground](#), [LayoutBackground](#), [YearHeaderTextColor](#) properties.

**XML**

```
<syncfusion:SfCalendar x:Name="calendar">
  <syncfusion:SfCalendar.YearViewSettings>
    <syncfusion:YearViewSettings
      LayoutBackground="#ffe4b5"
      DateTextColor="#E6E6FA"
      HeaderBackground="#8B4513"
      YearHeaderTextColor="#FFFFFF"/>
    </syncfusion:SfCalendar.YearViewSettings>
  </syncfusion:SfCalendar>
```

**C#**

```
SfCalendar calendar = new SfCalendar();
YearViewSettings yearViewSettings = new YearViewSettings();
yearViewSettings.LayoutBackground = Color.FromHex("#ffe4b5");
yearViewSettings.DateTextColor = Color.FromHex("#008000");
yearViewSettings.YearHeaderTextColor = Color.FromHex("#ff0000");
calendar.YearViewSettings = yearViewSettings;
this.Content = calendar;
```



### Decade view

This view displays the period of 10 years. By default, current year range of 10 years will be displayed on loading. You can easily navigate between month/year view to decade view by tapping the calendar header. The year can be navigated quickly by selecting a particular year from decade view.

### XML

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="Decade"/>
```

### C#

```
SfCalendar calendar = new SfCalendar();
calendar.ViewMode=ViewMode.Decade;
this.Content = calendar;
```



### Decade view customization

You can customize the decade view of calendar by using [YearViewSettings](#).

- Year text color can be modified using [MonthHeaderTextColor](#).
- You can customize the decade view header text and background color by using the [YearHeaderTextColor](#) and [HeaderBackground](#) property.
- You can customize the background of decade view by using [LayoutBackground](#) and [MonthLayoutBackground](#).

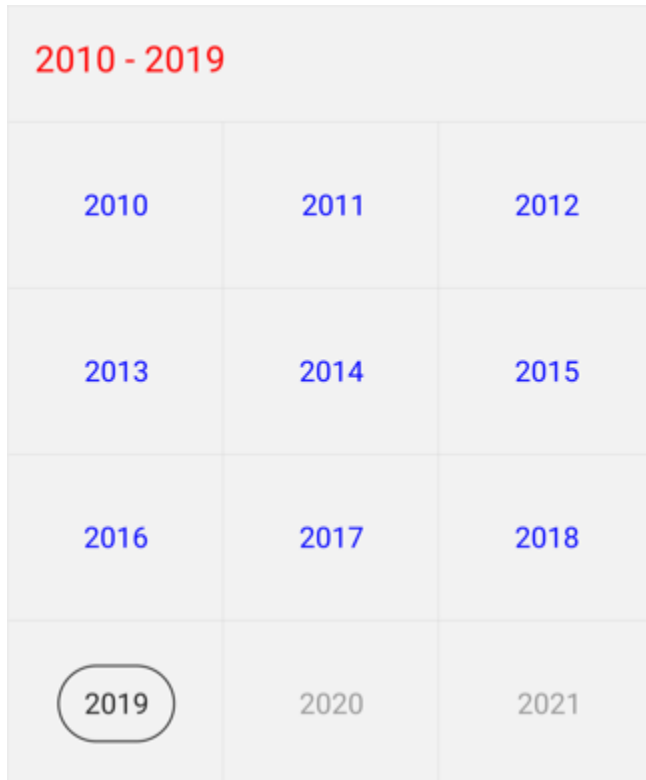
### XML

```
<syncfusion:SfCalendar x:Name="calendar">
  <syncfusion:SfCalendar.YearViewSettings>
    <syncfusion:YearViewSettings
      LayoutBackground = "#D3D3D3"
      HeaderBackground = "#F2F2F2"
      MonthHeaderTextColor = "#0000FF"
      YearHeaderTextColor = "#ff0000"
      MonthLayoutBackground = "#F2F2F2"/>
    </syncfusion:SfCalendar.YearViewSettings>
  </syncfusion:SfCalendar>
```

### C#

```
SfCalendar calendar = new SfCalendar();
YearViewSettings yearViewSettings = new YearViewSettings();
yearViewSettings.LayoutBackground = Color.FromHex("#D3D3D3");
yearViewSettings.MonthHeaderTextColor = Color.FromHex("#0000FF");
```

```
yearViewSettings.MonthLayoutBackground = Color.FromHex("#F2F2F2");
yearViewSettings.YearHeaderTextColor = Color.FromHex("#ff0000");
yearViewSettings.HeaderBackground = Color.FromHex("#F2F2F2");
calendar.YearViewSettings = yearViewSettings;
this.Content = calendar;
```



### Century view

This view displays the period of 100 years. By default, current year range of 100 years will be displayed on loading. You can easily navigate between month/year/decade view to century view by tapping the calendar header. You can easily navigate to decade view by selecting decade years in century view.

### XML

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="Century"/>
```

### C#

```
SfCalendar calendar = new SfCalendar();
calendar.ViewMode=ViewMode.Century;
this.Content = calendar;
```



### Century view customization

You can customize the century view of calendar using [YearViewSettings](#).

- Year text color can be modified using [MonthHeaderTextColor](#).
- You can customize the century view header text and background color by using the [YearHeaderTextColor](#) and [HeaderBackground](#) property.
- You can customize the background of century view by using [LayoutBackground](#) and [MonthLayoutBackground](#).

### XML

```
<syncfusion:SfCalendar x:Name="calendar">
  <syncfusion:SfCalendar.YearViewSettings>
    <syncfusion:YearViewSettings
      LayoutBackground="#D3D3D3"
      HeaderBackground="#F2F2F2"
      MonthHeaderTextColor="#0000FF"
      YearHeaderTextColor="#ff0000"
      MonthLayoutBackground="#F2F2F2"/>
    </syncfusion:SfCalendar.YearViewSettings>
  </syncfusion:SfCalendar>
```

### C#

```
SfCalendar calendar = new SfCalendar();
YearViewSettings yearViewSettings = new YearViewSettings();
yearViewSettings.LayoutBackground = Color.FromHex("#D3D3D3");
yearViewSettings.MonthHeaderTextColor = Color.FromHex("#0000FF");
```

```

yearViewSettings.MonthLayoutBackground = Color.FromHex("#F2F2F2");
yearViewSettings.YearHeaderTextColor = Color.FromHex("#ff0000");
yearViewSettings.HeaderBackground = Color.FromHex("#F2F2F2");
calendar.YearViewSettings = yearViewSettings;
this.Content = calendar;

```



### Date Navigation

**SfCalendar** control provides option to navigate through items either programmatically or by using gesture.

#### Programmatic Navigation

By using the following methods, we can navigate the months or year in **SfCalendar** with programmatically without applying gesture.

1. Forward
2. Backward

#### Forward

By default, the date can be navigated to next view using touch gesture and swiping the control in right to left direction. The view can also be changed programmatically using **Forward** method available in **SfCalendar**. It will move to next month, next year, next period of decade years, next period of century years based on the **ViewMode**.

#### C#

```
calendar.Forward();
```



---

**Note:** It can be navigated until it reaches the MaxDate.

---

#### *Backward*

By default, the date can be navigated to previous view using touch gesture and swiping the control in left to right direction. The view can also be changed programmatically using [Backward](#) method available in `SfCalendar`. It will move to previous month, previous year, previous period of decade years, previous period century years based on the `ViewMode`.

#### **C#**

```
calendar.Backward();
```

---

**Note:** It can be navigated until it reaches the MinDate.

---

#### *Move to Date*

Visible dates can be moved to specific date using [MoveToDate](#) property available in `SfCalendar`. it will move to any specific month, year, decade, century view based on the `ViewMode`.

**Note:** The specified date should lie between MinDate and MaxDate, if the specified date is greater than MaxDate then the view will be moved to MaxDate and if the specified date is lesser than the MinDate then the view will be moved to MinDate.

#### **C#**

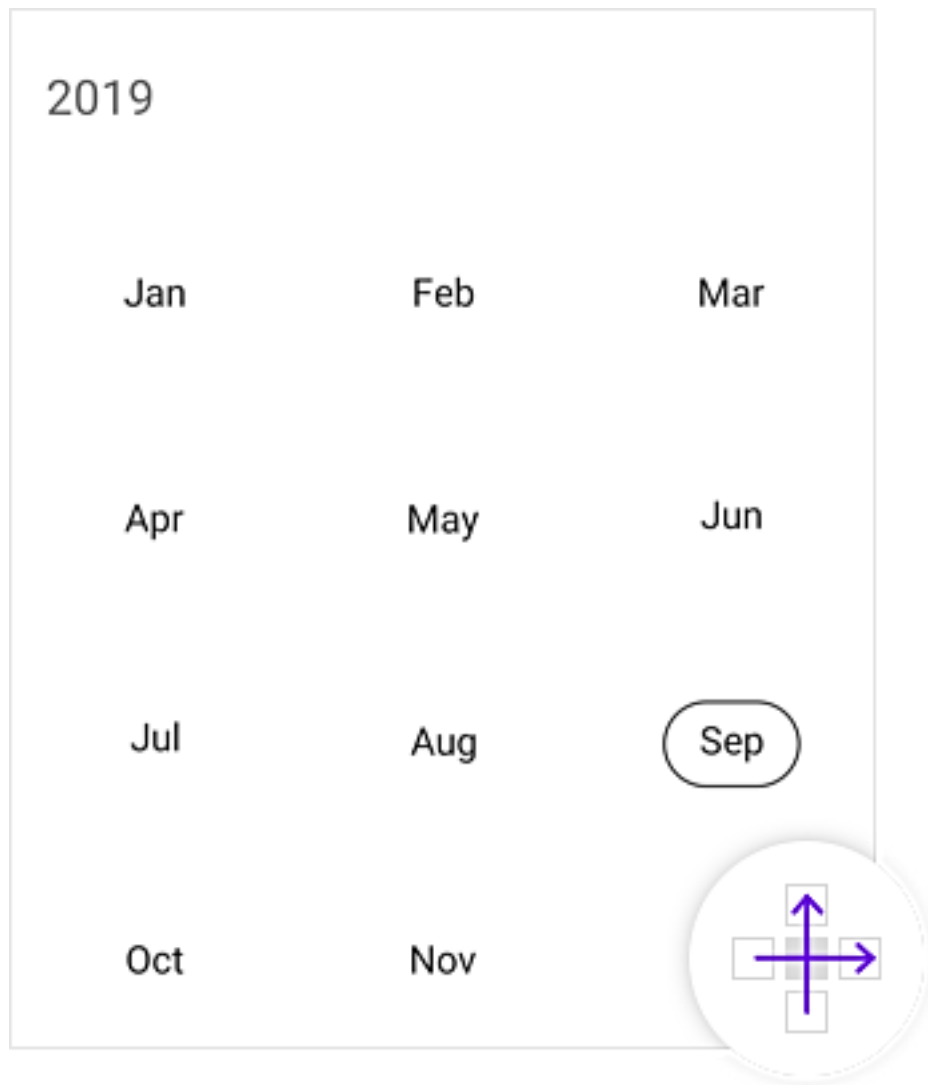
```
SfCalendar calendar = new SfCalendar();  
calendar.MoveToDate = new DateTime(2017, 5, 5);  
this.Content = calendar;
```

#### *Navigation Direction*

You can navigate the calendar `MonthView`, `YearView`, `DecadeView` and `CenturyView` either `Vertical` or `Horizontal` directions by setting the [NavigationDirection](#).

#### **C#**

```
SfCalendar calendar = new SfCalendar();  
Calendar.NavigationDirection = NavigationDirection.Vertical;
```



### Events (Appointments)

**SfCalendar** control provides support to add appointments on calendar's dates. By the way of adding collection of appointments, it will show the event with indicator on the desired dates.

Calendar's events can be added to **SfCalendar** using the following ways. [CalendarEventCollection](#) holds the details about the events to be rendered in calendar. Events contains the following attributes

1. [StartTime](#)
2. [EndTime](#)
3. [Subject](#)
4. [Color](#)

Finally add this collection of **CalendarInlineEvents** into [DataSource](#) of **SfCalendar**. The following code example will help to create an appointments on calendar's date. For events to be listed for a particular day, enable the inline feature in month view cell.

---

**Information:** Inline event support can be toggled on / off with **ShowInlineEvents** property.

---

### Month Appointment Display

You can handle the calendar month view appointment display by using [InlineViewMode](#) property of SfCalendar. By default, [InlineViewMode](#) is set as [Inline](#). Using the [InlineViewMode](#), you can set the month view appointments display as follows.

- **Inline** - Show the selected date's events in-line. In this mode, two rows of calendar will be hidden to show the events.
- **AgendaView** - Show the selected date's events below the month.

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:CalendarSample" x:Class="CalendarSample.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfCalendar.XForms;assembly=Syncfusion.SfCalendar.XForms
">
<syncfusion:SfCalendar.BindingContext>
<local:CalendarViewModel/>
</syncfusion:SfCalendar.BindingContext>
<syncfusion:SfCalendar x:Name="calendar" ViewMode="MonthView"
ShowInlineEvents="True" InlineViewMode="Inline"
MaximumEventIndicatorCount="1" DataSource="{Binding CalendarInlineEvents}">
</syncfusion:SfCalendar>
</ContentPage>
```

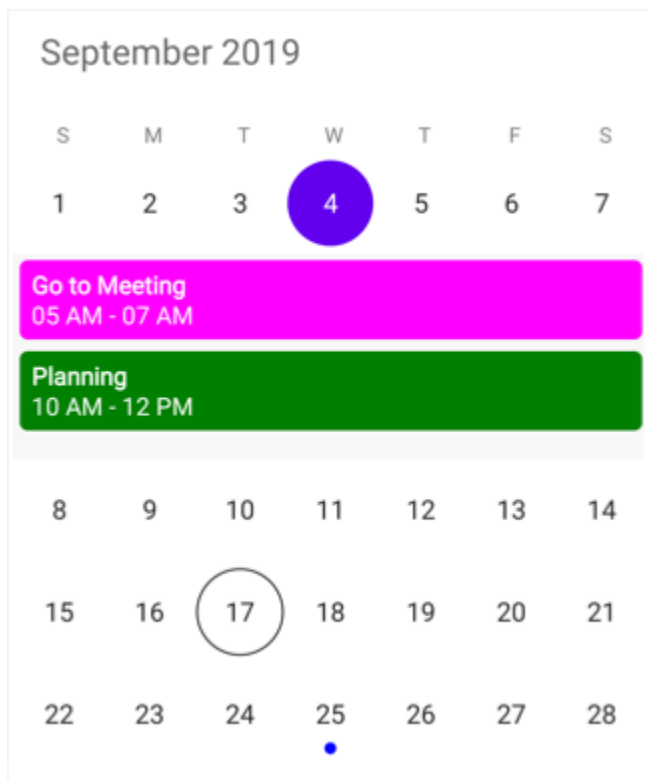
### C#

```
using System;
using Syncfusion.SfCalendar.XForms;
using Xamarin.Forms;
namespace CalendarSample
{
    public class CalendarViewModel
    {
        public CalendarEventCollection CalendarInlineEvents { get; set; } = new
        CalendarEventCollection();
        public CalendarViewModel()
        {
            CalendarInlineEvent event1 = new CalendarInlineEvent();
            event1.StartTime = new DateTime(2017, 5, 1, 5, 0, 0);
            event1.EndTime = new DateTime(2017, 5, 1, 7, 0, 0);
            event1.Subject = "Go to Meeting";
            event1.Color = Color.Fuchsia;
            CalendarInlineEvent event2 = new CalendarInlineEvent();
            event2.StartTime = new DateTime(2017, 5, 1, 10, 0, 0);
            event2.EndTime = new DateTime(2017, 5, 1, 12, 0, 0);
            event2.Subject = "Planning";
            event2.Color = Color.Green;
            CalendarInlineEvents.Add(event1);
            CalendarInlineEvents.Add(event2);
        }
    }
}
```

```
}

```

You can download the entire source code of this demo for Xamarin.Forms from here [CalendarEvents](#)



**Note:** If there is no appointment for the selected day, `Inline` view and `AgendaView` displays the text as “No Appointments”.

The `Inline` view and `AgendaView` will be available only in month view with single selection mode.

#### Month Appointment Indicator

You can customize the number of appointment indicators displayed in month cell using `MaximumEventIndicatorCount` property of ‘SfCalendar’. The default value of `MaximumEventIndicatorCount` is 5.

**Note:** If appointments count are lesser than the Appointment Indicator count value in the particular day, then according to number of appointments available, indicator will be displayed in the month cell.

#### Customize inline/agenda view using DataTemplate

The default appearance of the appointment can be customized by using the `InlineItemTemplate` property of the `MonthViewSettings`.

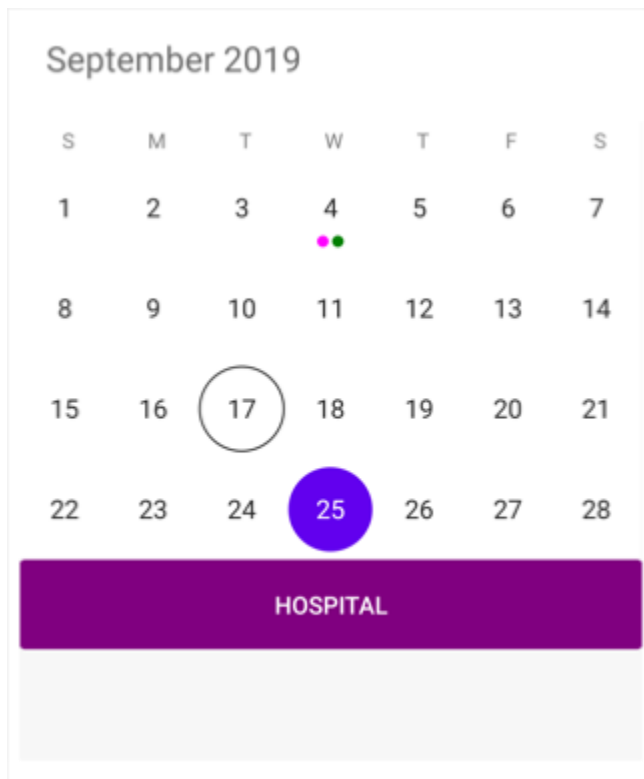
#### XML

```
<syncfusion:SfCalendar x:Name="calendar" ShowInlineEvents="True">
  <syncfusion:SfCalendar.MonthViewSettings>
    <syncfusion:MonthViewSettings>
      <syncfusion:MonthViewSettings.InlineItemTemplate>
        <DataTemplate>

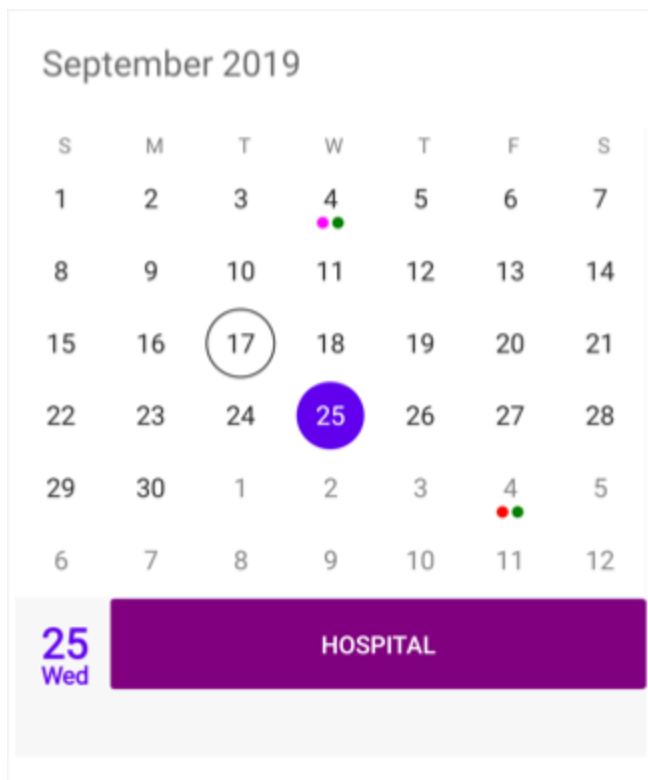
```

```
<Button BackgroundColor="Purple" Text="{Binding Subject}" TextColor="White"
/>
</DataTemplate>
</syncfusion:MonthViewSettings.InlineItemTemplate>
</syncfusion:MonthViewSettings>
</syncfusion:SfCalendar.MonthViewSettings>
</syncfusion:SfCalendar>
```

Inline view mode



Agenda view mode



### Customize inline/agenda view using Template Selector

Inline template selector can be used to choose a `DataTemplate` at runtime based on the value of a data-bound to inline appointment property through `InlineltemTemplate`. It lets you choose a different data template for each appointment, customizing the appearance of a particular inline appointment based on certain conditions. `DataTemplateSelector` for inline appointment as object and calendar as bindable object.

#### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<local:AppointmentSelector x:Key="TemplateSelector" />
</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfCalendar x:Name="calendar" ShowInlineEvents="True">
<syncfusion:SfCalendar.MonthViewSettings>
<syncfusion:MonthViewSettings InlineItemTemplate="{StaticResource
TemplateSelector}" />
</syncfusion:SfCalendar.MonthViewSettings>
</syncfusion:SfCalendar>
```

### Creating a DataTemplateSelector

#### C#

```
public class AppointmentSelector : DataTemplateSelector
{
public DataTemplate AppointmentTemplate { get; set; }
public DataTemplate AllDayAppointmentTemplate { get; set; }
public AppointmentSelector()
{
}
```

```

{
AppointmentTemplate = new DataTemplate(typeof(AppointmentTemplate));
AllDayAppointmentTemplate = new
DataTemplate(typeof(AllDayAppointmentTemplate));
}
protected override DataTemplate OnSelectTemplate(object item, BindableObject
container)
{
var calendar = (container as SfCalendar);
if (calendar == null)
{
return null;
}
if ((item as CalendarInlineEvent).IsAllDay)
{
return AllDayAppointmentTemplate;
}
else
{
return AppointmentTemplate;
}
}
}
}

```

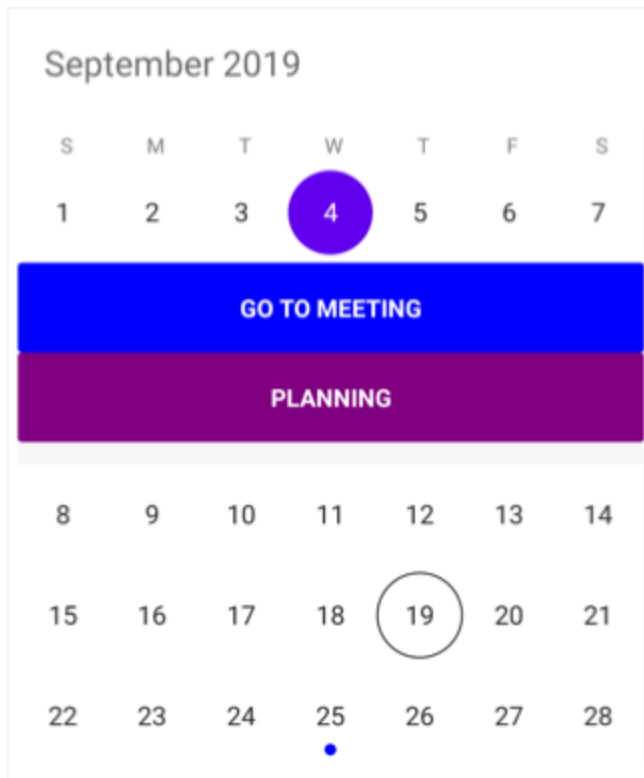
### XML

```

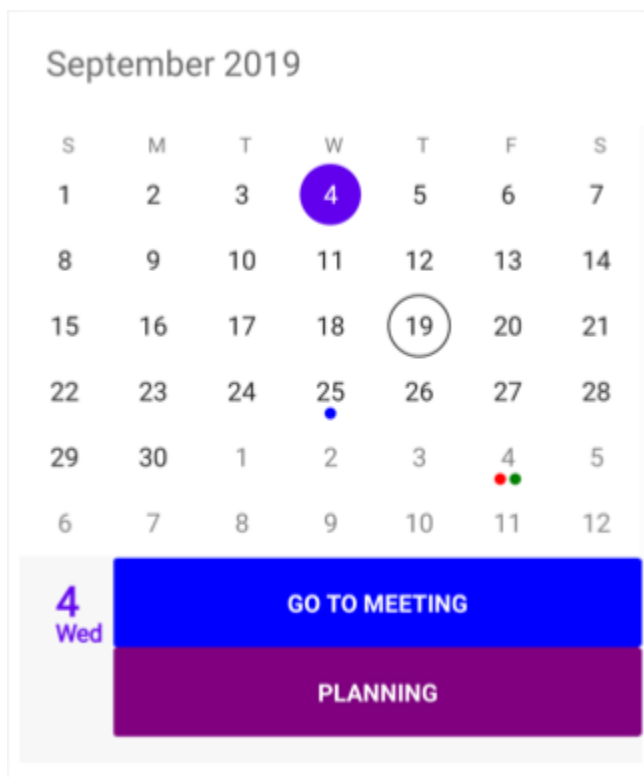
<!--<Button as Template for inline Appointment>-->
<Button xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="Calendar_Sample.AppointmentTemplate"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand"
BackgroundColor="{Binding Color}"
Text="{Binding Subject}"
FontAttributes="Bold"
TextColor="White" />
<!--<Button as Template for all day Appointment>-->
<Button xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="Calendar_Sample.AllDayAppointmentTemplate"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand"
BackgroundColor="{Binding Color}"
Text="{Binding Subject}"
FontAttributes="Bold"
TextColor="Black" />

```

Inline view mode



Agenda view mode



Getting inline/agenda view appointment details  
Using [InlineEvent](#) argument in the [InlineItemTappedEventArgs](#)

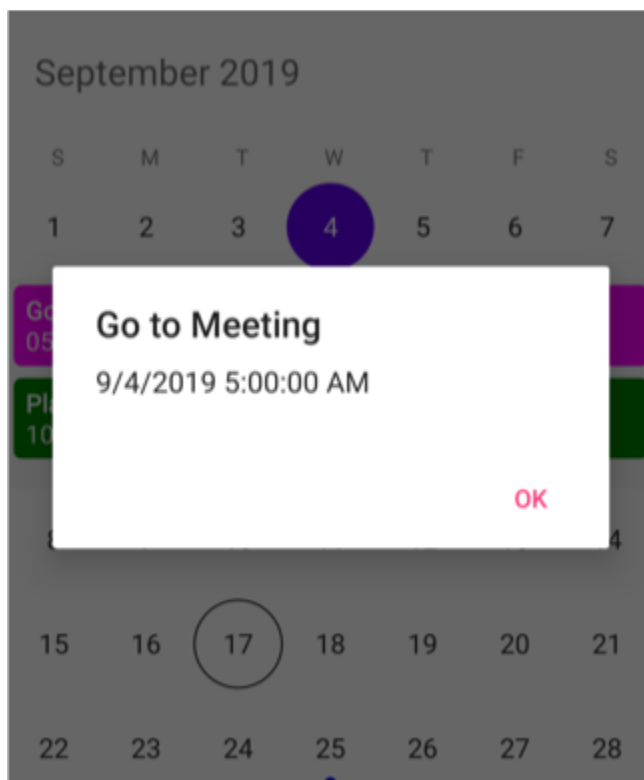


of [InlineItemTapped](#) event, you can get the month inline/agenda appointments details while tapping the specific appointment. You can do the required functions while tapping the inline/agenda appointment using this event.

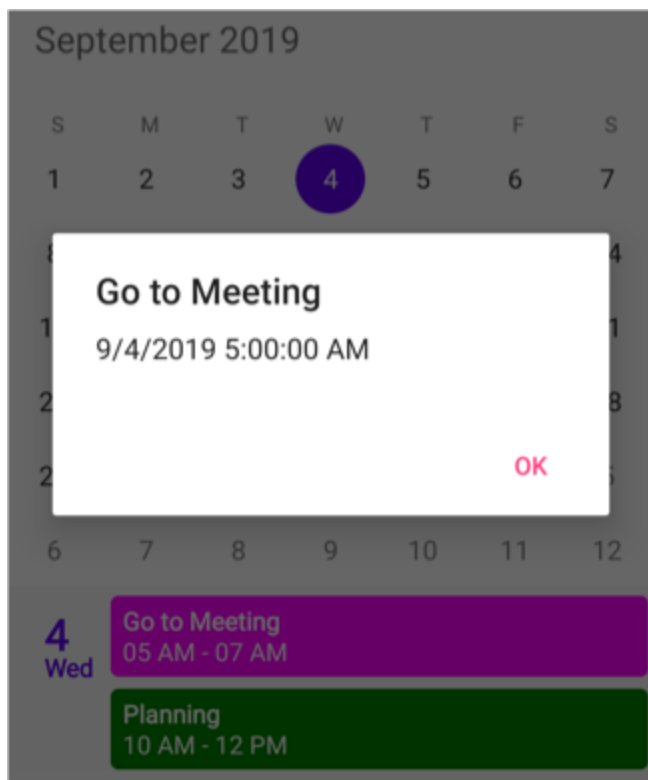
### C#

```
calendar.InlineItemTapped+= Calendar_InlineItemTapped;  
private void Calendar_InlineItemTapped(object sender,  
InlineItemTappedEventArgs e)  
{  
    var appointment = e.InlineEvent;  
    DisplayAlert(appointment.Subject, appointment.StartTime.ToString(), "ok");  
}
```

Inline view mode



Agenda view mode



### Select Multiple Dates

Dates can be selected by making a touch on month view cells. The default [SelectionMode](#) is Single which allows user to select one date at a time. **SfCalendar** provides support to select dates in two modes such as Single and Multiple selection.

- **SingleSelection** – A single date can be selected in a month view which can be equipped when user needs to select one date at a time / to view events.
- **MultiSelection** – More than one date can be selected in a random manner. Clicking again on selected dates can do deselection.
- **RangeSelection** – It allows us to select a single date range in **SfCalendar** month view.
- **MultiRangeSelection** – More than one date range can be selected in a month view.

The selected dates can be retrieved through **OnSelectionChanged** event which is raised on selecting.

### Multi selection mode

#### XML

```
<syncfusion:SfCalendar x:Name="calendar" SelectionMode="MultiSelection"/>
```

#### C#

```
SfCalendar calendar = new SfCalendar();
calendar.SelectionMode=SelectionMode.MultiSelection;
List<DateTime> selectedDates = new List<DateTime>();
selectedDates.Add(new DateTime(2019, 3, 06));
selectedDates.Add(new DateTime(2019, 3, 11));
selectedDates.Add(new DateTime(2019, 3, 15));
```

```
selectedDates.Add(new DateTime(2019, 3, 19));  
selectedDates.Add(new DateTime(2019, 3, 21));  
selectedDates.Add(new DateTime(2019, 3, 25));  
selectedDates.Add(new DateTime(2019, 3, 27));  
calendar.SelectedDates = selectedDates;  
this.Content = calendar;
```



**Note:** In range selection, navigation through swipe will be restricted and moving between months can be done by clicking on navigation button available at the top corner of SfCalendar control.

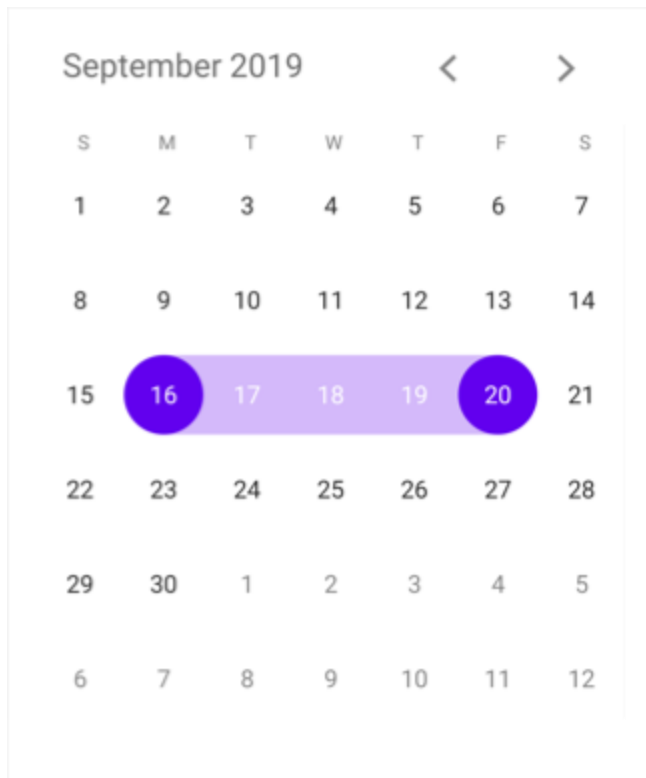
Range selection mode

#### XML

```
<syncfusion:SfCalendar x:Name="calendar" SelectionMode="RangeSelection"/>
```

#### C#

```
calendar.SelectionMode=SelectionMode.RangeSelection;
```



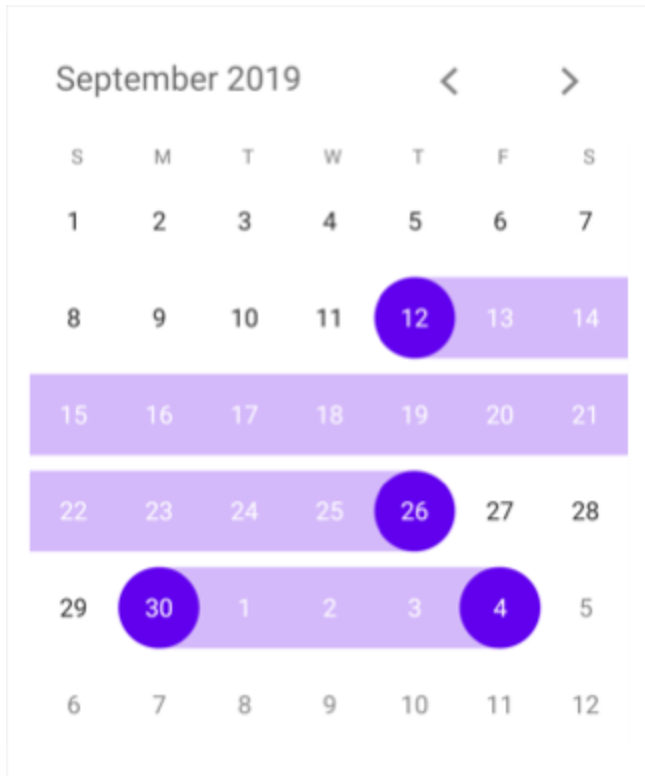
Multi range selection mode

#### **XML**

```
<syncfusion:SfCalendar x:Name="calendar"  
SelectionMode="MultiRangeSelection"/>
```

#### **C#**

```
calendar.SelectionMode=SelectionMode.MultiRangeSelection;
```



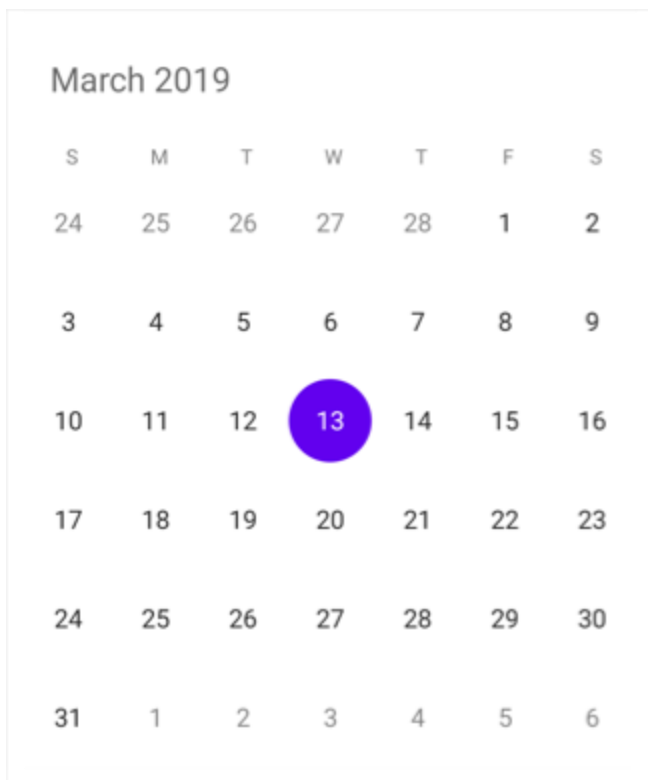
Single selection mode

#### XML

```
<syncfusion:SfCalendar x:Name="calendar" SelectionMode="SingleSelection"/>
```

#### C#

```
calendar.SelectionMode = SelectionMode.SingleSelection;
```



#### Programmatically clear the selected dates

You can clear the calendar selected dates pragmatically by using `ClearSelection` method, which is applicable for calendar `SelectionMode` such as `SingleSelection`, `MultiSelection`, `RangeSelection`, and `MultiRangeSelection`.

#### C#

```
calendar.ClearSelection();
```

#### Restrict Dates From Selection

Dates can be restricted or a collection of dates can be blacked out in `SfCalendar` Control.

##### Range of Min / Max Dates

Visible dates can be restricted between certain range of dates using `MinDate` and `MaxDate` properties available in `SfCalendar` control. It is applicable in all the calendar views.

The inline feature in month view will work only within the min max date range.

Beyond the min max date range, following restrictions will be applied.

- Date navigation features of move to date will be restricted.
- Cannot swipe the control using touch gesture.
- Selection does not work for month view.
- The tapped delegates will not be triggered while tapped on the `MonthCell`.

#### C#

```
SfCalendar calendar = new SfCalendar();
```

```
DateTime minDate=new DateTime(2015, 1, 1);
calendar.MinDate=minDate;
DateTime maxDate=new DateTime(2040, 12, 12);
calendar.MaxDate=maxDate;
this.Content = calendar;
```

### Blackout Dates

In **SfCalendar**, **BlackoutDates** refers the disabled dates that restrict the user from selecting it. These dates will be marked with slanted Stripes and Strikethrough, by using the **BlackoutDatesViewMode** property. By default, the value of this property is set to **Strikethrough**.

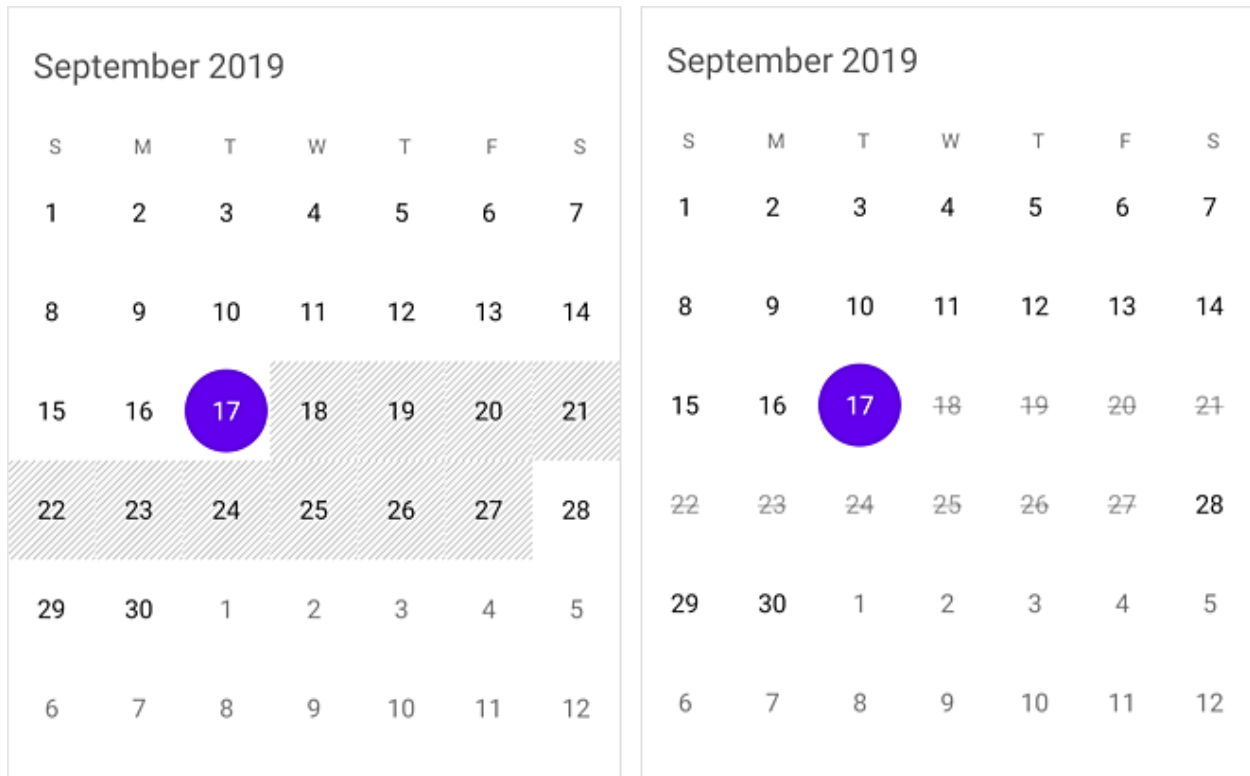
The Blackout dates can be achieved in two ways.

A date collection can be provided to set the **BlackoutDates**. This is useful when one wants to block dates where holidays or any other events occur.

By invoking the **AddDatesInPast** method, all past dates will be blacked out till current date.

### C#

```
SfCalendar calendar = new SfCalendar();
calendar.BlackoutDatesViewMode = BlackoutDatesViewMode.Stripes;
List<DateTime> black_Dates = new List<DateTime>();
black_Dates.Add(new DateTime(2019, 09, 18));
black_Dates.Add(new DateTime(2019, 09, 19));
black_Dates.Add(new DateTime(2019, 09, 20));
black_Dates.Add(new DateTime(2019, 09, 21));
black_Dates.Add(new DateTime(2019, 09, 22));
black_Dates.Add(new DateTime(2019, 09, 23));
black_Dates.Add(new DateTime(2019, 09, 24));
black_Dates.Add(new DateTime(2019, 09, 25));
black_Dates.Add(new DateTime(2019, 09, 26));
black_Dates.Add(new DateTime(2019, 09, 27));
calendar.BlackoutDates= black_Dates ;
this.Content = calendar;
```



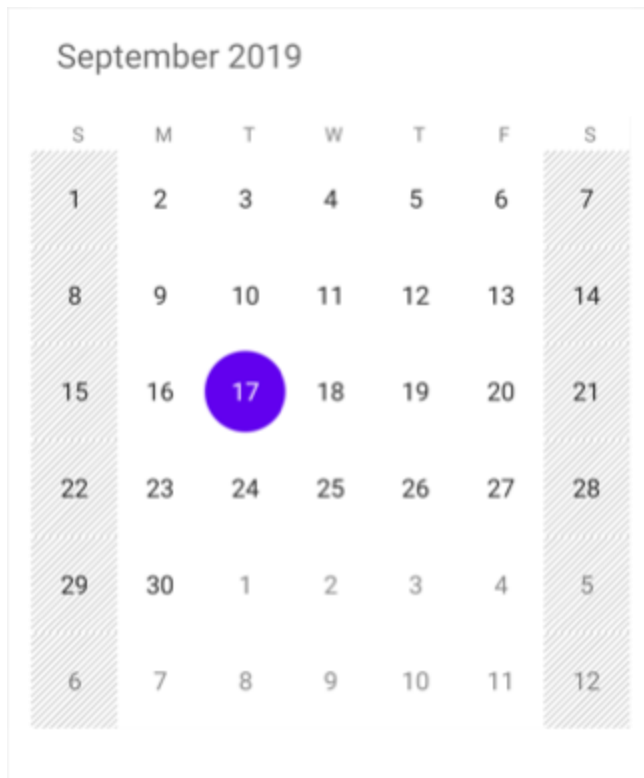
#### *Dynamic Blackout dates rendering*

Using [OnMonthCellLoaded](#) event you can render the black out dates.

#### **C#**

```
private void Calendar_OnMonthCellLoaded(object sender, EventArgs e)
{
    List<DateTime> blackoutDates = new List<DateTime>();
    var dayOfWeek = e.Date.DayOfWeek;
    if (dayOfWeek == DayOfWeek.Saturday || dayOfWeek == DayOfWeek.Sunday)
    {
        blackoutDates.Add(e.Date);
        calendar.BlackoutDates = blackoutDates;
    }
}
```





#### *Customize the blackout dates Color*

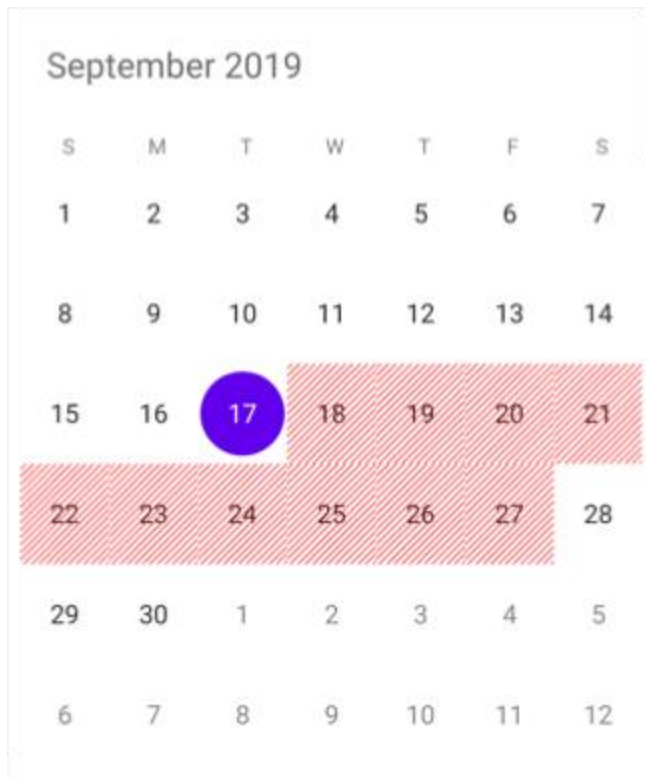
You can customize the color of [BlackoutDates](#) in month view mode using the [BlackOutColor](#) property of [MonthViewSettings](#).

#### **XML**

```
<syncfusion:SfCalendar x:Name="calendar">
  <syncfusion:SfCalendar.MonthViewSettings>
    <syncfusion:MonthViewSettings BlackOutColor="Red">
    </syncfusion:MonthViewSettings>
  </syncfusion:SfCalendar.MonthViewSettings>
</syncfusion:SfCalendar>
```

#### **C#**

```
SfCalendar calendar = new SfCalendar();
calendar.BlackoutDatesViewMode = BlackoutDatesViewMode.Stripes;
MonthViewSettings monthViewSettings = new MonthViewSettings();
monthViewSettings.BlackOutColor = Color.Red;
calendar.MonthViewSettings = monthViewSettings;
this.Content = calendar;
```



## Binding Properties in MVVM Pattern

### Binding SelectedDate

Calendar supports selecting a date programmatically by binding the [SelectedDate](#) property from your view model.

#### C#

```
calendar.SetBinding(SfCalendar.SelectedDateProperty, new  
Binding("SelectedDate", BindingMode.TwoWay));
```

#### XML

```
XAML:  
<calendar:SfCalendar x:Name="calendar"  
ShowInlineEvents="True"  
SelectionMode="SingleSelection"  
SelectedDate="{Binding SelectedDate, Mode=TwoWay}">  
</calendar:SfCalendar>
```

The following code sample demonstrates the ViewModel class.

#### C#

```
public class MainViewModel  
{  
    public DateTime SelectedDate { get; set; }  
    public MainViewModel()  
    {  
        SelectedDate = new DateTime(2019, 02, 05);  
    }  
}
```

```
}
}
```

---

**NOTE**


---

- You can bind the `SelectedDate` property only when the `SelectionMode` is set to `SingleSelection` in calendar.

### Binding SelectedDates

Calendar supports selecting dates programmatically by binding the `SelectedDates` property from your view model with the `List<DateTime>` type.

#### C#

```
calendar.SetBinding(SfCalendar.SelectedDatesProperty, new
Binding("SelectedDates", BindingMode.TwoWay));
```

#### XML

```
<calendar:SfCalendar x:Name="calendar"
ShowInlineEvents="True"
SelectionMode="MultiSelection"
SelectedDates="{Binding SelectedDates, Mode=TwoWay}">
</calendar:SfCalendar>
```

The following code sample demonstrates the ViewModel class.

#### C#

```
public class MainViewModel
{
    public List<DateTime> SelectedDates { get; set; }
    public MainViewModel()
    {
        SelectedDates = new List<DateTime>();
        SelectedDates.Add(new DateTime(2019, 02, 05));
        SelectedDates.Add(new DateTime(2019, 02, 08));
        SelectedDates.Add(new DateTime(2019, 02, 10));
        SelectedDates.Add(new DateTime(2019, 02, 14));
        SelectedDates.Add(new DateTime(2019, 02, 20));
    }
}
```

---

**NOTE**


---

- You can bind the `SelectedDates` property only when `SelectionMode` is set to `MultiSelection` in calendar.

### Binding SelectedRange

Calendar supports selecting a range of dates programmatically by binding the `SelectedRange` property with `SelectionRange` type from your view model.

**C#**

```
calendar.SetBinding(SfCalendar.SelectedRangeProperty, new  
Binding("SelectedRange", BindingMode.TwoWay));
```

**XML**

```
<calendar:SfCalendar x:Name="calendar"  
ShowInlineEvents="True"  
SelectionMode="RangeSelection"  
SelectedRange="{Binding SelectedRange, Mode=TwoWay}">  
</calendar:SfCalendar>
```

The following code sample demonstrates the ViewModel class.

**C#**

```
public class MainViewModel  
{  
    public SelectionRange SelectedRange { get; set; }  
    public MainViewModel()  
    {  
        SelectedRange = new SelectionRange();  
        SelectedRange.StartDate = new DateTime(2019, 02, 10);  
        SelectedRange.EndDate = new DateTime(2019, 02, 20);  
    }  
}
```

Calendar supports selecting multiple ranges of dates programmatically by binding the **SelectedRange** property with **ObservableCollection<SelectionRange>** type from your view model.

**C#**

```
calendar.SetBinding(SfCalendar.SelectedRangeProperty, new  
Binding("SelectedRanges", BindingMode.TwoWay));
```

**XML**

```
<calendar:SfCalendar x:Name="calendar"  
ShowInlineEvents="True"  
SelectionMode="MultiRangeSelection"  
SelectedRange="{Binding SelectedRanges, Mode=TwoWay}">  
</calendar:SfCalendar>
```

The following code sample demonstrates the ViewModel class.

**C#**

```
public class MainViewModel  
{  
    public ObservableCollection<SelectionRange> SelectedRanges { get; set; }  
    public MainViewModel()  
    {  
        SelectedRanges = new ObservableCollection<SelectionRange>();  
        SelectedRanges.Add(new SelectionRange());  
    }  
}
```

```
{
    StartDate = new DateTime(2019, 02, 10),
    EndDate = new DateTime(2019, 02, 20)
});
SelectedRanges.Add(new SelectionRange()
{
    StartDate = new DateTime(2019, 01, 31),
    EndDate = new DateTime(2019, 02, 05)
});
}
```

## NOTE

- You can bind the `SelectedRange` property only when `SelectionBinding` is set to `RangeSelection` and `MultiRangeSelection` in calendar.

## Commands

### Tap command

The [TapCommand](#) will be triggered whenever tapping the calendar cell and passing the [CalendarTappedEventArgs](#) as parameter.

## XML

```
<calendar:SfCalendar x:Name="calendar" TapCommand="{Binding
CalendarCellTapped}">
<calendar:SfCalendar.BindingContext>
<local:CalendarViewModel/>
</calendar:SfCalendar.BindingContext>
</calendar:SfCalendar>
```

## C#

```
public class CalendarViewModel
{
    private string commandText;
    public string CommandText
    {
        get { return commandText; }
        set { commandText = value; }
    }
    public ICommand CalendarCellTapped { get; set; }
    public CalendarViewModel()
    {
        CalendarCellTapped = new Command<CalendarTappedEventArgs>(CellTapped);
    }
    private void CellTapped(CalendarTappedEventArgs obj)
    {
        CommandText = obj.DateTime.ToString("dd/MM/yyyy") + " " +
        obj.SelectedAppointment.ToString();
    }
}
```

*Hold command*

The [HoldCommand](#) will be triggered whenever the calendar cell is long pressed and passing the [DayCellHoldingEventArgs](#) as parameter.

**XML**

```
<calendar:SfCalendar x:Name="calendar" HoldCommand="{Binding
OnDateCellHolding}">
<calendar:SfCalendar.BindingContext>
<local:CalendarViewModel/>
</calendar:SfCalendar.BindingContext>
</calendar:SfCalendar>
```

**C#**

```
public class CalendarViewModel
{
    private string commandText;
    public string CommandText
    {
        get { return commandText; }
        set { commandText = value; }
    }
    public ICommand OnDateCellHolding { get; set; }
    public CalendarViewModel()
    {
        OnDateCellHolding = new Command<DateTime>(DateCellHolding);
    }
    private void DateCellHolding(DateTime obj)
    {
        CommandText = obj.ToString("dd/MM/yyyy") ;
    }
}
```

*Month changed command*

The [MonthChangedCommand](#) will be triggered whenever the navigating between month and Forward()/backward() is called in calendar and passing the [MonthChangedEventArgs](#) as parameter.

**XML**

```
<calendar:SfCalendar x:Name="calendar" MonthChangedCommand="{Binding
OnMonthChanged}">
<calendar:SfCalendar.BindingContext>
<local:CalendarViewModel/>
</calendar:SfCalendar.BindingContext>
</calendar:SfCalendar>
```

**C#**

```
public class CalendarViewModel
{
    private string commandText;
    public string CommandText
    {
        get { return commandText; }
    }
}
```

```

set { commandText = value; }
}
public ICommand OnMonthChanged { get; set; }
public CalendarViewModel()
{
    OnMonthChanged = new Command<MonthChangedEventArgs>(MonthChanged);
}
private void MonthChanged(MonthChangedEventArgs obj)
{
    CommandText = " CurrentMonth -" + " " +obj.CurrentValue.ToString("dd/MM/yyyy")
    + " "+" PreviousMonth - "+" " + obj.PreviousValue.ToString("dd/MM/yyyy")
    + " "+" VisibleDate - "+" " "+"
    obj.VisibleDates[0].Date.ToString("dd/MM/yyyy");
}
}

```

### *Selection changed command*

The [SelectionChangedCommand](#) will be triggered whenever the selection is changed in calendar for the following selections and passing the [SelectionChangedEventArgs](#) as parameter.

- SingleSelection - A single date can be selected in a month view which can be equipped when user needs to select one date at a time.
- MultiSelection - More than one date can be selected.
- RangeSelection - It allows us to select a single date range in calendar month view.
- MultiRangeSelection - More than one date range can be selected in a month view.

### XML

```

<calendar:SfCalendar x:Name="calendar" SelectionChangedCommand="{Binding
OnSelectionChanged}">
<calendar:SfCalendar.BindingContext>
<local:CalendarViewModel/>
</calendar:SfCalendar.BindingContext>
</calendar:SfCalendar>

```

### C#

```

public class CalendarViewModel
{
    private string selectionChangedCommandText;
    public string SelectionChangedCommandText
    {
        get { return selectionChangedCommandText; }
        set { selectionChangedCommandText = value; }
    }
    public ICommand OnSelectionChanged { get; set; }
    public CalendarViewModel()
    {
        OnSelectionChanged = new
        Command<SelectionChangedEventArgs>(SelectionChanged);
    }
    private void SelectionChanged(SelectionChangedEventArgs obj)
    {

```

```

SelectionChangedCommandText = " DateAdded- "
+obj.DateAdded?.Count.ToString() +" "+ " DateRemoved- " +
obj.DateRemoved?.Count.ToString() + " " + " NewRangeAdded -"
+ obj.NewRangeAdded?.Count.ToString();
}
}

```

### Change First Day Of Week

By default, the starting day will be taken from the device culture. This can be modified using [FirstDayOfWeek](#) property. Changing the first day of week will be applied to both month and year view.

#### XML

```
<syncfusion:SfCalendar x:Name="calendar" FirstDayOfWeek="2"/>
```

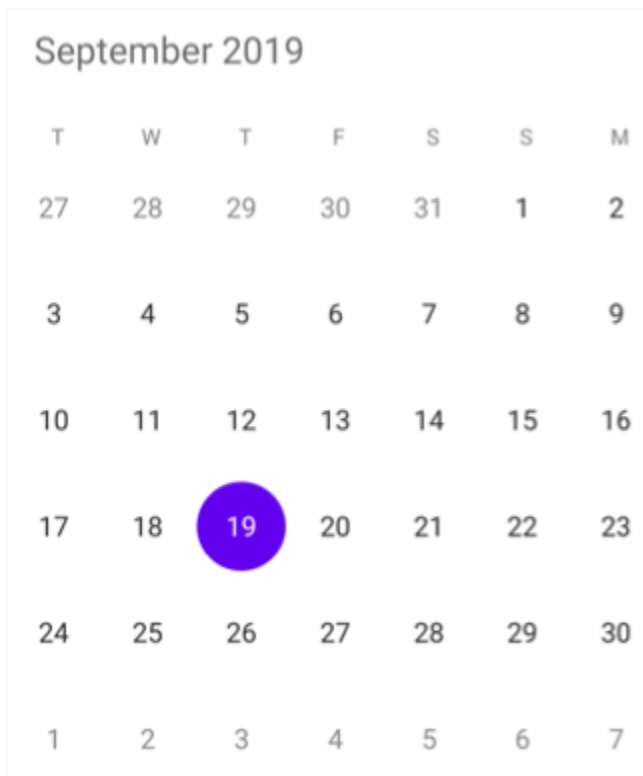
#### C#

```

SfCalendar calendar = new SfCalendar();
calendar.FirstDayOfWeek= 2;
this.Content = calendar;

```

**Note:** The value will be provided as integers starting from 0 as Sunday.



### Localization

SfCalendar control is available with complete localization support. Localization can be specified by setting the [Locale](#) property of the control using the format of Language code followed by Country code.



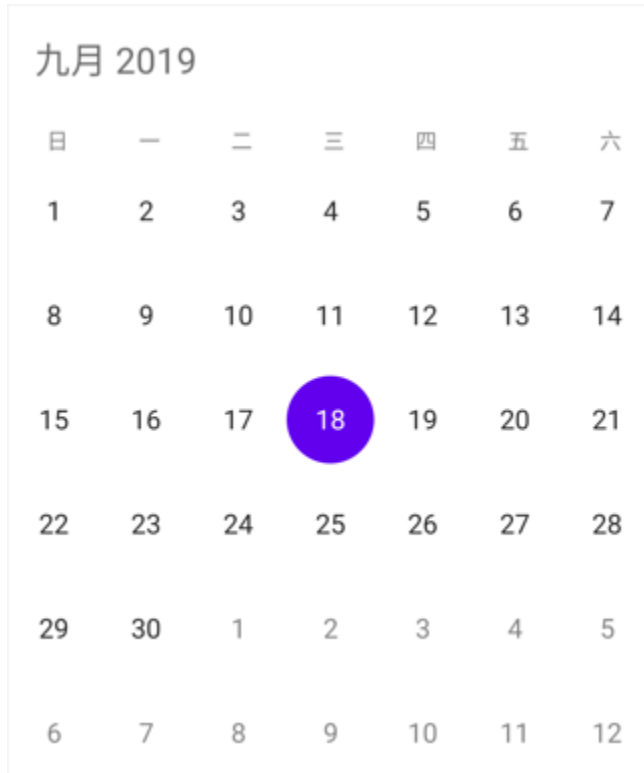
### Change default control language

Based on the locale specified, the strings in the control are localized accordingly.

#### C#

```
SfCalendar calendar = new SfCalendar ();  
calendar.Locale= new System.Globalization.CultureInfo("zh-CN");  
this.Content = calendar;
```

**Note:** By default, SfCalendar control is available with en-US locale.



### Localizing custom strings from PCL

You can localize the custom strings (All Day, No Events) used in the calendar control from PCL. It can be achieved by providing the custom strings to the specific language resx file and handling the required culture with the locale. In the below code, we have set Portugal as Calendar locale as well as custom strings.

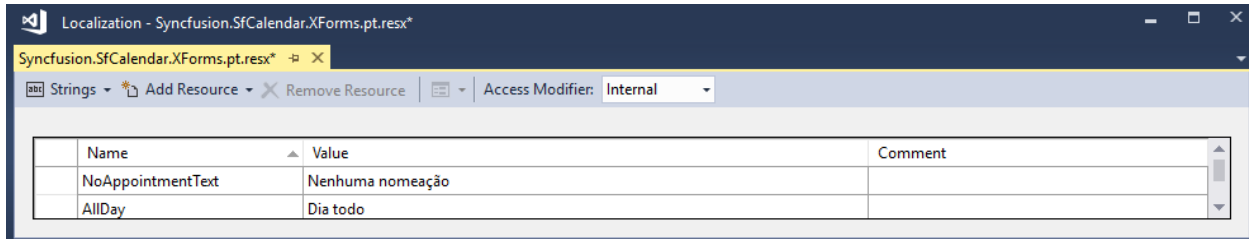
#### C#

```
calendar.Locale = new System.Globalization.CultureInfo("pt-PT");  
CalendarResourceManager.Manager = new  
System.Resources.ResourceManager("Sample  
name.Resources.Syncfusion.SfCalendar.XForms",  
GetType().GetTypeInfo().Assembly);  
CultureInfo.CurrentUICulture = new CultureInfo("pt");
```

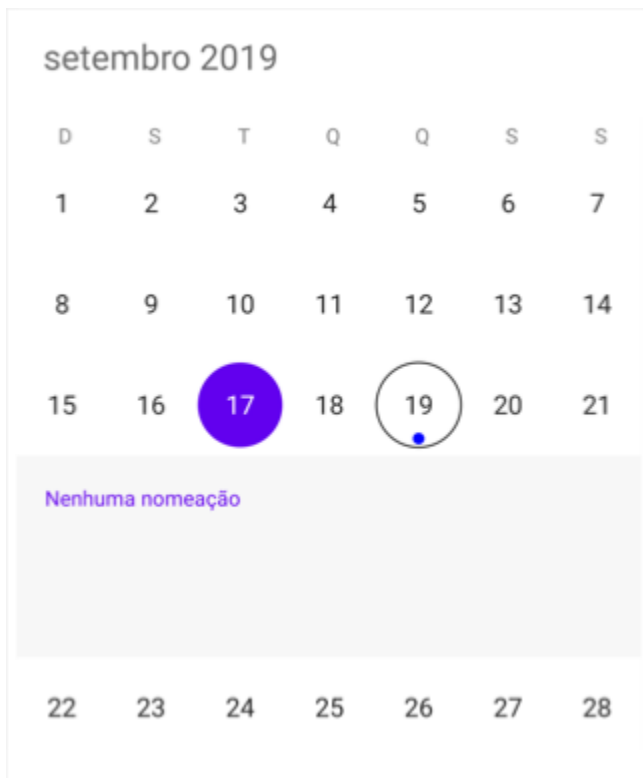
### Adding resx file

You need to add the required resx files under the Resources folder in the PCL project and the filename should be `Syncfusion.SfCalendar.Forms.LanguageCode.resx`.

Example: For Portuguese, `Syncfusion.SfCalendar.Forms.pt.resx`



Now, set the Build Action as `EmbeddedResource` for `Syncfusion.SfCalendar.Forms.pt.resx` file and Build Action as `Compile` for `Syncfusion.SfCalendar.Forms.pt.Designer.cs` file.



You can download the entire source code of this demo for Xamarin.Forms from here [LocaleFromPCL](#).

### Localizing the custom texts using platform renderer

You can localize the custom strings used in the calendar control. For that you need to configure it for each platform separately.

*Localizing custom text in Android renderer.* Localizing custom text in iOS renderer.

### Localizing custom text in Android renderer

You can localize custom text available in the control by adding equivalent localized string in the `string.xml` file.

## XML

```
<resources>
<string name="sfcalendar_inlineviewnoappointmenttext">Aucun
événement</string>
<string name="sfcalendar_inlineviewalldaytext">Toute la journée</string>
</resources>
```

Android can select and load resources from different directories, based on the current device configuration and locale, refer [here](#). For an example, if an application requires multiple languages you can follow the below steps.

The procedure for creating strings.xml files is as follows:

*Translate the strings.xml file to each language.* Create new folders under resource as values-ar, values-de, values-en and values-fr (The original values folder already exists). \* Place the translated strings.xml files in the respective folders.

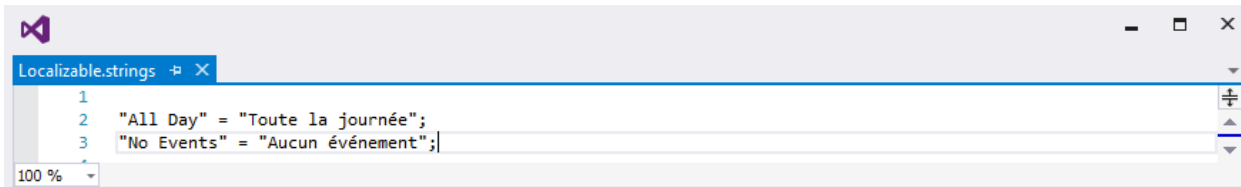


## NOTE

The corresponding Locale values folder updates only when the device language changes.

### *Localizing custom text in iOS renderer*

You can localize custom text available in the control by adding equivalent localized string in the Localizable.strings file, refer [here](#).



If an application requires multiple languages you can follow the below steps:

*Translate the Localizable.Strings file to each language.* Create new .lproj folders under resource as en.lproj, fr.lproj, de.lproj. \* Place the Localizable.Strings file in the respective .lproj folders.



---

**NOTE**

The corresponding `<Language>.lproj` folder updates only when the device language changes.

You can download the entire source code of this demo for Xamarin.Forms from here [Localization](#).

### Right to left(RTL)

SfCalendar supports to change the layout direction of the control in the right-to-left direction by setting the [FlowDirection](#) to `RightToLeft` or by changing the device language.

#### XML

```
<calendar:SfCalendar FlowDirection="RightToLeft">
</calendar:SfCalendar>
```

#### C#

```
calendar.FlowDirection = FlowDirection.RightToLeft;
```

---

**Note**

For implementing the `FlowDirection` in the control, Xamarin.Forms package version must be 3.0 and above. Please refer [RightToLeft](#) to get more details about `RightToLeft` flow direction in Xamarin.Forms.

#### *Android*

For Android, add `android:supportsRtl="true"` in your application tag of `AndroidManifest.xml` file, and make sure your `MinSDKVersion` is 17+. By changing the device language / enabling the device's Force RTL layout can achieve the `RightToLeft` layout direction in Calendar.

#### XML

```
<manifest ... >
<uses-sdk android:minSdkVersion="17" ... />
<application ... android:supportsRtl="true">
</application>
</manifest>
```

#### *iOS*

For iOS, add the `RightToLeft` language in the `CFBundleLocalizations` section of your `Info.plist` file, and make sure you're targeting iOS 9+.

#### XML

```
<resources>
<key>CFBundleDevelopmentRegion</key>
<string>en</string>
<key>CFBundleLocalizations</key>
<array>
<string>en</string>
<string>ar</string>
</array>
</resources>
```

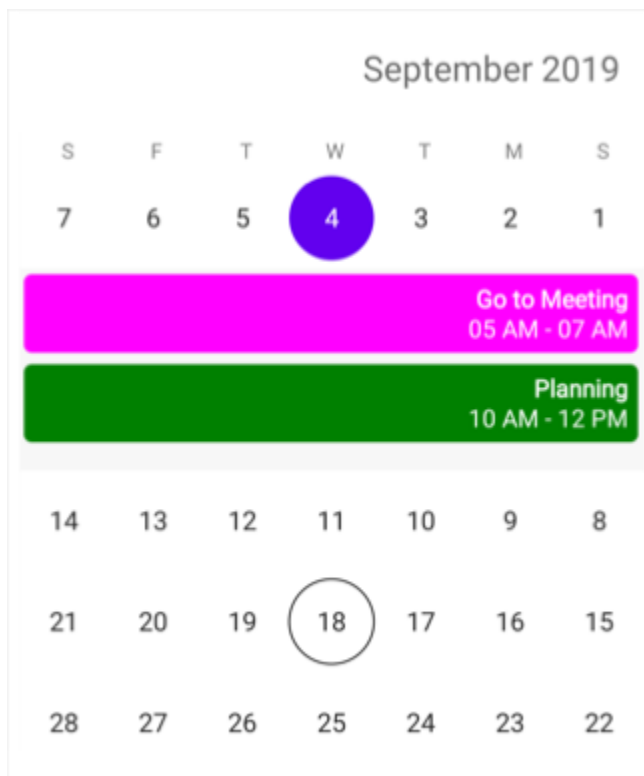
Localization native development region	String en
▼ Localizations	Array (2 items)
	String en
	String ar

### UWP

For UWP, you need to set `FlowDirection` to `RightToLeft` in the `MainPage.cs` file of the `UWP` project.

### C#

```
public MainPage()
{
    ...
    this.FlowDirection = FlowDirection.RightToLeft;
    LoadApplication (new App ());
    ...
}
```



## Customization of Calendar control

### How to Perform an Operation while a Calendar Cell is Tapped?

We can perform operation while the Calendar cell is Tapped using [CalendarTapped](#) event.

CalendarTapped event returns the date selected and the Appointments that are associated with the date selected.

Members	Description
(sender as SfCalendar)	Carries details about native control
DateTime	It shows the datetime in Calendar

### XML

```
<syncfusion:SfCalendar x:Name="calendar"
    OnCalendarTapped="Handle_OnCalendarTapped" />
```

### C#

```
void Handle_OnCalendarTapped(object sender, CalendarTappedEventArgs e)
{
    SfCalendar calendar = (sender as SfCalendar);
    DateTime date = e.datetime;
}
```

### How to Perform an Operation when the Selected Date Get Changed?

We can perform an operation when the selected date get changed using [SelectionChanged](#) event which returns the dates selected and dates deselected from the [SfCalendar](#).

Members	Description
(sender as SfCalendar)	Carries details about native control
DateAdded	Date selected from the calendar
DateRemoved	Date deselected from the calendar
NewRangeAdded	Carries details about the selected multi range

### XML

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="MonthView"
    SelectionChanged="Handle_SelectionChanged" />
```

### C#

```
void Handle_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    ////// Get added and removed dates in Single, Multi and Range selection modes.
    IList<DateTime> selectedDates = e.DateAdded;
    IList<DateTime> deselectedDates = e.DateRemoved;
```

```

//// Gets the added date range in Multi-range selection mode.
IList<SelectionRange> selectionRange = e.NewRangeAdded;
}

```

### How to Perform an Operation when Navigate to Next Month?

User defined operation can be performed using [MonthChanged](#) event when navigating to next month. This event returns the details about current value and previous value of month.

Members	Description
(sender as SfCalendar)	Carries details about native control
Args	Carries details about MonthEventParameters

### XML

```

<syncfusion:SfCalendar x:Name="calendar" ViewMode="MonthView"
MonthChanged="Handle_MonthChanged" />

```

### C#

```

void Handle_MonthChanged(object sender, MonthChangedEventArgs e)
{
    SfCalendar calendar = (sender as SfCalendar);
    DateTime oldMonth = e.PreviousValue;
    DateTime currentMonth = e.CurrentValue;
}

```

### How to Perform an Operation while Dealing with Appointments?

[InlineToggled](#) event returns the selected date along with the appointments it carries. Using this event user can perform operation while dealing with appointments.

Members	Description
(sender as SfCalendar)	Carries details about native control
SelectedAppointment	Appointments from the selected date

### XML

```

<syncfusion:SfCalendar x:Name="calendar"
InlineToggled="Handle_InlineToggled" />

```

### C#

```

void Handle_InlineToggled(object sender, InlineToggledEventArgs e)
{
    if ((args.SelectedAppointment as CalendarEventCollection).Count != 0)
    {
        string subject = (e.SelectedAppointment as
CalendarEventCollection)[0].Subject;
    }
}

```

```
DateTime startTime = (e.SelectedAppointment as  
CalendarEventCollection)[0].StartTime;  
DateTime endTime = (e.SelectedAppointment as  
CalendarEventCollection)[0].EndTime;  
}  
}
```

### How to Customize Cell or Month View?

[OnMonthCellLoaded](#) event allows us to customize **SfCalendar** control. It returns MonthCell args

Members	Description
Args	Carries details about MonthCell

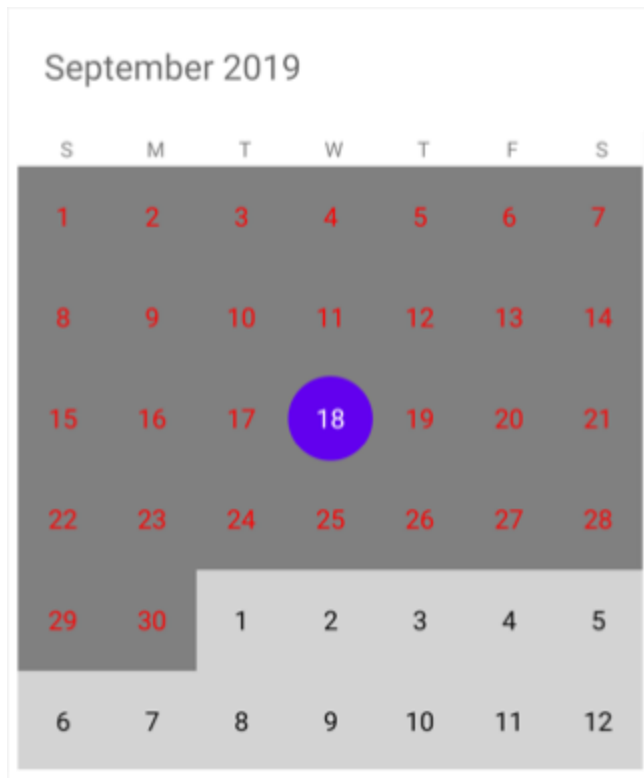
### XML

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="MonthView"  
OnMonthCellLoaded="Handle_OnMonthCellLoaded" />
```

### C#

```
void Handle_OnMonthCellLoaded(object sender, MonthCellLoadedEventArgs e)  
{  
    if (e.IsCurrentMonth)  
    {  
        e.BackgroundColor = Color.Gray;  
        e.TextColor = Color.Red;  
    }  
    else  
    {  
        e.BackgroundColor = Color.LightGray;  
        e.TextColor = Color.Black;  
    }  
}
```





### Create your own custom calendar month cell view

You can customize the month view with custom view using the `View` property of `MonthCellLoadedEventArgs` in the `OnMonthCellLoaded` event `SfCalendar` control.

#### C#

```
private void Calendar_OnMonthCellLoaded(object sender,
MonthCellLoadedEventArgs e)
{
    var button = new Button();
    button.Text = e.Date.Day.ToString();
    e.View = button;
}
```



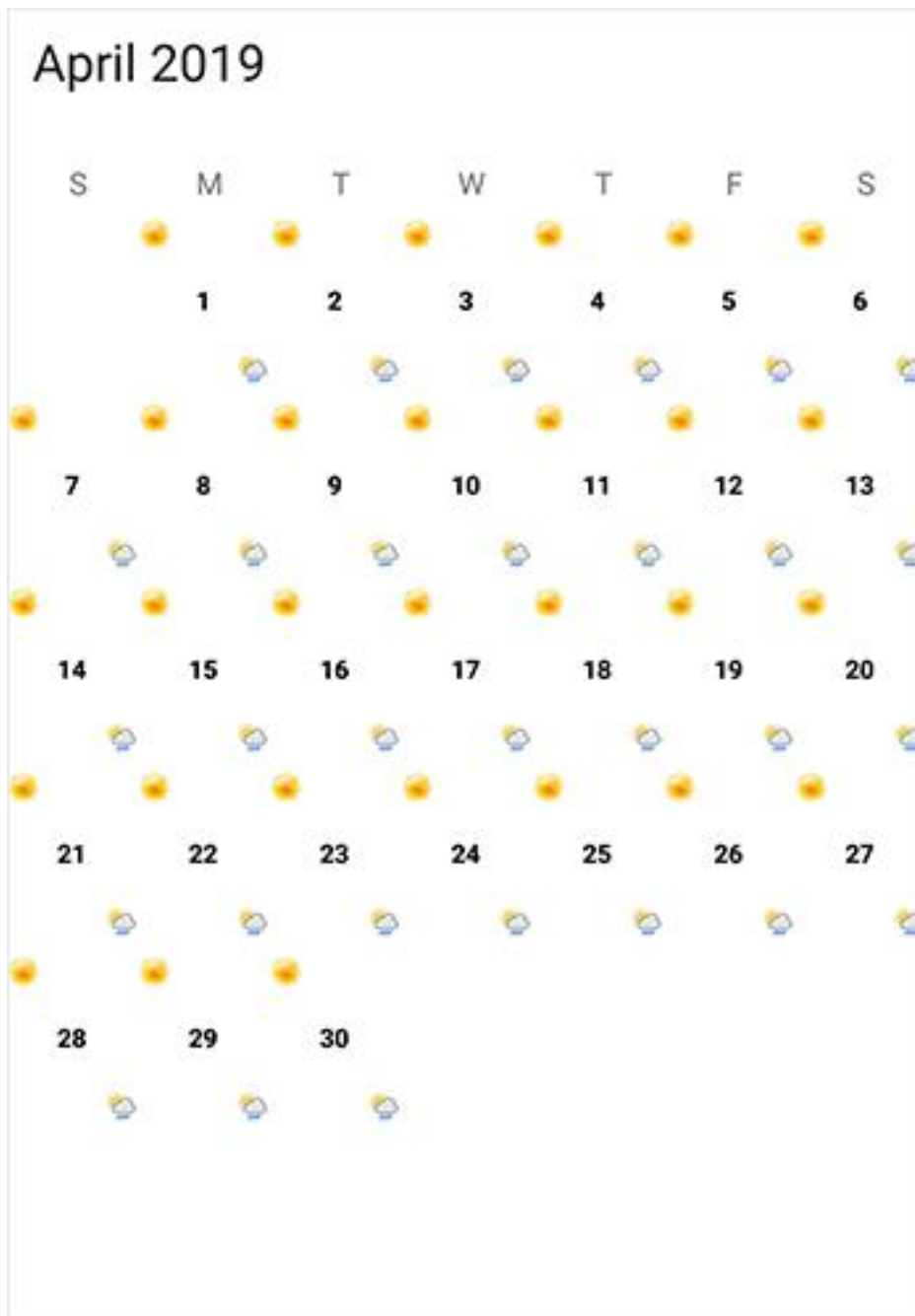
How to customize month view cell using a template?

You can customize the month cell of the **SfCalendar** using [CellTemplate](#) property of [MonthViewSettings](#).

#### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:MonthCellCustomization"
x:Class="MonthCellCustomization.MainPage"
xmlns:calendar="clr-namespace:Syncfusion.SfCalendar.XForms;assembly=Syncfusion.SfCalendar.XForms"
">
<ContentPage.Content>
<calendar:SfCalendar x:Name="calendar" ShowLeadingAndTrailingDays="true" BackgroundColor="White">
<calendar:SfCalendar.MonthViewSettings>
<calendar:MonthViewSettings>
<calendar:MonthViewSettings.CellTemplate>
<DataTemplate>
<Grid BackgroundColor="White">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
```

```
<RowDefinition Height="*" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Image HorizontalOptions="Center" Source="Weather1.png" Grid.Row="0" Grid.Column="0" />
<Label Text="{Binding Day}" FontAttributes="Bold" TextColor="Black" Grid.Row="1" HorizontalTextAlignment="Center" VerticalTextAlignment="Center" Grid.Column="1" FontSize="10" />
<Image HorizontalOptions="Center" Grid.Row="2" Source="Weather2.png" Grid.Column="2" />
</Grid>
</DataTemplate>
</calendar:MonthViewSettings.CellTemplate>
</calendar:MonthViewSettings>
</calendar:SfCalendar.MonthViewSettings>
</calendar:SfCalendar>
</ContentPage.Content>
</ContentPage>
```



How to Perform the Operation while long pressing the dateCell?

[OnDateCellHolding](#) event returns the long pressed date along with the **SfCalendar** it carries. Using this event user can perform operation while long pressing the date.

Members	Description
Args	Carries details with long pressed date and Calendar

### XML

```
<syncfusion:SfCalendar x:Name="sfcalendar"
    OnDateCellHolding="Handle_OnDateCellHolding" />
```

**C#**

```
void Handle_OnDateCellHolding(object sender,
    Syncfusion.SfCalendar.XForms.DayCellHoldingEventArgs e)
{
    // do the operation while long pressing the date cell
}
```

## How to Resize the SfCalendar Control?

SfCalendar control can be resized using **WidthRequest** and **HeightRequest** properties in SfCalendarcontrol.

**C#**

```
SfCalendar calendar = new SfCalendar();
calendar.WidthRequest = 200;
calendar.HeightRequest = 200;
```

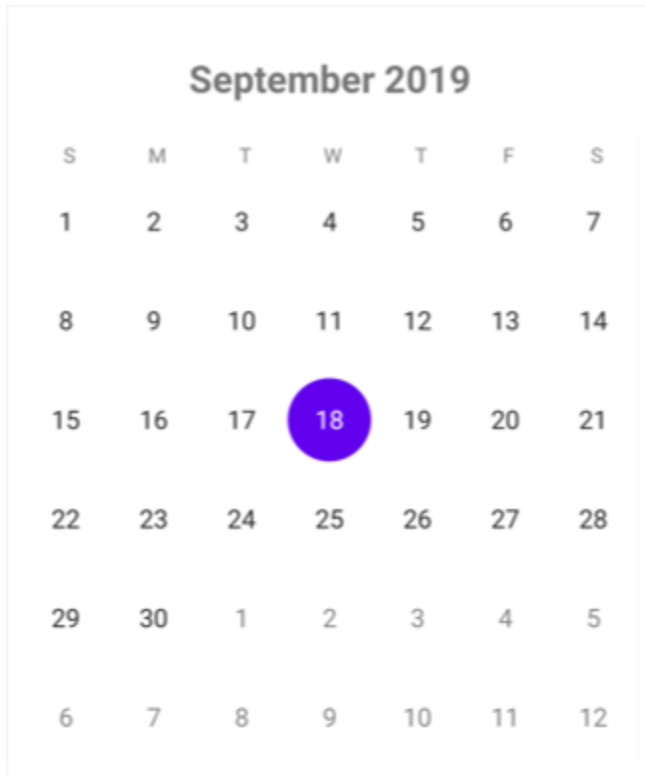
## How to Customize the SfCalendar Header?

[HeaderView](#) property of SfCalendar allows us to customize **SfCalendar** Header. It returns custom view for SfCalendarHeader

Members	Description
HeaderView	Carries custom view for Calendar Header

**XML**

```
<syncfusion:SfCalendar x:Name="calendar">
    <syncfusion:SfCalendar.HeaderView>
        <Label Text="{binding CalendarDate}" HorizontalTextAlignment="Center"
            VerticalTextAlignment="Center" FontAttributes="Bold" FontSize="Large"/>
    </syncfusion:SfCalendar.HeaderView>
</syncfusion:SfCalendar>
```



How to enable or disable the YearView in SfCalendar?

[ShowYearView](#) property of **SfCalendar** allows us to enable and disable the YearView of **SfCalendar**. The default value of ShowYearView is true.

Members	Description
ShowYearView	Carries boolean value which is used to enable or disable the YearView in SfCalendar

#### XML

```
<syncfusion:SfCalendar Grid.Row="1" ShowYearView="false" x:Name="calendar" />
```

How to enable or disable the Horizontal and Vertical cell grid lines in SfCalendar?

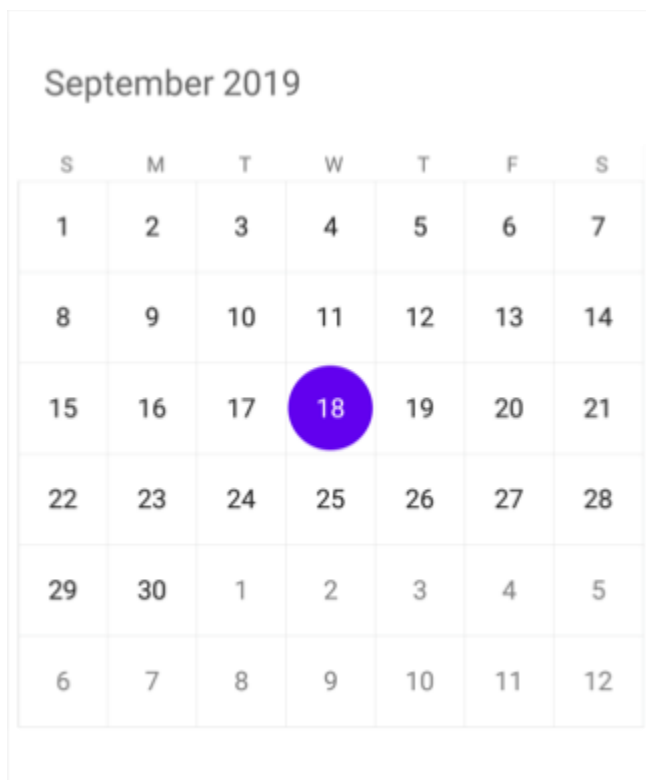
[CellGridOptions](#) property of **MonthViewSettings** allows us to enable and disable the horizontal and vertical border lines of **SfCalendar**.

Members	Description
CellGridOptions	Carries enum value which is used to enable or disable the vertical and horizontal border lines in SfCalendar
VerticalLines	This enum value of CellGridOptions is used to enable the vertical border lines in SfCalendar

HorizontalLines	This enum value of CellGridOptions is used to enable the horizontal border lines in SfCalendar
Both	This enum value of CellGridOptions is used to enable the vertical and horizontal border lines in SfCalendar
None	This enum value of CellGridOptions is used to disable the vertical and horizontal border lines in SfCalendar

**XML**

```
<syncfusion:SfCalendar x:Name="calendar" >
  <syncfusion:SfCalendar.MonthViewSettings>
    <syncfusion:MonthViewSettings CellGridOptions="Both" />
  </syncfusion:SfCalendar.MonthViewSettings>
</syncfusion:SfCalendar>
```

**Customize the year cell or year view**

You can customize the year cell of the **SfCalendar** control using the [OnYearCellLoaded](#) event, which returns [YearCellLoadedEventArgs](#). The [YearCellLoadedEventArgs](#) has the following properties to customize the year view: [BackgroundColor](#), [Font](#), [FontFamily](#), [Month](#), [MonthHeaderBackgroundColor](#), [MonthHeaderTextColor](#), [MonthLayoutBackgroundColor](#), [TextColor](#), and [View](#).

**C#**

```
private void Calendar_OnYearCellLoaded(object sender,
  YearCellLoadedEventArgs e)
```

```
{
e.BackgroundColor = Color.Red;
e.Font = Font.SystemFontOfSize(12, FontAttributes.Italic);
e.FontFamily = "Times New Roman";
e.MonthHeaderBackgroundColor = Color.Blue;
e.MonthHeaderTextColor = Color.Black;
e.MonthLayoutBackgroundColor = Color.Gray;
e.TextColor = Color.Green;
}
```



Customize the year view with custom UI

You can customize the YearView with Custom UI in the SfCalendar control using the View property of YearCellLoadedEventArgs in the OnYearCellLoaded event.

**C#**

```
private void Calendar_OnYearCellLoaded(object sender,
YearCellLoadedEventArgs e)
{
var button = new Button();
button.Text = e.Month.ToString("MMMM");
e.View = button;
}
```



### Deselect today selection on initial load

Initially, the calendar is loaded with the current day as selected date in **MonthView** when the **SelectionMode** is set to **SingleSelection**, but you can deselect the date on initial loading in **SfCalendar** by set the [SelectedDate](#) property as null.

### C#

```
calendar.SelectedDate = null;
```

### Feature Comparison across the different platforms in Xamarin.

This section covers the features of **SfCalendar** specific to Android,iOS and UWP platforms. The following comparison table help to find available features are in different platforms.

Features are in SfCalendar	Android	iOS	UWP	Achieved By
View of SfCalendar	MonthView	MonthView	MonthView	ViewMode
	YearView	YearView	YearView	
	Decade	Decade	Decade	
	Century/p>	Century/p>	Century/p>	
Date Selection	Single	Single	Single	SelectionMode
	Multiple	Multiple	Multiple	
	Range	Range	Range	
	MultiRange	MultiRange	MultiRange	
Blackout Dates	Yes	Yes	Yes	BlackoutDates

FirstDay of Week	Yes	Yes	Yes	FirstDayofWeek
Adding the appointments	Yes	Yes	Yes	DataSource
Visibility changes of SfCalendar	Yes	Yes	Yes	IsVisible
Localization support	Yes	Yes	Yes	Locale
Show Appointments	Yes	Yes	Yes	ShowInlineEvents
Move to Specific date	Yes	Yes	Yes	MoveToDate and MoveToDate(DateTime datetime)
Navigation of month Programmatically	Yes	Yes	Yes	Forward() and Backward()
Custom Day label support	Yes	Yes	Yes	CustomDayLabels
Restrict the date selection and navigation	Yes	Yes	Yes	MinDate and MaxDate
Dynamically select the dates in MultiSelection	Yes	Yes	Yes	SelectedDates
Calendar's Header height customization	Yes	Yes	Yes	HeaderHeight
Enabling and disabling the header of SfCalendar	Yes	Yes	Yes	ShowHeader
Customization of Navigation Buttons	Yes	Yes	Yes	NavigationButtonHeight NavigationButtonWidth

				NavigationArrowThickness
Enable or disable the navigation button visible	Yes	Yes	Yes	ShowNavigationButtons
Customization of MonthView	Yes	Yes	Yes	MonthViewSettings
Customization of YearView	Yes	Yes	No	YearViewSettings
Deselect the date selection in single selection mode	Yes	Yes	No	ToggleDaySelection
Customization of Month Day's text	Yes	Yes	Yes	MonthViewSettings
Option to move the next / previous month while clicking the previous/next month date in current active cell	Yes	Yes	Yes	NavigateToMonthOn- InactiveDatesSelection
Swiping behavior	Yes Swiping of SfCalendar has been restricted only with RangeSelection. With that RangeSelection mode We can swipe the month by using	Yes Swiping of SfCalendar has been restricted only with RangeSelection. With that RangeSelection mode We can swipe the month by using	Yes Swiping of SfCalendar has been restricted only with RangeSelection. With that RangeSelection mode We can swipe the month by using	—

	navigation buttons	navigation buttons	navigation buttons	
Restrict the swiping behavior	Yes	Yes	Yes	EnableSwiping
Maximum number of Inline event count indication	Yes	Yes	No	MaximumEventIndicator Count
Dynamically select the dates in RangeSelecti on  By setting the start and end range of date selection, RangeSelecti on is possible	Yes	Yes	Yes	SelectedRange
Customization of Cell Border Lines	Horizontal	Horizontal	Horizontal	CellGridOptions
	Vertical	Vertical	Vertical	
	Both	Both	Both	
	None	None	None	
Scrolling Direction	Horizontal  Vertical	Horizontal  Vertical	Horizontal  Vertical	NavigationDirection

Restrict the YearView Option	Yes	Yes	Yes	ShowYearView
Calendarâ€™s Header customization	Yes	Yes	Yes	HeaderView
Common DataTemplate	Yes	Yes	Yes	CellTemplate
DataTemplate with Custom Object on OnMonthCell	Yes	Yes	Yes	CellTemplate and CellBindingContext in an argument of OnMonthCellLoaded

## Theming

Theming is a set of resources which are used to provide the consistency look for calendar.

You can modify the default appearance of calendar using this support. By default calendar have default theme resources and it's located in `Syncfusion.Xamarin.Core` assembly. You need to merge them in your application's resource to apply the theme.

In the below code you can see the default color and key value for the default resources.

### XML

```
<Color x:Key="SfCalendarInlineTextColor">#414141</Color>
<Color x:Key="SfCalendarBlackoutColor">#C2C2C2</Color>
<Color x:Key="SfCalendarDateSelectionColor">#F5F5F5</Color>
<Color x:Key="SfCalendarInlineBackgroundColor">#F5F5F5</Color>
<Color x:Key="SfCalendarSelectedDayTextColor">#FFFFFF</Color>
<Color x:Key="SfCalendarBorderColor">#F0F0F0</Color>
<Color x:Key="SfCalendarWeekDayBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfCalendarWeekDayTextColor">#707070</Color>
<Color x:Key="SfCalendarDisabledBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfCalendarDisabledTextColor">#C2C2C2</Color>
<Color x:Key="SfCalendarPreviousMonthBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfCalendarPreviousMonthTextColor">#C2C2C2</Color>
<Color x:Key="SfCalendarCurrentMonthBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfCalendarCurrentMonthTextColor">#414141</Color>
<Color x:Key="SfCalendarWeekEndTextColor">#707070</Color>
<Color x:Key="SfCalendarDayHeaderTextColor">#707070</Color>
<Color x:Key="SfCalendarDayHeaderBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfCalendarWeekEndBackgroundColor">#C2C2C2</Color>
<Color x:Key="SfCalendarHeaderBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfCalendarHeaderTextColor">#707070</Color>
<Color x:Key="SfCalendarYearViewLayoutBackground">#FFFFFF</Color>
<Color x:Key="SfCalendarYearViewHeaderBackground">#FFFFFF</Color>
<Color x:Key="SfCalendarYearViewMonthLayoutBackground">#FFFFFF</Color>
<Color x:Key="SfCalendarYearViewMonthHeaderBackground">#FFFFFF</Color>
<Color x:Key="SfCalendarYearViewDateTextColor">#414141</Color>
<Color x:Key="SfCalendarYearViewHeaderTextColor">#707070</Color>
<Color x:Key="SfCalendarAgendaSelectedDateColor">#FFFFFF</Color>
```

You can apply the default theme (light theme) or customized theme to calendar.

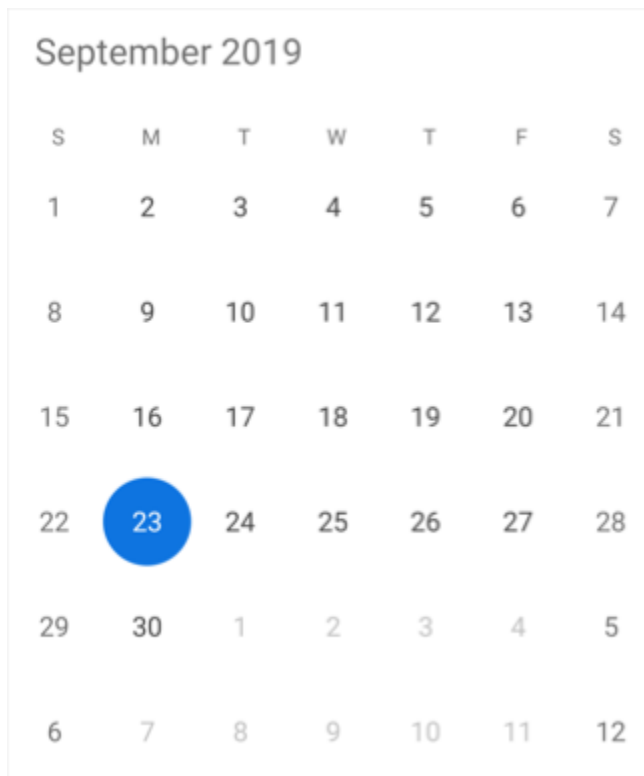
#### Default theme (light theme)

You need to apply the syncfusion theme dictionaries in your application to view the default theme (light theme).

#### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:CalendarSample" x:Class="CalendarSample.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfCalendar.XForms;assembly=Syncfusion.SfCalendar.XForms
" xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms">
<ContentPage.Resources>
<syncTheme:SyncfusionThemeDictionary>
<syncTheme:SyncfusionThemeDictionary.MergedDictionaries>
<syncTheme:LightTheme x:Name="LightTheme" />
</syncTheme:SyncfusionThemeDictionary.MergedDictionaries>
</syncTheme:SyncfusionThemeDictionary>
</ContentPage.Resources>
<syncfusion:SfCalendar x:Name="calendar" ViewMode="MonthView"
ShowInlineEvents="true" InlineViewMode="Inline" />
</ContentPage>
```

#### Month view



## Year view



## Customizing the default theme

You can customize the default theme by overriding the existing key and set the new value.

**XML**

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:CalendarSample" x:Class="CalendarSample.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfCalendar.XForms;assembly=Syncfusion.SfCalendar.XForms
" xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms">
<ContentPage.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<syncTheme:LightTheme />
<syncfusion:SfCalendarStyles />
<ResourceDictionary>
<Color x:Key="SfCalendarInlineBackgroundColor">Green</Color>
<Color x:Key="SfCalendarInlineTextColor">White</Color>
<Color x:Key="SfCalendarWeekEndBackgroundColor">Blue</Color>
<Color x:Key="SfCalendarYearViewHeaderTextColor">Blue</Color>
<Color x:Key="SfCalendarYearViewDateTextColor">Green</Color>
</ResourceDictionary>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</ContentPage.Resources>
```

```
<syncfusion:SfCalendar x:Name="calendar" ViewMode="MonthView"
ShowInlineEvents="true" InlineViewMode="Inline" />
</ContentPage>
```

---

**NOTE**

---

Xamarin.Forms version should be above 3 while customize the default theme.

Month view



Year view





You can download the entire source code of this demo for Xamarin.Forms from here [CalendarTheme](#).

#### AutomationId

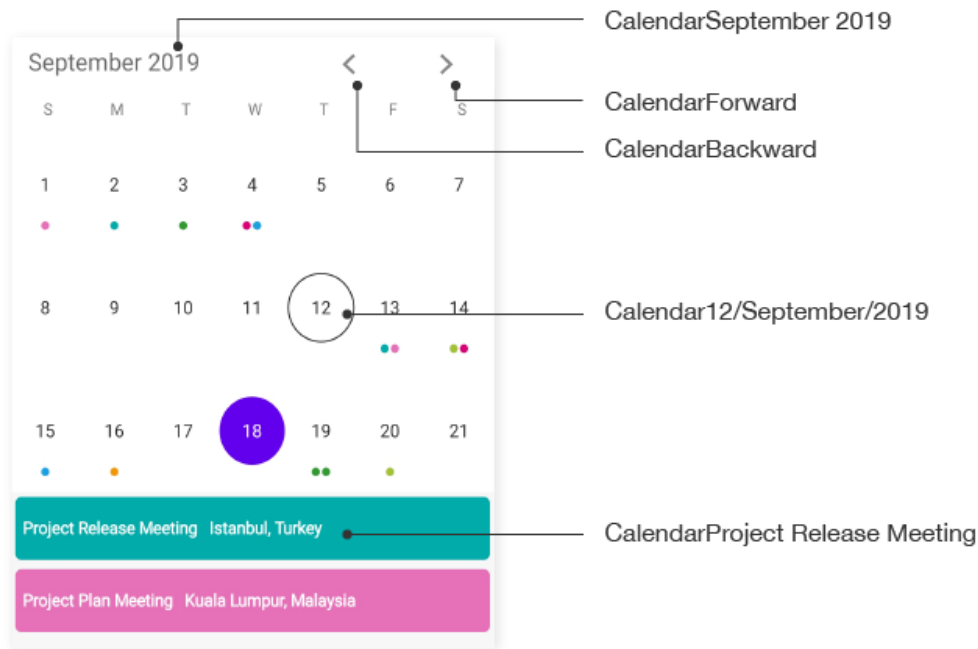
The SfCalendar control has built-in **AutomationId** for inner elements. Please find the following table of Automation IDs for inner elements.

View	AutomationId Format	Example
MonthCell	dddd dd/MMMM/yyyy	Tuesday 27/May/2018
Month Header - Month view	MMMM yyyy	May 2018
Month Header - Week view	MMMM â€” MMMM yyyy	November â€” December 2018
Left Arrow	string	Backward
Right Arrow	string	Forward
Inline View , Agenda View	subject text	Meeting
Year View - Header	yyyy	2018
Year View - Year Cell	MMMM yyyy	January 2018
Decade View - Header	yyyy - yyyy	2020 â€” 2029
Decade View - Year Cell	yyyy	2020
Century View - Header	yyyy - yyyy	2000 â€” 2099

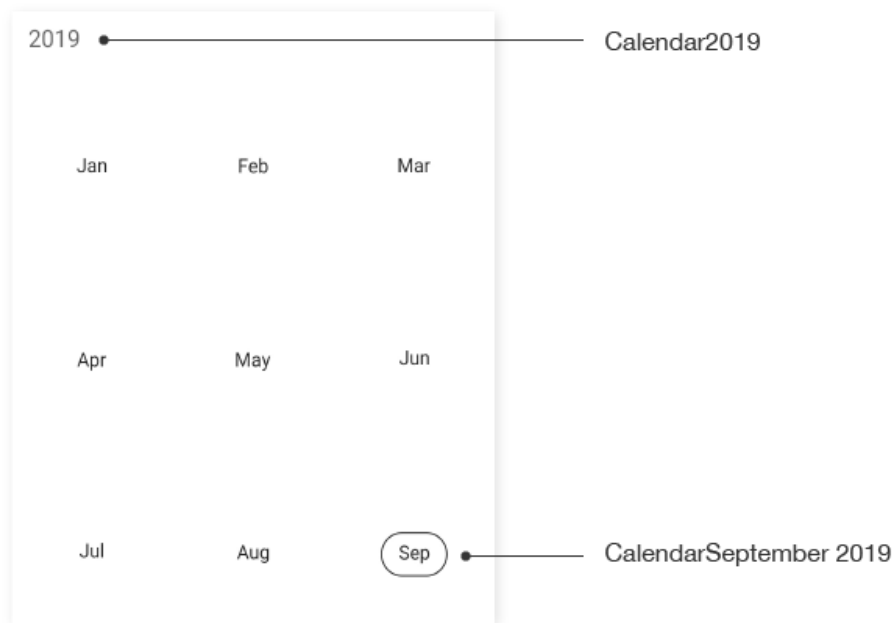
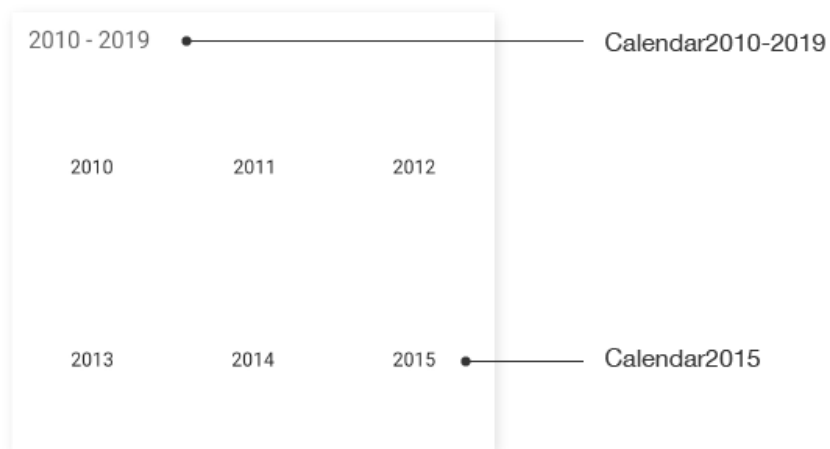
Century View - Year Cell	yyyy	2070 – 2079
--------------------------	------	-------------

To keep unique AutomationId, these inner elements' AutomationIds are updated based on the control's AutomationId. For example, if you set SfCalendar AutomationId as SfCalendar.AutomationId = Calendar, then the Automation framework will interact with the RightArrow button as CalendarForward. The following screenshots denote the AutomationIds for inner elements.

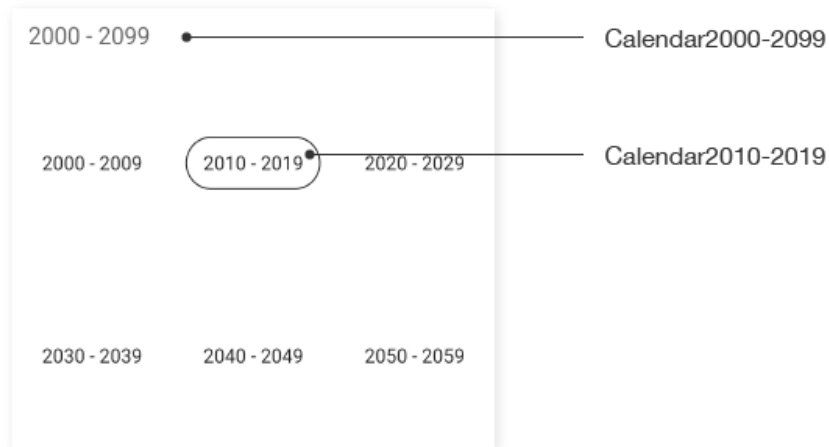
#### Month view



*Week view*

*Year view**Decade view*

### Century view

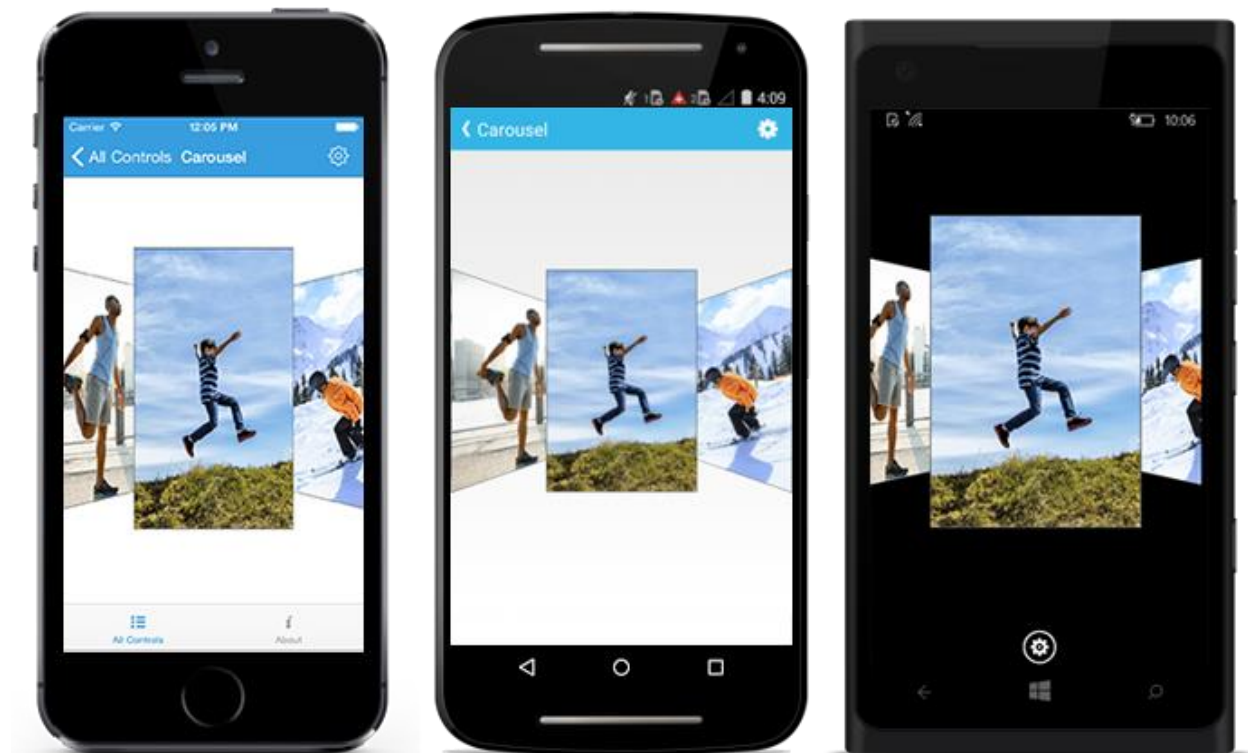


## SfCarousel

### Overview

The Essential Xamarin Carousel control allows navigating through image data in an interactive way so that they can be viewed or selected. Also, provides various customization options for its item arrangements.

It has been available in Xamarin.Forms, Xamarin.Android and Xamarin.iOS platforms also.



### Key Features:

- **Offset** - Support to specify the space between the unselected items in SfCarousel.
- **Rotation Angle** - Options to rotate all the items to a specified angle.
- **Duration** - Support to specify the time taken to move an item to the selected item position.

### Getting Started

This section explains how to showcase a Gallery of photos along with a Title using SfCarousel Control.

#### Adding SfCarousel reference

You can add SfCarousel reference using one of the following methods:

##### Method 1: Adding SfCarousel reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfCarousel). To add SfCarousel to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfCarousel](https://www.nuget.org/packages/Syncfusion.Xamarin.SfCarousel), and then install it.

![Adding SfCarousel reference from NuGet](images/Adding SfCarousel reference.png)

---

**Note:** Install the same version of SfCarousel NuGet in all the projects.

---

##### Method 2: Adding SfCarousel reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfCarousel control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

##### Method 3: Adding SfCarousel assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfCarousel.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfCarousel.Android.dll Syncfusion.SfCarousel.XForms.Android.dll Syncfusion.SfCarousel.XForms.dll Xamarin.Android.Support.v17.Leanback (from NuGet Packages) Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfCarousel.iOS.dll Syncfusion.SfCarousel.XForms.iOS.dll Syncfusion.SfCarousel.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll

UWP	Syncfusion.SfCarousel.UWP.dll Syncfusion.SfCarousel.XForms.UWP.dll Syncfusion.SfCarousel.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
-----	---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the SfCarousel on each platform

To use SfCarousel inside an application, each platform application must initialize the SfCarousel renderer. This initialization step varies from platform to platform and is discussed in the following sections.

#### Android and UWP

The Android and UWP launches the SfCarousel without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch SfCarousel in iOS, need to create an instance of SfCarouselRenderer in FinishedLaunching overridden method of AppDelegate class in iOS Project as shown below.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    new SfCarouselRenderer();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### ReleaseMode issue in UWP platform

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the SfCarousel assemblies in **App.xaml.cs** in UWP project as like in below code snippet.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
}
```

```

rootFrame.NavigationFailed += OnNavigationFailed;
// you'll need to add `using System.Reflection;`
List<Assembly> assembliesToInclude = new List<Assembly>();
//Now, add all the assemblies your app uses
assembliesToInclude.Add(typeof(SfCarouselRenderer).GetTypeInfo().Assembly);
// replaces Xamarin.Forms.Forms.Init(e);
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
...
}

```

### Create a Simple SfCarousel

The SfCarousel control is configured entirely in C# code or by using XAML markup. The following steps explain on how to create a SfCarousel and configure its elements,

- Adding namespace for the added assemblies.

#### XML

```

xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"

```

#### C#

```

using Syncfusion.SfCarousel.XForms;

```

- Now add the SfCarousel control with a required optimal name by using the included namespace.

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<carousel:SfCarousel x:Name="carousel" />
</ContentPage>

```

#### C#

```

using Syncfusion.SfCarousel.XForms;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();

```



```
SfCarousel carousel = new SfCarousel();  
this.Content = carousel;  
}  
}  
}
```

### Add Carousel Items

We can populate the carousel's items by using any one of the following ways,

- Through SfCarouselItem
- Through ItemTemplate

#### Through SfCarouselItem

By passing the list of `SfCarouselItem`, we can get the view of SfCarousel control. In that we can pass Images as well as Item content.

The following code example illustrates to add list of Images in Carousel ,

#### C#

```
using Syncfusion.SfCarousel.XForms;  
using System.Collections.ObjectModel;  
using Xamarin.Forms;  
namespace CarouselSample  
{  
    public partial class MainPage : ContentPage  
    {  
        public MainPage()  
        {  
            InitializeComponent();  
            SfCarousel carousel = new SfCarousel()  
            {  
                ItemWidth = 170,  
                ItemHeight = 250  
            };  
            ObservableCollection<SfCarouselItem> carouselItems = new  
            ObservableCollection<SfCarouselItem>();  
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person1.png" });  
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person2.png" });  
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person3.png" });  
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person4.png" });  
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person5.png" });  
            carousel.ItemsSource = carouselItems;  
            this.Content = carousel;  
        }  
    }  
}
```

The following code example illustrates to add list of Item in Carousel ,

**C#**

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfCarousel carousel = new SfCarousel()
            {
                ItemWidth = 170,
                ItemHeight = 250
            };
            ObservableCollection<SfCarouselItem> carouselItems = new
            ObservableCollection<SfCarouselItem>();
            carouselItems.Add(new SfCarouselItem()
            {
                ItemContent = new Button()
                {
                    Text = "ItemContent1",
                    TextColor = Color.White,
                    BackgroundColor = Color.FromHex("#7E6E6B"),
                    FontSize = 12
                }
            });
            carouselItems.Add(new SfCarouselItem()
            {
                ItemContent = new Label()
                {
                    Text = "ItemContent2",
                    BackgroundColor = Color.FromHex("#7E6E6B"),
                    HorizontalTextAlignment = TextAlignment.Center,
                    VerticalTextAlignment = TextAlignment.Center,
                    FontSize = 12
                }
            });
            carouselItems.Add(new SfCarouselItem()
            {
                ItemContent = new Image()
                {
                    Source = "carousel_person1.png",
                    Aspect = Aspect.AspectFit
                }
            });
            carousel.ItemsSource = carouselItems;
            this.Content = carousel;
        }
    }
}
```

*Through ItemTemplate*

**ItemTemplate** property of SfCarousel control is used to customize the contents of SfCarousel items.

- Create a model view which holds image data

**C#**

```
namespace CarouselSample
{
    public class CarouselModel
    {
        public CarouselModel(string imageString)
        {
            Image = imageString;
        }
        private string _image;
        public string Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }
}
```

- Populate carousel items collection in View model with the image data.

**C#**

```
using System.Collections.Generic;
namespace CarouselSample
{
    public class CarouselViewModel
    {
        public CarouselViewModel()
        {
            ImageCollection.Add(new CarouselModel("carousel_person1.png"));
            ImageCollection.Add(new CarouselModel("carousel_person2.png"));
            ImageCollection.Add(new CarouselModel("carousel_person3.png"));
            ImageCollection.Add(new CarouselModel("carousel_person4.png"));
            ImageCollection.Add(new CarouselModel("carousel_person5.png"));
        }
        private List<CarouselModel> imageCollection = new List<CarouselModel>();
        public List<CarouselModel> ImageCollection
        {
            get { return imageCollection; }
            set { imageCollection = value; }
        }
    }
}
```

The following code illustrates the way to use `ItemTemplate` in both XAML as well as C#

**XAML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```

xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel/>
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
HeightRequest="400"
WidthRequest="800" />
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            CarouselViewModel carouselViewModel = new CarouselViewModel();
            SfCarousel carousel = new SfCarousel()
            {
                HeightRequest = 400,
                WidthRequest = 800
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            carousel.ItemTemplate = itemTemplate;
            carousel.ItemsSource = carouselViewModel.ImageCollection;
            this.Content = carousel;
        }
    }
}

```

```

public class CarouselModel
{
    public CarouselModel(string imageString)
    {
        Image = imageString;
    }
    private string _image;
    public string Image
    {
        get { return _image; }
        set { _image = value; }
    }
}

public class CarouselViewModel
{
    public CarouselViewModel()
    {
        ImageCollection.Add(new CarouselModel("carousel_person1.png"));
        ImageCollection.Add(new CarouselModel("carousel_person2.png"));
        ImageCollection.Add(new CarouselModel("carousel_person3.png"));
        ImageCollection.Add(new CarouselModel("carousel_person4.png"));
        ImageCollection.Add(new CarouselModel("carousel_person5.png"));
    }
    private List<CarouselModel> imageCollection = new List<CarouselModel>();
    public List<CarouselModel> ImageCollection
    {
        get { return imageCollection; }
        set { imageCollection = value; }
    }
}

```

**Information:** Carousel's Images are placed within the application folder for Android, iOS and UWP with build action Android Resource, Bundled Resource and Content respectively.

**Note:** In addition, carousel provides a support to load the Images from URL and SD Card location.

### Setting the height and width of the carousel item

ItemHeight and ItemWidth properties are used to change the height and width of carouselItem in carousel panel.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<carousel:SfCarousel x:Name="carousel"
ItemHeight="170"
ItemWidth="270"/>
</ContentPage>

```

**C#**

```

using Syncfusion.SfCarousel.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfCarousel carousel = new SfCarousel()
            {
                ItemWidth = 170,
                ItemHeight = 250
            };
            ObservableCollection<SfCarouselItem> carouselItems = new
            ObservableCollection<SfCarouselItem>();
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person1.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person2.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person3.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person4.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person5.png"
            });
            carousel.ItemsSource = carouselItems;
            this.Content = carousel;
        }
    }
}

```

**Set Desire Item to be Selected**

We can bring particular item to the center of the screen using **SelectedIndex** property in SfCarousel control.

The items can be populated as described [above](#)

**XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<carousel:SfCarousel x:Name="carousel"
ItemHeight="170"
ItemWidth="270"
SelectedIndex="2"/>
</ContentPage>

```

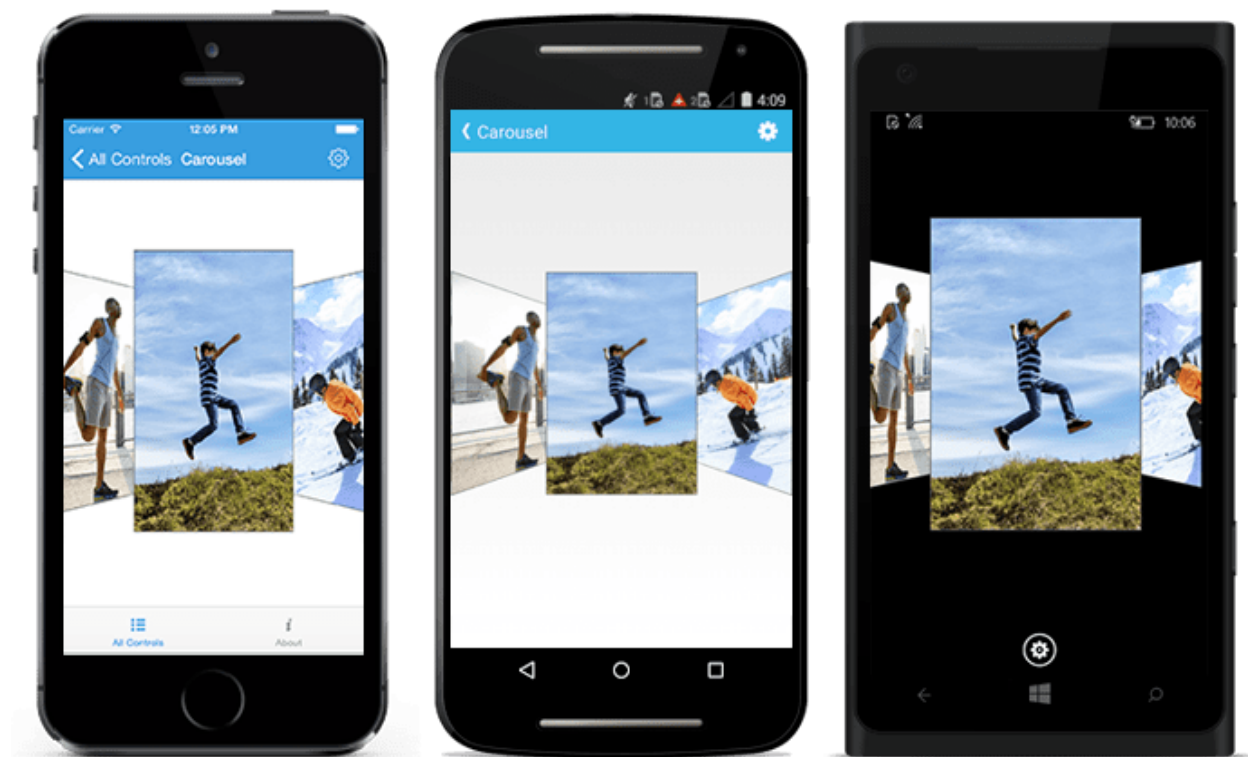
**C#**

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfCarousel carousel = new SfCarousel()
            {
                ItemWidth = 170,
                ItemHeight = 250,
                SelectedIndex = 2
            };
            ObservableCollection<SfCarouselItem> carouselItems = new
            ObservableCollection<SfCarouselItem>();
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person1.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person2.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person3.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person4.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person5.png"
            });
            carousel.ItemsSource = carouselItems;
            this.Content = carousel;
        }
    }
}
```

---

**Note:** The `SelectedIndex` property will be 0 by default.

---



You can find the complete getting started sample from this [link](#).

### Linear Arrangement

The Carousel items can be populated in the view in a stacked linear layout by setting the **ViewMode** property to Linear. The present option is **Default**.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
  <ContentPage.BindingContext>
    <local:CarouselViewModel/>
  </ContentPage.BindingContext>
  <ContentPage.Resources>
    <ResourceDictionary>
      <DataTemplate x:Key="itemTemplate">
        <Image Source="{Binding Image}"
Aspect="AspectFit"/>
      </DataTemplate>
    </ResourceDictionary>
  </ContentPage.Resources>
  <ContentPage.Content>
    <carousel:SfCarousel x:Name="carousel">
```



```

ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
HeightRequest="400"
WidthRequest="800"
ViewModel="Linear"/>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            CarouselViewModel carouselViewModel = new CarouselViewModel();
            SfCarousel carousel = new SfCarousel()
            {
                HeightRequest = 400,
                WidthRequest = 800,
                ViewModel = ViewModel.Linear
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            carousel.ItemTemplate = itemTemplate;
            carousel.ItemsSource = carouselViewModel.ImageCollection;
            this.Content = carousel;
        }
    }

    public class CarouselModel
    {
        public CarouselModel(string imageString)
        {
            Image = imageString;
        }
        private string _image;
        public string Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }

    public class CarouselViewModel
    {
        public CarouselViewModel()
    }

```

```
{
    ImageCollection.Add(new CarouselModel("carousel_person1.png"));
    ImageCollection.Add(new CarouselModel("carousel_person2.png"));
    ImageCollection.Add(new CarouselModel("carousel_person3.png"));
    ImageCollection.Add(new CarouselModel("carousel_person4.png"));
    ImageCollection.Add(new CarouselModel("carousel_person5.png"));
}
private List<CarouselModel> imageCollection = new List<CarouselModel>();
public List<CarouselModel> ImageCollection
{
    get { return imageCollection; }
    set { imageCollection = value; }
}
}
```

**Note:** It is important to include Xamarin.Android.Support.v17.Leanback library to use carousel linear mode in Android platform.



### Populating Data

SfCarousel control, supports binding to different data sources such as IList Data Source, Observable Collection Data Source.

### Through Binding

Items can be populated in SfCarousel control through data source and applying custom template as explained below.

#### *Create a Model with Data*

SfCarousel items can be populated with a collection of image data. For example, a user may want to create a SfCarousel control which will display a list of images.

The SfCarousel model looks as follows.

#### **C#**

```
namespace CarouselSample
{
    public class CarouselModel
    {
        public CarouselModel(string imageString)
        {
            Image = imageString;
        }
        private string _image;
        public string Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }
}
```

Populate carousel items collection in View model with the image data. The below code works when the images are placed within the application folder for Android, iOS and UWP with build action Android Resource, Bundled Resource and Content respectively.

#### **C#**

```
using System.Collections.Generic;
namespace CarouselSample
{
    public class CarouselViewModel
    {
        public CarouselViewModel()
        {
            ImageCollection.Add(new CarouselModel("carousel_person1.png"));
            ImageCollection.Add(new CarouselModel("carousel_person2.png"));
            ImageCollection.Add(new CarouselModel("carousel_person3.png"));
            ImageCollection.Add(new CarouselModel("carousel_person4.png"));
            ImageCollection.Add(new CarouselModel("carousel_person5.png"));
        }
        private List<CarouselModel> imageCollection = new List<CarouselModel>();
        public List<CarouselModel> ImageCollection
        {
            get { return imageCollection; }
            set { imageCollection = value; }
        }
    }
}
```

---

**Note:** Images can also be referred in PCL and from website URL as [instructed](#)

---

### *Binding the Data with Custom Template*

SfCarousel provides support to add a custom view as carousel items by designing a view inside its ItemTemplate. This template will be applied for all its items and its data will be binded.

#### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel/>
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
HeightRequest="400"
WidthRequest="800" />
</ContentPage.Content>
</ContentPage>
```

#### **C#**

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
CarouselViewModel carouselViewModel = new CarouselViewModel();
SfCarousel carousel = new SfCarousel()
{
HeightRequest = 400,
WidthRequest = 800
};
var itemTemplate = new DataTemplate(() =>
```

```

{
    var grid = new Grid();
    var nameLabel = new Image();
    nameLabel.SetBinding(Image.SourceProperty, "Image");
    grid.Children.Add(nameLabel);
    return grid;
});
carousel.ItemTemplate = itemTemplate;
carousel.ItemsSource = carouselViewModel.ImageCollection;
this.Content = carousel;
}
}
public class CarouselModel
{
    public CarouselModel(string imageString)
    {
        Image = imageString;
    }
    private string _image;
    public string Image
    {
        get { return _image; }
        set { _image = value; }
    }
}
public class CarouselViewModel
{
    public CarouselViewModel()
    {
        ImageCollection.Add(new CarouselModel("carousel_person1.png"));
        ImageCollection.Add(new CarouselModel("carousel_person2.png"));
        ImageCollection.Add(new CarouselModel("carousel_person3.png"));
        ImageCollection.Add(new CarouselModel("carousel_person4.png"));
        ImageCollection.Add(new CarouselModel("carousel_person5.png"));
    }
    private List<CarouselModel> imageCollection = new List<CarouselModel>();
    public List<CarouselModel> ImageCollection
    {
        get { return imageCollection; }
        set { imageCollection = value; }
    }
}
}

```

- Now set the **BindingContext** for the items collection in code behind.

### C#

```
carousel.BindingContext = new CarouselViewModel();
```

### Through Carousel Item

Different set of views can be provided to every items through **ItemContent** property available in **SfCarouselItem** class.

**C#**

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfCarousel carousel = new SfCarousel()
            {
                ItemWidth = 170,
                ItemHeight = 250
            };
            ObservableCollection<SfCarouselItem> carouselItems = new
            ObservableCollection<SfCarouselItem>();
            carouselItems.Add(new SfCarouselItem() { ItemContent = new Image() { Source
            = "carousel_person1.png", Aspect = Aspect.Fill } });
            carouselItems.Add(new SfCarouselItem() { ItemContent = new Image() { Source
            = "carousel_person2.png", Aspect = Aspect.Fill } });
            carouselItems.Add(new SfCarouselItem() { ItemContent = new Image() { Source
            = "carousel_person3.png", Aspect = Aspect.Fill } });
            carouselItems.Add(new SfCarouselItem() { ItemContent = new Image() { Source
            = "carousel_person4.png", Aspect = Aspect.Fill } });
            carouselItems.Add(new SfCarouselItem() { ItemContent = new Image() { Source
            = "carousel_person5.png", Aspect = Aspect.Fill } });
            carousel.ItemsSource = carouselItems;
            this.Content = carousel;
        }
    }
}
```

and also carousel provides a support to display only the Image data with **Image** property in **SfCarouselItem** class.

**C#**

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfCarousel carousel = new SfCarousel()
            {
                ItemWidth = 170,
                ItemHeight = 250
            };
        }
    }
}
```

```

ObservableCollection<SfCarouselItem> carouselItems = new
ObservableCollection<SfCarouselItem>();
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person1.png"
});
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person2.png"
});
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person3.png"
});
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person4.png"
});
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person5.png"
});
carousel.ItemsSource = carouselItems;
this.Content = carousel;
}
}
}

```

Similar way every item can be created and customized in case of different carousel item view is needed.

### LoadMore

Virtualization can be achieved by using the Load more concept. This support is used to handle the numerous items in the carousel control. A particular items are maintained in the view port based on the `LoadMoreItemsCount` property. The LoadMore view is added after the last item in the collection of carousel view. When tapping the LoadMore view, the next set of items in the collection can be added to the carousel.

The following properties are used to achieve this support:

`AllowLoadMore` `LoadMoreItemsCount` \* `LoadMoreView`

### AllowLoadMore

By enabling the `AllowLoadMore` property, the LoadMore support works in the carousel view.

**Note:** The default value of the `AllowLoadMore` property is false.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel />
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>
</DataTemplate>
</ResourceDictionary>

```

```
</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
ItemHeight="200"
ItemWidth="200"
ItemSpacing="2"
AllowLoadMore="True"
ViewMode="Linear">
</carousel:SfCarousel>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = new CarouselViewModel();
            SfCarousel carousel = new SfCarousel()
            {
                ItemHeight = 200,
                ItemWidth = 200,
                ItemSpacing = 2,
                AllowLoadMore = true,
                ViewMode = ViewMode.Linear
            };
            carousel.ItemTemplate = new DataTemplate(() =>
            {
                Image image = new Image();
                image.SetBinding(Image.SourceProperty, "Image");
                image.Aspect = Aspect.AspectFit;
                return image;
            });
            carousel.SetBinding(SfCarousel.ItemsSourceProperty, "ImageCollection");
            this.Content = carousel;
        }
    }

    public class CarouselModel
    {
        public CarouselModel(string imageString)
        {
            Image = imageString;
        }
        private string _image;
        public string Image
        {
            get { return _image; }
        }
    }
}
```



```

set { _image = value; }
}
}
public class CarouselViewModel
{
public CarouselViewModel()
{
ImageCollection.Add(new CarouselModel("carousel_person1.png"));
ImageCollection.Add(new CarouselModel("carousel_person2.png"));
ImageCollection.Add(new CarouselModel("carousel_person3.png"));
ImageCollection.Add(new CarouselModel("carousel_person4.png"));
ImageCollection.Add(new CarouselModel("carousel_person5.png"));
}
private List<CarouselModel> imageCollection = new List<CarouselModel>();
public List<CarouselModel> ImageCollection
{
get { return imageCollection; }
set { imageCollection = value; }
}
}
}

```

### LoadMoreItemsCount

Number of items can be maintained in the carousel control by using the **LoadMoreItemsCount** property. By using the **LoadMoreItemsCount** property, numerous items can be separated.

---

**Note:** The default value of the **LoadMoreItemsCount** property is 3.

---

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:LoadMore"
x:Class="LoadMore.MainPage"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
">
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel />
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>
</DataTemplate>
</ResourceDictionary>

```

```

</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
ItemHeight="200"
ItemWidth="200"
ItemSpacing="2"
AllowLoadMore="True"
ViewMode="Linear"
LoadMoreItemsCount="2">
</carousel:SfCarousel>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
this.BindingContext = new CarouselViewModel();
SfCarousel carousel = new SfCarousel()
{
ItemHeight = 200,
ItemWidth = 200,
ItemSpacing = 2,
AllowLoadMore = true,
LoadMoreItemsCount = 2,
ViewMode = ViewMode.Linear
};
carousel.ItemTemplate = new DataTemplate(() =>
{
Image image = new Image();
image.SetBinding(Image.SourceProperty, "Image");
image.Aspect = Aspect.AspectFit;
return image;
});
carousel.SetBinding(SfCarousel.ItemsSourceProperty, "ImageCollection");
this.Content = carousel;
}
}
public class CarouselModel
{
public CarouselModel(string imageString)
{
Image = imageString;
}
private string _image;
public string Image

```

```

{
    get { return _image; }
    set { _image = value; }
}
}

public class CarouselViewModel
{
    public CarouselViewModel()
    {
        ImageCollection.Add(new CarouselModel("carousel_person1.png"));
        ImageCollection.Add(new CarouselModel("carousel_person2.png"));
        ImageCollection.Add(new CarouselModel("carousel_person3.png"));
        ImageCollection.Add(new CarouselModel("carousel_person4.png"));
        ImageCollection.Add(new CarouselModel("carousel_person5.png"));
    }
    private List<CarouselModel> imageCollection = new List<CarouselModel>();
    public List<CarouselModel> ImageCollection
    {
        get { return imageCollection; }
        set { imageCollection = value; }
    }
}
}

```

### LoadMoreView

Custom view can be passed instead of the LoadMore label by using the **LoadMoreView** property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel />
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
ItemHeight="200"
ItemWidth="200"
ItemSpacing="2"
AllowLoadMore="True"

```

```

ViewMode="Linear"
LoadMoreItemsCount="2">
<carousel:SfCarousel.LoadMoreView>
<Grid BackgroundColor="#FFFFFFF">
<Label
Text="Load More..."
FontSize="14"
TextColor="#FF000000"
FontAttributes="Bold"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"
HorizontalOptions="Center"
VerticalOptions="Center"/>
</Grid>
</carousel:SfCarousel.LoadMoreView>
</carousel:SfCarousel>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

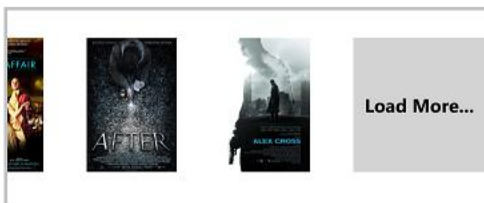
using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
this.BindingContext = new CarouselViewModel();
SfCarousel carousel = new SfCarousel()
{
ItemHeight = 200,
ItemWidth = 200,
ItemSpacing = 2,
AllowLoadMore = true,
LoadMoreItemsCount = 2,
ViewMode = ViewMode.Linear
};
carousel.ItemTemplate = new DataTemplate(() =>
{
Image image = new Image();
image.SetBinding(Image.SourceProperty, "Image");
image.Aspect = Aspect.AspectFit;
return image;
});
carousel.SetBinding(SfCarousel.ItemsSourceProperty, "ImageCollection");
Grid grid = new Grid()
{
BackgroundColor = Color.White
};
Label label = new Label()
{
Text = "Load More...",

```

```

FontSize = 14,
TextColor = Color.Black,
FontAttributes = FontAttributes.Bold,
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center,
HorizontalTextAlignment = TextAlignment.Center,
VerticalTextAlignment = TextAlignment.Center
};
grid.Children.Add(label);
carousel.LoadMoreView = grid;
this.Content = carousel;
}
}
public class CarouselModel
{
public CarouselModel(string imageString)
{
Image = imageString;
}
private string _image;
public string Image
{
get { return _image; }
set { _image = value; }
}
}
public class CarouselViewModel
{
public CarouselViewModel()
{
ImageCollection.Add(new CarouselModel("cart.png"));
ImageCollection.Add(new CarouselModel("cart.png"));
ImageCollection.Add(new CarouselModel("cart.png"));
ImageCollection.Add(new CarouselModel("cart.png"));
ImageCollection.Add(new CarouselModel("cart.png"));
}
private List<CarouselModel> imageCollection = new List<CarouselModel>();
public List<CarouselModel> ImageCollection
{
get { return imageCollection; }
set { imageCollection = value; }
}
}
}

```



You can find the complete Load More sample from this [link](#).

## UI Virtualization

In UI virtualization concept, only the number of items that can be adaptable to the viewport of our device are arranged. Even, if the numerous items have been added to the collection, it loads only the viewport adaptable count of the carousel Items. Items are added at the right of the view when swiping the countable items in forward direction. At the same time, same number of items are removed at the left of the view for maintaining the same viewport items count. Similarly, items are added at the left of the view when swiping in backward direction for maintaining the same viewport items count. At the time, the same number of items are removed at the right of the view. Using this mechanism, virtualization concept is achieved in the carousel control.

The following property has been used in UIVirtualization support:

- EnableVirtualization

### EnableVirtualization

UI Virtualization concept is achieved by enabling the EnableVirtualization property.

---

**Note:** The default value of the EnableVirtualization property is false.

---

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel />
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
ItemHeight="200"
ItemWidth="200"
ItemSpacing="2"
ViewMode="Linear"
EnableVirtualization="true">
</carousel:SfCarousel>
</ContentPage.Content>
</ContentPage>
```

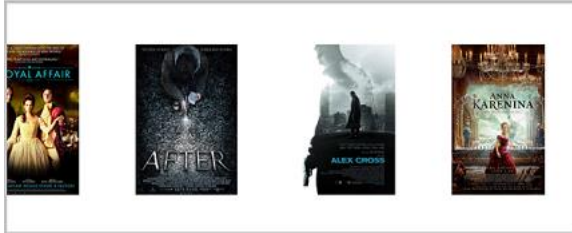
### C#

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = new CarouselViewModel();
            SfCarousel carousel = new SfCarousel()
            {
                ItemHeight = 200,
                ItemWidth = 200,
                ItemSpacing = 2,
                EnableVirtualization = true,
                ViewMode = ViewMode.Linear
            };
            carousel.ItemTemplate = new DataTemplate(() =>
            {
                Image image = new Image();
                image.SetBinding(Image.SourceProperty, "Image");
                image.Aspect = Aspect.AspectFit;
                return image;
            });
            carousel.SetBinding(SfCarousel.ItemsSourceProperty, "ImageCollection");
            this.Content = carousel;
        }
    }

    public class CarouselModel
    {
        public CarouselModel(string imageString)
        {
            Image = imageString;
        }
        private string _image;
        public string Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }

    public class CarouselViewModel
    {
        public CarouselViewModel()
        {
            ImageCollection.Add(new CarouselModel("carousel_person1.png"));
            ImageCollection.Add(new CarouselModel("carousel_person2.png"));
            ImageCollection.Add(new CarouselModel("carousel_person3.png"));
            ImageCollection.Add(new CarouselModel("carousel_person4.png"));
            ImageCollection.Add(new CarouselModel("carousel_person5.png"));
        }
        private List<CarouselModel> imageCollection = new List<CarouselModel>();
        public List<CarouselModel> ImageCollection
        {
            get { return imageCollection; }
        }
    }
}
```

```
set { imageCollection = value; }
}
}
```



You can find the complete UIVirtualization sample from this [link](#).

### Transformation

The Offset between selected and unselected item can be customized in SfCarousel control. And also the items can be scaled to the specified value.

### Tilt Non Selected Items

The **RotationAngle** property in the SfCarousel control is used to tilt all the unselected items in a specified angle.

**Note:** If the angle value is positive, then the rotation is in the clockwise direction. If the angle value is negative, the rotation is in the counterclockwise direction.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<carousel:SfCarousel x:Name="carousel"
ItemHeight="170"
ItemWidth="270"
RotationAngle="90"/>
</ContentPage>
```

### C#

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```



```

SfCarousel carousel = new SfCarousel()
{
    ItemWidth = 170,
    ItemHeight = 250,
    RotationAngle = 90
};
ObservableCollection<SfCarouselItem> carouselItems = new
ObservableCollection<SfCarouselItem>();
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person1.png"
});
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person2.png"
});
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person3.png"
});
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person4.png"
});
carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person5.png"
});
carousel.ItemsSource = carouselItems;
this.Content = carousel;
}
}
}

```



### Set Gap between Items

The **Offset** property is used to specify the accurate distance between unselected items in SfCarousel panel.

**Note:** The default value is 20.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<carousel:SfCarousel x:Name="carousel"
ItemHeight="170"
ItemWidth="270"
Offset="30"/>

```

```
</ContentPage>
```

## C#

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfCarousel carousel = new SfCarousel()
            {
                ItemWidth = 170,
                ItemHeight = 250,
                Offset = 30
            };
            ObservableCollection<SfCarouselItem> carouselItems = new
            ObservableCollection<SfCarouselItem>();
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person1.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person2.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person3.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person4.png"
            });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person5.png"
            });
            carousel.ItemsSource = carouselItems;
            this.Content = carousel;
        }
    }
}
```



## Set Gap between Selected and unselected Item

Distance between the selected item and other items can be customized by using `SelectedItemOffset` property of the SfCarousel control.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<carousel:SfCarousel x:Name="carousel"
ItemHeight="170"
ItemWidth="270"
SelectedItemOffset="5"/>
</ContentPage>
```

## C#

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfCarousel carousel = new SfCarousel()
            {
                ItemWidth = 170,
                ItemHeight = 250,
                SelectedItemOffset = 5
            };
            ObservableCollection<SfCarouselItem> carouselItems = new
            ObservableCollection<SfCarouselItem>();
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person1.png" });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person2.png" });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person3.png" });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person4.png" });
            carouselItems.Add(new SfCarouselItem() { ImageName = "carousel_person5.png" });
            carousel.ItemsSource = carouselItems;
            this.Content = carousel;
        }
    }
}
```

## Set Scaling for Carousel Items

The **ScaleOffset** property in the SfCarousel control is used to scale all the items to the specified scale value.

## XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel />
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
ItemHeight="200"
ItemWidth="200"
ScaleOffset="0.7"/>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

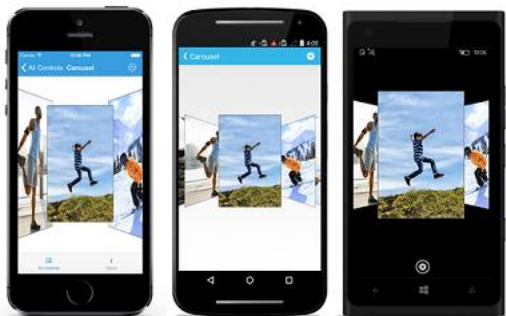
using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = new CarouselViewModel();
            SfCarousel carousel = new SfCarousel()
            {
                ItemHeight = 200,
                ItemWidth = 200,
                ScaleOffset = 0.7f
            };
            carousel.ItemTemplate = new DataTemplate(() =>
            {
                Image image = new Image();
                image.SetBinding(Image.SourceProperty, "Image");
                image.Aspect = Aspect.AspectFit;
                return image;
            });
            carousel.SetBinding(SfCarousel.ItemsSourceProperty, "ImageCollection");

```

```

this.Content = carousel;
}
}
public class CarouselModel
{
public CarouselModel(string imageString)
{
Image = imageString;
}
private string _image;
public string Image
{
get { return _image; }
set { _image = value; }
}
}
public class CarouselViewModel
{
public CarouselViewModel()
{
ImageCollection.Add(new CarouselModel("carousel_person1.png"));
ImageCollection.Add(new CarouselModel("carousel_person2.png"));
ImageCollection.Add(new CarouselModel("carousel_person3.png"));
ImageCollection.Add(new CarouselModel("carousel_person4.png"));
ImageCollection.Add(new CarouselModel("carousel_person5.png"));
}
private List<CarouselModel> imageCollection = new List<CarouselModel>();
public List<CarouselModel> ImageCollection
{
get { return imageCollection; }
set { imageCollection = value; }
}
}
}

```



### Spacing between the Items in Linear mode

Spacing of all the items in Linear mode can be determined by using `ItemSpacing` property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

```

```

xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel />
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
ItemHeight="200"
ItemWidth="200"
ItemSpacing="10"
ViewMode="Linear"/>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = new CarouselViewModel();
            SfCarousel carousel = new SfCarousel()
            {
                ItemHeight = 200,
                ItemWidth = 200,
                ItemSpacing = 10,
                ViewMode = ViewMode.Linear
            };
            carousel.ItemTemplate = new DataTemplate(() =>
            {
                Image image = new Image();
                image.SetBinding(Image.SourceProperty, "Image");
                image.Aspect = Aspect.AspectFit;
                return image;
            });
            carousel.SetBinding(SfCarousel.ItemsSourceProperty, "ImageCollection");
            this.Content = carousel;
        }
    }
}

```

```

}
}
public class CarouselModel
{
public CarouselModel(string imageString)
{
Image = imageString;
}
private string _image;
public string Image
{
get { return _image; }
set { _image = value; }
}
}
public class CarouselViewModel
{
public CarouselViewModel()
{
ImageCollection.Add(new CarouselModel("carousel_person1.png"));
ImageCollection.Add(new CarouselModel("carousel_person2.png"));
ImageCollection.Add(new CarouselModel("carousel_person3.png"));
ImageCollection.Add(new CarouselModel("carousel_person4.png"));
ImageCollection.Add(new CarouselModel("carousel_person5.png"));
}
private List<CarouselModel> imageCollection = new List<CarouselModel>();
public List<CarouselModel> ImageCollection
{
get { return imageCollection; }
set { imageCollection = value; }
}
}
}

```

## Animation

The **Duration** property of the SfCarousel control is used to specify the time taken to move an item to the selected item position. The duration is specified in seconds. The default value is 300 ms.

## XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel />
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>

```

```

</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
ItemHeight="200"
ItemWidth="200"
Duration="1000"/>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = new CarouselViewModel();
            SfCarousel carousel = new SfCarousel()
            {
                ItemHeight = 200,
                ItemWidth = 200,
                Duration = 1000
            };
            carousel.ItemTemplate = new DataTemplate(() =>
            {
                Image image = new Image();
                image.SetBinding(Image.SourceProperty, "Image");
                image.Aspect = Aspect.AspectFit;
                return image;
            });
            carousel.SetBinding(SfCarousel.ItemsSourceProperty, "ImageCollection");
            this.Content = carousel;
        }
    }
    public class CarouselModel
    {
        public CarouselModel(string imageString)
        {
            Image = imageString;
        }
        private string _image;
        public string Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }
}

```



```

public class CarouselViewModel
{
    public CarouselViewModel()
    {
        ImageCollection.Add(new CarouselModel("carousel_person1.png"));
        ImageCollection.Add(new CarouselModel("carousel_person2.png"));
        ImageCollection.Add(new CarouselModel("carousel_person3.png"));
        ImageCollection.Add(new CarouselModel("carousel_person4.png"));
        ImageCollection.Add(new CarouselModel("carousel_person5.png"));
    }
    private List<CarouselModel> imageCollection = new List<CarouselModel>();
    public List<CarouselModel> ImageCollection
    {
        get { return imageCollection; }
        set { imageCollection = value; }
    }
}

```

### How to perform an operation while changing the carouselItem?

We can perform operation while changing the carouselItem using **SelectionChanged** event.

SelectionChanged event returns the index and selected carouselItem

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:carousel="clr-
namespace:Syncfusion.SfCarousel.XForms;assembly=Syncfusion.SfCarousel.XForms
"
xmlns:local="clr-namespace:CarouselSample"
x:Class="CarouselSample.MainPage">
<ContentPage.BindingContext>
<local:CarouselViewModel />
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="itemTemplate">
<Image Source="{Binding Image}"
Aspect="AspectFit"/>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<carousel:SfCarousel x:Name="carousel"
ItemTemplate="{StaticResource itemTemplate}"
ItemsSource="{Binding ImageCollection}"
ItemHeight="200"
ItemWidth="200"
SelectionChanged="Carousel_SelectionChanged"/>
</ContentPage.Content>
</ContentPage>

```

#### C#

```
using Syncfusion.SfCarousel.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
namespace CarouselSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = new CarouselViewModel();
            SfCarousel carousel = new SfCarousel()
            {
                ItemHeight = 200,
                ItemWidth = 200
            };
            carousel.ItemTemplate = new DataTemplate(() =>
            {
                Image image = new Image();
                image.SetBinding(Image.SourceProperty, "Image");
                image.Aspect = Aspect.AspectFit;
                return image;
            });
            carousel.SelectionChanged += Carousel_SelectionChanged;
            carousel.SetBinding(SfCarousel.ItemsSourceProperty, "ImageCollection");
            this.Content = carousel;
        }
        private void Carousel_SelectionChanged(object sender,
            SelectionChangedEventArgs e)
        {
            int count = (sender as SfCarousel).SelectedIndex + 1;
            DisplayAlert("SelectionChanged", "Carousel item:" + count + " has Selected",
                "Ok");
        }
        public class CarouselModel
        {
            public CarouselModel(string imageString)
            {
                Image = imageString;
            }
            private string _image;
            public string Image
            {
                get { return _image; }
                set { _image = value; }
            }
        }
        public class CarouselViewModel
        {
            public CarouselViewModel()
            {
                ImageCollection.Add(new CarouselModel("carousel_person1.png"));
                ImageCollection.Add(new CarouselModel("carousel_person2.png"));
                ImageCollection.Add(new CarouselModel("carousel_person3.png"));
                ImageCollection.Add(new CarouselModel("carousel_person4.png"));
                ImageCollection.Add(new CarouselModel("carousel_person5.png"));
            }
        }
    }
}
```

```
}  
private List<CarouselModel> imageCollection = new List<CarouselModel>();  
public List<CarouselModel> ImageCollection  
{  
    get { return imageCollection; }  
    set { imageCollection = value; }  
}  
}  
}
```

### AutomationId

The SfCarousel control has built-in **AutomationId** for inner elements. The **AutomationId** API allows the automation framework to find and interact with the inner elements of the SfCarousel control. To keep unique AutomationId, these inner elements' AutomationIds are updated based on the control's **AutomationId**.

For example, if you set SfCarouselItem's **AutomationId** as "Person", then the automation framework will interact the fourth carousel item as "Person SfCarouselItem 4 of 6" (6 denotes the total count).

The following screenshot illustrates the AutomationIds of inner elements. If the SfCarousel's **AutomationId** as Gallery, then the Automation framework will interact the LoadMore as "Gallery LoadMore. Tap to load more items". You cannot interact with the carousel item when you want to select an index that is not visible in the view.

---

**Note:** You cannot provide AutomationId when the carousel item is populated with custom template.

---

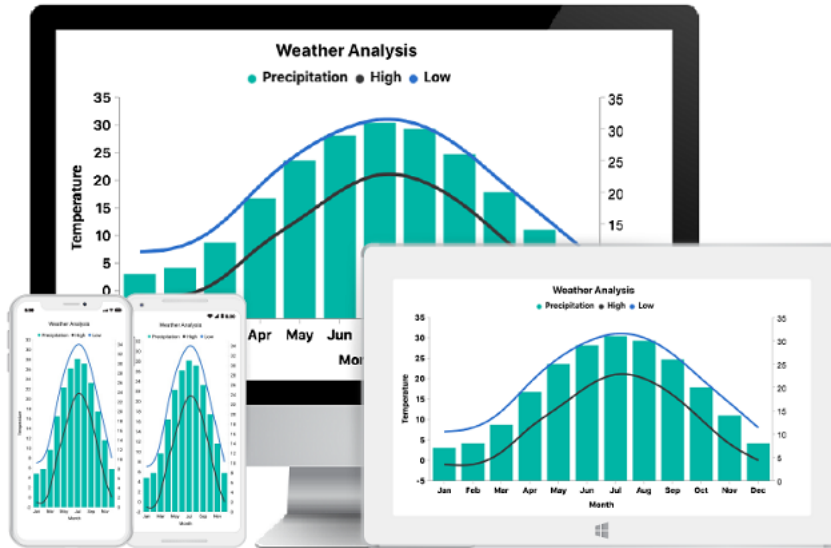


Person SfCarouselItem 4 of 6

## SfChart

### Overview

Essential Chart for Xamarin.Forms provide a perfect way to visualize data with a high level of user interactivity that focuses on development, productivity and simplicity of use. Essential Chart also provides a wide variety of charting features that are used to visualize large quantities of data, flexible data binding and user customization.



### Key features

- Chart supports more than 25 different types of series, ranging from simple bar series to complex financial charts. Each type of chart represents a unique style of representing data that is more user friendly and has greater UI visualization.
- Data can be plotted against multiple scales that helps to visualize the mixed types of data in a single chart.
- Chart provides support to render multiple series at the same time, with options to compare and visualize two different chart series, simultaneously.
- User friendly customization support. [SfChart](#) provides various options for you to customize chart features, axis, labels, legends, series, etc., and visualize them accordingly.

### Getting Started

This section explains you the steps required to populate the Chart with data, title, add data labels and tooltips to the Chart. This section covers only the minimal features that you need to know to get started with the Chart.

---

**Note:** Watch the video: [Xamarin.Forms SfChart - Getting Started \(YouTube\)](#)

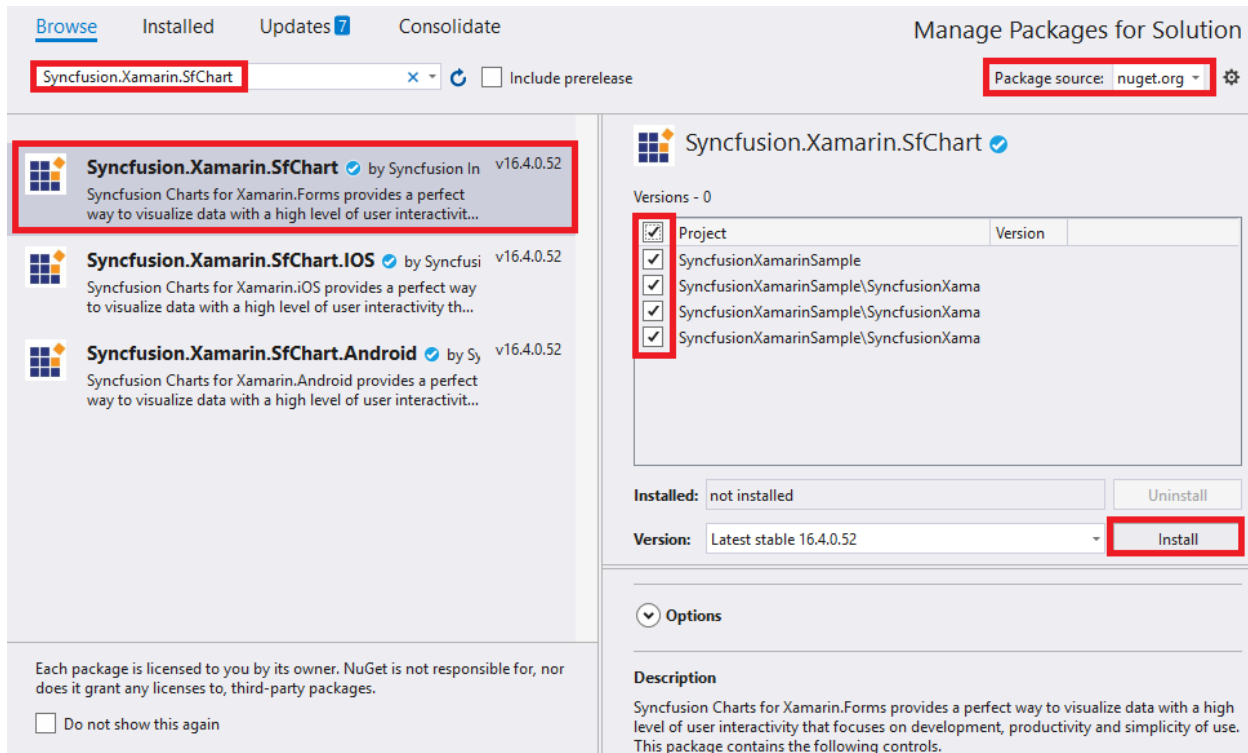
---

#### Adding SfChart reference

You can add SfChart reference using one of the following methods:

##### Method 1: Adding SfChart reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](#). To add chart to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfChart](#), and then install it.

**Note:**

- Install the same version of the chart NuGet in all the projects.
- In addition, you need to install the [Syncfusion.Xamarin.SfChart.WPF](#) package for Xamarin.Forms WPF platform only.

**Method 2: Adding SfChart reference from toolbox**

Syncfusion provides Xamarin Toolbox. Using this toolbox, you can drag the SfChart control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

**Method 3: Adding SfChart assemblies manually from the installed location**

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location : {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfChart.XForms.Android.dll Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll

iOS	Syncfusion.SfChart.XForms.iOS.dll Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
macOS	Syncfusion.SfChart.XForms.macOS.dll Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.macOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfChart.UWP.dll Syncfusion.SfChart.XForms.UWP.dll Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
WPF	Syncfusion.SfChart.WPF.dll Syncfusion.SfChart.XForms.WPF.dll Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.WPF.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

#### Launching the application on each platform with chart

To use the chart inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

##### iOS

To launch the chart in iOS, call the `SfChartRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework initialization and before the `LoadApplication` method is called as demonstrated in the following code sample:

**Note:** If you are adding the references from toolbox, this step is not needed.

##### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    ...
    global::Xamarin.Forms.Forms.Init();
}
```

```
Syncfusion.SfChart.XForms.iOS.Renderers.SfChartRenderer.Init();  
LoadApplication(new App());  
...  
}
```

### macOS

To launch the chart in macOS, call the `SfChartRenderer.Init()` method in the `DidFinishLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework initialization and before the `LoadApplication` method is called as demonstrated in the following code sample:

### C#

```
public override void DidFinishLaunching(NSNotification notification)  
{  
    ...  
    Forms.Init();  
    Syncfusion.SfChart.XForms.MacOS.SfChartRenderer.Init();  
    LoadApplication(new App());  
    ...  
}
```

### Windows Presentation Foundation (WPF)

To launch the chart in WPF, call the `SfChartRenderer.Init()` method in the `MainWindow` constructor of the `MainWindow` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

### C#

```
public partial class MainWindow : FormsApplicationPage  
{  
    public MainWindow()  
    {  
        InitializeComponent();  
        Forms.Init();  
        Syncfusion.SfChart.XForms.WPF.SfChartRenderer.Init();  
        LoadApplication(new App());  
    }  
}
```

### Universal Windows Platform (UWP)

To deploy the chart in **Release** mode, you need to initialize the chart assemblies in `App.xaml.cs` in UWP project as shown in the below code snippets.

### C#

```
// In App.xaml.cs  
protected override void OnLaunched(LaunchActivatedEventArgs e)  
{  
    ...  
    if (rootFrame == null)  
    {  
        List<Assembly> assembliesToInclude = new List<Assembly>();  
        assembliesToInclude.Add(typeof(Syncfusion.SfChart.XForms.UWP.SfChartRenderer)  
            .GetTypeInfo().Assembly);  
    }  
}
```



```
Xamarin.Forms.Forms.Init(e, assembliesToInclude);  
}  
...  
}
```

### Android

The Android platform does not require any additional configuration to render the chart.

### Initialize Chart

Import the [SfChart](#) namespace as shown below in your respective Page,

#### XML

```
xmlns:chart="clr-  
namespace:Syncfusion.SfChart.XForms;assembly=Syncfusion.SfChart.XForms"
```

#### C#

```
using Syncfusion.SfChart.XForms;
```

Then initialize an empty chart with [PrimaryAxis](#) and [SecondaryAxis](#) as shown below,

#### XML

```
<chart:SfChart>  
  <chart:SfChart.PrimaryAxis>  
    <chart:CategoryAxis>  
  </chart:CategoryAxis>  
  </chart:SfChart.PrimaryAxis>  
  <chart:SfChart.SecondaryAxis>  
    <chart:NumericalAxis>  
  </chart:NumericalAxis>  
  </chart:SfChart.SecondaryAxis>  
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart();  
//Initializing Primary Axis  
CategoryAxis primaryAxis = new CategoryAxis();  
chart.PrimaryAxis = primaryAxis;  
//Initializing Secondary Axis  
NumericalAxis secondaryAxis = new NumericalAxis ();  
chart.SecondaryAxis = secondaryAxis;  
this.Content = chart;
```

Run the project and check if you get following output to make sure you have configured your project properly to add [SfChart](#).



### Initialize view model

Now, let us define a simple data model that represents a data point in [SfChart](#).

#### C#

```
public class Person
{
    public string Name { get; set; }
    public double Height { get; set; }
}
```

Next, create a view model class and initialize a list of `Person` objects as shown below,

#### C#

```
public class ViewModel
{
    public List<Person> Data { get; set; }
    public ViewModel()
    {
        Data = new List<Person>()
        {
            new Person { Name = "David", Height = 180 },
            new Person { Name = "Michael", Height = 170 },
            new Person { Name = "Steve", Height = 160 },
            new Person { Name = "Joel", Height = 182 }
        };
    }
}
```

Set the `ViewModel` instance as the `BindingContext` of your Page; this is done to bind properties of `ViewModel` to [SfChart](#).

**Note:** Add namespace of `ViewModel` class in your XAML page if you prefer to set `BindingContext` in XAML.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="ChartDemo.MainPage"
xmlns:chart="clr-
namespace:Syncfusion.SfChart.XForms;assembly=Syncfusion.SfChart.XForms"
xmlns:local="clr-namespace:ChartDemo">
<ContentPage.BindingContext>
<local:ViewModel></local:ViewModel>
</ContentPage.BindingContext>
</ContentPage>
```

### C#

```
this.BindingContext = new ViewModel();
```

### Populate Chart with data

As we are going to visualize the comparison of heights in the data model, add [ColumnSeries](#) to [SfChart.Series](#) property, and then bind the Data property of the above `ViewModel` to the [ColumnSeries.ItemsSource](#) property as shown below.

**Note:** You need to set [XBindingPath](#) and [YBindingPath](#) properties, so that [SfChart](#) would fetch values from the respective properties in the data model to plot the series.

### XML

```
<chart:SfChart>
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis>
<chart:CategoryAxis.Title>
<chart:ChartAxisTitle Text="Name"> </chart:ChartAxisTitle>
</chart:CategoryAxis.Title>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis>
<chart:NumericalAxis.Title>
<chart:ChartAxisTitle Text="Height (in cm)"></chart:ChartAxisTitle>
</chart:NumericalAxis.Title>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
<chart:SfChart.Series>
<chart:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Name"
YBindingPath="Height">
</chart:ColumnSeries>
</chart:SfChart.Series>
</chart:SfChart>
```

## C#

```
//Initializing primary axis
CategoryAxis primaryAxis = new CategoryAxis();
primaryAxis.Title.Text = "Name";
chart.PrimaryAxis = primaryAxis;
//Initializing secondary Axis
NumericalAxis secondaryAxis = new NumericalAxis();
secondaryAxis.Title.Text = "Height (in cm)";
chart.SecondaryAxis = secondaryAxis;
//Initializing column series
ColumnSeries series = new ColumnSeries();
series.SetBinding(ChartSeries.ItemsSourceProperty, "Data");
series.XBindingPath = "Name";
series.YBindingPath = "Height";
chart.Series.Add(series);
```

### Add Title

You can add title to chart to provide quick information to the user about the data being plotted in the chart. You can set title using [SfChart.Title](#) property as shown below.

## XML

```
<chart:SfChart>
...
<chart:SfChart.Title>
<chart:ChartTitle Text="Chart"/>
</chart:SfChart.Title>
...
</chart:SfChart>
```

## C#

```
chart.Title.Text = "Chart";
```

Refer this [link](#) to learn more about the options available in [SfChart](#) to customize chart title.

### Enable data labels

You can add data labels to improve the readability of the chart. This can be achieved using [ChartSeries.DataMarkers](#) property as shown below.

## XML

```
<chart:SfChart>
...
<chart:ColumnSeries>
<chart:ColumnSeries.DataMarker>
<chart:ChartDataMarker/>
</chart:ColumnSeries.DataMarker>
</chart:ColumnSeries>
...
</chart:SfChart>
```

## C#

```
series.DataMarker = new ChartDataMarker();
```

Refer this [link](#) to learn more about the options available in [SfChart](#) to customize data markers.

#### Enable legend

You can enable legend using [SfChart.Legend](#) property as shown below,

#### XML

```
<chart:SfChart>
...
<chart:SfChart.Legend>
<chart:ChartLegend/>
</chart:SfChart.Legend>
...
</chart:SfChart>
```

#### C#

```
chart.Legend = new ChartLegend ();
```

Additionally, you need to set label for each series using [ChartSeries.Label](#) property, which will be displayed in corresponding legend.

#### XML

```
<chart:SfChart>
...
<chart:SfChart.Series>
<chart:ColumnSeries Label="Heights" ItemsSource="{Binding Data}"
XBindingPath="Name" YBindingPath="Height">
</chart:ColumnSeries>
</chart:SfChart.Series>
...
</chart:SfChart>
```

#### C#

```
series.Label = "Heights";
```

Refer this [link](#) to learn more about the options available in [SfChart](#) to customize legend.

#### Enable tooltip

Tooltips are used to show information about the segment, when you tap on the segment. You can enable tooltip by setting [ChartSeries.EnableTooltip](#) property to true.

#### XML

```
<chart:SfChart>
...
<chart:SfChart.Series>
<chart:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Name"
YBindingPath="Height" EnableTooltip ="True">
</chart:ColumnSeries>
```

```

</chart:SfChart.Series>
...
</chart:SfChart>

```

## C#

```
series.EnableTooltip = true;
```

Refer this [link](#) to learn more about the options available in [SfChart](#) to customize tooltip.

The following code example gives you the complete code of above configurations.

## XML

```

<ContentPage xmlns:chart="clr-
namespace:Syncfusion.SfChart.XForms;assembly=Syncfusion.SfChart.XForms"
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:local="clr-namespace:
ChartGettingStarted;assembly=ChartGettingStarted"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="ChartGettingStarted.ChartSample">
  <chart:SfChart x:Name="Chart" HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
    <chart:SfChart.BindingContext>
      <local:ViewModel/>
    </chart:SfChart.BindingContext>
    <chart:SfChart.Legend>
      <chart:ChartLegend />
    </chart:SfChart.Legend>
    <chart:SfChart.Title>
      <chart:ChartTitle Text="Chart"/>
    </chart:SfChart.Title>
    <chart:SfChart.PrimaryAxis>
      <chart:CategoryAxis>
        <chart:CategoryAxis.Title>
          <chart:ChartAxisTitle Text="Name"/>
        </chart:CategoryAxis.Title>
      </chart:CategoryAxis>
    </chart:SfChart.PrimaryAxis>
    <chart:SfChart.SecondaryAxis>
      <chart:NumericalAxis>
        <chart:NumericalAxis.Title>
          <chart:ChartAxisTitle Text="Height (in cm)"/>
        </chart:NumericalAxis.Title>
      </chart:NumericalAxis>
    </chart:SfChart.SecondaryAxis>
    <chart:SfChart.Series>
      <chart:ColumnSeries ItemsSource="{Binding Data}" Label="Heights"
XBindingPath="Name" YBindingPath="Height" EnableTooltip="True">
        <chart:ColumnSeries.DataMarker>
          <chart:ChartDataMarker/>
        </chart:ColumnSeries.DataMarker>
      </chart:ColumnSeries>
    </chart:SfChart.Series>
  </chart:SfChart>
</ContentPage>

```

**C#**

```
using Syncfusion.SfChart.XForms;
namespace ChartGettingStarted
{
    public partial class ChartSample : ContentPage
    {
        public ChartSample()
        {
            InitializeComponent();
            SfChart chart = new SfChart();
            chart.Title.Text = "Chart";
            //Initializing primary axis
            CategoryAxis primaryAxis = new CategoryAxis();
            primaryAxis.Title.Text = "Name";
            chart.PrimaryAxis = primaryAxis;
            //Initializing secondary Axis
            NumericalAxis secondaryAxis = new NumericalAxis();
            secondaryAxis.Title.Text = "Height (in cm)";
            chart.SecondaryAxis = secondaryAxis;
            //Initializing column series
            ColumnSeries series = new ColumnSeries();
            series.ItemsSource = viewModel.Data;
            series.XBindingPath = "Name";
            series.YBindingPath = "Height";
            series.Label = "Heights";
            series.DataMarker = new ChartDataMarker();
            series.EnableTooltip = true;
            chart.Legend = new ChartLegend();
            chart.Series.Add(series);
            this.Content = chart;
        }
    }
}
```

The following chart is created as a result of the above codes.



You can find the complete getting started sample from this [link](#).

## Populating Data

**SfChart** control can be configured with data points using [ItemsSource](#) property of [ChartSeries](#). There are two ways, you can create data points for chart.

### Chart Data Point

One way is to create a collection of [ChartDataPoint](#) objects and assign this collection to [ItemsSource](#) property. Here, each [ChartDataPoint](#) object represents a data point in a chart series.

**Note:** [ChartDataPoint](#) class has few overloaded constructors depending on the number of y-values required to plot a data point for a particular series type. For example, you can use a constructor with two parameters to instantiate data point for [XYDataSeries](#) like Line, Spline, and Pie etc.

Following code snippet illustrates this,

### C#

```
public class DataModel
{
    public ObservableCollection<ChartDataPoint> HighTemperature{ get; set; }
    public DataModel ()
    {
        HighTemperature = new ObservableCollection<ChartDataPoint> ();
        HighTemperature.Add (new ChartDataPoint ("Jan", 42));
        HighTemperature.Add (new ChartDataPoint ("Feb", 44));
        HighTemperature.Add (new ChartDataPoint ("Mar", 53));
        HighTemperature.Add (new ChartDataPoint ("Apr", 64));
        HighTemperature.Add (new ChartDataPoint ("May", 75));
        HighTemperature.Add (new ChartDataPoint ("Jun", 83));
        HighTemperature.Add (new ChartDataPoint ("Jul", 87));
        HighTemperature.Add (new ChartDataPoint ("Aug", 84));
        HighTemperature.Add (new ChartDataPoint ("Sep", 78));
        HighTemperature.Add (new ChartDataPoint ("Oct", 67));
    }
}
```



```
HighTemperature.Add (new ChartDataPoint ("Nov", 55));
HighTemperature.Add (new ChartDataPoint ("Dec", 45));
}
```

**XML**

```
<chart:SfChart>
<chart:SfChart.Series>
<chart:ColumnSeries ItemsSource="{Binding HighTemperature}"/>
</chart:SfChart.Series>
</chart:SfChart>
```

**C#**

```
//Adding the series to the chart and set the ItemsSource property
chart.Series.Add (new ColumnSeries () {
ItemsSource = dataModel.HighTemperature
});
```

## Custom Object

Another way is to assign a collection of custom objects to the [ItemsSource](#) property. In this case, you need to set the [XBindingPath](#) and [YBindingPath](#) properties of chart series with the property names of the custom object which contains the x-value/category and y-value respectively.

**Note:** While using custom objects, XBindingPath property is required for all types of chart series. You need to set [YBindingPath](#) property only for the [XYDataSeries](#) types which needs single y-value for a data point. For example, Line, Spline, Column, Bar, Pie etc. For [BubbleSeries](#) type, you need to set both YBindingPath and [Size](#) properties since it requires two y-values to plot a single bubble data point. In the case of financial series types like Candle and HiLoOpenClose, which requires four y-values for a single data point, you need to set [High](#), [Low](#), [Open](#) and [Close](#) properties with the property names of a custom object which contains respective values.

**C#**

```
[C#]
public class MonthDemand
{
    public MonthDemand(string demand, double year2010, double year2011)
    {
        this.Demand = demand;
        this.Year2010 = year2010;
        this.Year2011 = year2011;
    }
    public string Demand { get; set; }
    public double Year2010 { get; set; }
    public double Year2011 { get; set; }
}
public class DataModel
{
    public ObservableCollection<MonthDemand> Demands{ get; set; }
    public DataModel ()
    {
        Demands = new ObservableCollection<MonthDemand>();
    }
}
```

```

Demands.Add(new MonthDemand("Jan", 42, 27));
Demands.Add(new MonthDemand("Feb", 44, 28));
Demands.Add(new MonthDemand("Mar", 53, 35));
Demands.Add(new MonthDemand("Apr", 64, 44));
Demands.Add(new MonthDemand("May", 75, 54));
Demands.Add(new MonthDemand("Jun", 83, 63));
Demands.Add(new MonthDemand("Jul", 87, 68));
Demands.Add(new MonthDemand("Aug", 84, 66));
Demands.Add(new MonthDemand("Sep", 78, 59));
Demands.Add(new MonthDemand("Oct", 67, 48));
Demands.Add(new MonthDemand("Nov", 55, 38));
Demands.Add(new MonthDemand("Dec", 45, 29));
}
}

```

**XML**

```

<chart:SfChart>
...
<chart:SfChart.Series>
<chart:ColumnSeries ItemsSource="{Binding Demands}" XBindingPath="Demand"
YBindingPath="Year2010"/>
</chart:SfChart.Series>
</chart:SfChart>

```

**C#**

```

chart.Series.Add(new ColumnSeries() {
    ItemsSource = dataModel.Demands,
    XBindingPath = "Demand",
    YBindingPath = "Year2010"
});

```

**Title**

You can define and customize the Chart title using [Title](#) property of [SfChart](#). The [Text](#) property of [ChartTitle](#) is used to set the text for the title.

Following properties are used to customize its appearance.

- [TextColor](#) – used to change the color of the text.
- [BackgroundColor](#) – used to change the background color.
- [BorderColor](#) – used to change the border color.
- [BorderWidth](#) – used to change the border width.
- [Font](#) – used to change the text size, font family, and font weight. (This is deprecated API. Use [FontSize](#), [FontFamily](#), and [FontAttributes](#) properties instead of this.)
- [FontFamily](#) - used to change the font family for chart title.
- [FontAttributes](#) – used to change the font style for the chart title.
- [FontSize](#) - used to change the font size for the chart title.
- [Margin](#) - used to change the margin for title.

**XML**

```
<chart:SfChart>
<chart:SfChart.Title>
<chart:ChartTitle Text="Efficiency of oil-fired power production"
TextColor="Blue"/>
</chart:SfChart.Title>
</chart:SfChart>
```

## C#

```
SfChart sfChart = new SfChart();
sfChart.Title.Text = "Efficiency of oil-fired power production";
sfChart.Title.TextColor = Color.Blue;
```



## Text Alignment

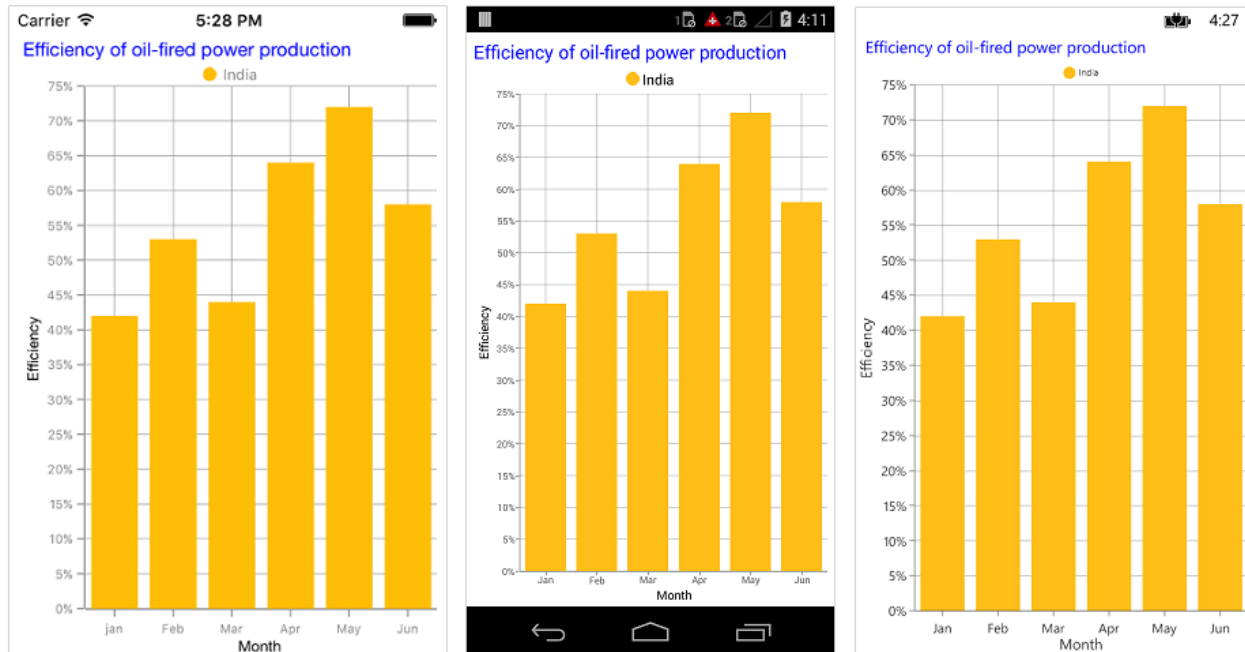
You can align the title text content to the Start, Center or End of the title using the [TextAlignment](#) property of the [ChartTitle](#).

## XML

```
<chart:SfChart.Title>
<chart:ChartTitle Text="Efficiency of oil-fired power production"
TextAlignment="Start" TextColor="Blue"/>
</chart:SfChart.Title>
```

## C#

```
sfChart.Title.Text = "Efficiency of oil-fired power production";
sfChart.Title.TextAlignment = TextAlignment.Start;
sfChart.Title.TextColor = Color.Blue;
```



## Axis

Charts typically have two axes that are used to measure and categorize data: a vertical (Y) axis, and a horizontal (X) axis.

Vertical(Y) axis always uses numerical scale. Horizontal(X) axis supports the following types of scale:

- Category
- Numeric
- Date time
- Logarithmic Axis

## Category Axis

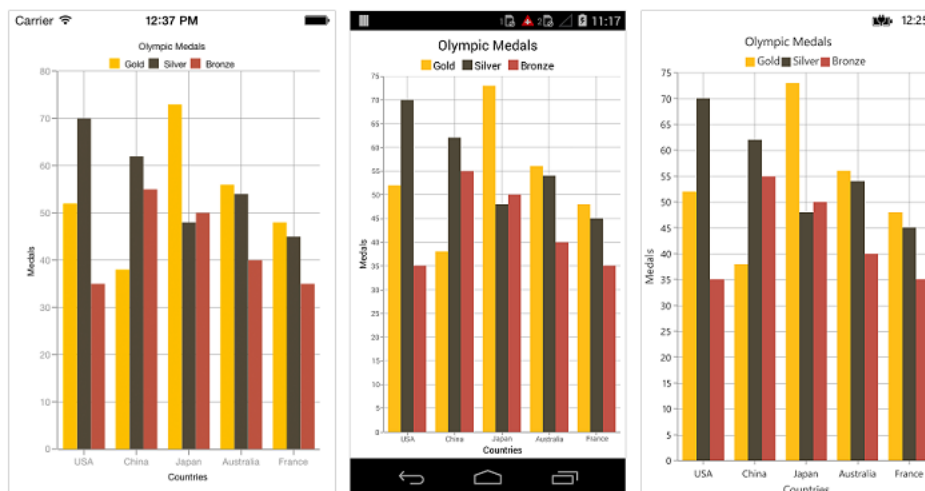
Category axis displays text labels instead of numbers.

## XML

```
<chart:SfChart.PrimaryAxis >
<chart:CategoryAxis />
</chart:SfChart.PrimaryAxis>
```

## C#

```
chart.PrimaryAxis = new CategoryAxis();
```



### Placing labels between ticks

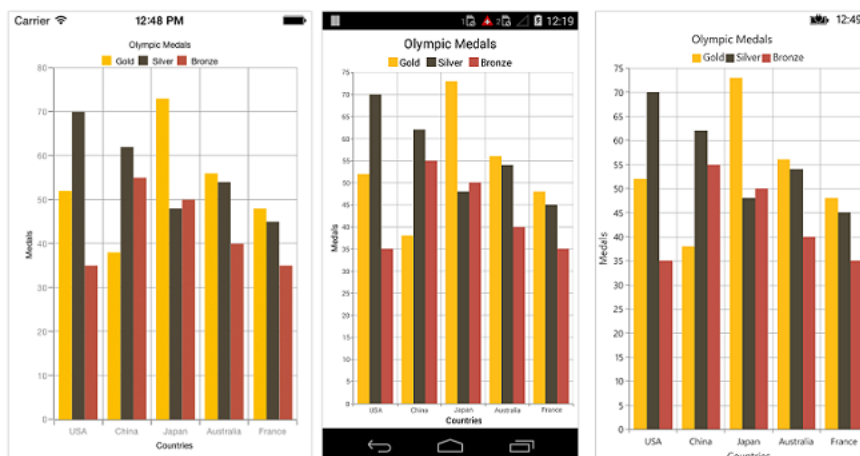
Labels in category axis can be placed between the ticks by setting [LabelPlacement](#) to [BetweenTicks](#). Default value of [LabelPlacement](#) property is [OnTicks](#) i.e. labels will be placed on the ticks by default.

### XML

```
<chart:SfChart.PrimaryAxis >
<chart:CategoryAxis LabelPlacement="BetweenTicks"/>
</chart:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis() { LabelPlacement =
LabelPlacement.BetweenTicks };
```



### Displaying labels after a fixed interval

To display labels after a fixed interval n, you can set [Interval](#) property of [ChartAxis](#) as n. Default value of interval is 1 i.e. all the labels will be displayed by default.

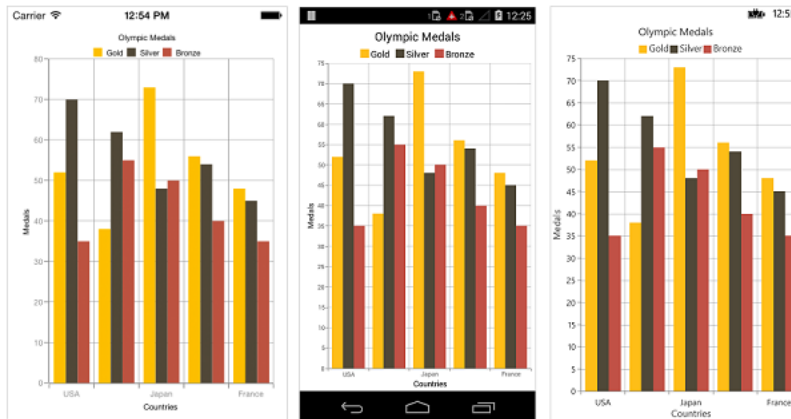
### XML

```
<chart:SfChart.PrimaryAxis >
<chart:CategoryAxis Interval="2" LabelPlacement="BetweenTicks"/>
```

```
</chart:SfChart.PrimaryAxis>
```

## C#

```
chart.PrimaryAxis = new CategoryAxis() { Interval = 2, LabelPlacement =
LabelPlacement.BetweenTicks };
```



### Indexed category axis

Category axis can also be rendered based on the index values of data source by setting the `[ArrangeByIndex]()` property to true in the axis.

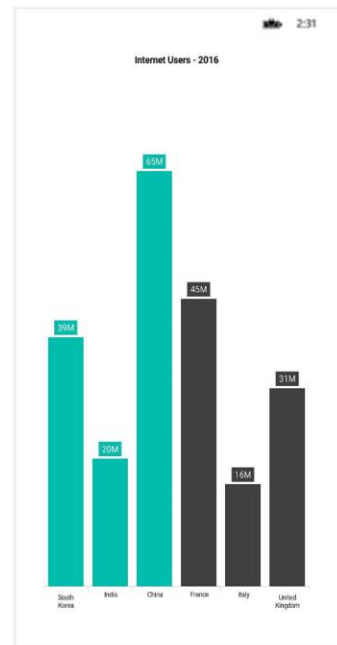
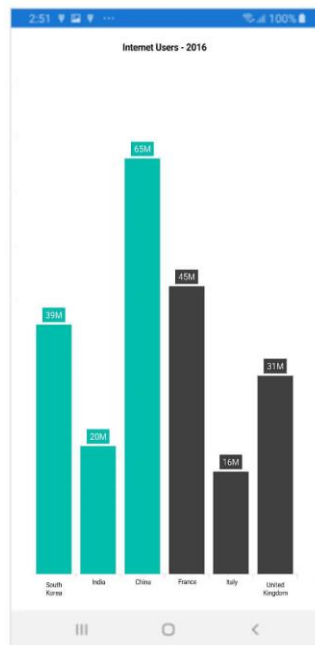
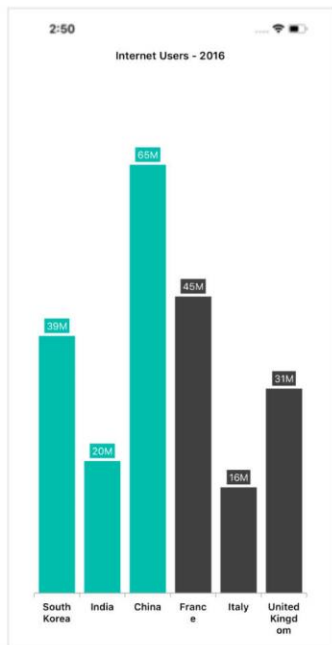
## XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis ArrangeByIndex="False" />
</chart:SfChart.PrimaryAxis>
<chart:SfChart.Series>
<chart:ColumnSeries ItemsSource="{Binding Data1}" XBindingPath="Country"
YBindingPath="Year2016"/>
<chart:ColumnSeries ItemsSource="{Binding Data2}" XBindingPath="Country"
YBindingPath="Year2016"/>
</chart:SfChart.Series>
```

## C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    ArrangeByIndex = false
};
ColumnSeries series1 = new ColumnSeries()
{
    ItemsSource = viewmodel.Data1,
    XBindingPath = "Country",
    YBindingPath = "Year2016"
};
ColumnSeries series2 = new ColumnSeries()
{
    ItemsSource = viewmodel.Data2,
    XBindingPath = "Country",
    YBindingPath = "Year2016",
```

```
};
chart.Series.Add(series1);
chart.Series.Add(series2);
```



## Numeric Axis

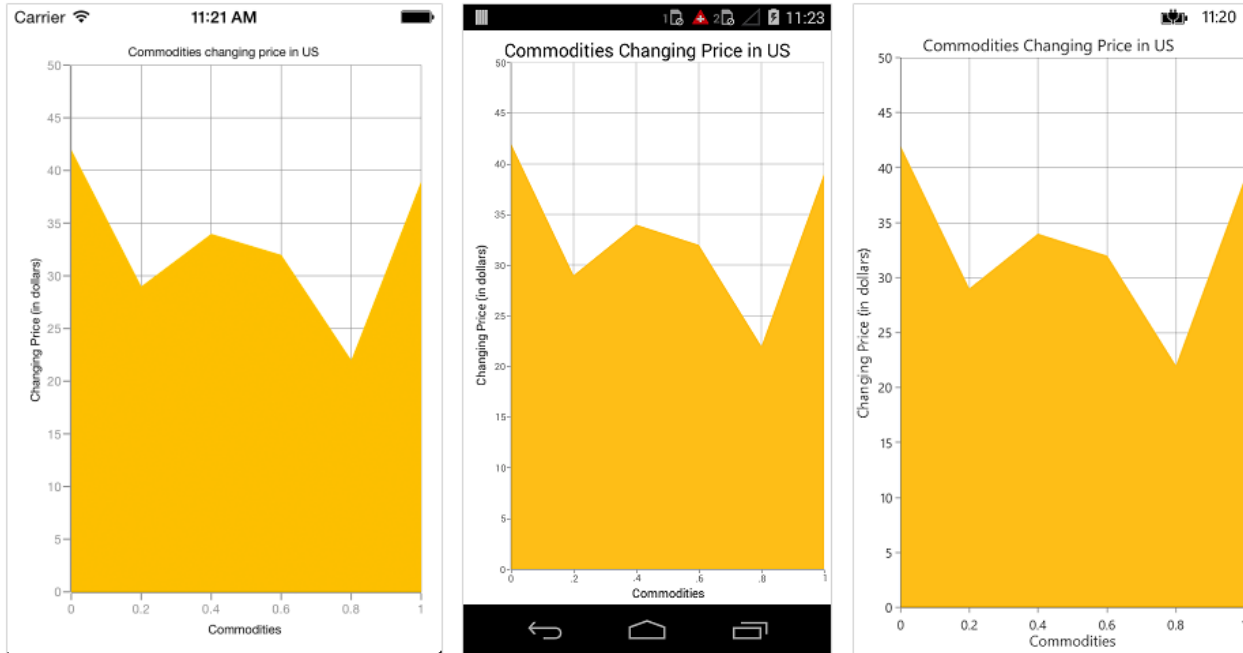
Numeric axis uses numerical scale and displays numbers as labels.

### XML

```
<chart:SfChart.PrimaryAxis >
<chart:NumericalAxis/>
</chart:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis();
```



### Customizing numeric range

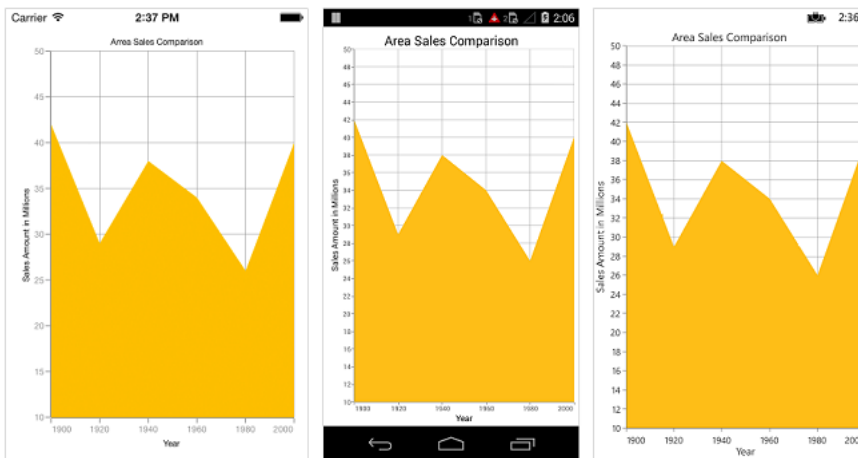
To customize the range of an axis, you can use the [Minimum](#) and [Maximum](#) properties of [NumericalAxis](#). By default, nice range will be calculated automatically based on the provided data.

### XML

```
<chart:SfChart.SecondaryAxis >
<chart:NumericalAxis Minimum="10" Maximum="50"/>
</chart:SfChart.SecondaryAxis >
```

### C#

```
chart.SecondaryAxis = new NumericalAxis() { Minimum = 10, Maximum = 50 };
```



### Customizing numeric interval

Axis interval can be customized using the [Interval](#) property of [ChartAxis](#). By default, nice interval will be calculated based on the minimum and maximum value of the provided data.



**XML**

```
<chart:SfChart.SecondaryAxis >
<chart:NumericalAxis Interval="10"/>
</chart:SfChart.SecondaryAxis >
```

**C#**

```
chart.SecondaryAxis = new NumericalAxis() { Interval = 10 };
```

*Apply padding to the range*

Padding can be applied to the minimum and maximum extremes of the axis range by using [RangePadding](#) property. Numeric axis supports the following types of padding.

- None
- Round
- Additional
- Normal

**None**

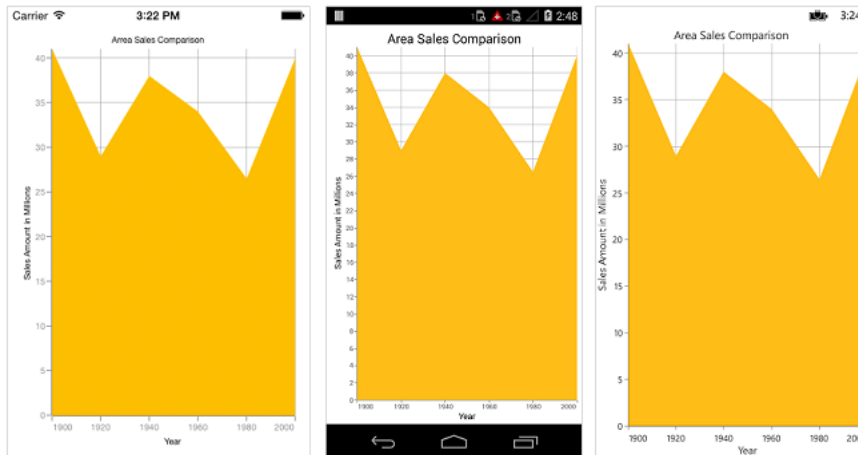
When the value of [RangePadding](#) property is [None](#), padding will not be applied to the axis. This is also the default value of [RangePadding](#) for horizontal axis.

**XML**

```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis RangePadding="None"/>
</chart:SfChart.SecondaryAxis>
```

**C#**

```
chart.SecondaryAxis = new NumericalAxis() { RangePadding =
NumericalPadding.None };
```



### Round

When the value of [RangePadding](#) property is [Round](#), axis range will be rounded to the nearest possible value divided by the interval.

### XML

```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis RangePadding="Round"/>
</chart:SfChart.SecondaryAxis>
```

### C#

```
chart.SecondaryAxis = new NumericalAxis() { RangePadding =
NumericalPadding.Round };
```



### Additional

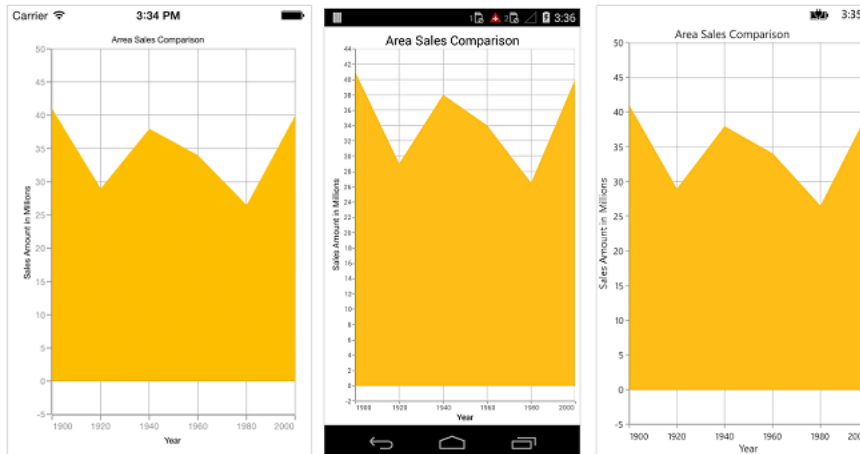
When the value of [RangePadding](#) property is [Additional](#), axis range will be rounded and an interval of the axis will be added as padding to the minimum and maximum values of the range.

**XML**

```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis RangePadding="Additional"/>
</chart:SfChart.SecondaryAxis>
```

**C#**

```
chart.SecondaryAxis = new NumericalAxis() { RangePadding =
NumericalPadding.Additional };
```

**Normal**

When the value of [RangePadding](#) property is [Normal](#), nice range will be calculated for the axis based on the best readability of the data. This is also the default for vertical axis.

**XML**

```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis RangePadding="Normal"/>
</chart:SfChart.SecondaryAxis>
```

**C#**

```
chart.SecondaryAxis = new NumericalAxis() { RangePadding =
NumericalPadding.Normal };
```



### Date Time Axis

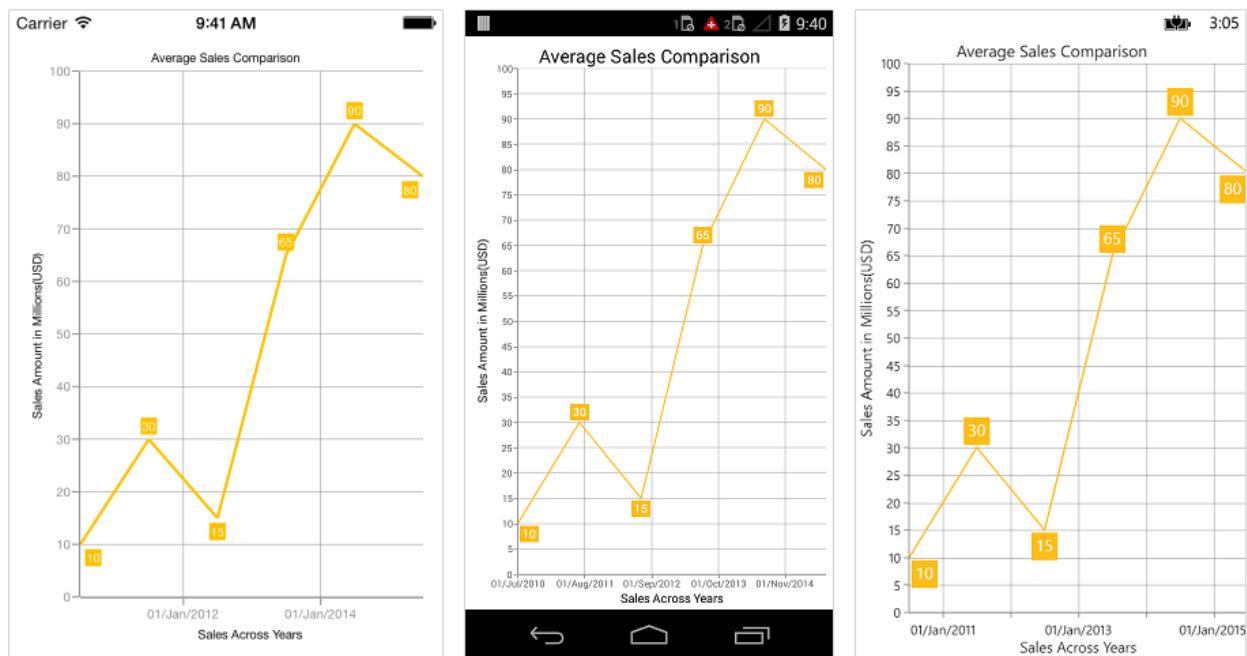
Date time axis uses date time scale and displays date time values as axis labels in specified format.

### XML

```
<chart:SfChart.PrimaryAxis >
<chart:DateTimeAxis/>
</chart:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis();
```



### Customizing date time range

To customize the range of an axis, you can use the [Minimum](#) and [Maximum](#) properties of [DateTimeAxis](#). By default, nice range will be calculated automatically based on the provided data.

**XML**

```

Namespace:
xmlns:sys="clr-namespace:System;assembly=mscorlib"
...
<chart:SfChart.PrimaryAxis >
<chart:DateTimeAxis>
<chart:DateTimeAxis.Minimum>
<sys:DateTime x:FactoryMethod="Parse">
<x:Arguments>
<x:String>Jan 1 2010</x:String>
</x:Arguments>
</sys:DateTime>
</chart:DateTimeAxis.Minimum>
<chart:DateTimeAxis.Maximum>
<sys:DateTime x:FactoryMethod="Parse">
<x:Arguments>
<x:String>Dec 30 2015</x:String>
</x:Arguments>
</sys:DateTime>
</chart:DateTimeAxis.Maximum>
</chart:DateTimeAxis>
</chart:SfChart.PrimaryAxis>

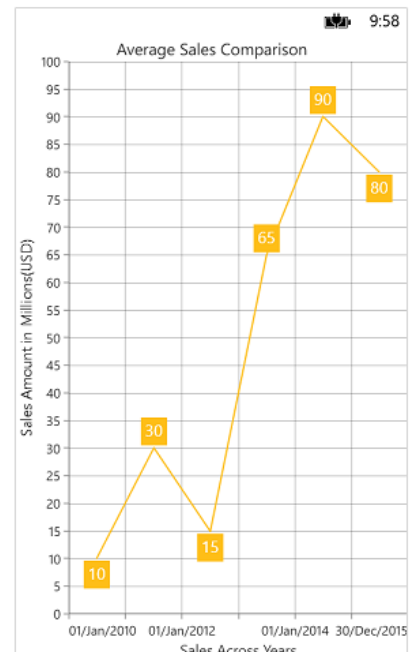
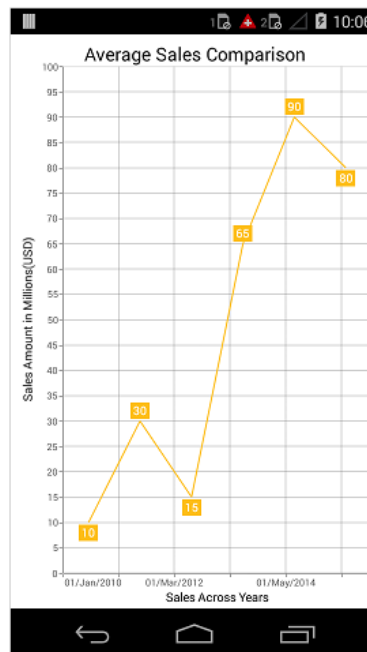
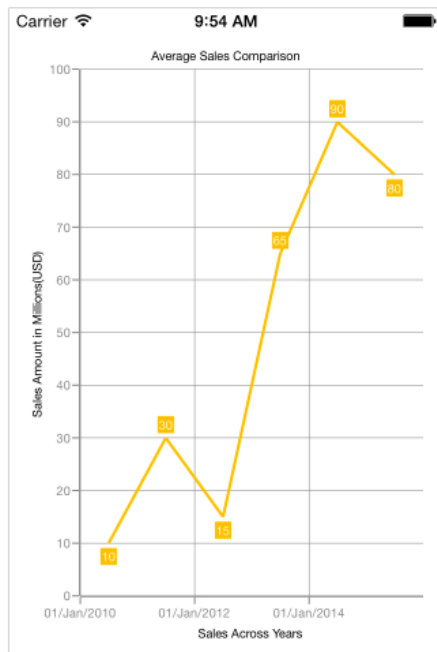
```

**C#**

```

chart.PrimaryAxis = new DateTimeAxis() {
Minimum = new DateTime(2010, 1, 1),
Maximum = new DateTime(2015, 12, 30)
};

```



### Date time intervals

Date time intervals can be customized using [Interval](#) and [IntervalType](#) properties of the [DateTimeAxis](#). For example, setting [Interval](#) as 2 and [IntervalType](#) as [Years](#) will consider 2 years as interval.

Essential Chart supports the following types of interval for date time axis

- Years
- Months
- Days
- Hours
- Minutes
- Seconds
- Milliseconds

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:DateTimeAxis IntervalType="Months">
<chart:DateTimeAxis.Interval>
<Double>6</Double>
</chart:DateTimeAxis.Interval>
</chart:DateTimeAxis>
</chart:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis() {
IntervalType = DateTimeIntervalType.Months,
Interval = 6
};
```



### Get actual interval type

The [GetActualIntervalType](#) method is used to get the interval that is calculated from actual data in [DateTimeAxis](#).

### C#

```
var intervalType = dateTimeAxis.GetActualIntervalType();
```

### Apply padding to range

Padding can be applied to the minimum and maximum extremes of range by using the [RangePadding](#) property. Date-time axis supports the following types of padding:

- None
- Round
- Additional

### None

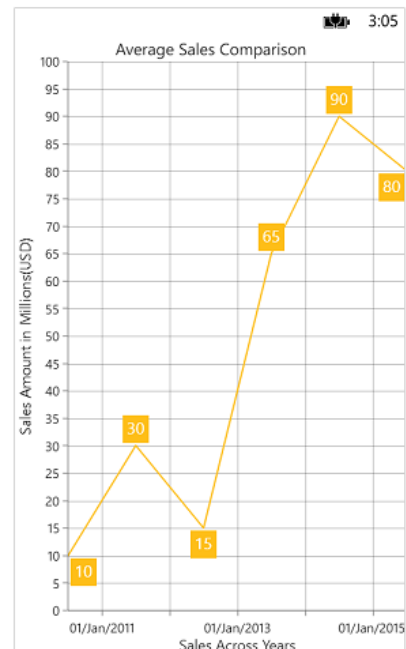
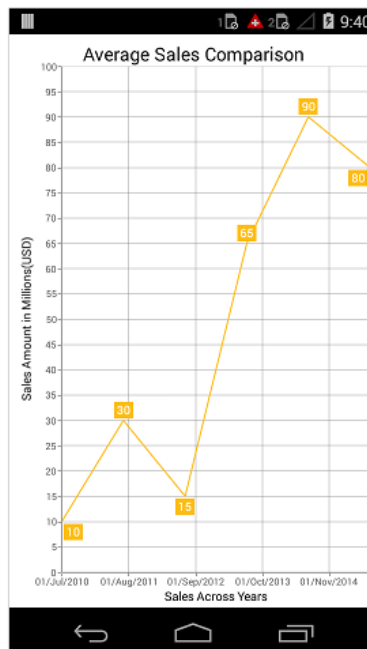
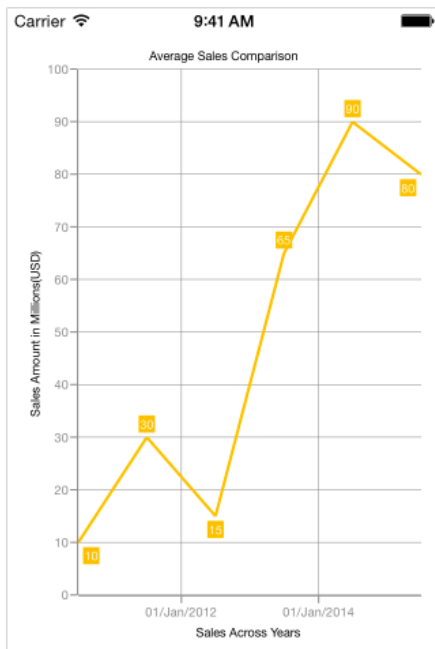
When the value of [RangePadding](#) property is [None](#), padding will not be applied to the axis. This is also the default value of [RangePadding](#).

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:DateTimeAxis RangePadding="None"/>
</chart:SfChart.PrimaryAxis >
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis() { RangePadding =
DateTimeRangePadding.None };
```



### Round

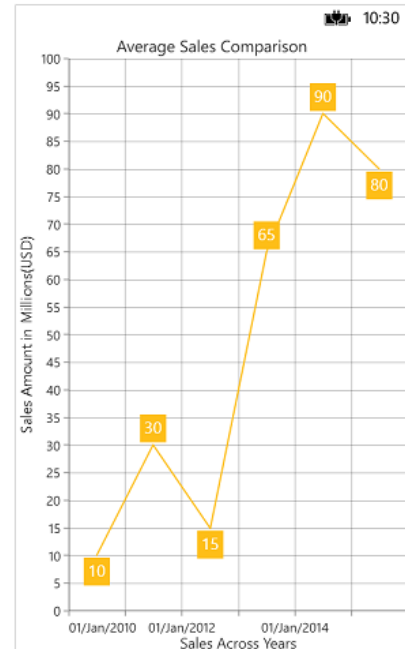
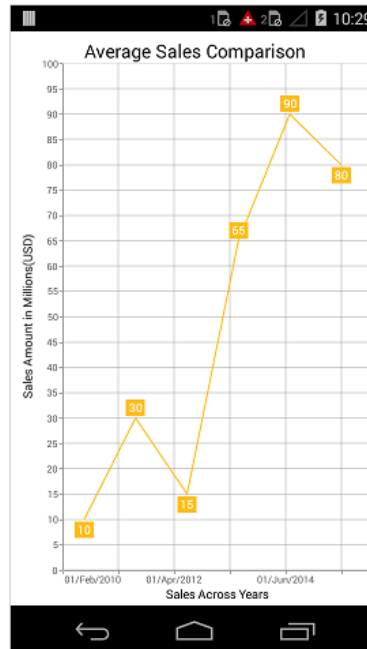
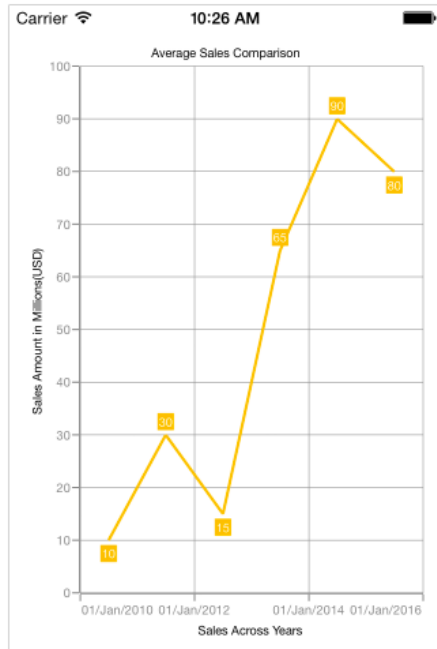
When the value of [RangePadding](#) property is [Round](#), axis range will be rounded to the nearest possible date time value.

**XML**

```
<chart:SfChart.PrimaryAxis >
<chart:DateTimeAxis RangePadding="Round"/>
</chart:SfChart.PrimaryAxis >
```

**C#**

```
chart.PrimaryAxis = new DateTimeAxis() { RangePadding =
DateTimeRangePadding.Round };
```

**Additional**

When the value of [RangePadding](#) property is [Additional](#), range will be rounded and date time interval of the axis will be added as padding to the minimum and maximum extremes of the range.

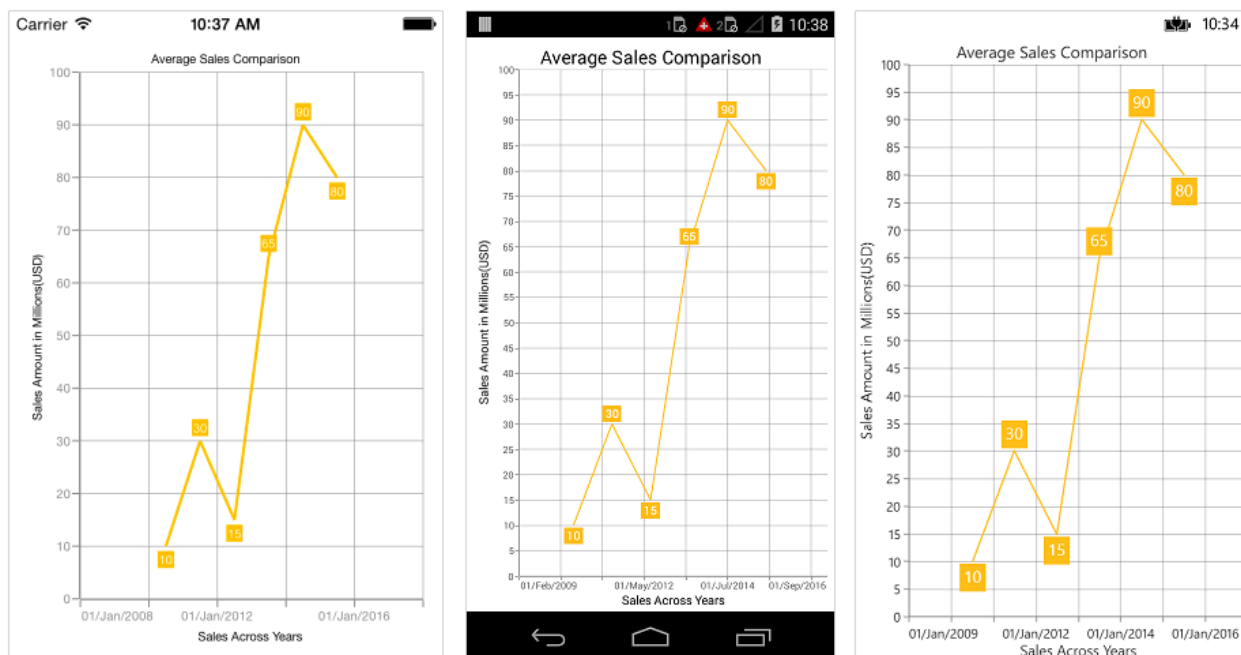
**XML**

```
<chart:SfChart.PrimaryAxis >
<chart:DateTimeAxis RangePadding="Additional"/>
</chart:SfChart.PrimaryAxis >
```

**C#**

```
chart.PrimaryAxis = new DateTimeAxis() { RangePadding =
DateTimeRangePadding.Additional };
```





### Date-time category axis

The [DateTimeCategoryAxis](#) is a unique type of axis used mainly with financial series. Like [CategoryAxis](#), all the data points are plotted with equal spaces by removing space for missing dates. Intervals and ranges for the axis are calculated similar to [DateTimeAxis](#). There will be no visual gaps between points even when the difference between two points is more than a year. The following APIs are used to customize the interval of [DateTimeCategoryAxis](#).

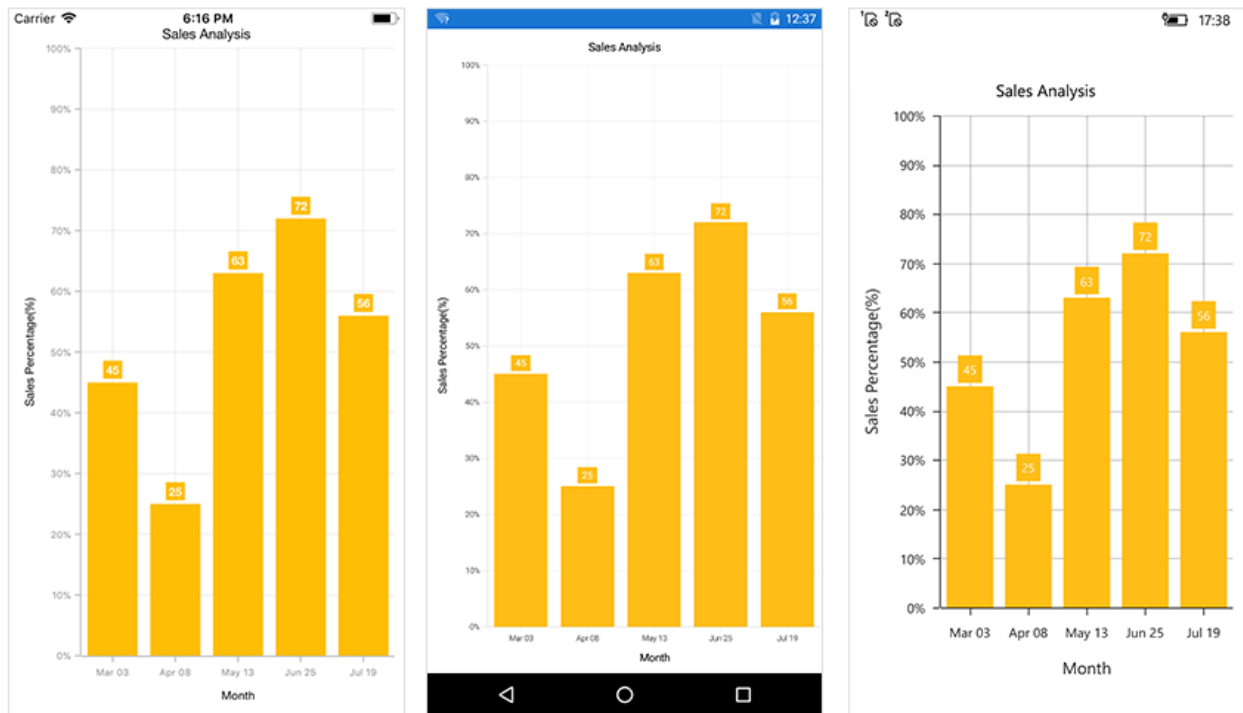
- [Interval](#) - Gets or sets the double value that represents the interval between the labels.
- [IntervalType](#) - Gets or sets the [DateTimeIntervalType](#) that represents the type of the interval to be displayed.

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:DateTimeCategoryAxis Interval="1" IntervalType="Months" />
</chart:SfChart.PrimaryAxis>
```

### C#

```
DateTimeCategoryAxis xAxis = new DateTimeCategoryAxis();
xAxis.Interval = 1;
xAxis.IntervalType = DateTimeIntervalType.Months;
chart.PrimaryAxis = xAxis;
```



### Logarithmic axis

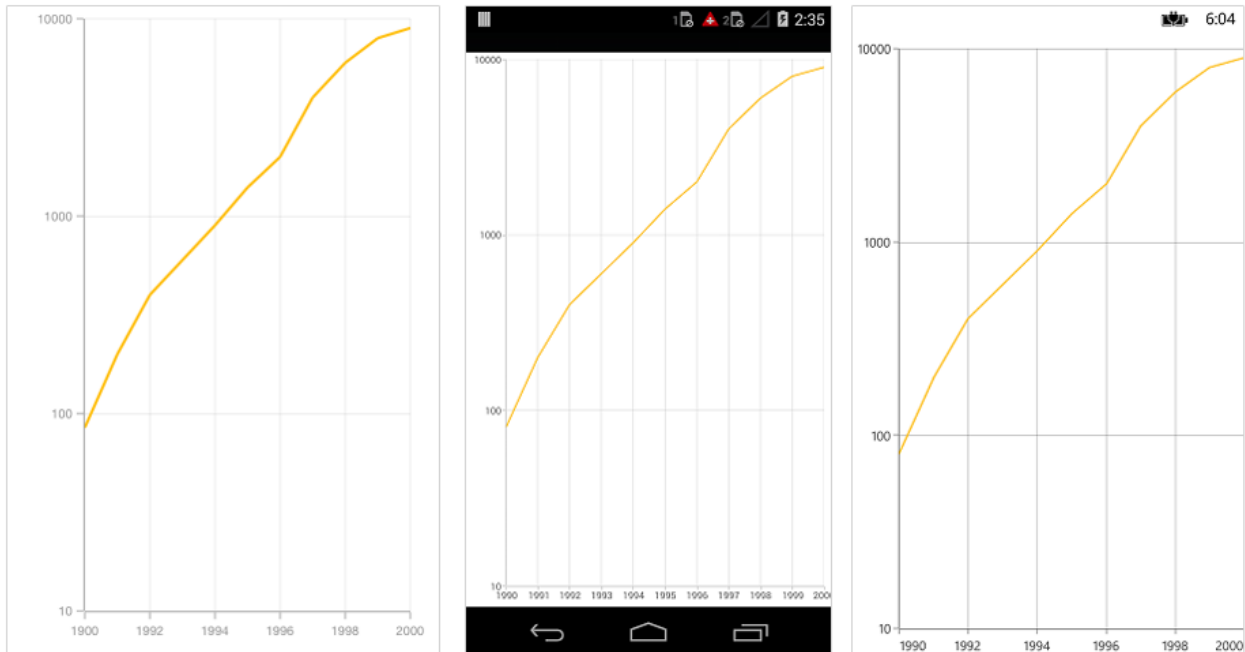
Logarithmic axis uses logarithmic scale and displays numbers as axis labels.

### XML

```
<chart:SfChart.SecondaryAxis>
<chart:LogarithmicAxis />
</chart:SfChart.SecondaryAxis>
```

### C#

```
chart.SecondaryAxis = new LogarithmicAxis ();
```



### Customizing the logarithmic range

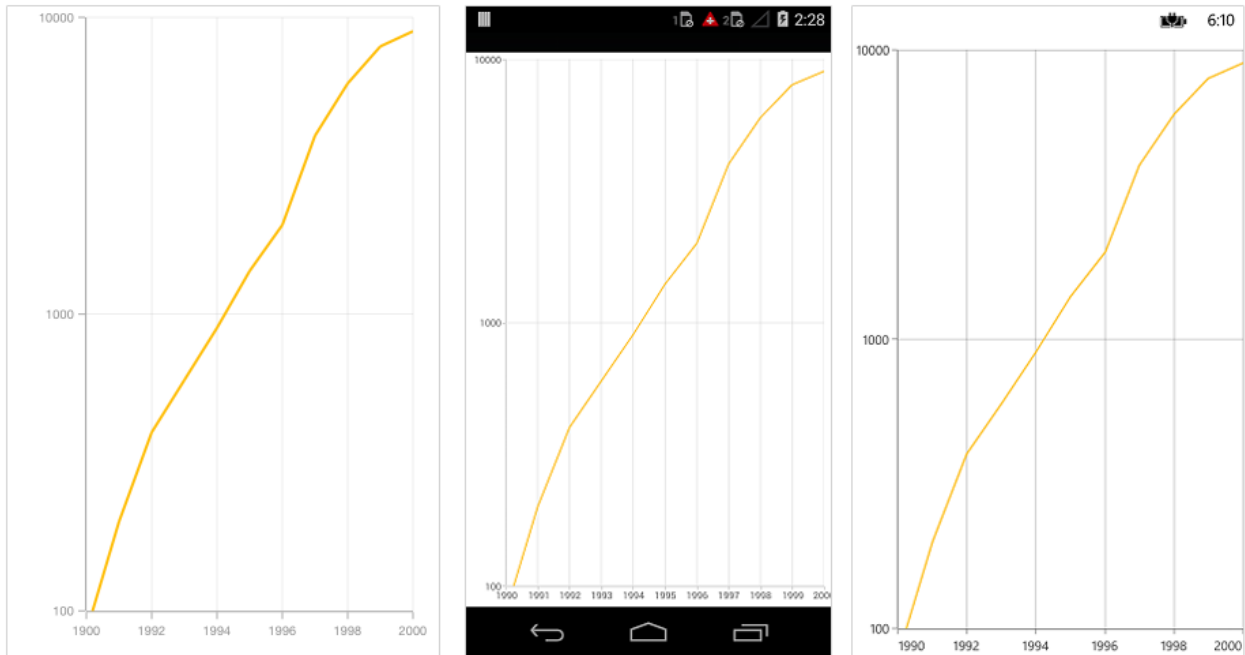
To customize the range of log axis, you can use the [Minimum](#), [Maximum](#) and [Interval](#) properties of [LogarithmicAxis](#). By default, nice range will be calculated automatically based on the provided data.

### XML

```
<chart:SfChart.SecondaryAxis>
<chart:LogarithmicAxis >
<chart:LogarithmicAxis.Minimum>
<x:Double>100</x:Double>
</chart:LogarithmicAxis.Minimum>
<chart:LogarithmicAxis.Maximum>
<x:Double>10000</x:Double>
</chart:LogarithmicAxis.Maximum>
</chart:LogarithmicAxis>
</chart:SfChart.SecondaryAxis>
```

### C#

```
chart.SecondaryAxis = new LogarithmicAxis() {
    Minimum = 100,
    Maximum = 10000
};
```



### *Customizing the logarithmic base*

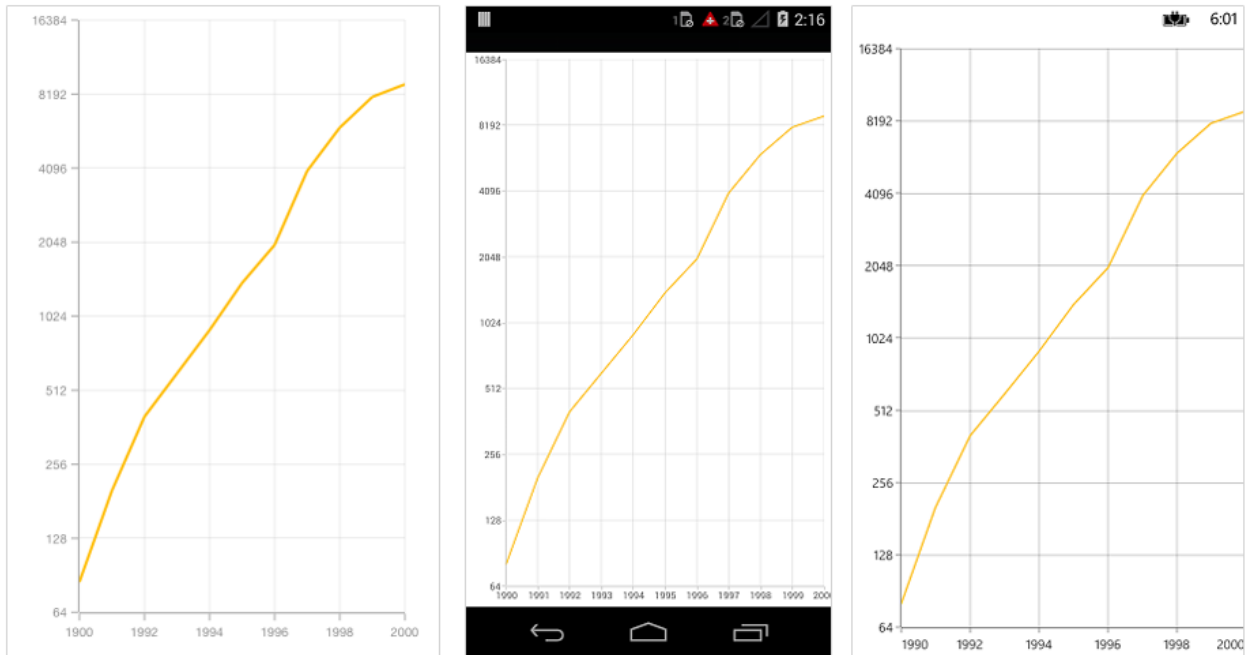
To customize the log base value, you can use [LogarithmicBase](#) property.

### **XML**

```
<chart:SfChart.SecondaryAxis>  
<chart:LogarithmicAxis LogarithmicBase="2" />  
</chart:SfChart.SecondaryAxis>
```

### **C#**

```
chart.SecondaryAxis = new LogarithmicAxis() { LogarithmicBase = 2 };
```



### Common axis features

Customization of features such as axis title, labels, grid lines and tick lines are common to all the axes. Each of these features are explained in this section.

#### Axis Visibility

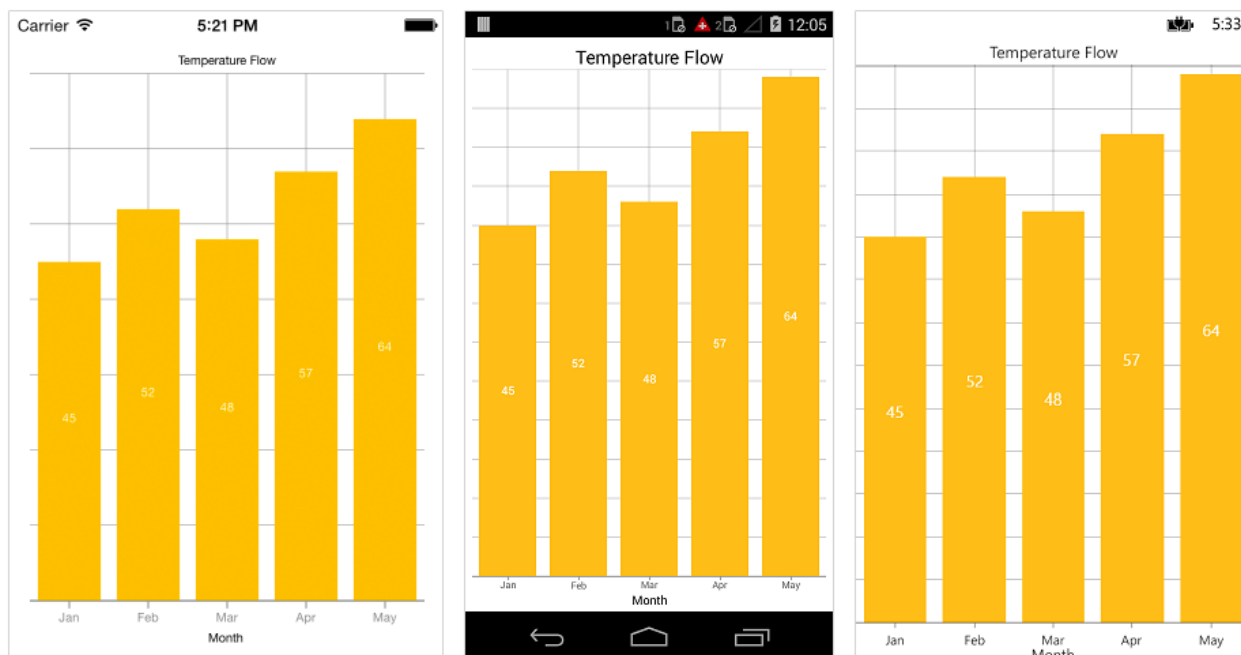
Axis visibility can be controlled using the [IsVisible](#) property of axis. Default value of [IsVisible](#) property is **True**.

#### XML

```
<chart:SfChart.SecondaryAxis>  
<chart:NumericalAxis IsVisible="False"/>  
</chart:SfChart.SecondaryAxis >
```

#### C#

```
chart.SecondaryAxis.IsVisible = false;
```



### Axis title

The [Title](#) property in axis provides options to customize the text and font of axis title. Axis does not display title by default. The title can be customized using following properties,

- [Text](#) – used to set the title for axis.
- [TextColor](#) – used to change the color of the label.
- [BackgroundColor](#) – used to change the label background color.
- [BorderColor](#) – used to change the border color.
- [BorderWidth](#) – used to change the width of the border.
- [Font](#) – used to change the text size, font family, and font weight. (This is deprecated API. Use [FontSize](#), [FontFamily](#), and [FontAttributes](#) properties instead of this.)
- [FontFamily](#) - used to change the font family for the axis title.
- [FontAttributes](#) - used to change the font style for the axis title.
- [FontSize](#) - used to change the font size for the axis title.
- [Margin](#) - used to change the margin size for labels.

Following code snippet illustrates how to enable and customize the axis title.

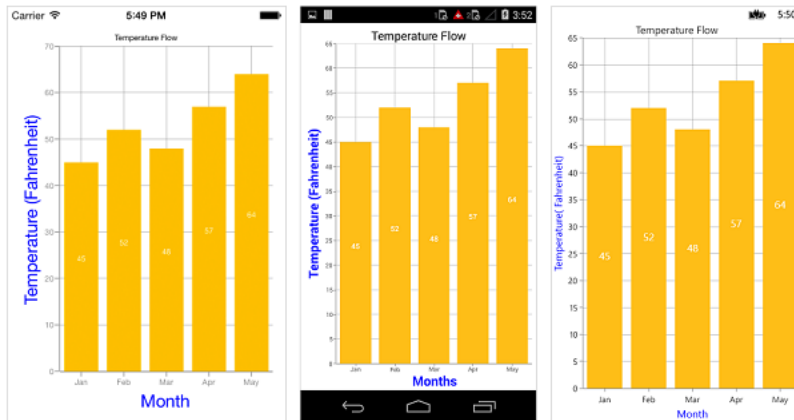
### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis >
<chart:CategoryAxis.Title>
<chart:ChartAxisTitle Text="Month" TextColor="Blue" Font="Bold,20"/>
</chart:CategoryAxis.Title>
</chart:CategoryAxis >
</chart:SfChart.PrimaryAxis >
```

### C#

```
chart.PrimaryAxis.Title.Text = "Month";
```

```
chart.PrimaryAxis.Title.TextColor = Color.Blue;
chart.PrimaryAxis.Title.Font = Font.SystemFontOfSize(20,
FontAttributes.Bold);
```



### Axis label rotation

The [LabelRotationAngle](#) property of axis can be used to rotate the axis labels position. Default value of [LabelRotationAngle](#) property is 0d.

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis LabelRotationAngle="-45"/>
</chart:SfChart.PrimaryAxis>
```

### C#

```
CategoryAxis categoryAxis = new CategoryAxis();
categoryAxis.LabelRotationAngle = -45;
chart.PrimaryAxis = categoryAxis;
```



### Axis line customization

[SfChart](#) provides support to customize the style of the axis line by defining the [AxisLineStyle](#) property as shown in the below code snippet.

- [StrokeColor](#) – used to change the stroke color of axis line.
- [StrokeWidth](#) – used to change the stroke width of axis line.
- [StrokeDashArray](#) - used to render axis line series with dashes.

### XML

```

Namespace:
xmlns:sys="clr-namespace:System;assembly=mscorlib"
...
<chart:SfChart.PrimaryAxis>
<chart:DateTimeAxis>
<chart:DateTimeAxis.AxisLineStyle>
<chart:ChartLineStyle StrokeWidth="10" StrokeColor="Red">
<chart:ChartLineStyle.StrokeDashArray>
<x:Array Type="{x:Type x:Double}">
<sys:Double>5</sys:Double>
<sys:Double>6</sys:Double>
</x:Array>
</chart:ChartLineStyle.StrokeDashArray>
</chart:ChartLineStyle>
</chart:DateTimeAxis.AxisLineStyle>
</chart:DateTimeAxis>
</chart:SfChart.PrimaryAxis>

```

### C#

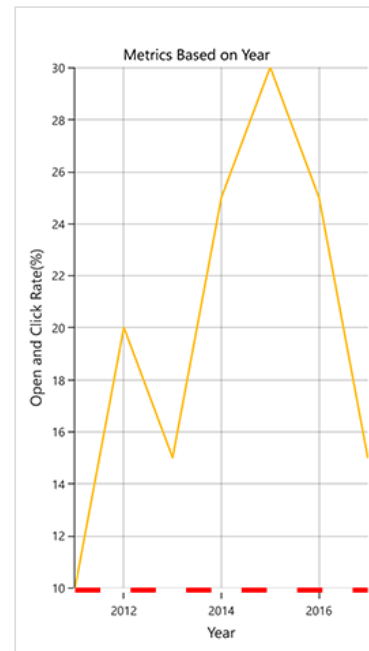
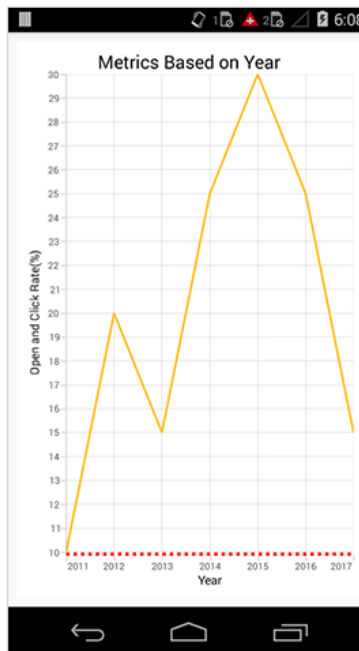
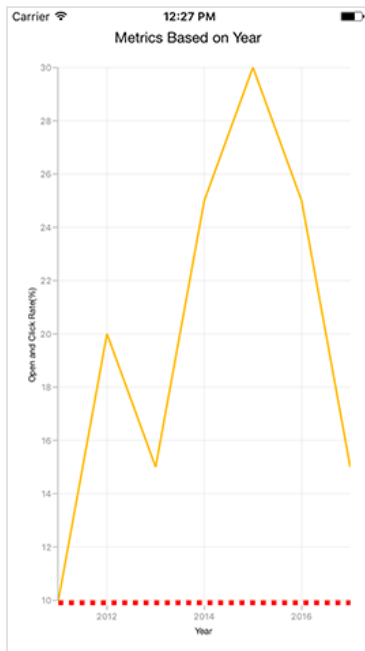
```

CategoryAxis primaryAxis = new CategoryAxis();
ChartLineStyle axisLineStyle = new ChartLineStyle();

```



```
axisLineStyle.StrokeColor = Color.Red;
axisLineStyle.StrokeWidth = 10;
axisLineStyle.StrokeDashArray = new double[2] { 5, 6 };
primaryAxis.AxisLineStyle = axisLineStyle;
chart.PrimaryAxis = primaryAxis;
```



### Axis line offset

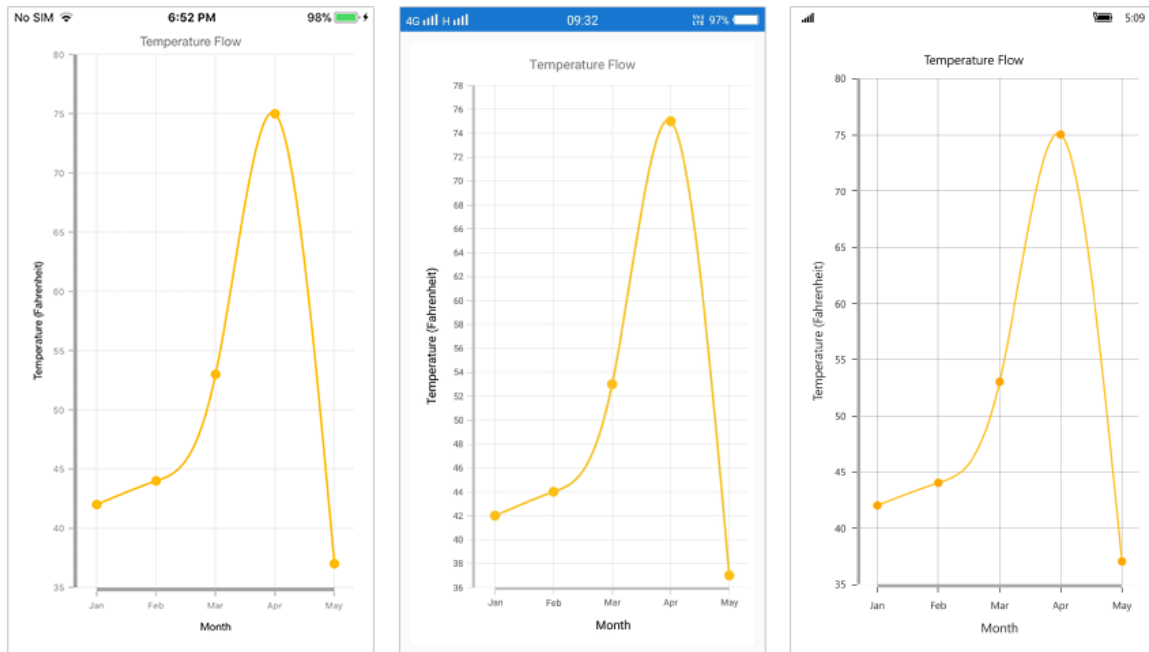
The [AxisLineOffset](#) property is used to offset the rendering of axis line.

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis AxisLineOffset="20" PlotOffset="20">
<chart:CategoryAxis.AxisLineStyle>
<chart:ChartLineStyle StrokeWidth="5"/>
</chart:CategoryAxis.AxisLineStyle>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
```

### C#

```
CategoryAxis primaryAxis = new CategoryAxis();
primaryAxis.PlotOffset = 20;
primaryAxis.AxisLineOffset = 20;
primaryAxis.AxisLineStyle.StrokeWidth = 5;
chart.PrimaryAxis = primaryAxis;
```



### Label customization

The [LabelStyle](#) property of axis provides options to customize the font-family, color, size and font-weight of axis labels. The axis labels can be customized using following properties:

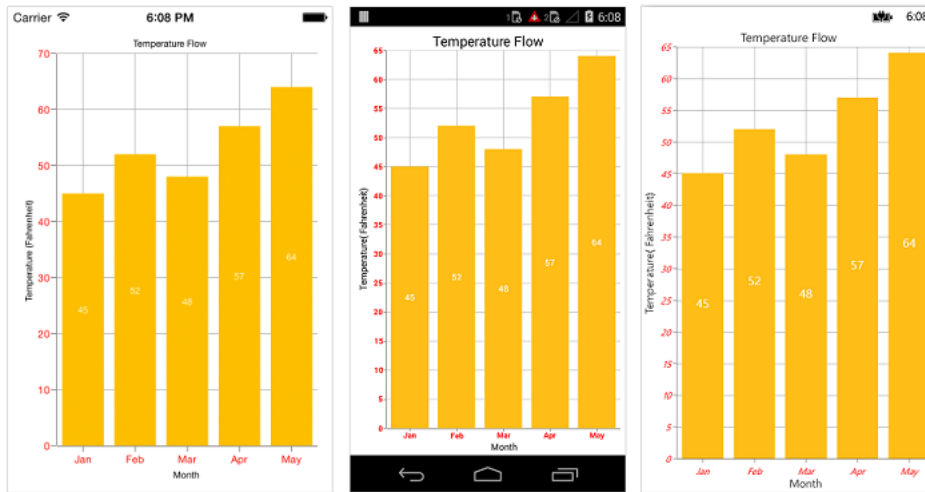
- [TextColor](#) – used to change the color of the labels.
- [BackgroundColor](#) – used to change the label background color.
- [BorderColor](#) – used to change the border color.
- [BorderThickness](#) – used to change the thickness of the border.
- [Font](#) – used to change the text size, font family and font weight.
- [Margin](#) - used to change the margin size for labels.
- [CornerRadius](#) - Used to change the corner radius of the background of labels.
- [LabelAlignment](#) - Used to align the label at the [Start](#), [Center](#), or [End](#).
- [LabelFormat](#) - used to set numeric or date time format to the chart axis label.
- [MaxWidth](#) - Provides the maximum text width of the axis label and wraps into the next line when exceeds the maximum width.
- [WrappedLabelAlignment](#) - Positions the wrapped text at the start, center, or end. The default value of the [WrappedLabelAlignment](#) property is [Start](#).

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis >
<chart:CategoryAxis.LabelStyle>
<chart:ChartAxisLabelStyle TextColor="Red" Font="Bold,20"/>
</chart:CategoryAxis.LabelStyle>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis.LabelStyle.Font = Font.SystemFontOfSize(20,
FontAttributes.Bold);
chart.PrimaryAxis.LabelStyle.TextColor = Color.Red;
```



### Label and tick positioning

Axis labels and ticks can be positioned [Inside](#) or [Outside](#) the chart area by using [LabelStyle.LabelsPosition](#) and [TickPosition](#) properties of ChartAxis. By default labels and ticks will be positioned outside the chart area.

### XML

```
<chart:SfChart.PrimaryAxis >
<chart:DateTimeAxis TickPosition="Inside">
<chart:DateTimeAxis.LabelStyle>
<chart:ChartAxisLabelStyle LabelsPosition="Inside"/>
</chart:DateTimeAxis.LabelStyle>
</chart:DateTimeAxis>
</chart:SfChart.PrimaryAxis >
```

### C#

```
chart.PrimaryAxis.LabelStyle.LabelsPosition = AxisElementPosition.Inside;
chart.PrimaryAxis.TickPosition = AxisElementPosition.Inside;
```



### Edge labels placement

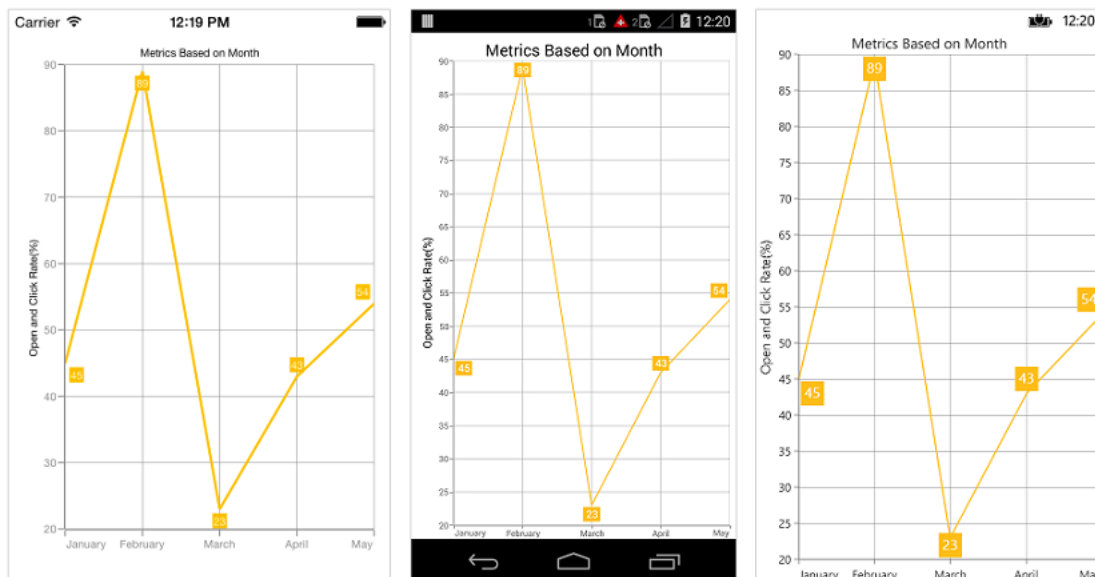
Labels with long text at the edges of an axis may appear partially outside the chart. The [EdgeLabelsDrawingMode](#) property can be used to avoid the partial appearance of labels at the corners. Default value of this property is [Center](#). Other available options of [EdgeLabelsDrawingMode](#) are [Fit](#), [Shift](#) and [Hide](#).

### XML

```
<chart:CategoryAxis EdgeLabelsDrawingMode="Shift"/>
```

### C#

```
chart.PrimaryAxis.EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Shift;
```



### Edge labels visibility

The visibility of the edge labels in an axis can be controlled using [EdgeLabelsVisibilityMode](#) property.

The following options are available in [EdgeLabelsVisibilityMode](#),

- [Default](#) - used to display the edge label based on auto interval calculations
- [Visible](#) - used to display the edge labels (first and last label) irrespective of the auto interval calculation until zooming (i.e., in normal state).
- [AlwaysVisible](#) - used to always display the edge labels even while zooming the chart.

The following code example demonstrates the AlwaysVisible option while zooming.

#### **XML**

```
<chart:SfChart.PrimaryAxis>  
<chart:NumericalAxis EdgeLabelsVisibilityMode="AlwaysVisible">  
</chart:NumericalAxis>  
</chart:SfChart.PrimaryAxis>
```

#### **C#**

```
Chart.PrimaryAxis = new NumericalAxis();  
chart.SecondaryAxis.EdgeLabelsVisibilityMode =  
EdgeLabelsVisibilityMode.AlwaysVisible;
```

#### *Label extent*

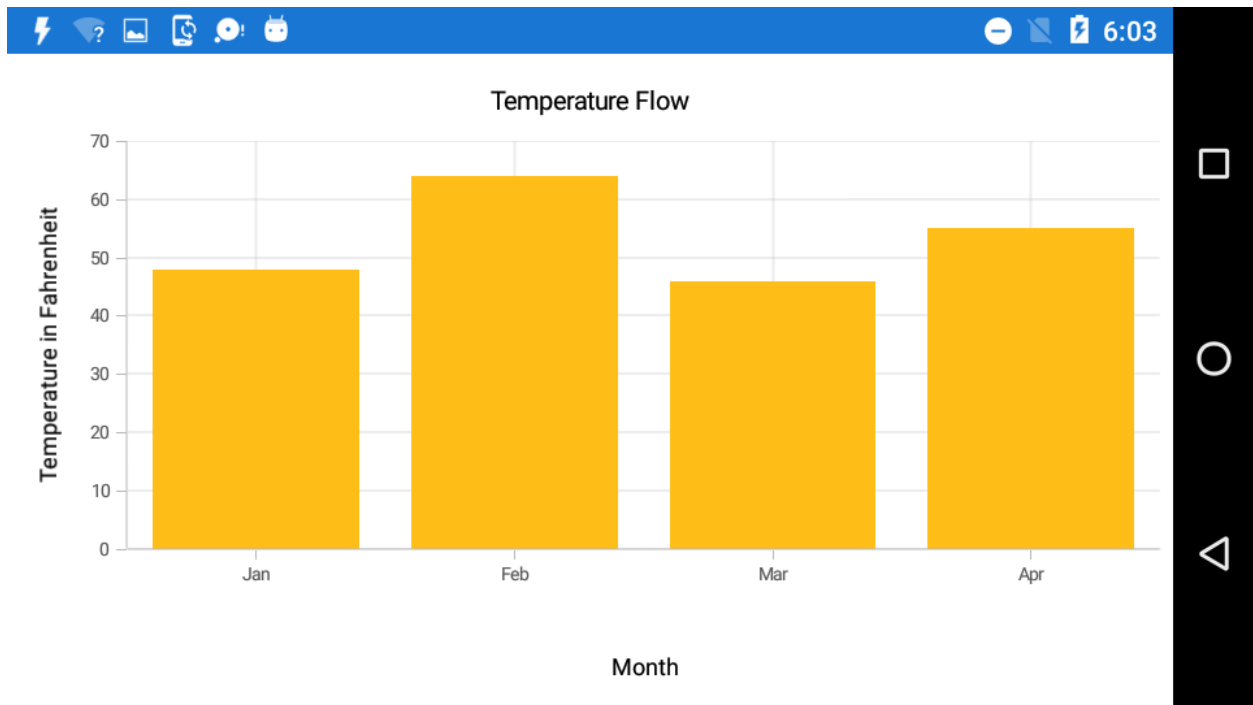
The [LabelExtent](#) property allows to set the gap between axis labels and title. This is typically used to maintain the fixed gap between axis labels and title when the digits of the axis value changed in live update.

#### **XML**

```
<chart:SfChart.PrimaryAxis>  
<chart:CategoryAxis LabelExtent="60" >  
<chart:CategoryAxis.Title>  
<chart:ChartAxisTitle Text="Month"/>  
</chart:CategoryAxis.Title>  
</chart:CategoryAxis>  
</chart:SfChart.PrimaryAxis>
```

#### **C#**

```
Chart.PrimaryAxis = new CategoryAxis();  
Chart.PrimaryAxis.LabelExtent = 60;  
Chart.PrimaryAxis.Title.Text = "Month";
```



#### Grid lines customization

The [ShowMajorGridLines](#) and [ShowMinorGridLines](#) properties are used to control the visibility of grid lines. [MajorGridLineStyle](#) and [MinorGridLineStyle](#) properties in axis are used to customize the major grid lines and minor grid lines of an axis respectively. They provide options to change the width, dashes, color of grid lines. By default minor grid lines will not be visible.

#### XML

```
<chart:SfChart.SecondaryAxis >  
  <chart:NumericalAxis ShowMajorGridLines="True" ShowMinorGridLines="True"  
    MinorTicksPerInterval="1"/>  
</chart:SfChart.SecondaryAxis >
```

#### C#

```
chart.SecondaryAxis = new NumericalAxis ()  
{  
    ShowMajorGridLines = true,  
    ShowMinorGridLines = true,  
    MinorTicksPerInterval = 1  
};
```



### *Tick lines customization*

The [MajorTickStyle](#) and [MinorTickStyle](#) properties in axis are used to customize the major tick lines of an axis and minor tick lines of an axis respectively. They provide options to change the [StrokeWidth](#), [TickSize](#), [StrokeColor](#) and [MinorTicksPerInterval](#) of tick lines. By default minor tick lines will not be visible.

### **XML**

```
<chart:SfChart.SecondaryAxis >
  <chart:NumericalAxis ShowMinorGridLines="True" MinorTicksPerInterval="1">
    <chart:NumericalAxis.MajorTickStyle >
      <chart:ChartAxisTickStyle TickSize="7" StrokeColor="Blue" StrokeWidth="3"/>
    </chart:NumericalAxis.MajorTickStyle>
    <chart:NumericalAxis.MinorTickStyle>
      <chart:ChartAxisTickStyle TickSize="5" StrokeColor="Green" StrokeWidth="2"/>
    </chart:NumericalAxis.MinorTickStyle>
    <chart:NumericalAxis/>
  </chart:SfChart.SecondaryAxis >
```

### **C#**

```
NumericalAxis numerical = new NumericalAxis();
numerical.MajorTickStyle.TickSize = 7;
numerical.MajorTickStyle.StrokeWidth = 3;
numerical.MajorTickStyle.StrokeColor = Color.Red;
numerical.ShowMinorGridLines = true;
numerical.MinorTicksPerInterval = 1;
numerical.MinorTickStyle.TickSize = 5;
numerical.MinorTickStyle.StrokeWidth = 2;
numerical.MinorTickStyle.StrokeColor = Color.Green;
chart.SecondaryAxis = numerical;
```



### Customize individual axis elements

The [RangeStyles](#) can be used to customize the gridlines, ticks and axis labels for a specific region of [ChartAxis](#). The following properties are used to customize the specific range in an axis:

- [Start](#) - Sets the start range of an axis
- [End](#) - Sets the end range of an axis
- [MajorGridLineStyle](#) - Customizes the major grid lines of an axis.
- [MinorGridLineStyle](#) - Customizes the minor grid lines of an axis.
- [MajorTickStyle](#) - Customizes the major tick lines of an axis.
- [MinorTickStyle](#) - Customizes the minor tick lines of an axis.
- [LabelStyle](#) - Customizes the axis labels for a specific range.

**Note:** Grid lines [StrokeDashArray](#) is not supported if the [RangeStyles](#) of axis is set in UWP.

### XML

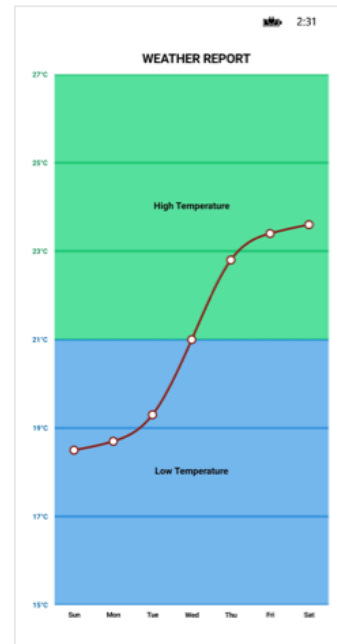
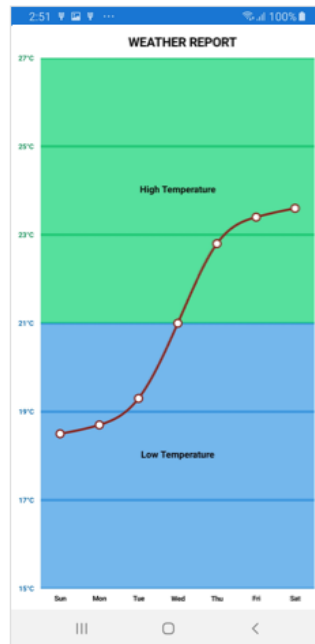
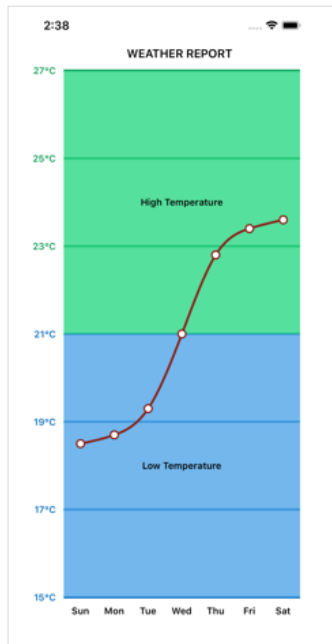
```
<chart:NumericalAxis.RangeStyles>
  <chart:ChartAxisRangeStyleCollection>
    <chart:ChartAxisRangeStyle Start="15" End="21" >
      <chart:ChartAxisRangeStyle.MajorGridLineStyle>
        <chart:ChartLineStyle StrokeColor="#096EBF" StrokeWidth="3"/>
      </chart:ChartAxisRangeStyle.MajorGridLineStyle>
      <chart:ChartAxisRangeStyle.LabelStyle>
        <chart:ChartAxisLabelStyle TextColor="#096EBF" FontAttributes="Bold"/>
      </chart:ChartAxisRangeStyle.LabelStyle>
    </chart:ChartAxisRangeStyle>
    . . . .
  </chart:ChartAxisRangeStyleCollection>
</chart:NumericalAxis.RangeStyles>
```

### C#

```
NumericalAxis numericalAxis = new NumericalAxis() { Minimum = 15, Maximum = 27 };
ChartAxisRangeStyleCollection axisRangeStyles = new
ChartAxisRangeStyleCollection();
ChartAxisRangeStyle rangeStyle = new ChartAxisRangeStyle() { Start = 15, End = 21};
```



```
rangeStyle.MajorGridLineStyle = new ChartLineStyle() { StrokeColor =
Color.FromHex("#096EBF"), StrokeWidth = 3 };
rangeStyle.LabelStyle = new ChartAxisLabelStyle() { TextColor =
Color.FromHex("#096EBF"), FontAttributes = FontAttributes.Bold };
axisRangeStyles.Add(rangeStyle);
....
numericalAxis.RangeStyles = axisRangeStyles;
```



### Inverting axis

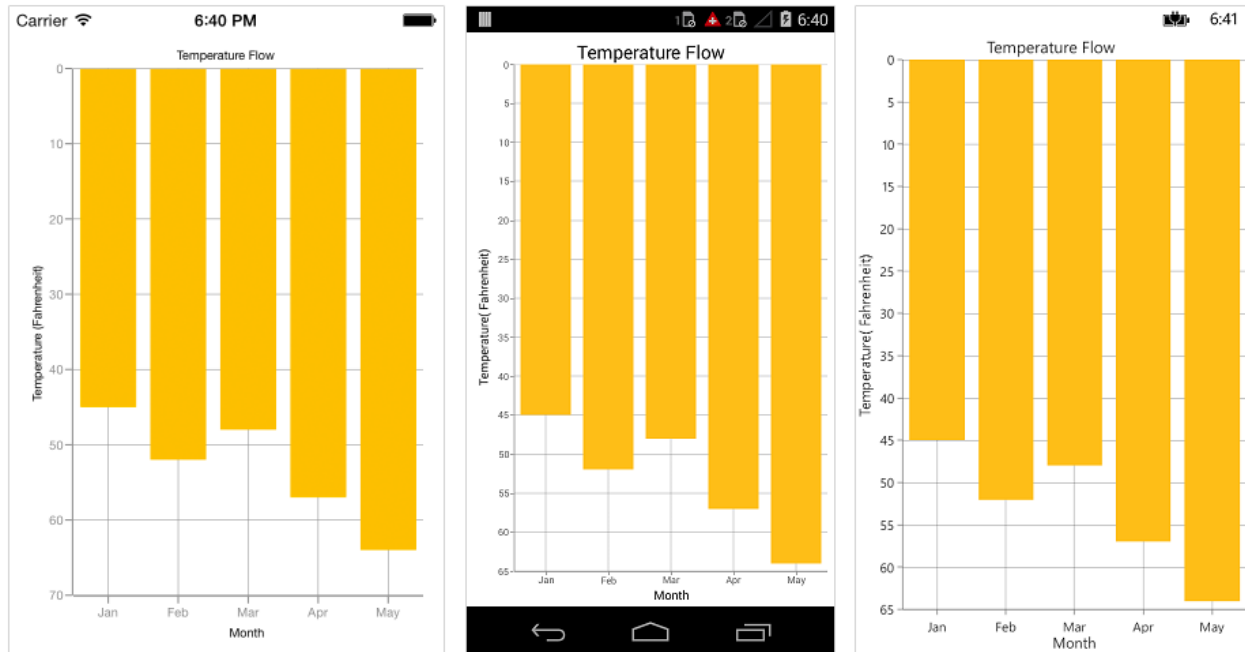
Axis can be inverted using the [IsInversed](#) property of axis. Default value of [IsInversed](#) property is `False`.

### XML

```
<chart:SfChart.SecondaryAxis >
<chart:NumericalAxis IsInversed="True">
</chart:SfChart.SecondaryAxis >
```

### C#

```
chart.SecondaryAxis.IsInversed = true;
```



### Placing axes at the opposite side

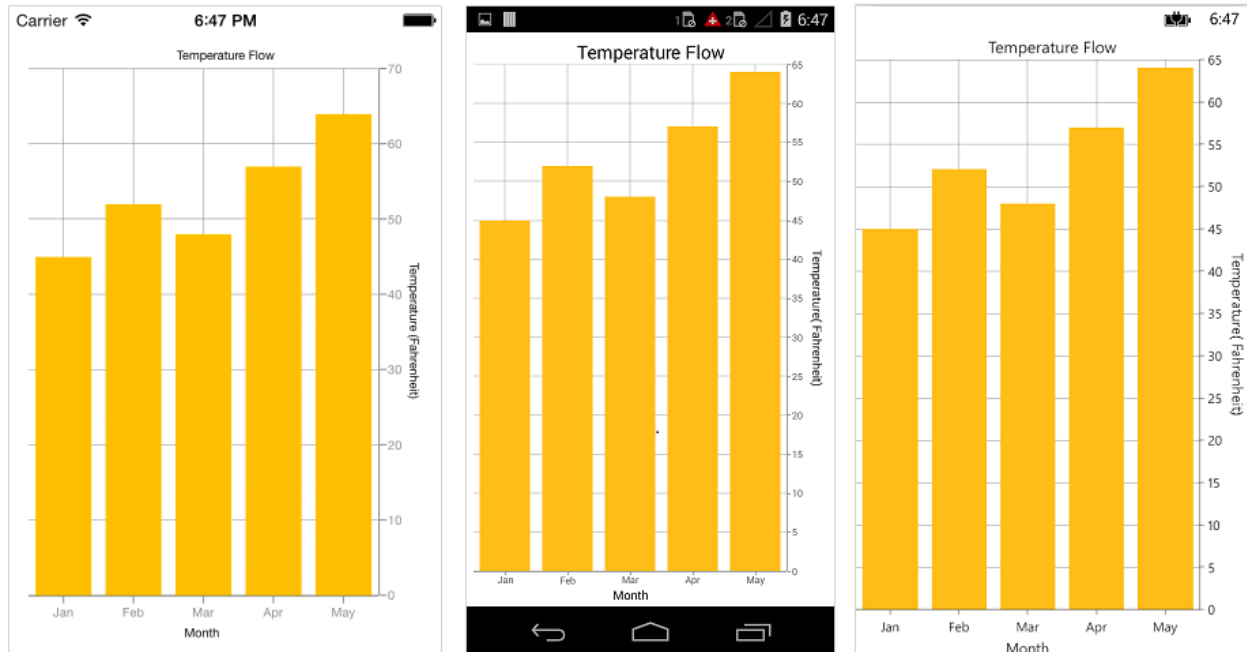
The [OpposedPosition](#) property of axis can be used to place the axis at the opposite side of its default position. Default value of [OpposedPosition](#) property is `False`.

### XML

```
<chart:SfChart.SecondaryAxis >
  <chart:NumericalAxis OpposedPosition="True">
</chart:SfChart.SecondaryAxis >
```

### C#

```
chart.SecondaryAxis.OpposedPosition = true;
```



### Offset the rendering

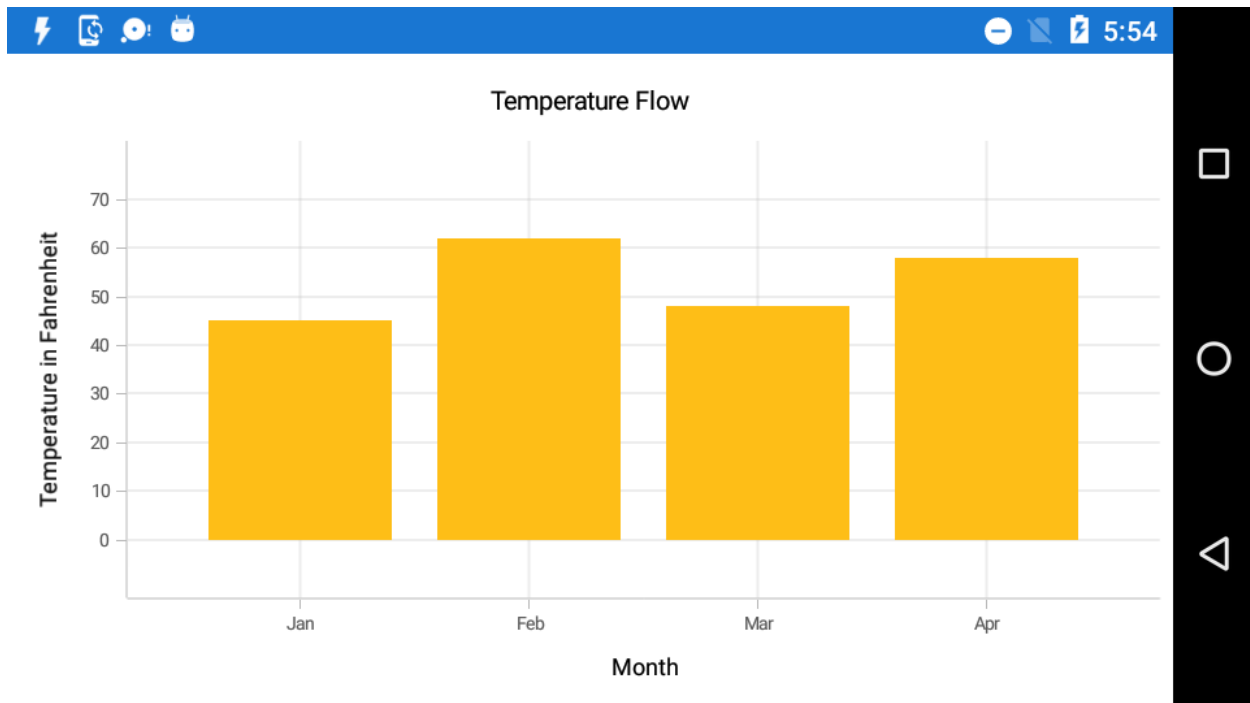
The [PlotOffset](#) property is used to offset the rendering of the axis at start and end position. The following code snippet demonstrates to apply the plot offset to both x and y axes.

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis PlotOffset="30" />
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis PlotOffset="30"/>
</chart:SfChart.SecondaryAxis>
```

### C#

```
Chart.PrimaryAxis = new CategoryAxis()
{
    PlotOffset = 30
};
Chart.SecondaryAxis = new NumericalAxis()
{
    PlotOffset = 30
};
```



#### Maximum number of labels per 100 pixels

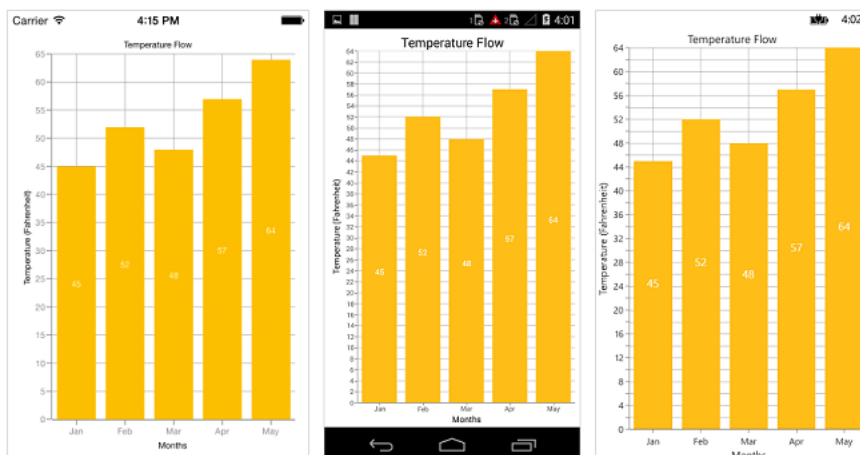
By default, a maximum of 3 labels are displayed for each 100 pixels in axis. The maximum number of labels that should be present within 100 pixels length can be customized using the [MaximumLabels](#) property of an axis. This property is applicable only for automatic range calculation and will not work if you set value for [Interval](#) property of an axis.

#### XML

```
<chart:SfChart.SecondaryAxis >
<chart:NumericalAxis MaximumLabels="5"/>
</chart:SfChart.SecondaryAxis >
```

#### C#

```
Chart.SecondaryAxis.MaximumLabels = 5;
```



### *AutoScrollingDelta*

[AutoScrollingDelta](#) is used to ensure that the specified range of data is always visible in the chart. It always shows the recently added data points at the end and scrolling will be reset to the end of the range whenever a new point is added.

By adding [ChartZoomPanBehavior](#) to the chart, you can scroll to see the previous datapoints.

### *AutoScrollingDeltaType*

In [DateTimeAxis](#), you can apply auto scrolling delta value in [Years](#), [Months](#), [Days](#), [Hours](#), [Minutes](#), [Seconds](#) and [Milliseconds](#) by setting [AutoScrollingDeltaType](#) property. Default value of this property is [Auto](#) and the delta will be calculated automatically based on range.

### **XML**

```
<chart:SfChart.PrimaryAxis>
<chart:DateTimeAxis AutoScrollingDelta="3" AutoScrollingDeltaType="Days">
</chart:SfChart.PrimaryAxis>
```

### **C#**

```
chart.PrimaryAxis = new DateTimeAxis()
{
    AutoScrollingDelta = 3,
    AutoScrollingDeltaType = DateTimeDeltaType.Days
};
```

### *AutoScrollingMode*

[AutoScrollingMode](#) property can be used to determine whether the axis should be scrolled from start position or end position. The default value of [AutoScrollingMode](#) is [End](#).

### **XML**

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis AutoScrollingDelta = "3" AutoScrollingMode = "Start">
</chart:SfChart.PrimaryAxis>
```

### **C#**

```
chart.PrimaryAxis = new DateTimeAxis()
{
    AutoScrollingDelta = 3,
    AutoScrollingMode = ChartAutoScrollingMode.Start
};
```

### *VisibleMinimum*

The [VisibleMinimum](#) property of the chart axis can be used only to get the double value that represents the minimum observable value of the axis range in runtime.

### *VisibleMaximum*

The [VisibleMaximum](#) property of the chart axis can be used only to get the double value that represents the maximum observable value of the axis range in runtime.

### Axis Crossing

Axis can be positioned anywhere in the chart area by using [CrossesAt](#) property. This property specifies where the horizontal axis should intersect or cross the vertical axis or vice-versa. Default value of [CrossesAt](#) property is null.

#### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis CrossesAt = "0" />
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis CrossesAt = "8" />
</chart:SfChart.SecondaryAxis >
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis() { CrossesAt = 0 };
chart.SecondaryAxis = new NumericalAxis() { CrossesAt = 8 };
```



### Crossing in date time axis

For crossing in date time horizontal axis, date object should be provided as value for [CrossesAt](#) property of vertical axis.

#### XML

```
Namespace:
xmlns:sys="clr-namespace:System;assembly=mscorlib"
...
<chart:SfChart.PrimaryAxis>
<chart:DateTimeAxis CrossesAt="0" />
</chart:SfChart.PrimaryAxis >
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis>
```

```
<chart:NumericalAxis.CrossesAt>
<sys:Datetime x:FactoryMethod = "Parse">
<x:Arguments>
<x:String>Jan 1 2003</x:String>
</x:Arguments>
</sys:Datetime>
</chart:NumericalAxis.CrossesAt>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

## C#

```
chart.PrimaryAxis = new DateTimeAxis() { CrossesAt = 0 };
chart.SecondaryAxis = new NumericalAxis() { CrossesAt = new DateTime(2003,
1, 1) };
```



### Positioning the axis elements while crossing

[RenderNextToCrossingValue](#) property is used to determine whether the crossing axis should be placed at crossing position or not. The default value of [RenderNextToCrossingValue](#) property is true.

## XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis CrossesAt="0" RenderNextToCrossingValue="False"/>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis CrossesAt="5" />
</chart:SfChart.SecondaryAxis>
```

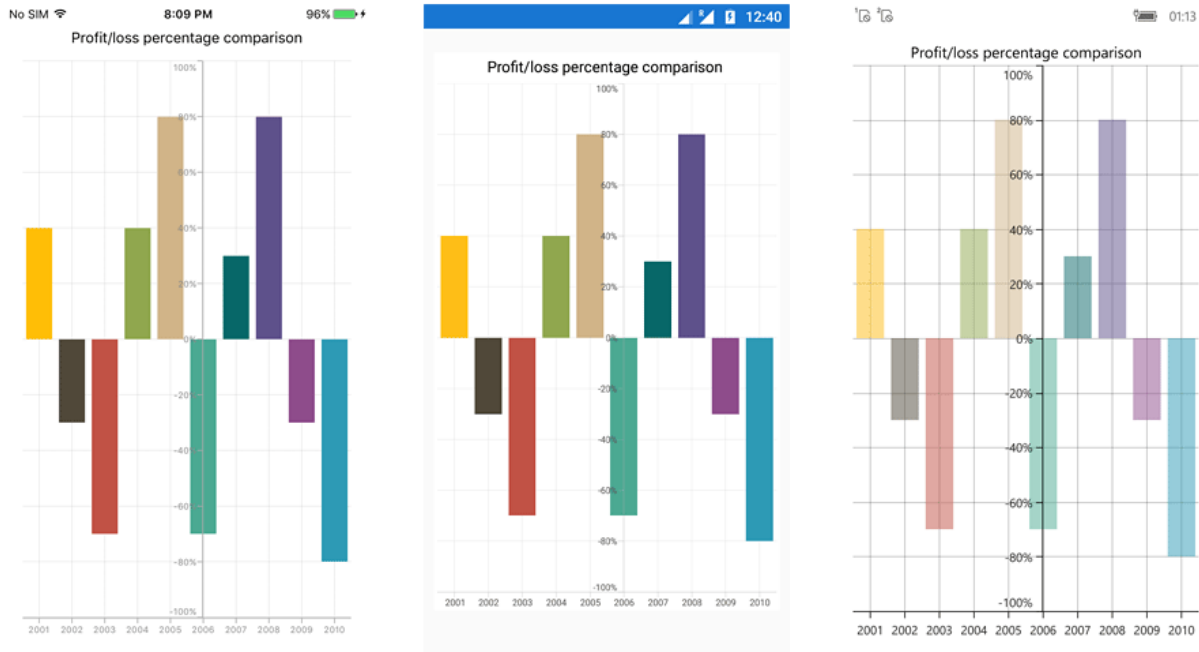
## C#

```
chart.PrimaryAxis = new CategoryAxis()
{
```

```

CrossesAt = 0,
RenderNextToCrossingValue = false
};
chart.SecondaryAxis = new NumericalAxis()
{
    CrossesAt = 5,
};

```



### Smart Axis Labels

Axis labels may overlap with each other based on chart dimensions and label size. The [LabelsIntersectAction](#) property of axis is used to avoid overlapping of axis labels. The default value of the [LabelsIntersectAction](#) is [None](#); other available values are [MultipleRows](#), [Hide](#), and [Wrap](#).

### XML

```

<chart:SfChart.PrimaryAxis >
<chart:CategoryAxis LabelsIntersectAction="MultipleRows"/>
</chart:SfChart.PrimaryAxis >

```

### C#

```

chart.PrimaryAxis.LabelsIntersectAction =
AxisLabelsIntersectAction.MultipleRows;

```





## Event

### ActualRangeChanged

The [ActualRangeChanged](#) event is triggered when the actual range of the axis is changed. The argument contains the following information.

- [ActualMinimum](#) - used to get or set the actual minimum value of the axis.
- [ActualMaximum](#) - used to get or set the actual maximum value of the axis.
- [VisibleMinimum](#) - used to get or set the visible minimum value of the axis.
- [VisibleMaximum](#) - used to get or set the visible maximum value of the axis.

---

**Note:** Actual range and visible range are similar unless the range is changed by specifying the [ZoomPosition](#) and [ZoomFactor](#) properties or zoom the chart interactively. Visible range is always the range which you see visually in the screen.

---

### LabelCreated

The [LabelCreated](#) event is triggered when the axis label is created. The argument contains the following information.

- [LabelContent](#) - Used to get or set the text of axis label based on condition.
- [Position](#) - Used to get the position of label.
- [LabelStyle](#) - Used to customize the appearance of axis labels based on condition. The properties listed in [label customization](#) can be customized using LabelStyle property.

### LabelClicked

The [LabelClicked](#) event is triggered when the axis label is clicked. The argument contains the following informations:

- [LabelContent](#) - Used to get the text of label.
- [Position](#) - Used to get the position of label.

## Chart Series

[ChartSeries](#) is the visual representation of the data. [SfChart](#) offers many types of series ranging from line series to financial series like HiLo and Candle. Based on your requirements and specifications, any type of Series can be added for data visualization.

The following APIs are common for the most of the series types:

[IsVisible](#) - controls the visibility of the series.

[ItemsSource](#) - used to set the data source for the series. Refer the [Populating Data](#) page to configure the items source and set the binding paths.

[Color](#) - used to change the color of the series.

[LegendIcon](#) - used to change the icon type in corresponding legend item.

[Label](#) - used to set the label that displays in corresponding legend item.

[IsVisibleOnLegend](#) - used to control the visibility of the series in legend.

[Opacity](#) - used to control the transparency of the series.

## Multiple Series

You can add multiple series to [Series](#) property of [SfChart](#) class. By default, all the series rendered based on the [PrimaryAxis](#) and [SecondaryAxis](#) of [SfChart](#). But if you want to plot different unit or value that is specific to particular series, you can specify the separate axis for that series using [XAxis](#) and [YAxis](#) properties of [ChartSeries](#).

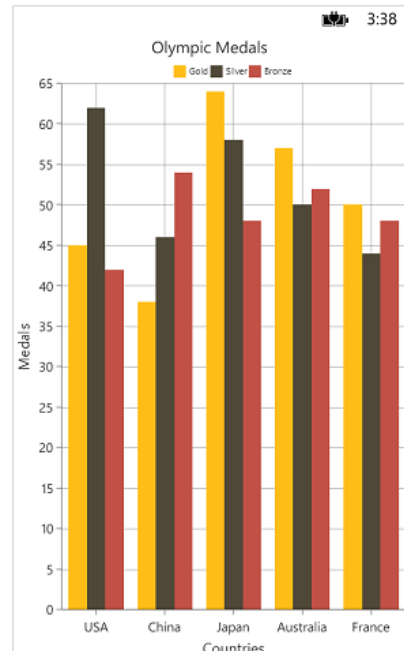
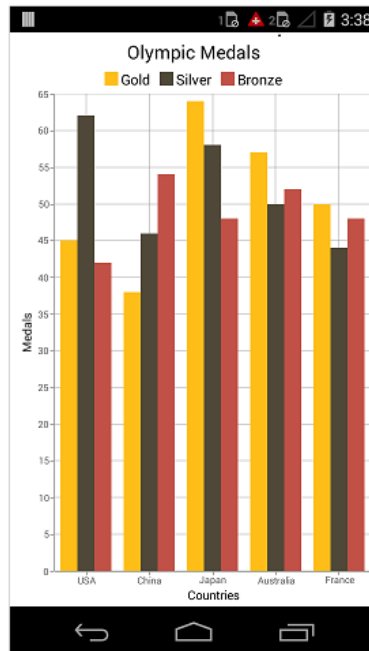
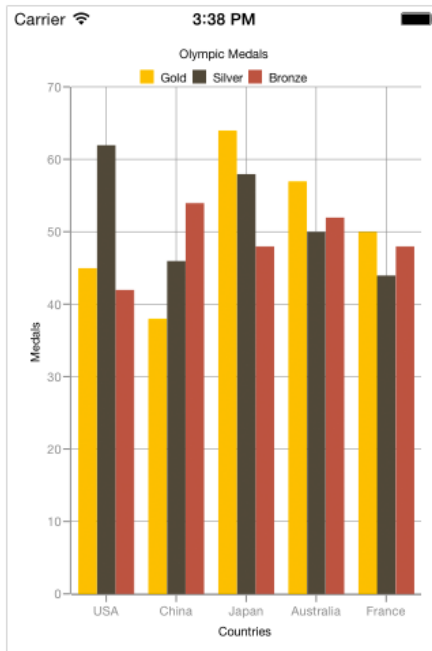
## XML

```
<chart:SfChart>
...
<chart:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Country"
YBindingPath="Value"/>
<chart:ColumnSeries ItemsSource="{Binding Data1}" XBindingPath="Country"
YBindingPath="Value"/>
<chart:ColumnSeries ItemsSource="{Binding Data2}" XBindingPath="Country"
YBindingPath="Value"/>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
ColumnSeries columnSeries = new ColumnSeries() {
    ItemsSource = Data,
    XBindingPath = "Country",
    YBindingPath = "Value"
};
ColumnSeries columnSeries1 = new ColumnSeries() {
    ItemsSource = Data1,
    XBindingPath = "Country",
    YBindingPath = "Value"
};
ColumnSeries columnSeries2 = new ColumnSeries() {
    ItemsSource = Data2,
    XBindingPath = "Country",
```

```
YBindingPath = "Value"
};
chart.Series.Add(columnSeries);
chart.Series.Add(columnSeries1);
chart.Series.Add(columnSeries2);
```



Following code snippet and screenshot shows how to apply the Y axis to individual series to plot different values.

### XML

```
<chart:SfChart.Series>
<chart:ColumnSeries Label="Revenue" ItemsSource="{Binding Demands}"
XBindingPath="XValue" YBindingPath="YValue" />
<chart:LineSeries Label="Customers" ItemsSource="{Binding Demands}"
XBindingPath="XValue" YBindingPath="YValue" >
<chart:LineSeries.YAxis>
<chart:NumericalAxis OpposedPosition="true" >
<chart:NumericalAxis.Title>
<chart:ChartAxisTitle Text = "Number of Customers" />
</chart:NumericalAxis.Title>
</chart:LineSeries.YAxis>
</chart:LineSeries>
</chart:SfChart.Series>
```

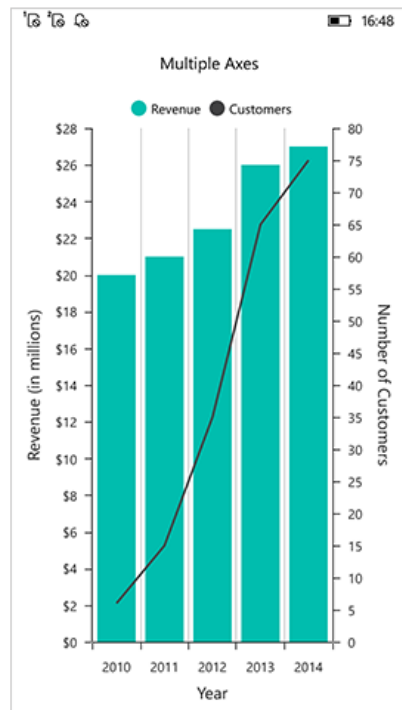
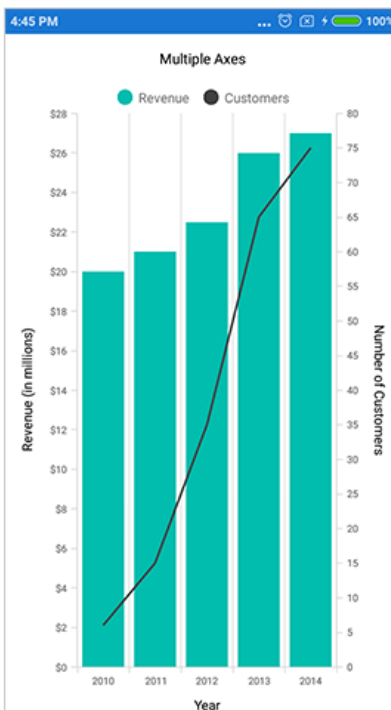
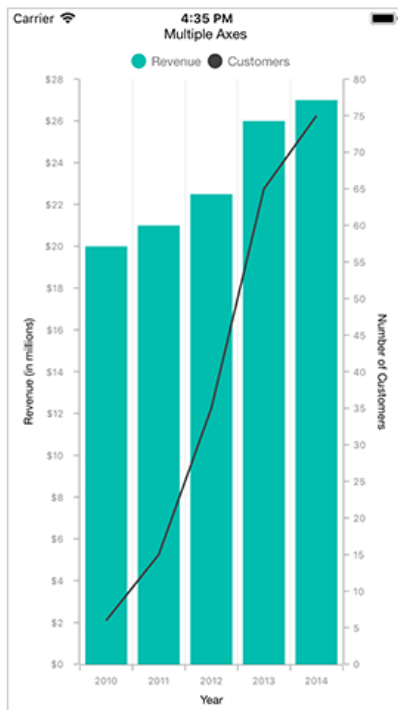
### C#

```
ColumnSeries series = new ColumnSeries();
series.ItemsSource = model.Demands;
series.XBindingPath = "XValue";
series.YBindingPath = "YValue";
series.Label = "Revenue";
```

```

chart.Series.Add(series);
LineSeries lineSeries = new LineSeries();
lineSeries.ItemsSource = model.Demands;
lineSeries.XBindingPath = "XValue";
lineSeries.YBindingPath = "YValue";
lineSeries.Label = "Customers";
NumericalAxis yAxis = new NumericalAxis();
yAxis.OpposedPosition = true;
yAxis.Title.Text = "Number of Customers";
lineSeries.YAxis = yAxis;
chart.Series.Add(lineSeries);

```



### Combination Series

[SfChart](#) allows you to render the combination of different types of series.

### XML

```

<chart:SfChart>
...
<chart:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Month"
YBindingPath="Value"/>
<chart:LineSeries ItemsSource="{Binding Data1}" XBindingPath="Month"
YBindingPath="Value"/>
</chart:SfChart>

```

### C#

```

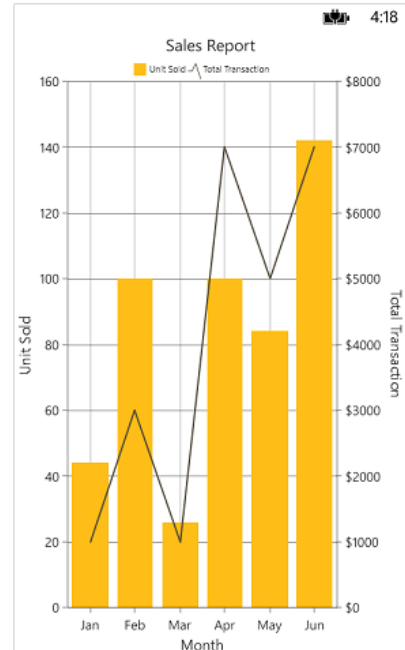
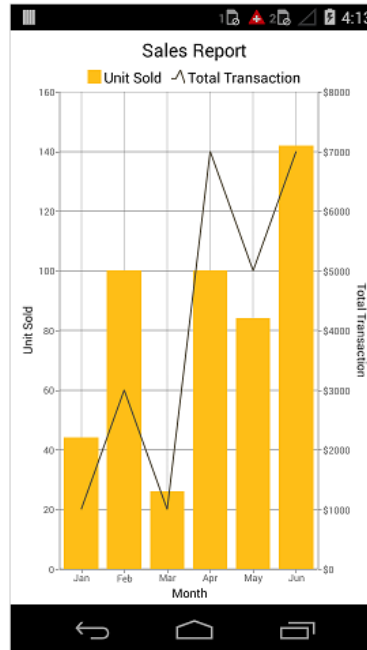
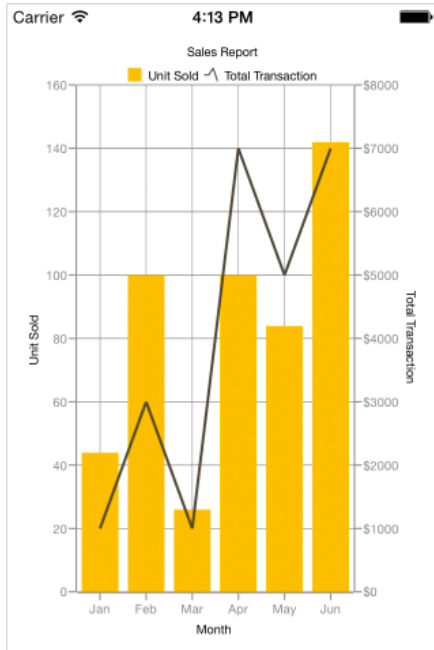
SfChart chart = new SfChart();
...
ColumnSeries columnSeries = new ColumnSeries() {
    ItemsSource = Data,

```

```

XBindingPath = "Month",
YBindingPath = "Value"
};
LineSeries lineSeries = new LineSeries() {
ItemsSource = Data1,
XBindingPath = "Month",
YBindingPath = "Value"
};
chart.Series.Add(columnSeries);
chart.Series.Add(lineSeries);

```



### Limitation of Combination Chart

- Bar, StackingBar, and StackingBar100 cannot be combined with the other Cartesian type series.
- Cartesian type series cannot be combined with Accumulation series (pie, doughnut, funnel, and pyramid).

When the combination of Cartesian and Accumulation series types are added to the [Series](#) property, the series which are similar to the first series will be rendered and other series will be ignored. Following code snippet illustrates this.

### XML

```

<chart:SfChart>
...
<chart:LineSeries ItemsSource="{Binding Data}" XBindingPath="Month"
YBindingPath="Value"/>
<chart:PieSeries ItemsSource="{Binding Data1}" XBindingPath="Month"
YBindingPath="Value"/>
</chart:SfChart>

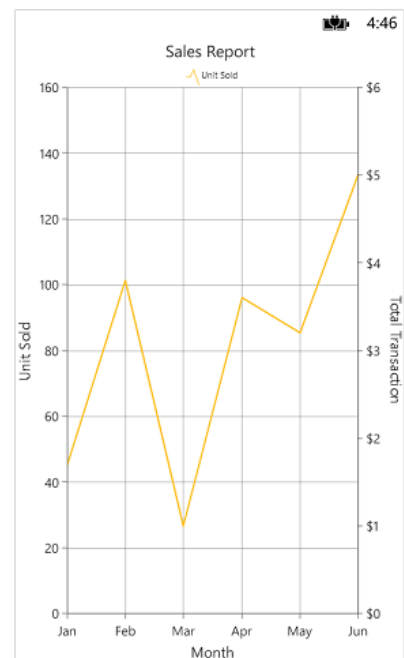
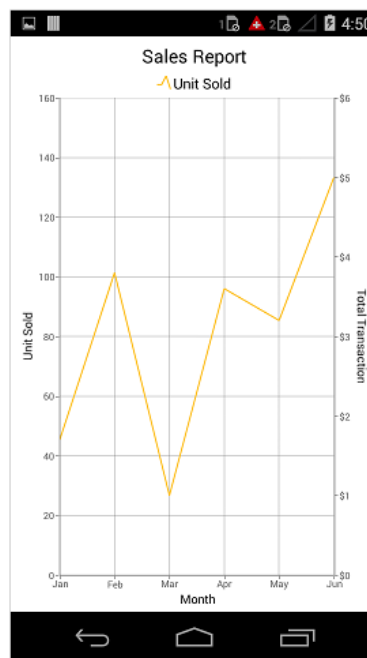
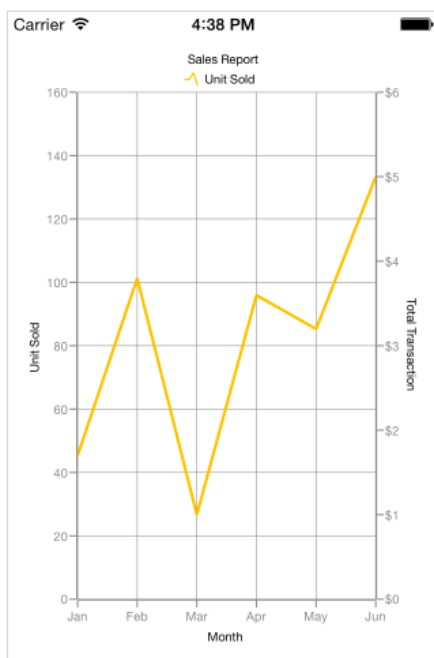
```

### C#

```

SfChart chart = new SfChart();
...
LineSeries lineSeries = new LineSeries() {
    ItemsSource = Data,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
PieSeries pieSeries = new PieSeries() {
    ItemsSource = Data1,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
chart.Series.Add(lineSeries);
chart.Series.Add(pieSeries);

```



### Grouping Stacked Series

You can group and stack the similar stacked series types using [GroupingLabel](#) property of stacked series. The stacked series which contains the same [GroupingLabel](#) will be stacked in a single group.

### XML

```

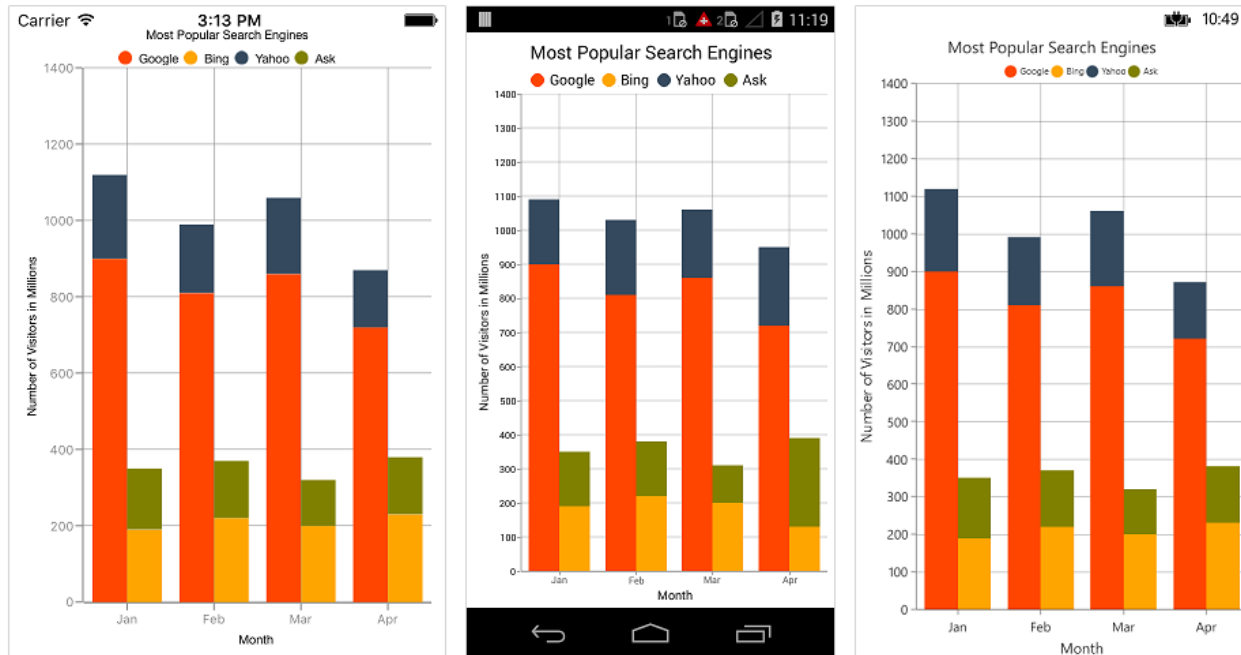
<chart:SfChart>
...
<chart:StackingColumnSeries ItemsSource="{Binding Data1}"
    GroupingLabel="GroupOne"
    Label="Google" XBindingPath="Month" YBindingPath="Value"/>
<chart:StackingColumnSeries ItemsSource="{Binding Data2}"
    GroupingLabel="GroupTwo"
    Label="Bing" XBindingPath="Month" YBindingPath="Value"/>
<chart:StackingColumnSeries ItemsSource="{Binding Data3}"
    GroupingLabel="GroupOne"
    Label="Yahoo" XBindingPath="Month" YBindingPath="Value"/>
<chart:StackingColumnSeries ItemsSource="{Binding Data4}"
    GroupingLabel="GroupTwo"

```

```
Label="Ask" XBindingPath="Month" YBindingPath="Value"/>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
StackingColumnSeries stackingColumnSeries1 = new StackingColumnSeries()
{
    ItemsSource = Data1,
    GroupingLabel = "GroupOne",
    Label = "Google",
    XBindingPath = "Month",
    YBindingPath = "Value"
};
StackingColumnSeries stackingColumnSeries2 = new StackingColumnSeries()
{
    ItemsSource = Data2,
    GroupingLabel = "GroupTwo",
    Label = "Bing",
    XBindingPath = "Month",
    YBindingPath = "Value"
};
StackingColumnSeries stackingColumnSeries3 = new StackingColumnSeries()
{
    ItemsSource = Data3,
    GroupingLabel = "GroupOne",
    Label = "Yahoo",
    XBindingPath = "Month",
    YBindingPath = "Value"
};
StackingColumnSeries stackingColumnSeries4 = new StackingColumnSeries()
{
    ItemsSource = Data4,
    GroupingLabel = "GroupTwo",
    Label = "Ask",
    XBindingPath = "Month",
    YBindingPath = "Value"
};
chart.Series.Add(stackingColumnSeries1);
chart.Series.Add(stackingColumnSeries2);
chart.Series.Add(stackingColumnSeries3);
chart.Series.Add(stackingColumnSeries4);
```



### Animation

[SfChart](#) provides animation support for data series. [Series](#) will be animated whenever the items source changes. Animation can be enabled by setting the [EnableAnimation](#) property to `true`. You can also control the duration of the animation using [AnimationDuration](#) property.

### XML

```
<chart:SfChart>
...
<chart:ColumnSeries
ItemsSource="{Binding ColumnData}"
EnableAnimation = "true"
AnimationDuration="0.8"
XBindingPath="Name"
YBindingPath="Value" />
</chart:SfChart>
```

### C#

```
ColumnSeries column = new ColumnSeries ();
column.ItemsSource = viewModel.ColumnData;
column.XBindingPath = "Name";
column.YBindingPath = "Value";
column.EnableAnimation = true;
column.AnimationDuration = 0.8;
```

### Transpose the Series (Vertical Chart)

The [IsTransposed](#) property of [CartesianSeries](#) is used to plot the chart vertically and view the data in a different perspective.

### XML

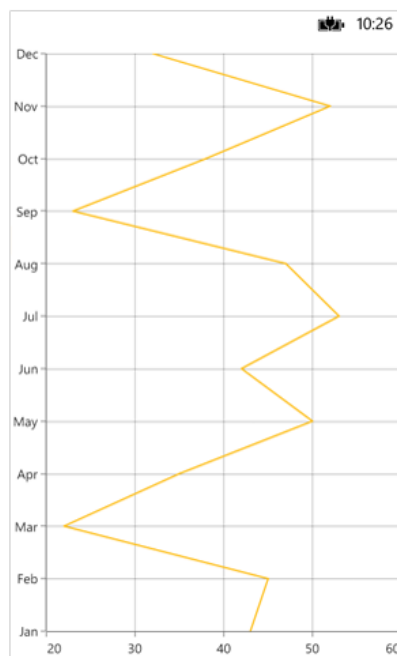
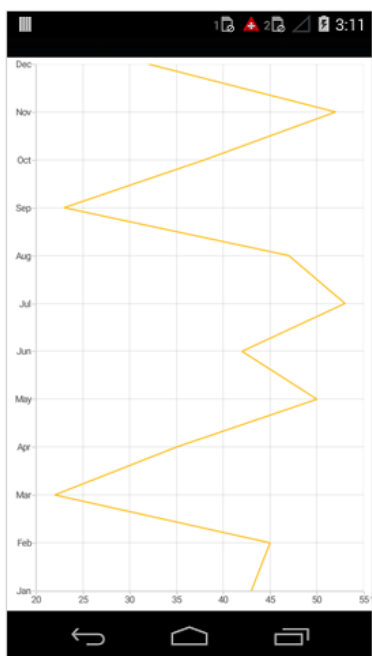
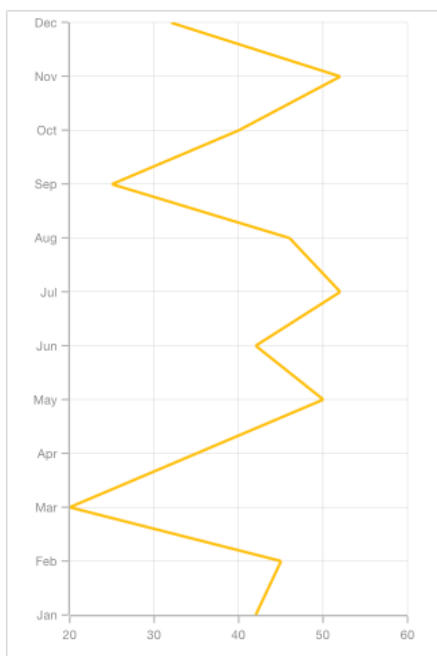
```
<chart:SfChart.Series>
```



```
<chart:LineSeries ItemsSource="{Binding Data}" XBindingPath="Month"
YBindingPath="Value" IsTransposed="True"/>
</chart:SfChart.Series>
```

**C#**

```
SfChart chart = new SfChart();
...
LineSeries lineSeries = new LineSeries();
lineSeries.XBindingPath = "Month";
lineSeries.YBindingPath = "Value";
lineSeries.ItemsSource = Data;
lineSeries.IsTransposed = true;
chart.Series.Add(lineSeries);
```

**Methods**

[GetDataPointIndex\(float pointX, float pointY\)](#)

The [GetDataPointIndex](#) method is used to get the actual data point index for corresponding screen point.

**C#**

```
ColumnSeries series = new ColumnSeries();
int index = series.GetDataPointIndex(400, 400);
```

**Note:** The output of this method will be -1 if there is no data point under the given x and y positions.

**Chart Types****Line Chart**

To render a line chart, create an instance of [LineSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the appearance.

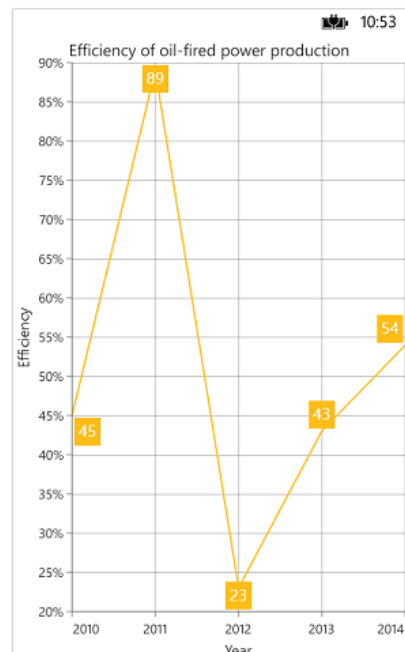
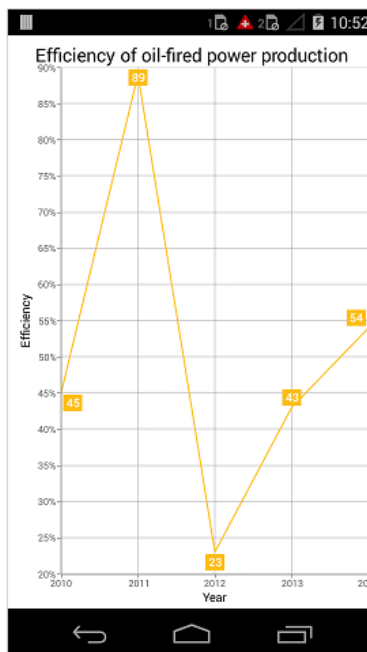
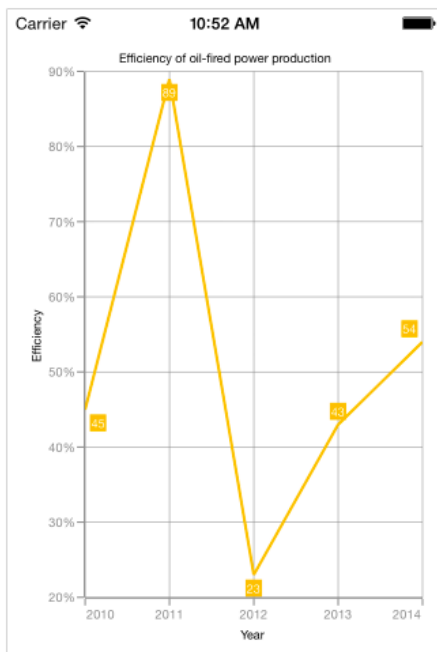
- [Color](#) – used to change the color of the line.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the line.

### XML

```
<chart:SfChart>
...
<chart:LineSeries ItemsSource="{Binding Data}" XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
LineSeries lineSeries = new LineSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(lineSeries);
```



### Fast Line Chart

[FastLineSeries](#) is a line chart, but it loads faster than [LineSeries](#). You can use this when there are large number of points to be loaded in chart. To render a fast line chart, create an instance of [FastLineSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the fast line segment appearance.

- [Color](#) – used to change the color of the series.

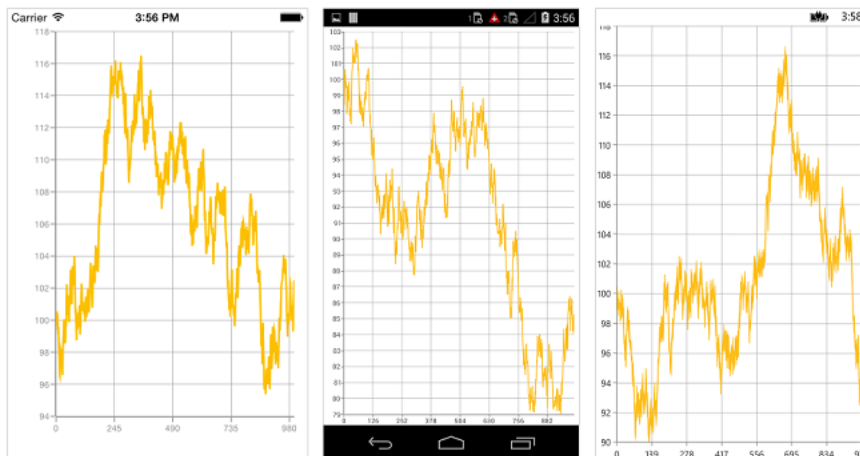
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.

**XML**

```
<chart:SfChart>
...
<chart:FastLineSeries ItemsSource="{Binding Data}" XBindingPath="XValue"
YBindingPath="YValue"/>
</chart:SfChart>
```

**C#**

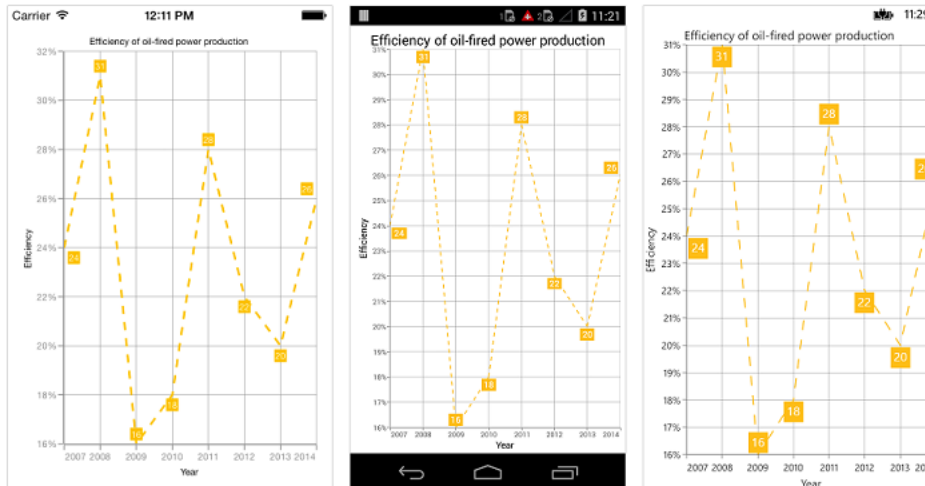
```
SfChart chart = new SfChart();
...
FastLineSeries fastLineSeries = new FastLineSeries()
{
    ItemsSource = Data,
    XBindingPath = "XValue",
    YBindingPath = "YValue"
};
chart.Series.Add(fastLineSeries);
```

*Dashed Lines*

[StrokeDashArray](#) property of the [FastLineSeries](#) is used to render fast line series with dashes.

**C#**

```
[C#]
FastLineSeries fastLineSeries = new FastLineSeries()
{
    ItemsSource = Data,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
fastLineSeries.StrokeDashArray = new double[2] { 2, 3 };
```



### EnableAntiAliasing

Since [FastLineSeries](#) can be loaded with a large number of points, the rendering of series should be smooth. This requirement can be achieved by setting [EnableAntiAliasing](#) property of [FastLineSeries](#) as false.

### Area Chart

To render an area chart, create an instance of [AreaSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.

### XML

```
<chart:SfChart>
...
<chart:AreaSeries ItemsSource="{Binding Data}" XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
AreaSeries areaSeries = new AreaSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(areaSeries);
```



### Spline Area Chart

To render a spline area chart, create an instance of [SplineAreaSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the spline area appearance.

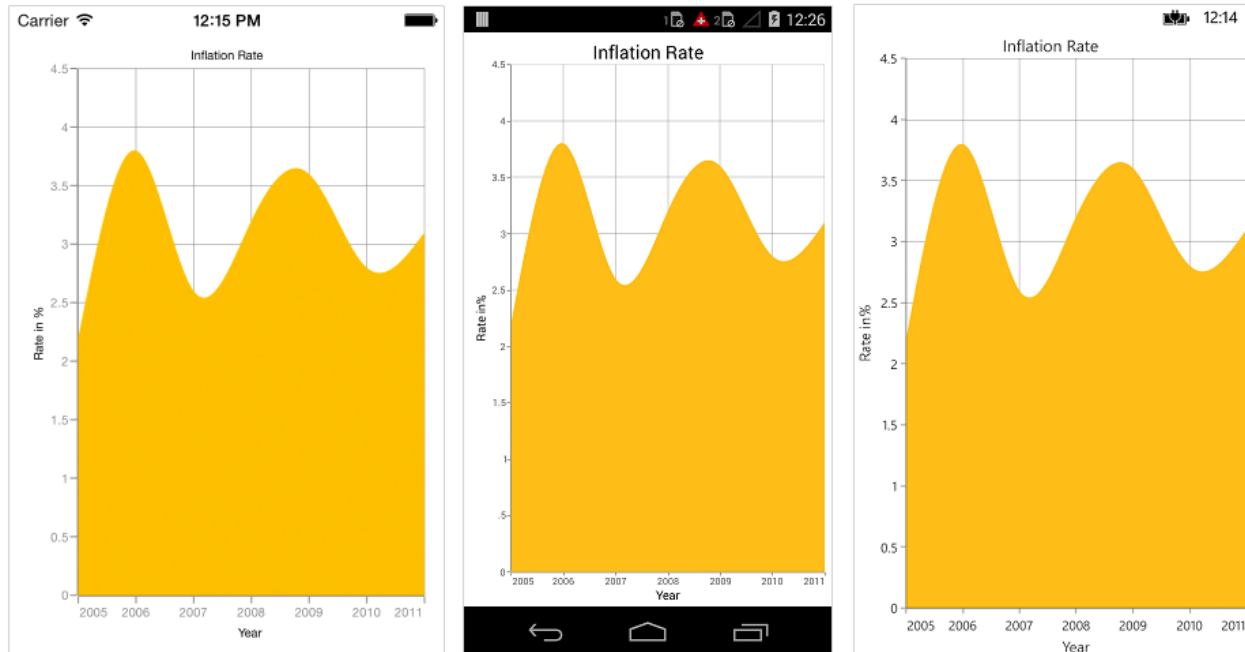
- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.

### XML

```
<chart:SfChart>
...
<chart:SplineAreaSeries ItemsSource="{Binding Data}" XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
SplineAreaSeries splineAreaSeries = new SplineAreaSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(splineAreaSeries);
```



### Spline Rendering Types

[SplineType](#) allows you to change the spline area curve in series.

The following types are used in [SplineAreaSeries](#) as

- [Natural](#)
- [Monotonic](#)
- [Cardinal](#)
- [Clamped](#)

By default [SplineType](#) value is [Natural](#).

The following code shows how to set the [SplineType](#) value as [Cardinal](#)

### XML

```
<chart:SfChart>
...
<chart:SplineAreaSeries ItemsSource="{Binding Data}" XBindingPath="Month"
YBindingPath="Value" SplineType="Cardinal" />
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
SplineAreaSeries splineAreaSeries = new SplineAreaSeries()
{
    ItemsSource = Data,
    XBindingPath = "Month",
    YBindingPath = "Value",
    SplineType = SplineType.Cardinal
};
```

```
chart.Series.Add(splineAreaSeries);
```

### Step Area Chart

To render a step area chart, create an instance of [StepAreaSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the appearance.

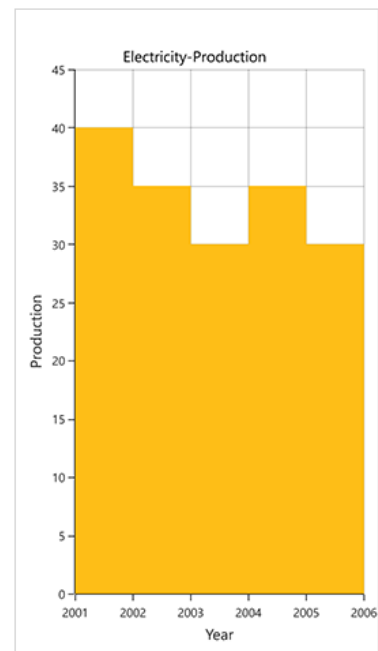
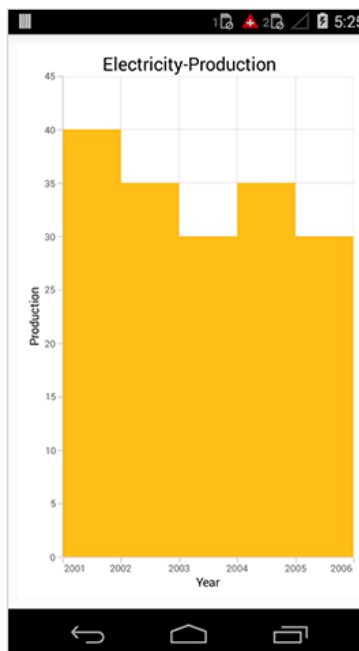
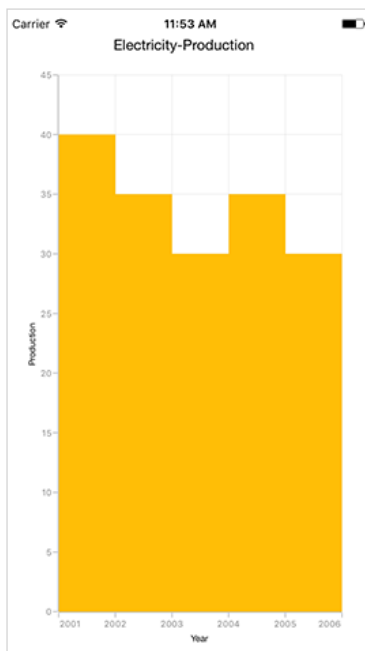
- [Color](#) - used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) - used to change the stroke width of the series.
- [StrokeColor](#) - used to change the stroke color of the series.

### XML

```
<chart:SfChart>
...
<chart:StepAreaSeries ItemsSource="{Binding Data}" XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
StepAreaSeries stepAreaSeries = new StepAreaSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(stepAreaSeries);
```



### Range Area Chart

To render a range area chart, create an instance of [RangeAreaSeries](#) and add to the [Series](#) collection property of [SfChart](#).

Since the [RangeAreaSeries](#) requires two Y values for a point, your data should contain high and low values. High and low value specifies the maximum and minimum range of the point.

There are two ways you can provide data to range area chart,

1. You can use [ChartDataPoint's](#) three parameter constructor to pass [XValue](#), [High](#) and [Low](#) values to [RangeAreaSeries](#),

#### C#

```
[C#]
SfChart chart = new SfChart();
...
ObservableCollection<ChartDataPoint> data = new
ObservableCollection<ChartDataPoint>()
{
    new ChartDataPoint("Jan/10", 30, 18),
    new ChartDataPoint("Feb/10", 24, 12),
    new ChartDataPoint("Mar/10", 29, 15),
    new ChartDataPoint("Apr/10", 24, 10),
    new ChartDataPoint("May/10", 30, 18),
    new ChartDataPoint("Jun/10", 24, 10),
};
RangeAreaSeries rangeAreaSeries = new RangeAreaSeries()
{
    ItemsSource = data
};
chart.Series.Add(rangeAreaSeries);
```

2. Or else you can use [High](#) and [Low](#) properties of [RangeAreaSeries](#) to map the high and low values from custom object to chart.

#### XML

```
<chart:SfChart>
...
<chart:SfChart.Series>
<chart:RangeAreaSeries ItemsSource="{Binding RangeAreaData}"
XBindingPath="Name" High="High" Low="Low"/>
</chart:SfChart.Series>
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart();
...
RangeAreaSeries rangeAreaSeries = new RangeAreaSeries()
{
    ItemsSource = RangeAreaData,
    XBindingPath = "Name",
    High = "High",
    Low = "Low"
```



```
};
chart.Series.Add(rangeAreaSeries);
```

You can use the following properties to customize the appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.



### Spline Range Area Chart

To render a spline range area chart, create an instance of the [SplineRangeAreaSeries](#), and add that instance to the [Series](#) collection property of [SfChart](#).

Since the [SplineRangeAreaSeries](#) requires two Y values for a point, data should contain high and low values. The high and low values specify the maximum and minimum ranges of a point.

The data can be provided to a spline range area chart by using the following two ways:

1. Using the [ChartDataPoint's](#) three parameter constructor to pass [XValue](#), [High](#), and [Low](#) values to the [SplineRangeAreaSeries](#).

### C#

```
[C#]
SfChart chart = new SfChart();
...
ObservableCollection<ChartDataPoint> data = new
ObservableCollection<ChartDataPoint>()
{
    new ChartDataPoint("Jan", 30, 18),
    new ChartDataPoint("Feb", 24, 12),
    new ChartDataPoint("Mar", 29, 15),
```

```
new ChartDataPoint("Apr", 24, 10),
new ChartDataPoint("May", 30, 18),
new ChartDataPoint("Jun", 24, 10),
};
SplineRangeAreaSeries splineRangeAreaSeries = new SplineRangeAreaSeries()
{
    ItemsSource = data
};
chart.Series.Add(splineRangeAreaSeries);
```

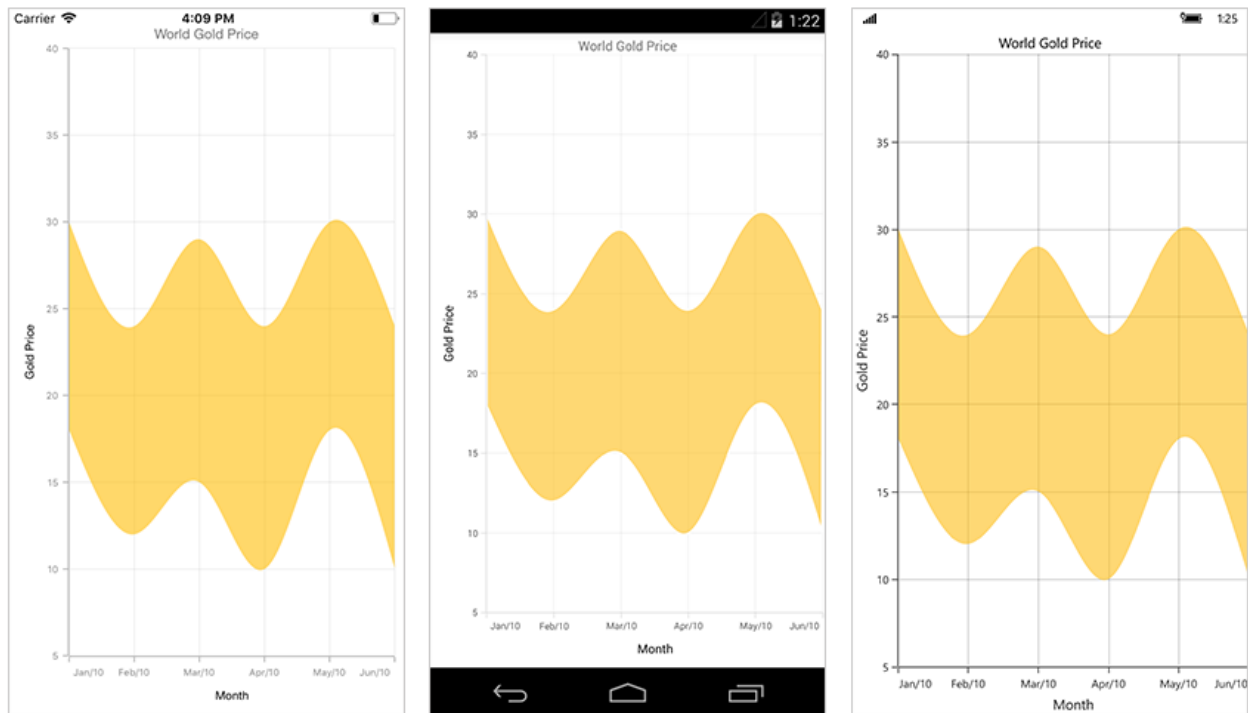
2.Or else, using the [high](#) and [low](#) properties of [SplineRangeAreaSeries](#) to map the high and low values from custom object to chart.

### XML

```
<chart:SfChart>
...
<chart:SfChart.Series>
<chart:SplineRangeAreaSeries ItemsSource="{Binding SplineRangeAreaData}"
XBindingPath="Name" High="High" Low="Low"/>
</chart:SfChart.Series>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
SplineRangeAreaSeries splineRangeAreaSeries = new SplineRangeAreaSeries()
{
    ItemsSource = SplineRangeAreaData,
    XBindingPath = "Name",
    High = "High",
    Low = "Low"
};
chart.Series.Add(splineRangeAreaSeries);
```



### *Spline Rendering Types*

[SplineType](#) property allows you to change the spline range area curve in series.

The following types can be used for SplineRangeAreaSeries

*Natural Monotonic Cardinal Clamped*

By default [SplineType](#) value is Natural.

The following code shows how to set the [SplineType](#) value as Cardinal.

### **XML**

```
<chart:SfChart>
...
<chart:SplineRangeAreaSeries ItemsSource="{Binding SplineRangeAreaData}"
XBindingPath="Name" High="High" Low="Low" SplineType="Cardinal"/>
</chart:SfChart>
```

### **C#**

```
SplineRangeAreaSeries splineRangeAreaSeries = new SplineRangeAreaSeries
{
    ItemsSource = SplineRangeAreaData,
    XBindingPath = "Name",
    High = "High",
    Low = "Low",
    SplineType = SplineType.Cardinal
};
chart.Series.Add(splineRangeAreaSeries);
```

### Stacked Area Chart

To render a stacked area chart, create an instance of [StackingAreaSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the stacked area appearance.

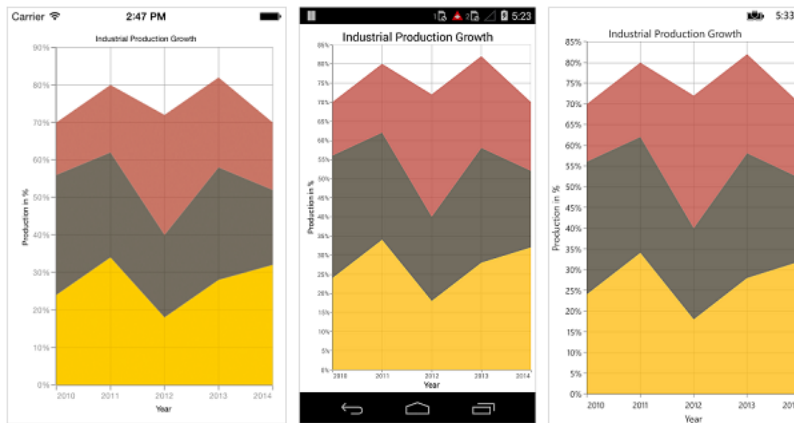
- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.

### XML

```
<chart:SfChart>
...
<chart:StackingAreaSeries ItemsSource="{Binding Data1}" XBindingPath="Year"
YBindingPath="Value"/>
<chart:StackingAreaSeries ItemsSource="{Binding Data2}" XBindingPath="Year"
YBindingPath="Value"/>
<chart:StackingAreaSeries ItemsSource="{Binding Data3}" XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
StackingAreaSeries stackingAreaSeries1 = new StackingAreaSeries()
{
    ItemsSource = Data1,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
StackingAreaSeries stackingAreaSeries2 = new StackingAreaSeries()
{
    ItemsSource = Data2,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
StackingAreaSeries stackingAreaSeries3 = new StackingAreaSeries()
{
    ItemsSource = Data3,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(stackingAreaSeries1);
chart.Series.Add(stackingAreaSeries2);
chart.Series.Add(stackingAreaSeries3);
```



### 100% Stacked Area Chart

To render a 100% stacked area chart, create an instance of [StackingArea100Series](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the 100% stacked area appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.

### XML

```
<chart:SfChart>
...
<chart:StackingArea100Series ItemsSource="{Binding Data1}"
XBindingPath="Year"
YBindingPath="Value"/>
<chart:StackingArea100Series ItemsSource="{Binding Data2}"
XBindingPath="Year"
YBindingPath="Value"/>
<chart:StackingArea100Series ItemsSource="{Binding Data3}"
XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

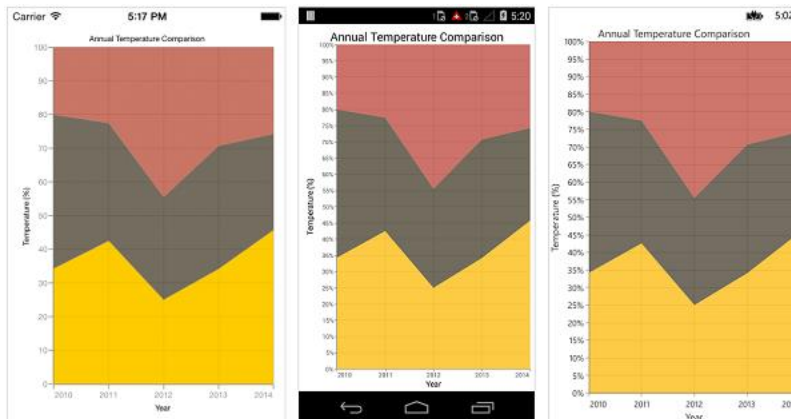
### C#

```
SfChart chart = new SfChart();
...
StackingArea100Series stackingArea100Series1 = new StackingArea100Series()
{
    ItemsSource = Data1,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
StackingArea100Series stackingArea100Series2 = new StackingArea100Series()
{
    ItemsSource = Data2,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
```

```

};
StackingArea100Series stackingArea100Series3 = new StackingArea100Series()
{
    ItemsSource = Data3,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(stackingArea100Series1);
chart.Series.Add(stackingArea100Series2);
chart.Series.Add(stackingArea100Series3);

```



### Column Chart

To render a column chart, create an instance of [ColumnSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [CornerRadius](#) - used to add the rounded corners to the rectangle. The [TopLeft](#), [TopRight](#), [BottomLeft](#) and [BottomRight](#) of [ChartCornerRadius](#) properties are used to set the radius value for each corner.
- [DataMarkerPosition](#) - used to position the data marker at [Bottom](#), [Top](#) and [Center](#) of the rectangle.
- [Spacing](#) - used to change the spacing between two segments. The default value of spacing is 0, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available space, respectively.
- [Width](#) - used to change the width of the rectangle. The default value of the width is 0.8, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available width, respectively.

### XML

```

<chart:SfChart>
...
<chart:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Country"
YBindingPath="Value"/>
</chart:SfChart>

```

**C#**

```
SfChart chart = new SfChart();
...
ColumnSeries columnSeries = new ColumnSeries()
{
    ItemsSource = Data,
    XBindingPath = "Country",
    YBindingPath = "Value"
};
chart.Series.Add(columnSeries);
```

*Overlapped placement*

By default, all the column series which has the same x and y axes are placed side by side in a chart. If you want place the series one over the other (overlapped), set the [SideBySideSeriesPlacement](#) property of [SfChart](#) to false and configure the [Width](#) property to differentiate the series. The following code snippet and screenshot illustrate the overlapped placement of column series.

**XML**

```
<chart:SfChart x:Name="Chart" SideBySideSeriesPlacement="False" >
    . . .
    <chart:SfChart.Series>
        <chart:ColumnSeries ItemsSource="{Binding Data1}" Label="2014"
            XBindingPath="Month" YBindingPath="Year2014">
        </chart:ColumnSeries>
        <chart:ColumnSeries Width="0.5" ItemsSource="{Binding Data2}" Label="2015"
            XBindingPath="Month" YBindingPath="Year2015" >
        </chart:ColumnSeries>
    </chart:SfChart.Series>
    . . .
</chart:SfChart>
```

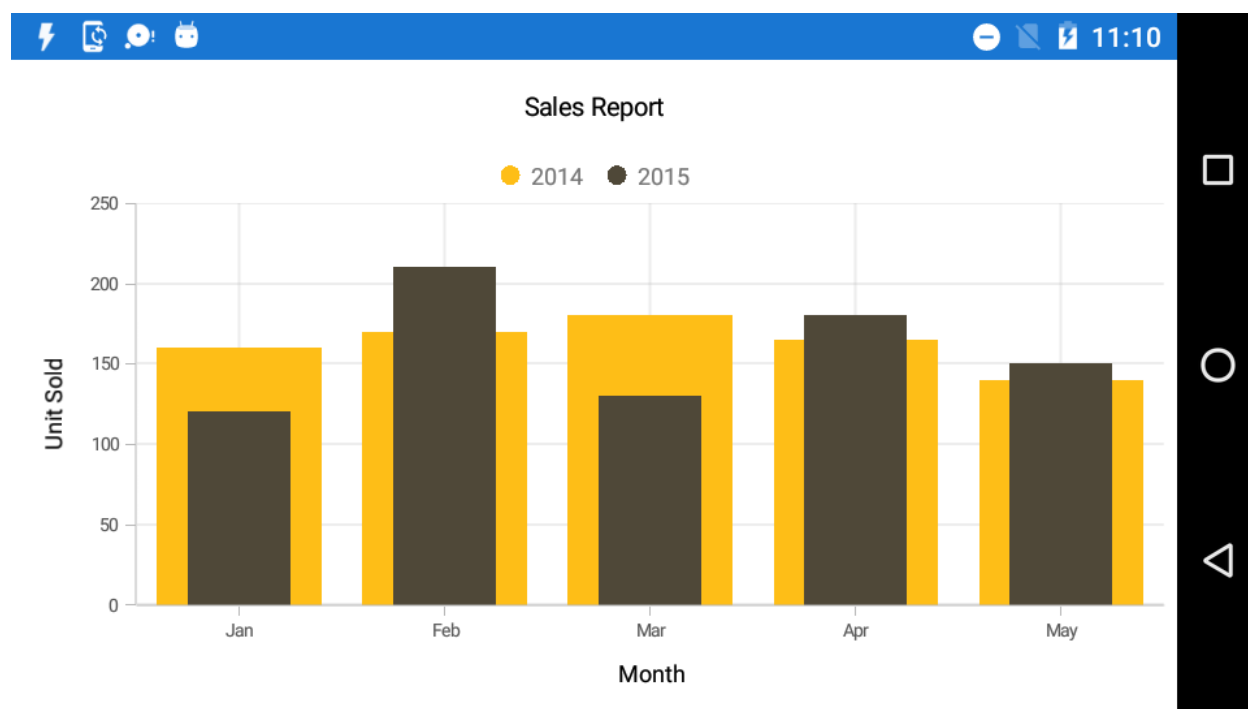
**C#**

```
SfChart chart = new SfChart()
```

```

{
    SideBySideSeriesPlacement = false
};
chart.PrimaryAxis = new CategoryAxis();
chart.SecondaryAxis = new NumericalAxis();
ColumnSeries series1 = new ColumnSeries()
{
    ItemsSource = view.Data1,
    XBindingPath = "Month",
    YBindingPath = "Year2014"
};
ColumnSeries series2 = new ColumnSeries()
{
    ItemsSource = view.Data2,
    XBindingPath = "Month",
    YBindingPath = "Year2015",
    Width="0.5"
};
chart.Series.Add(series1);
chart.Series.Add(series2);

```



### Histogram Chart

To render a histogram chart, create an instance of [HistogramSeries](#), and add it to the series collection of [SfChart](#).

Histogram chart provides a visual display of large amount of data that are difficult to understand in a tabular or data grid form.

You can customize intervals using the [Interval](#) property and collapse the normal distribution curve using the [ShowNormalDistributionCurve](#) property. You can use the following properties to customize the appearance.



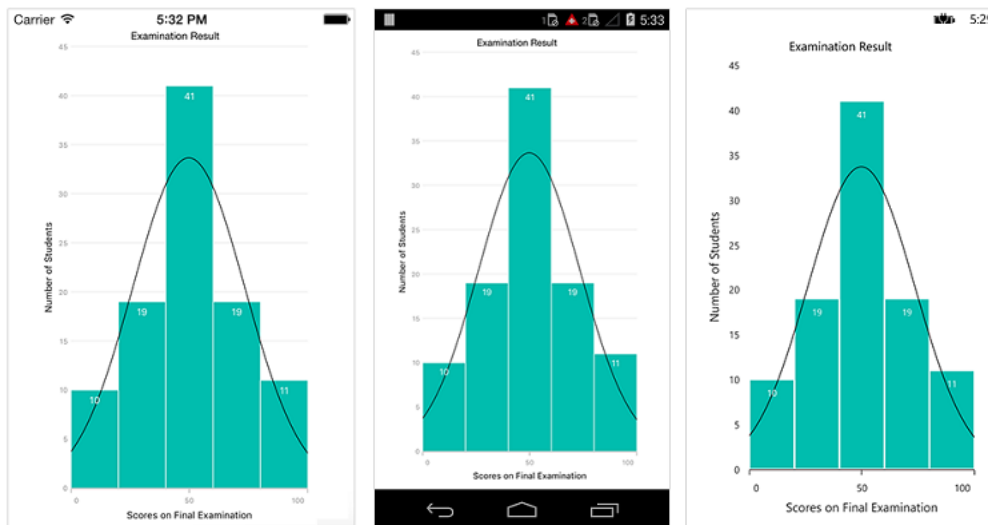
- [Color](#) – used to change the color of the series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [CurveColor](#) – used to change the color of the normal distribution curve.
- [DataMarkerPosition](#) - used to position the data marker at Bottom, Top and Center of the rectangle.

## XML

```
<chart:SfChart>
...
<chart:HistogramSeries ItemsSource="{Binding Data}" XBindingPath="XValue"
YBindingPath="YValue" Interval="20"/>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
HistogramSeries histogramSeries = new HistogramSeries()
{
    ItemsSource = Data,
    XBindingPath = "XValue",
    YBindingPath = "YValue",
    Interval = 20
};
chart.Series.Add(histogramSeries);
```



## Range Column Chart

To render a range column chart, create an instance of [RangeColumnSeries](#) and add to the [Series](#) collection property of [SfChart](#).

Since the [RangeColumnSeries](#) requires two Y values for a point, your data should contain high and low values. High and low value specifies the maximum and minimum range of the point.

There are two ways you can provide data to RangeColumn chart,

1.You can use [ChartDataPoint's](#) three parameter constructor to pass [XValue](#), [High](#) and [Low](#) values to [RangeColumnSeries](#),

### C#

```
[C#]
SfChart chart = new SfChart();
...
ObservableCollection<ChartDataPoint> data = new
ObservableCollection<ChartDataPoint>()
{
    new ChartDataPoint("Jan", 7.6, 1.8),
    new ChartDataPoint("Feb", 10, 3),
    new ChartDataPoint("Mar", 7.5, 1.7),
    new ChartDataPoint("Apr", 7.8, 4.5),
    new ChartDataPoint("May", 11.4, 5),
    new ChartDataPoint("Jun", 10.1, 4.2),
};
RangeColumnSeries rangeColumnSeries = new RangeColumnSeries()
{
    ItemsSource = data
};
chart.Series.Add(rangeColumnSeries);
```

2.Or else you can use [High](#) and [Low](#) properties of [RangeColumnSeries](#) to map the high and low values from custom object to chart.

### XML

```
<chart:SfChart>
...
<chart:RangeColumnSeries ItemsSource="{Binding Data}"
XBindingPath="Month"
High="Value1"
Low="Value2"/>
</chart:SfChart>
```

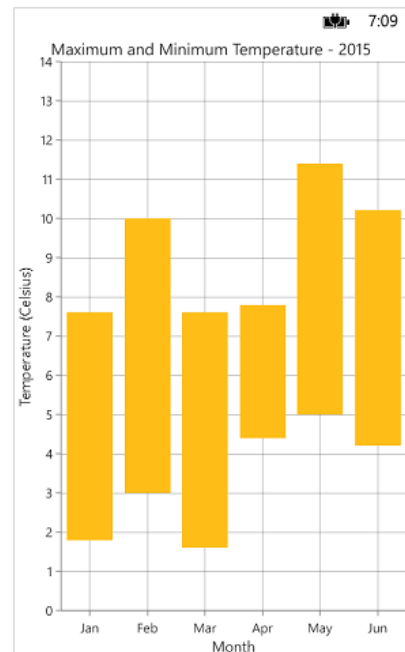
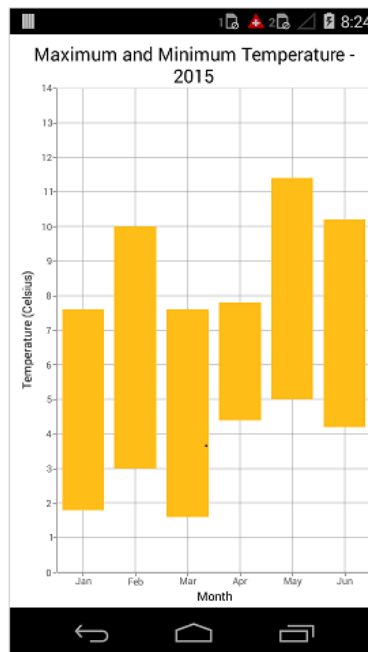
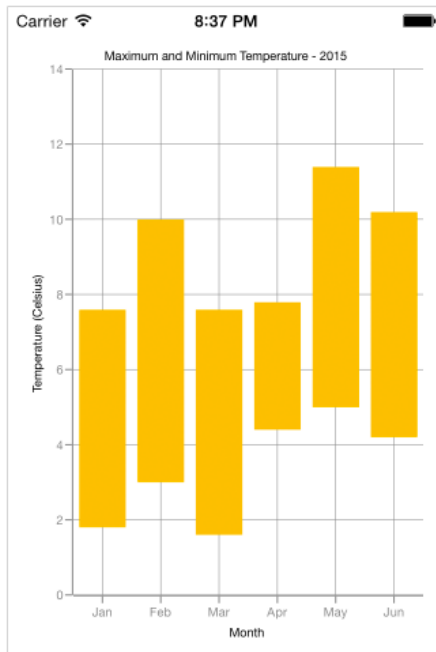
### C#

```
SfChart chart = new SfChart();
...
RangeColumnSeries rangeColumnSeries = new RangeColumnSeries()
{
    ItemsSource = Data,
    XBindingPath = "Month",
    High = "Value1",
    Low = "Value2"
};
chart.Series.Add(rangeColumnSeries);
```

Following properties are used to customize the range column segment appearance,

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.

- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [CornerRadius](#) - used to add the rounded corners to the rectangle. The [TopLeft](#), [TopRight](#), [BottomLeft](#) and [BottomRight](#) of [ChartCornerRadius](#) properties are used to set the radius value for each corner.
- [Spacing](#) - used to change the spacing between two segments. The default value of spacing is 0, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available space, respectively.
- [Width](#) - used to change the width of the rectangle. The default value of the width is 0.8, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available width, respectively.



### Stacked Column Chart

To render a stacked column chart, create an instance of [StackingColumnSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the stacked column segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [CornerRadius](#) - used to add the rounded corners to the rectangle. The [TopLeft](#), [TopRight](#), [BottomLeft](#) and [BottomRight](#) of [ChartCornerRadius](#) properties are used to set the radius value for each corner.
- [DataMarkerPosition](#) - used to position the data marker at [Bottom](#), [Top](#) and [Center](#) of the rectangle.
- [Spacing](#) - used to change the spacing between two segments. The default value of spacing is 0, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available space, respectively.

- [Width](#) - used to change the width of the rectangle. The default value of the width is 0.8, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available width, respectively.

### XML

```
<chart:SfChart>
...
<chart:StackingColumnSeries ItemsSource="{Binding Data1}"
XBindingPath="Month"
YBindingPath="Value"/>
<chart:StackingColumnSeries ItemsSource="{Binding Data2}"
XBindingPath="Month"
YBindingPath="Value"/>
<chart:StackingColumnSeries ItemsSource="{Binding Data3}"
XBindingPath="Month"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
StackingColumnSeries stackingColumnSeries1 = new StackingColumnSeries()
{
    ItemsSource = Data1,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
StackingColumnSeries stackingColumnSeries2 = new StackingColumnSeries()
{
    ItemsSource = Data2,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
StackingColumnSeries stackingColumnSeries3 = new StackingColumnSeries()
{
    ItemsSource = Data3,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
chart.Series.Add(stackingColumnSeries1);
chart.Series.Add(stackingColumnSeries2);
chart.Series.Add(stackingColumnSeries3);
```



### 100% Stacked Column Chart

To render a 100% stacked column chart, create an instance of [StackingColumn100Series](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the series appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [CornerRadius](#) - used to add the rounded corners to the rectangle. The [TopLeft](#), [TopRight](#), [BottomLeft](#) and [BottomRight](#) of [ChartCornerRadius](#) properties are used to set the radius value for each corner.
- [DataMarkerPosition](#) - used to position the data marker at [Bottom](#), [Top](#) and [Center](#) of the rectangle.
- [Spacing](#) - used to change the spacing between two segments. The default value of spacing is 0, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available space, respectively.
- [Width](#) - used to change the width of the rectangle. The default value of the width is 0.8, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available width, respectively.

### XML

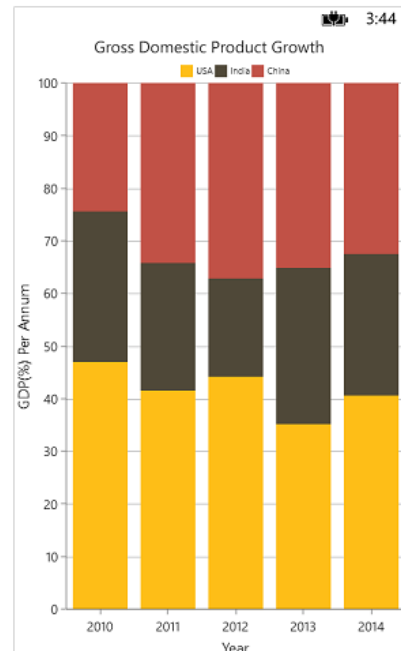
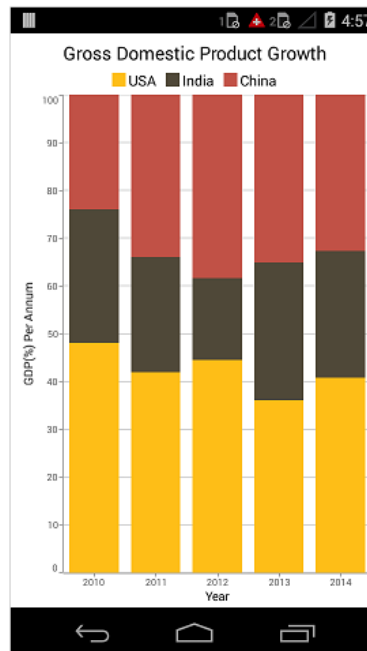
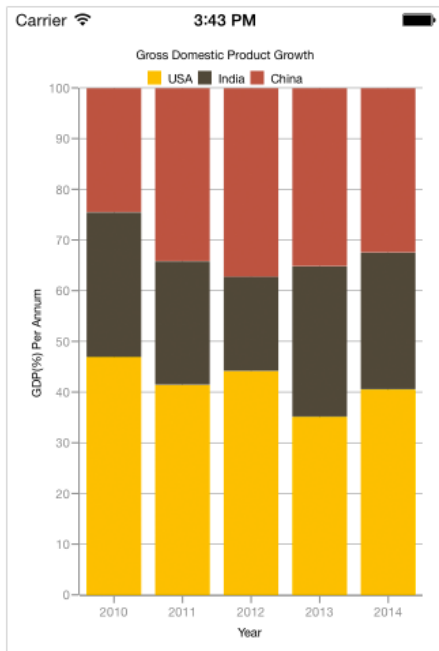
```
<chart:SfChart>
...
<chart:StackingColumn100Series ItemsSource="{Binding Data1}"
XBindingPath="Year"
YBindingPath="Value"/>
<chart:StackingColumn100Series ItemsSource="{Binding Data2}"
XBindingPath="Year"
YBindingPath="Value"/>
<chart:StackingColumn100Series ItemsSource="{Binding Data3}"
XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

**C#**

```

SfChart chart = new SfChart();
...
StackingColumn100Series stackingColumn100Series1 = new
StackingColumn100Series()
{
    ItemsSource = Data1,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
StackingColumn100Series stackingColumn100Series2 = new
StackingColumn100Series()
{
    ItemsSource = Data2,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
StackingColumn100Series stackingColumn100Series3 = new
StackingColumn100Series()
{
    ItemsSource = Data3,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(stackingColumn100Series1);
chart.Series.Add(stackingColumn100Series2);
chart.Series.Add(stackingColumn100Series3);

```

**Bar Chart**

To render a bar chart, create an instance of [BarSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the bar segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [CornerRadius](#) - used to add the rounded corners to the rectangle. The [TopLeft](#), [TopRight](#), [BottomLeft](#) and [BottomRight](#) of [ChartCornerRadius](#) properties are used to set the radius value for each corner.
- [DataMarkerPosition](#) - used to position the data marker at left, right and center of the rectangle.
- [Spacing](#) - used to change the spacing between two segments. The default value of spacing is 0, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available space, respectively.
- [Width](#) - used to change the width of the rectangle. The default value of the width is 0.8, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available width, respectively.

### **XML**

```
<chart:SfChart>
...
<chart:BarSeries ItemsSource="{Binding Data}" XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

### **C#**

```
SfChart chart = new SfChart();
...
BarSeries barSeries = new BarSeries ()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(barSeries);
```



### Stacked Bar Chart

To render a stacked bar chart, create an instance of [StackingBarSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the stacked bar segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [CornerRadius](#) - used to add the rounded corners to the rectangle. The [TopLeft](#), [TopRight](#), [BottomLeft](#) and [BottomRight](#) of [ChartCornerRadius](#) properties are used to set the radius value for each corner.
- [DataMarkerPosition](#) - used to position the data marker at left, right and center of the rectangle.
- [Spacing](#) - used to change the spacing between two segments. The default value of spacing is 0 and the value ranges from 0 to 1. Here 1 and 0 corresponds to 100% and 0% of available space respectively.
- [Width](#) - used to change the width of the rectangle. The default value of width is 0.8 and the value ranges from 0 to 1. Here 1 and 0 corresponds to 100% and 0% of available width respectively.

### XML

```
<chart:SfChart>
...
<chart:StackingBarSeries ItemsSource="{Binding Data1}" XBindingPath="Month"
YBindingPath="Value"/>
<chart:StackingBarSeries ItemsSource="{Binding Data2}" XBindingPath="Month"
YBindingPath="Value"/>
<chart:StackingBarSeries ItemsSource="{Binding Data3}" XBindingPath="Month"
YBindingPath="Value"/>
</chart:SfChart>
```



**C#**

```

SfChart chart = new SfChart();
...
StackingBarSeries stackingBarSeries1 = new StackingBarSeries()
{
    ItemsSource = Data1,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
StackingBarSeries stackingBarSeries2 = new StackingBarSeries()
{
    ItemsSource = Data2,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
StackingBarSeries stackingBarSeries3 = new StackingBarSeries()
{
    ItemsSource = Data3,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
chart.Series.Add(stackingBarSeries1);
chart.Series.Add(stackingBarSeries2);
chart.Series.Add(stackingBarSeries3);

```

**100% Stacked Bar Chart**

To render a 100% stacked bar chart, create an instance of [StackingBar100Series](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the series appearance.

- [Color](#) – used to change the color of the series.

- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [CornerRadius](#) - used to add the rounded corners to the rectangle. The [TopLeft](#), [TopRight](#), [BottomLeft](#) and [BottomRight](#) of [ChartCornerRadius](#) properties are used to set the radius value for each corner.
- [DataMarkerPosition](#) - used to position the data marker at left, right and center of the rectangle.
- [Spacing](#) - used to change the spacing between two segments. The default value of spacing is 0 and the value ranges from 0 to 1. Here 1 and 0 corresponds to 100% and 0% of available space respectively.
- [Width](#) - used to change the width of the rectangle. The default value of width is 0.8 and the value ranges from 0 to 1. Here 1 and 0 corresponds to 100% and 0% of available width respectively.

### XML

```
<chart:SfChart>
...
<chart:StackingBar100Series ItemsSource="{Binding Data1}"
XBindingPath="Year"
YBindingPath="Value"/>
<chart:StackingBar100Series ItemsSource="{Binding Data2}"
XBindingPath="Year"
YBindingPath="Value"/>
<chart:StackingBar100Series ItemsSource="{Binding Data3}"
XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
StackingBar100Series stackingBar100Series1 = new StackingBar100Series()
{
    ItemsSource = Data1,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
StackingBar100Series stackingBar100Series2 = new StackingBar100Series()
{
    ItemsSource = Data2,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
StackingBar100Series stackingBar100Series3 = new StackingBar100Series()
{
    ItemsSource = Data3,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(stackingBar100Series1);
chart.Series.Add(stackingBar100Series2);
chart.Series.Add(stackingBar100Series3);
```



## Spline Chart

To render a spline chart, create an instance of [SplineSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the spline segment appearance.

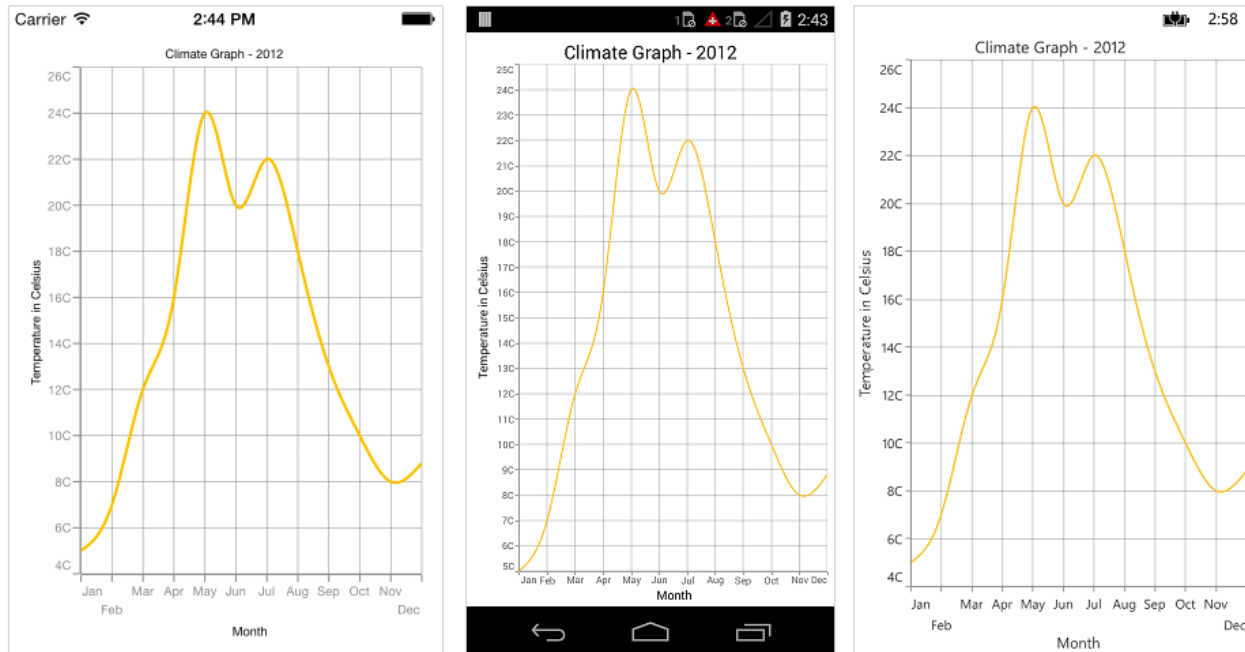
- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.

## XML

```
<chart:SfChart>
...
<chart:SplineSeries ItemsSource="{Binding Data}" XBindingPath="Month"
YBindingPath="Value"/>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
SplineSeries splineSeries = new SplineSeries()
{
    ItemsSource = Data,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
chart.Series.Add(splineSeries);
```



### Dashed Lines

[StrokeDashArray](#) property of the [SplineSeries](#) is used to render spline series with dashes.

### XML

```
<chart:SplineSeries ItemsSource="{Binding Data}" XBindingPath="Month"
YBindingPath="Value">
  <chart:SplineSeries.StrokeDashArray>
    <x:Array Type="{x:Type x:Double}">
      <sys:Double>5</sys:Double>
      <sys:Double>6</sys:Double>
    </x:Array>
  </chart:SplineSeries.StrokeDashArray>
</chart:SplineSeries>
```

### C#

```
SplineSeries splineSeries = new SplineSeries()
{
    ItemsSource = Data,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
splineSeries.StrokeDashArray = new double[2] { 5, 6 };
chart.Series.Add(splineSeries);
```

### Spline Rendering Types

[SplineType](#) allows you to change the spline curve in series.

The following types are used in [SplineSeries](#) as

- Natural
- Monotonic

- Cardinal
- Clamped

By default [SplineType](#) value is [Natural](#).

The following code shows how to set the [SplineType](#) value as [Cardinal](#)

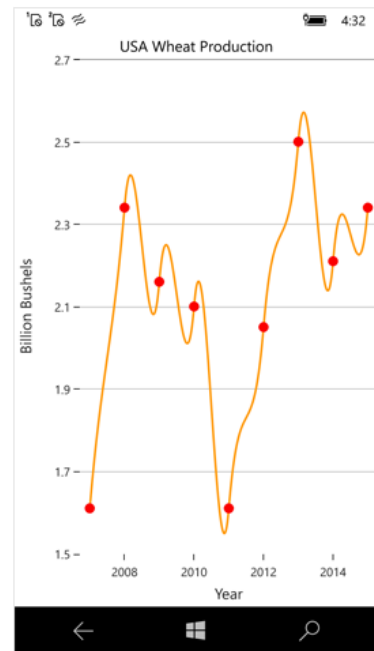
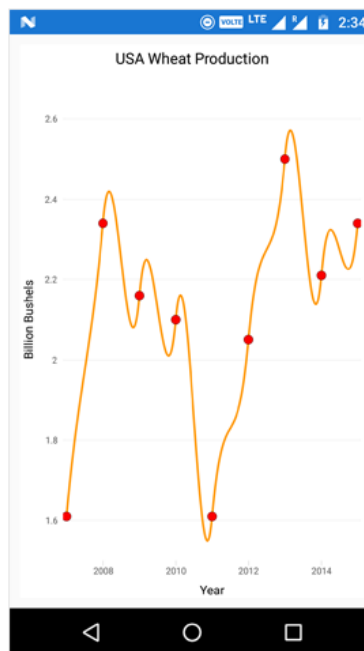
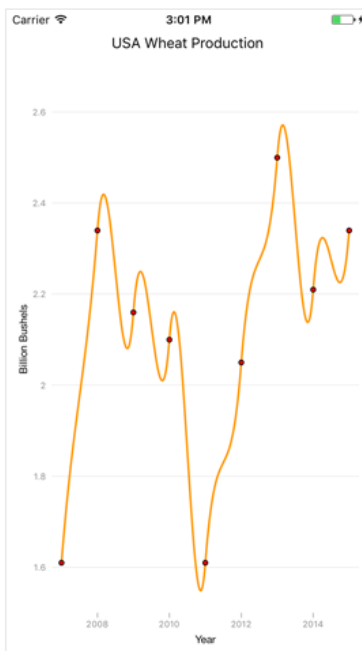
### XML

```
<chart:SfChart>
...
<chart:SplineSeries ItemsSource="{Binding Data}" XBindingPath="Month"
YBindingPath="Value" SplineType="Cardinal" />
</chart:SfChart>
```

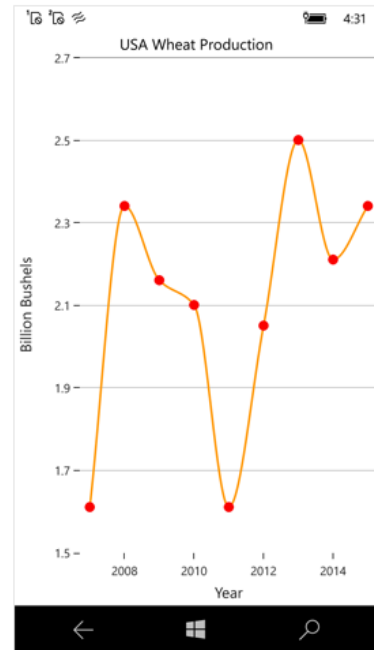
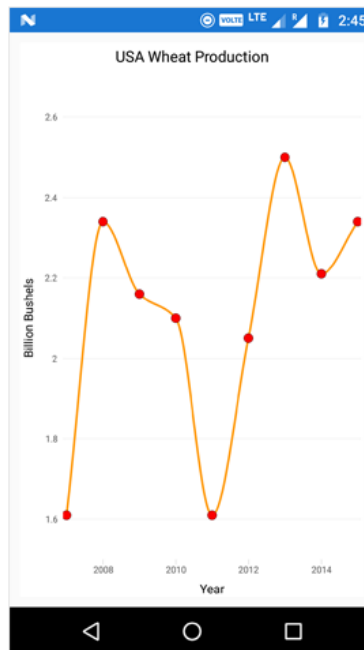
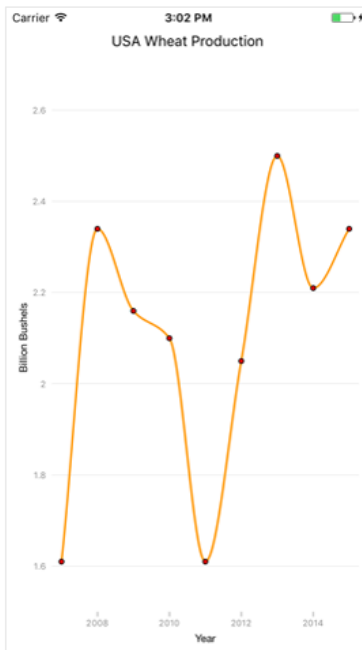
### C#

```
SfChart chart = new SfChart();
...
SplineSeries splineSeries = new SplineSeries()
{
    ItemsSource = Data,
    XBindingPath = "Month",
    YBindingPath = "Value",
    SplineType = SplineType.Cardinal
};
chart.Series.Add(splineSeries);
```

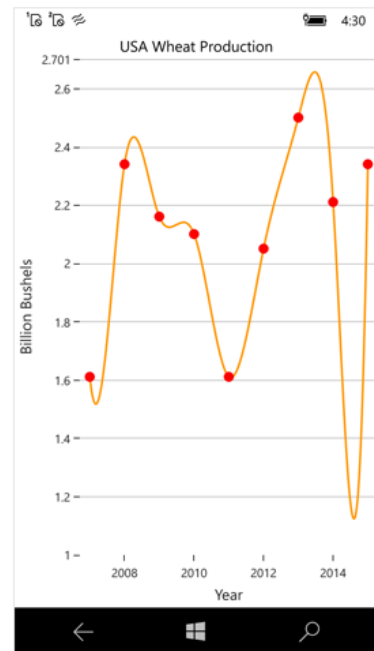
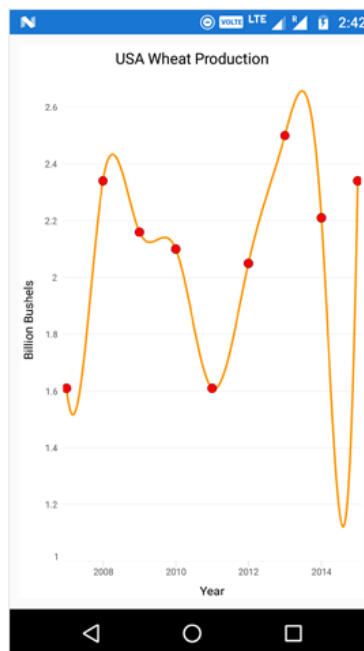
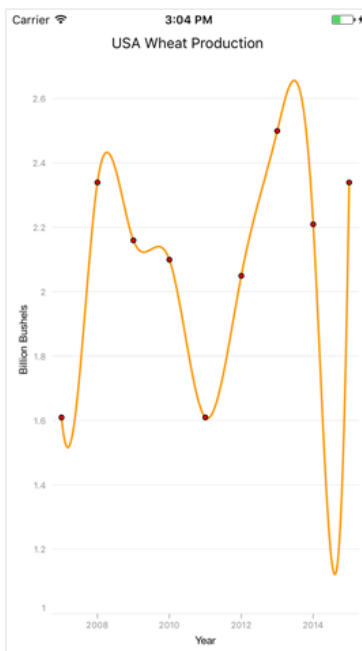
### Cardinal



### Monotonic



### Clamped



### Step Line Chart

To render a step line chart, create an instance of [StepLineSeries](#) and add to the [Series](#) collection property of

[SfChart](#). You can use the following properties to customize the spline segment appearance.

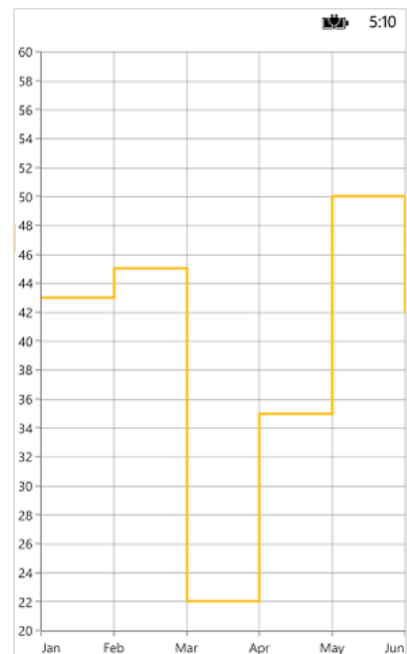
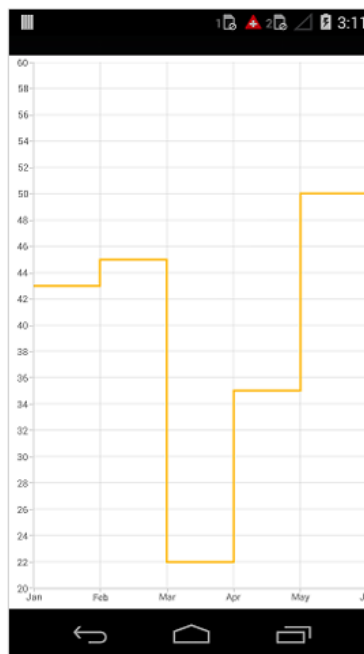
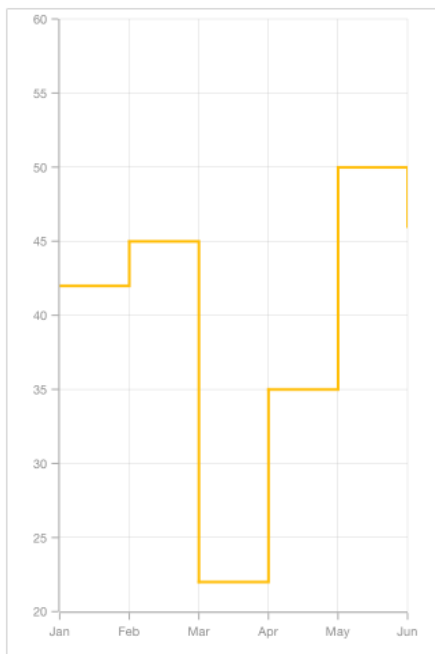
- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.

**XML**

```
<chart:SfChart.Series>
<chart:StepLineSeries ItemsSource="{Binding Data}" XBindingPath="Month"
YBindingPath="Value" />
</chart:SfChart.Series>
```

**C#**

```
SfChart chart = new SfChart();
...
StepLineSeries stepLine = new StepLineSeries()
{
    ItemsSource = Data,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
chart.Series.Add(stepLine);
```

**Bubble Chart**

To render a bubble chart, create an instance of [BubbleSeries](#) and add to the [Series](#) collection property of [SfChart](#).

Bubble chart requires 3 fields (X, Y and Size) to plot a point. Here [Size](#) is used to specify the size of each bubble segment.

There are two ways you can provide data to bubble chart,

1. You can use [ChartDataPoint's](#) three parameter constructor to pass [XValue](#), [YValue](#) and [Size](#) values to [BubbleSeries](#),

**C#**

```
[C#]
```

```
SfChart chart = new SfChart();
...
ObservableCollection<ChartDataPoint> data = new
ObservableCollection<ChartDataPoint>()
{
    new ChartDataPoint(64, 14.4, 20),
    new ChartDataPoint(71, 2, 15),
    new ChartDataPoint(74, 7, 30),
    new ChartDataPoint(80, 4, 22),
    new ChartDataPoint(82, 10.3, 28),
    new ChartDataPoint(94, 1, 8),
    new ChartDataPoint(96, 6, 18),
    new ChartDataPoint(98, 12.3, 28),
};
BubbleSeries bubbleSeries = new BubbleSeries()
{
    ItemsSource = data
};
chart.Series.Add(bubbleSeries);
```

2.Or else you can use [YBindingPath](#) and [Size](#) properties of [BubbleSeries](#) to map the Y value and size from custom object to chart.

#### XML

```
<chart:SfChart>
...
<chart:BubbleSeries ItemsSource="{Binding Data}"
XBindingPath="XValue"
YBindingPath="YValue"
Size="Size"/>
</chart:SfChart>
```

#### C#

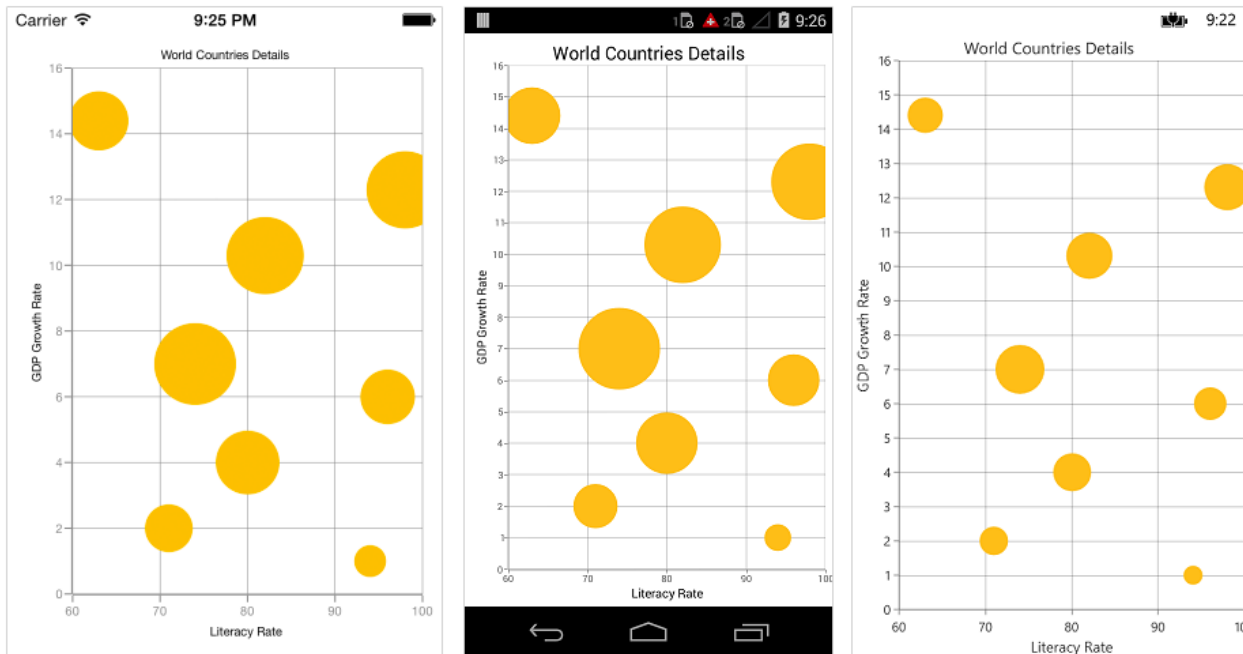
```
SfChart chart = new SfChart();
...
BubbleSeries bubbleSeries = new BubbleSeries()
{
    ItemsSource = Data,
    XBindingPath = "XValue",
    YBindingPath = "YValue",
    Size = "Size"
};
chart.Series.Add(bubbleSeries);
```

Following properties are used to customize the bubble segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [MinimumRadius](#) – used to change the minimum size of the series.



- [MaximumRadius](#) – used to change the maximum size of the series.



### Scatter Chart

To render a scatter chart, create an instance of [ScatterSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the scatter segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [ScatterWidth](#) – used to change the width of the series.
- [ScatterHeight](#) – used to change the height of the series.
- [ShapeType](#) - used to change the rendering shape of scatter series. The available shapes are [Cross](#), [Diamond](#), [Ellipse](#), [Hexagon](#), [InvertedTriangle](#), [Pentagon](#), [Plus](#), [Rectangle](#) and [Triangle](#).

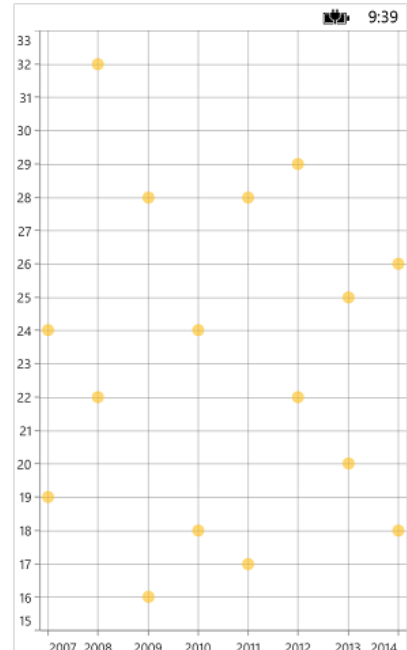
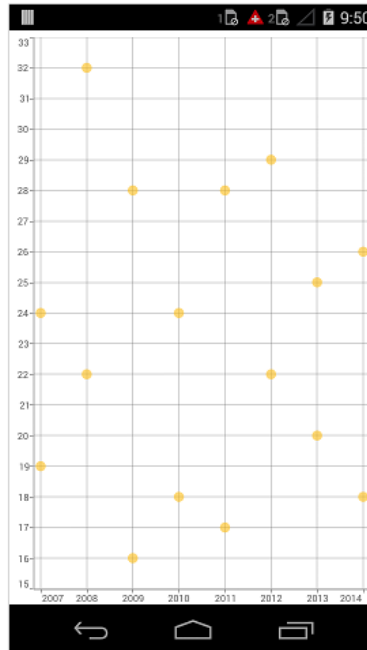
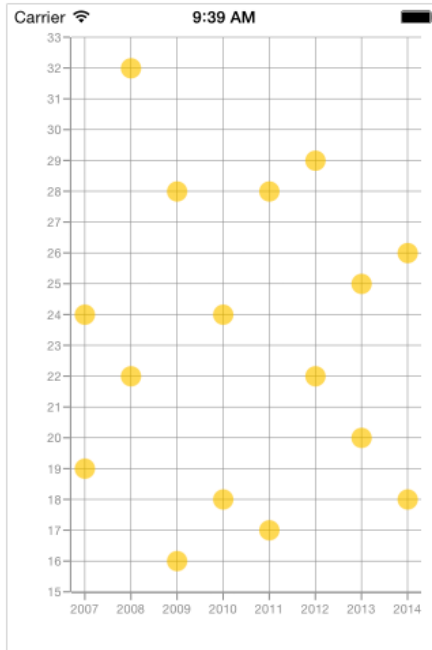
### XML

```
<chart:SfChart>
...
<chart:ScatterSeries ScatterHeight="15"
ScatterWidth="15"
ShapeType="Ellipse"
ItemsSource="{Binding Data}"
XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
```

```
ScatterSeries scatterSeries = new ScatterSeries ()
{
    ItemsSource = Data,
    ScatterHeight = 15,
    ScatterWidth = 15,
    ShapeType = ChartScatterShapeType.Ellipse,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(scatterSeries);
```



### Fast Scatter Chart

The [FastScatterSeries](#) is a special kind of scatter series that renders a collection with a huge number of data points. You can use the following properties to customize the appearance of a fast scatter point.

- [Color](#) – used to change the color of series.
- [Opacity](#) – used to control the transparency of chart series.
- [StrokeWidth](#) – used to change the stroke width of series.
- [StrokeColor](#) – used to change the stroke color of series.
- [ScatterWidth](#) – used to change the width of series.
- [ScatterHeight](#) – used to change the height of series.
- [ShapeType](#) – used to change the rendering shape of fast scatter series. The available shapes are [Cross](#), [Diamond](#), [Ellipse](#), [Hexagon](#), [InvertedTriangle](#), [Pentagon](#), [Plus](#), [Rectangle](#) and [Triangle](#).
- [EnableAntiAliasing](#) – Enables or disables the smoothness of series. Default value of [EnableAntiAliasing](#) property is true.

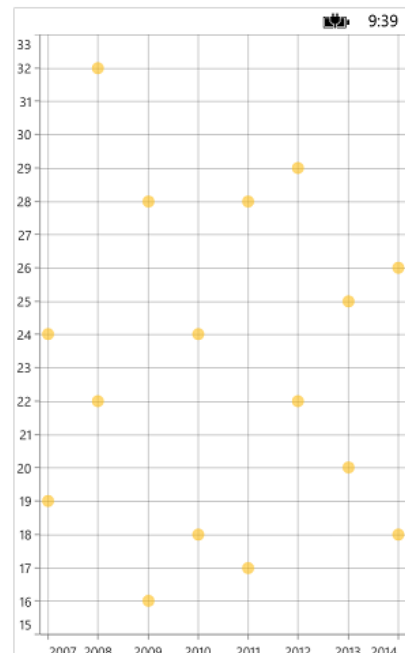
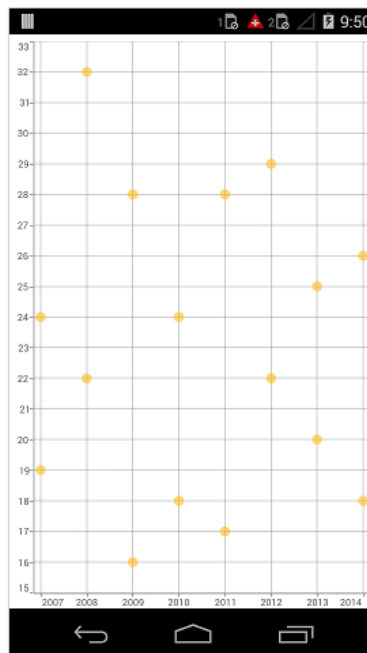
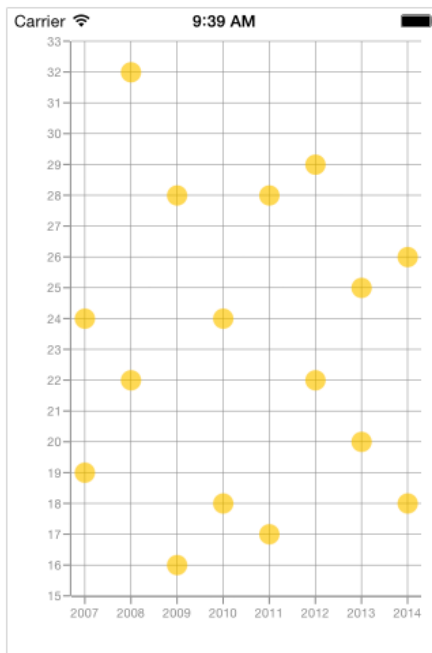
### XML

```
<chart:SfChart>
...
<chart:FastScatterSeries ScatterHeight="15"
```

```
ScatterWidth="15"
ShapeType="Ellipse"
ItemsSource="{Binding Data}"
XBindingPath="Year"
YBindingPath="Value"/>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
FastScatterSeries fastScatterSeries = new FastScatterSeries ()
{
    ItemsSource = Data,
    ScatterHeight = 15,
    ScatterWidth = 15,
    ShapeType = ChartScatterShapeType.Ellipse,
    XBindingPath = "Year",
    YBindingPath = "Value"
};
chart.Series.Add(fastScatterSeries);
```



## OHLC Chart

To render an OHLC chart, create an instance of [HiLoOpenCloseSeries](#) and add to the [Series](#) collection property of [SfChart](#).

OHLC chart requires five values (X, Open, High, Low and Close) to plot a point.

There are two ways you can provide data to an OHLC chart,

1. You can use [ChartDataPoint's](#) five parameter constructor to pass [XValue](#), [Open](#), [High](#), [Low](#), [Close](#), [Volume](#) values to [HiLoOpenCloseSeries](#) and [technical indicators](#).

**C#**

```
[C#]
SfChart chart = new SfChart();
...
ObservableCollection<ChartDataPoint> data = new
ObservableCollection<ChartDataPoint>()
{
    new ChartDataPoint("2010", 873.8, 878.85, 855.5, 860.5),
    new ChartDataPoint("2011", 861, 868.4, 835.2, 843.45),
    new ChartDataPoint("2012", 846.15, 853, 838.5, 847.5),
    new ChartDataPoint("2013", 846, 860.75, 841, 855),
    new ChartDataPoint("2014", 841, 845, 827.85, 838.65)
};
HiLoOpenCloseSeries hiLoOpenCloseSeries = new HiLoOpenCloseSeries()
{
    ItemsSource = data
};
chart.Series.Add(hiLoOpenCloseSeries);
```

2.Or else you can use [Open,High,Low](#) and [Close](#) properties of [HiLoOpenCloseSeries](#) to map Open, High, Low and Close values from custom object to chart.

**XML**

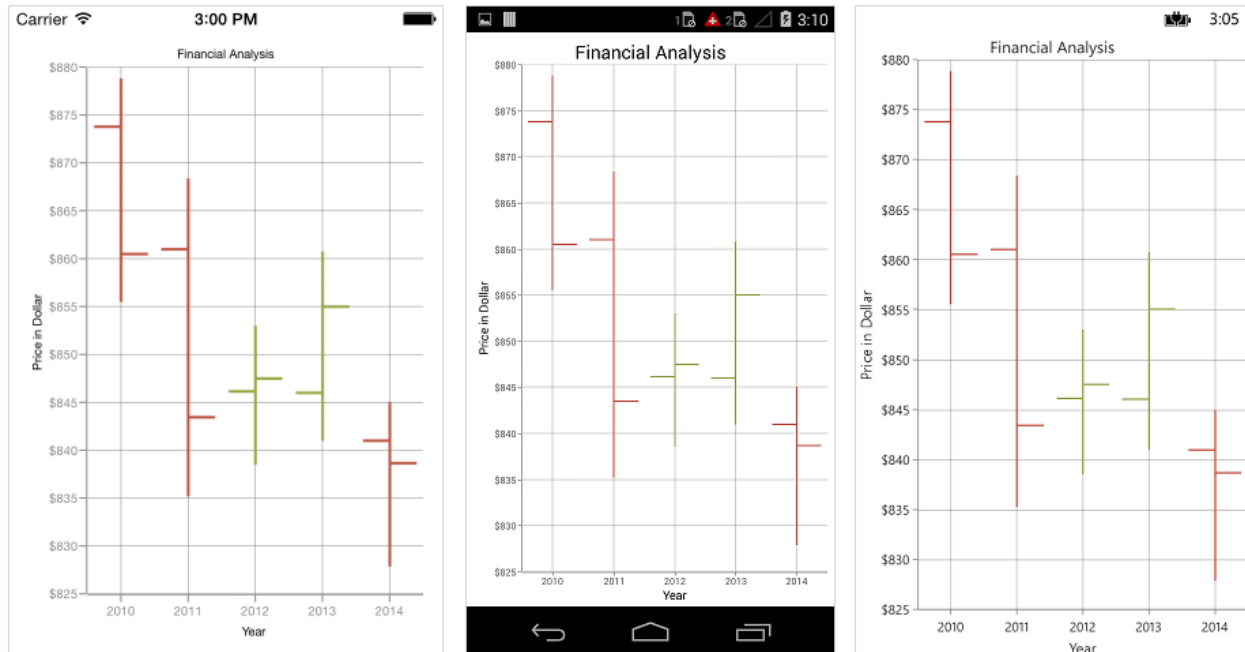
```
<chart:SfChart>
...
<chart:HiLoOpenCloseSeries ItemsSource="{Binding Data}" XBindingPath="Year"
High="Value1" Low="Value2"
Open="Value3" Close="Value4"/>
</chart:SfChart>
```

**C#**

```
SfChart chart = new SfChart();
...
HiLoOpenCloseSeries hiLoOpenCloseSeries = new HiLoOpenCloseSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    Open = "Value1",
    High = "Value2",
    Low = "Value3",
    Close = "Value4"
};
chart.Series.Add(hiLoOpenCloseSeries);
```

You can use the following properties to customize the [HiLoOpenCloseSeries](#) segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [Spacing](#) - used to change the spacing between two segments.
- [Width](#) - used to change the width of the rectangle.



### Bull and Bear Color

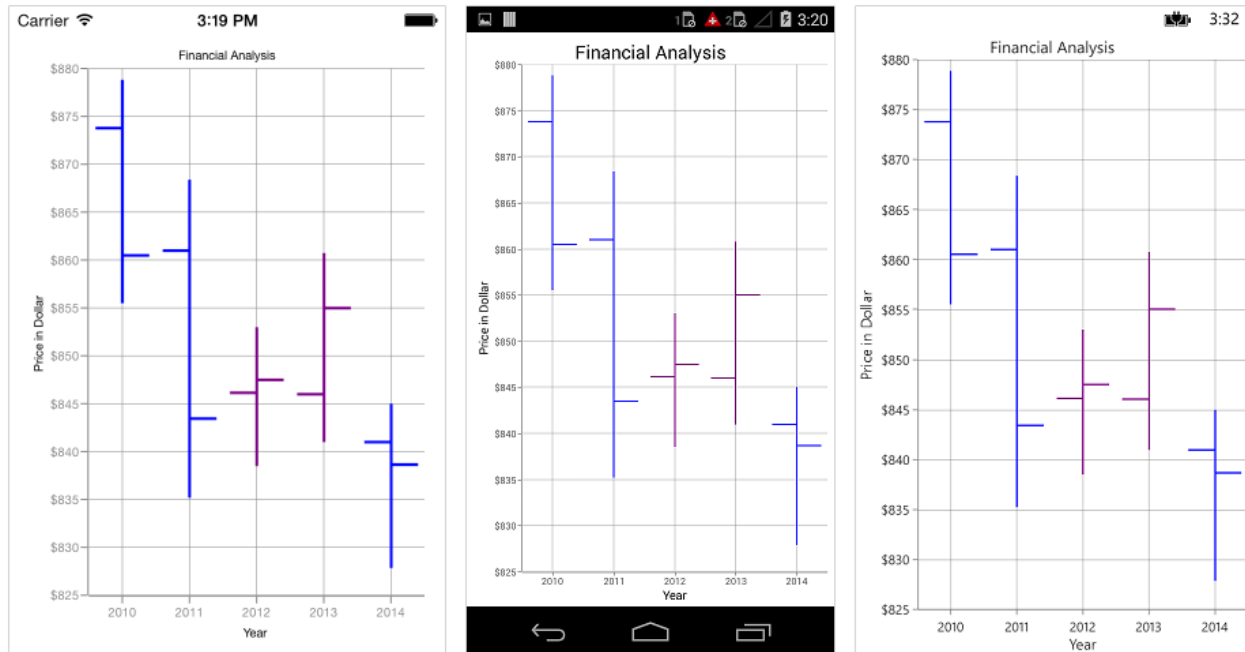
In OHLC chart, [BullFillColor](#) property is used to specify a fill color for the segments that indicates an increase in stock price in the measured time interval and [BearFillColor](#) property is used to specify a fill color for the segments that indicates a decrease in stock price in the measured time interval.

### XML

```
<chart:HiLoOpenCloseSeries BearFillColor="Blue"
BullFillColor="Purple"
ItemsSource="{Binding Data}"
XBindingPath="Year"
High="Value1"
Low="Value2"
Open="Value3"
Close="Value4"/>
```

### C#

```
HiLoOpenCloseSeries hiLoOpenCloseSeries = new HiLoOpenCloseSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    Open = "Value1",
    High = "Value2",
    Low = "Value3",
    Close = "Value4",
    BearFillColor = Color.Blue,
    BullFillColor = Color.Purple
};
```



### Candle Chart

To render a candle chart, create an instance of [CandleSeries](#) and add to the [Series](#) collection property of [SfChart](#).

Candle chart requires five values (X, Open, High, Low and Close) to plot a point.

There are two ways you can provide data to an candle chart,

1. You can use [ChartDataPoint's](#) five parameter constructor to pass [XValue](#), [Open](#), [High](#), [Low](#) and close values to [CandleSeries](#),

### C#

```
[C#]
SfChart chart = new SfChart();
...
ObservableCollection<ChartDataPoint> data = new
ObservableCollection<ChartDataPoint>()
{
    new ChartDataPoint("2010", 873.8, 878.85, 855.5, 860.5),
    new ChartDataPoint("2011", 861, 868.4, 835.2, 843.45),
    new ChartDataPoint("2012", 846.15, 853, 838.5, 847.5),
    new ChartDataPoint("2013", 846, 860.75, 841, 855),
    new ChartDataPoint("2014", 841, 845, 827.85, 838.65)
};
CandleSeries candleSeries = new CandleSeries ()
{
    ItemsSource = data
};
chart.Series.Add(candleSeries);
```

2. Or else you can use [Open, High, Low](#) and [Close](#) property of [CandleSeries](#) to map Open, High, Low and Close values from custom object to chart.

**XML**

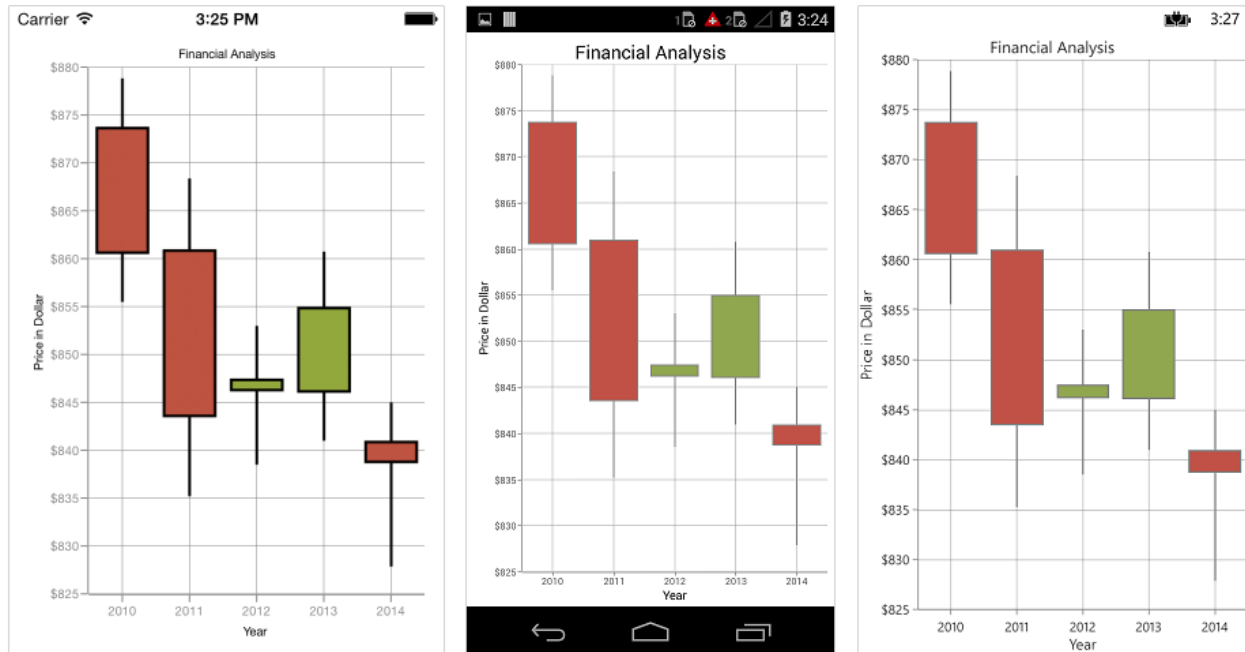
```
<chart:SfChart>
...
<chart:CandleSeries ItemsSource="{Binding Data}"
XBindingPath="Year"
High="Value1"
Low="Value2"
Open="Value3"
Close="Value4"/>
</chart:SfChart>
```

**C#**

```
SfChart chart = new SfChart();
...
CandleSeries candleSeries = new CandleSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    Open = "Value1",
    High = "Value2",
    Low = "Value3",
    Close = "Value4"
};
chart.Series.Add(candleSeries);
```

You can use the following properties to customize the candle segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.



### Bull and Bear Color

In Candle chart, [BullFillColor](#) property is used to specify a fill color for the segments that indicates an increase in stock price in the measured time interval and [BearFillColor](#) property is used to specify a fill color for the segments that indicates a decrease in stock price in the measured time interval.

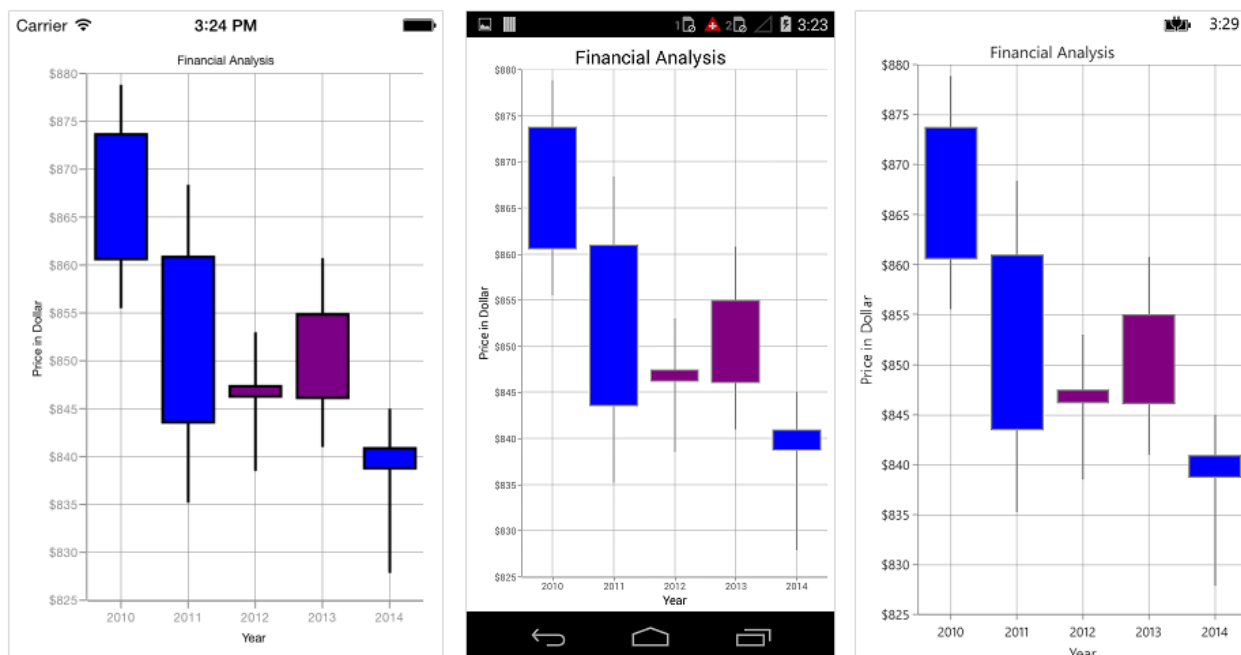
### XML

```
<chart:CandleSeries BearFillColor="Blue"
BullFillColor="Purple"
ItemsSource="{Binding Data}"
XBindingPath="Year"
High="Value1"
Low="Value2"
Open="Value3"
Close="Value4" />
```

### C#

```
CandleSeries candleSeries = new CandleSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    Open = "Value1",
    High = "Value2",
    Low = "Value3",
    Close = "Value4",
    BearFillColor = Color.Blue,
    BullFillColor = Color.Purple
};
```





### EnableSolidCandles

In Candle Series, [EnableSolidCandles](#) property is used to specify whether the candle segment should be filled or hollow. The default value of this property is false.

### XML

```
<chart:SfChart>
...
<chart:SfChart.Series>
<chart:CandleSeries ItemsSource="{Binding FinancialData}"
EnableSolidCandles="True"/>
</chart:SfChart.Series>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart()
{
...
Series =
{
new CandleSeries()
{
ItemsSource = viewModel.FinancialData,
EnableSolidCandles = true,
}
};
```

### Radar Chart

To render a radar chart, create an instance of [RadarSeries](#) and add to the [Series](#) collection property of [SfChart](#).

### Draw type

[DrawType](#) property is used to specify the radar series rendering type. Following are the two options you can set to this property,

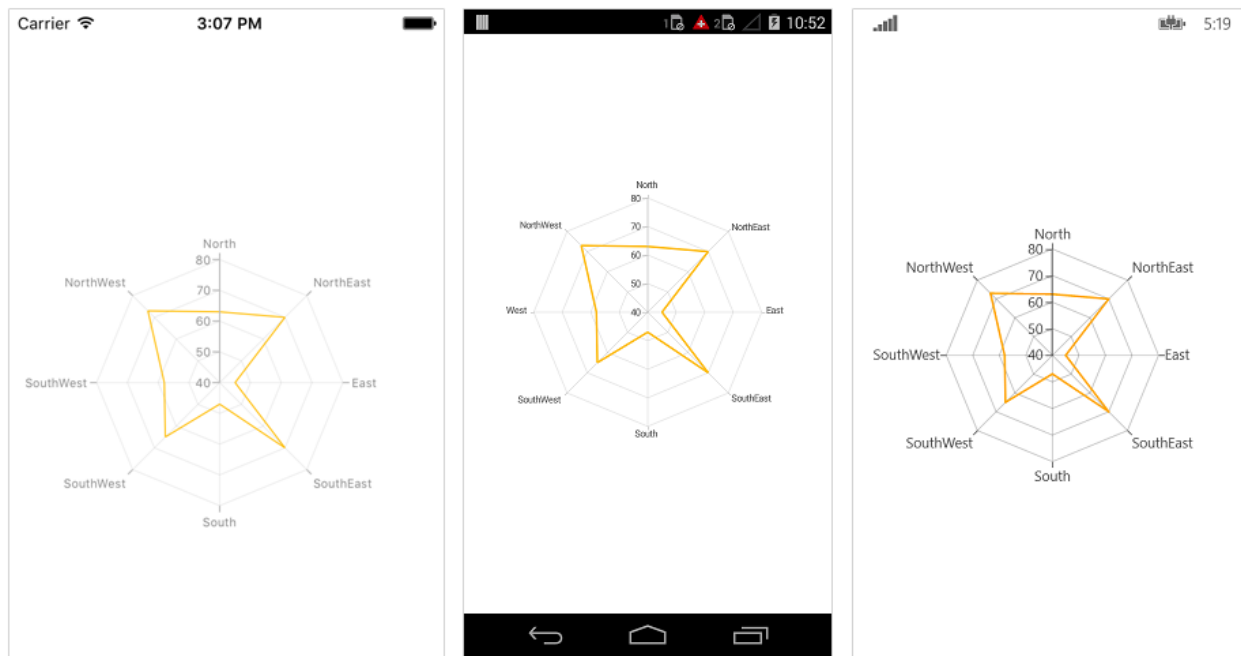
- [Line](#) – data points are visualized using line series.
- [Area](#) – data points are visualized using area series.

### XML

```
<chart:RadarSeries ItemsSource="{Binding RadarData}" DrawType="Line"
XBindingPath="Name" YBindingPath="Value" />
```

### C#

```
RadarSeries radar = new RadarSeries ();
radar.ItemsSource = viewModel.RadarData;
radar.XBindingPath = "Name";
radar.YBindingPath = "Value";
radar.DrawType = PolarRadarSeriesDrawType.Line;
```



### Customize the appearance

You can use the following properties to customize the appearance.

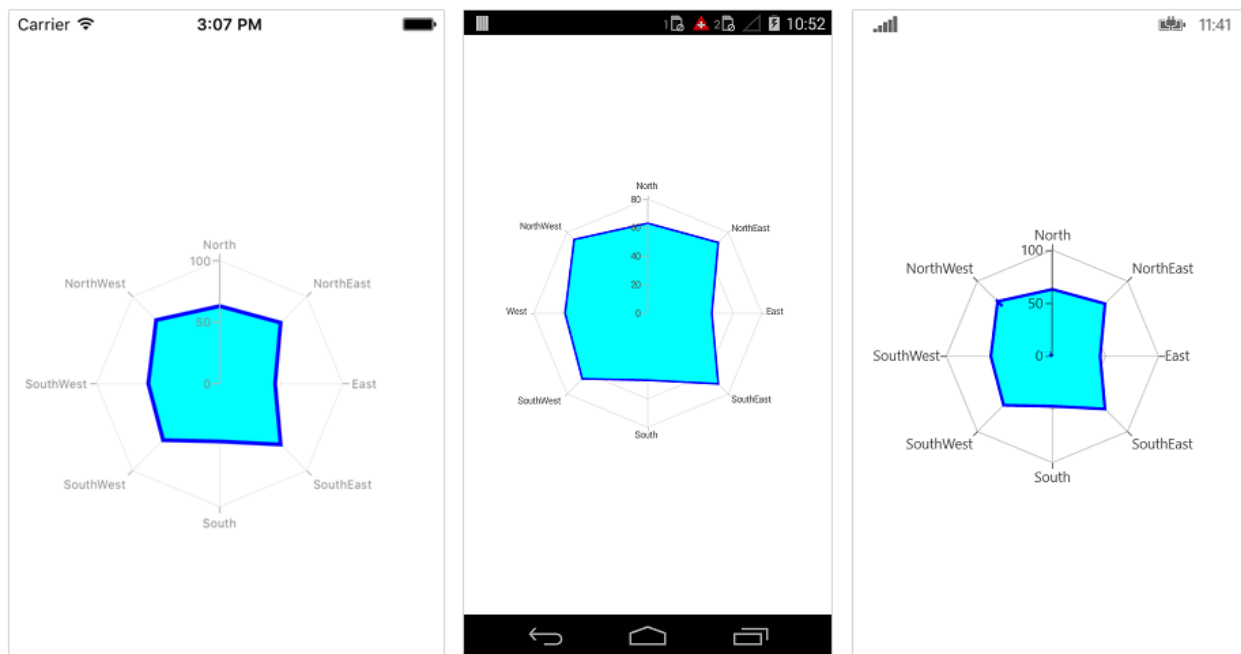
- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series when draw types is set to **Area**
- [StrokeDashArray](#) – used to render lines with dashes when the draw type is set to **Line**.

### XML

```
<chart:RadarSeries ItemsSource="{Binding RadarData}" Color="Aqua" StrokeColor="Blue"
StrokeWidth="3" XBindingPath="Name" YBindingPath="Value" />
```

**C#**

```
RadarSeries radar = new RadarSeries ();
radar.ItemsSource = viewModel.RadarData;
radar.XBindingPath = "Name";
radar.YBindingPath = "Value";
radar.Color = Color.Aqua;
radar.StrokeColor = Color.Blue;
radar.StrokeWidth = 3;
```

*Closed*

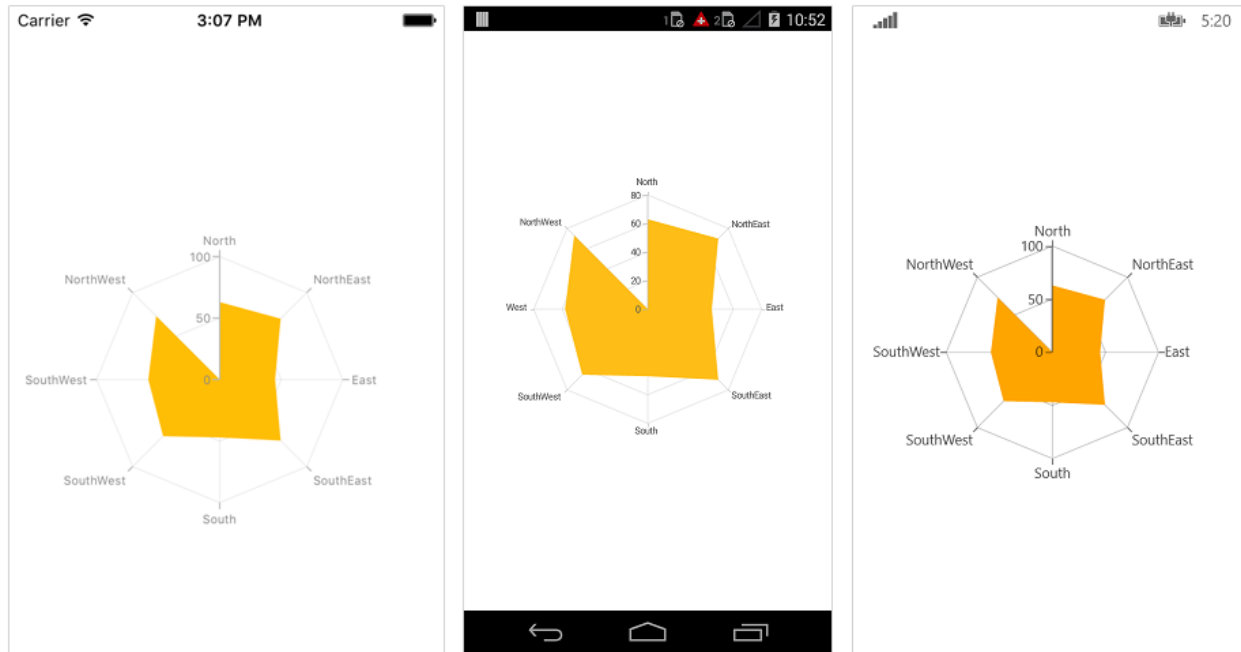
**Closed** property is used to determine, whether to connect the first and last data point of the series. By default, the property is set to **true**.

**XML**

```
<chart:RadarSeries ItemsSource="{Binding RadarData}" IsClosed="false"
XBindingPath="Name" YBindingPath="Value" />
```

**C#**

```
RadarSeries radar = new RadarSeries ();
radar.ItemsSource = viewModel.RadarData;
radar.XBindingPath = "Name";
radar.YBindingPath = "Value";
radar.IsClosed = false;
```



#### *Radar start angle for primary axis*

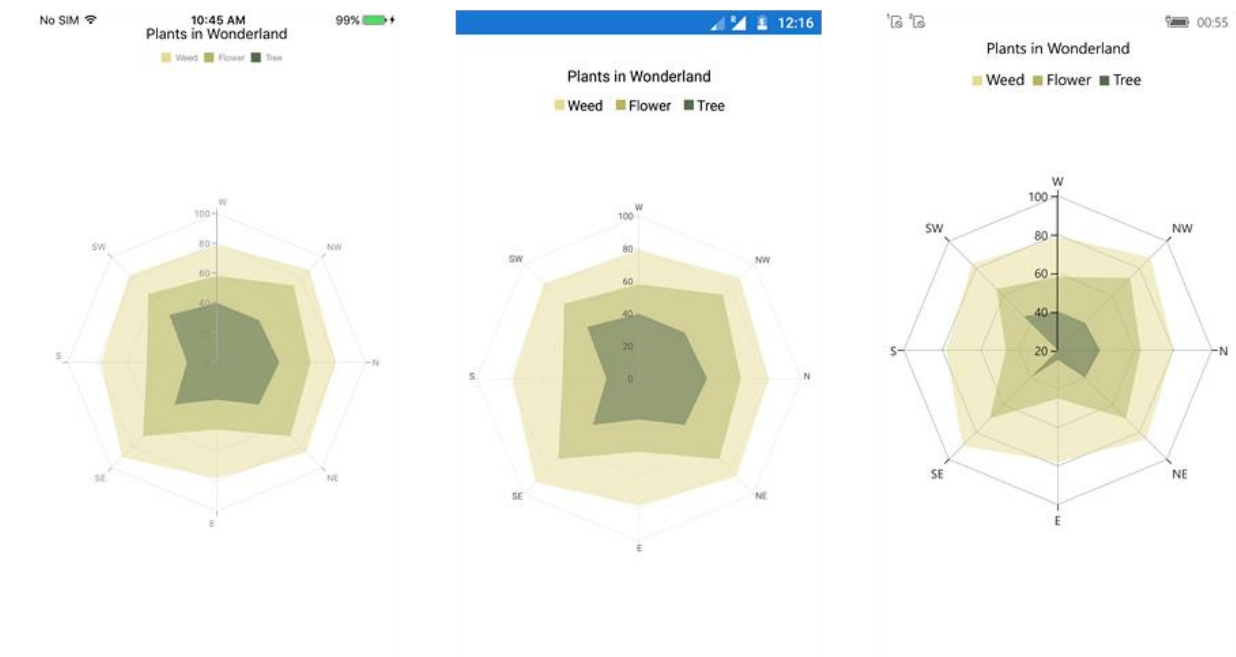
The start position of the radar series can be set by using [PolarAngle](#) property of axis. Default value of [PolarAngle](#) property is [Rotate270](#). [PolarAngle](#) property can be set for primary axis, secondary axis, or both axes

#### **XML**

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis PolarAngle = "Rotate0"/>
</chart:SfChart.PrimaryAxis >
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis/>
</chart:SfChart.SecondaryAxis >
```

#### **C#**

```
chart.PrimaryAxis = new CategoryAxis(){ PolarAngle =
ChartPolarAngle.Rotate0 };
chart.SecondaryAxis = new NumericalAxis();
```



*Radar start angle for secondary axis*

#### **XML**

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis/>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis PolarAngle="Rotate0"/>
</chart:SfChart.SecondaryAxis>
```

#### **C#**

```
chart.PrimaryAxis = new CategoryAxis();
chart.SecondaryAxis = new NumericalAxis() { PolarAngle =
ChartPolarAngle.Rotate0 };
```



*Radar start angle for both axis*

#### **XML**

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis PolarAngle = "Rotate0" />
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis PolarAngle = "Rotate0" />
</chart:SfChart.SecondaryAxis >
```

#### **C#**

```
chart.PrimaryAxis = new CategoryAxis() { PolarAngle =
ChartPolarAngle.Rotate0 };
chart.SecondaryAxis = new NumericalAxis() { PolarAngle =
ChartPolarAngle.Rotate0 };
```



### Polar Chart

To render a polar chart, create an instance of [PolarSeries](#) and add to the [Series](#) collection property of [SfChart](#).

#### Draw type

[DrawType](#) property is used to specify the polar series rendering type. Following are the two options you can set to this property,

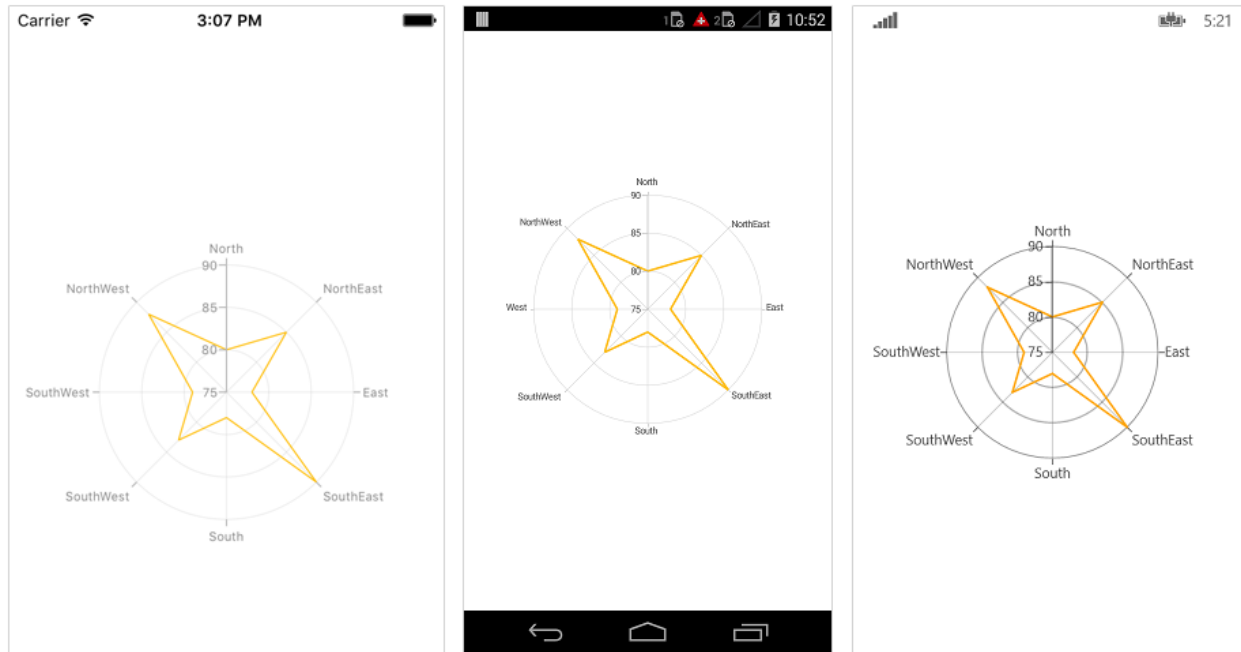
- [Line](#) – data points are visualized using line series.
- [Area](#) – data points are visualized using area series.

### XML

```
<chart:PolarSeries ItemsSource="{Binding PolarData}" DrawType="Line"
XBindingPath="Name" YBindingPath="Value" />
```

### C#

```
PolarSeries polar = new PolarSeries ();
polar.ItemsSource = viewModel.PolarData;
polar.XBindingPath = "Name";
polar.YBindingPath = "Value";
polar.DrawType = PolarRadarSeriesDrawType.Line;
```



### Customize the appearance

You can use the following properties to customize the appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series when draw types is set to [Area](#)
- [StrokeDashArray](#) – used to render lines with dashes when the draw type is set to [Line](#)

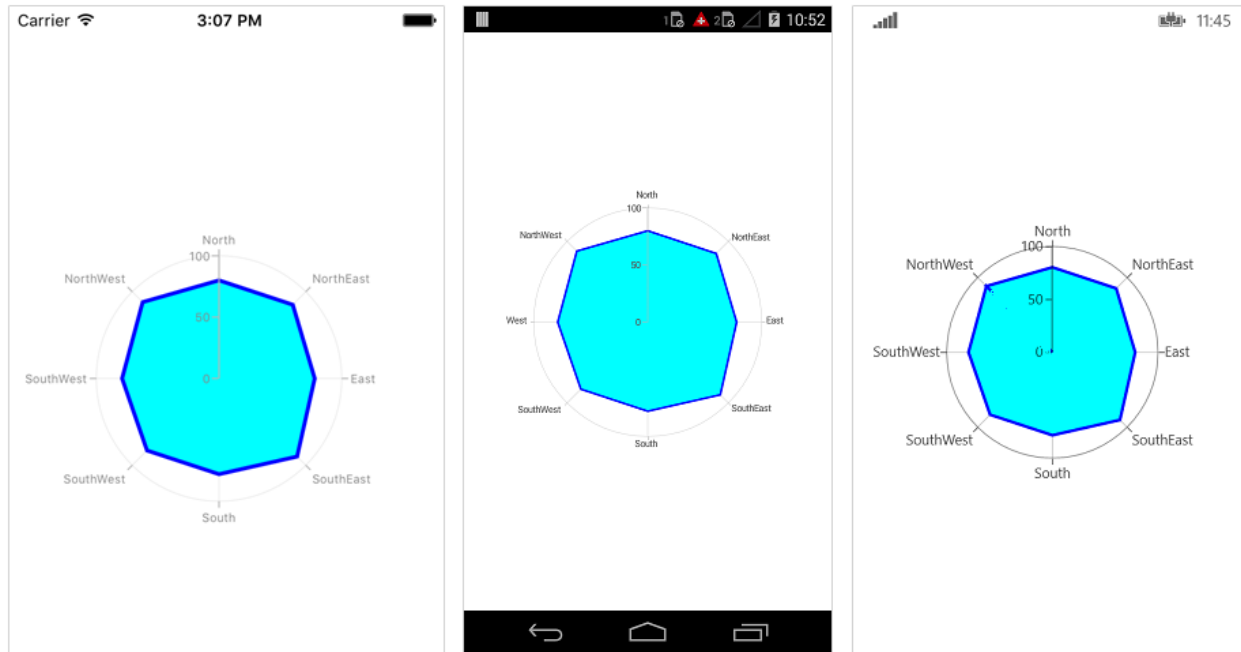
### XML

```
<chart:PolarSeries ItemsSource="{Binding PolarData}" StrokeColor="Blue" Color="Aqua"
StrokeWidth="3" XBindingPath="Name" YBindingPath="Value" />
```

### C#

```
PolarSeries polar = new PolarSeries ();
polar.ItemsSource = viewModel.PolarData;
polar.XBindingPath = "Name";
polar.YBindingPath = "Value";
polar.Color = Color.Aqua;
polar.StrokeColor = Color.Blue;
polar.StrokeWidth = 3;
```





### Closed

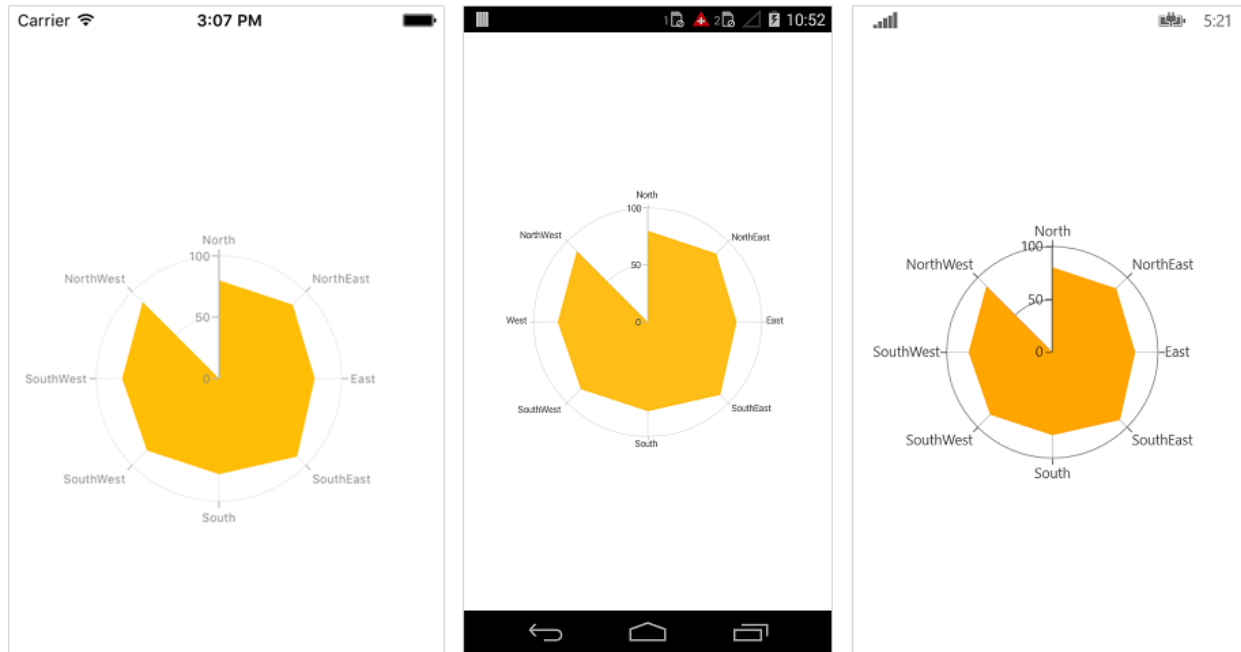
[Closed](#) property is used to determine, whether to connect the first and last data point of the series. By default, the property is set to `true`.

### XML

```
<chart:PolarSeries ItemsSource="{Binding PolarData}" IsClosed="false"
XBindingPath="Name" YBindingPath="Value" />
```

### C#

```
PolarSeries polar = new PolarSeries ();
polar.ItemsSource = viewModel.PolarData;
polar.XBindingPath = "Name";
polar.YBindingPath = "Value";
polar.IsClosed = false;
```



### *Polar start angle for primary axis*

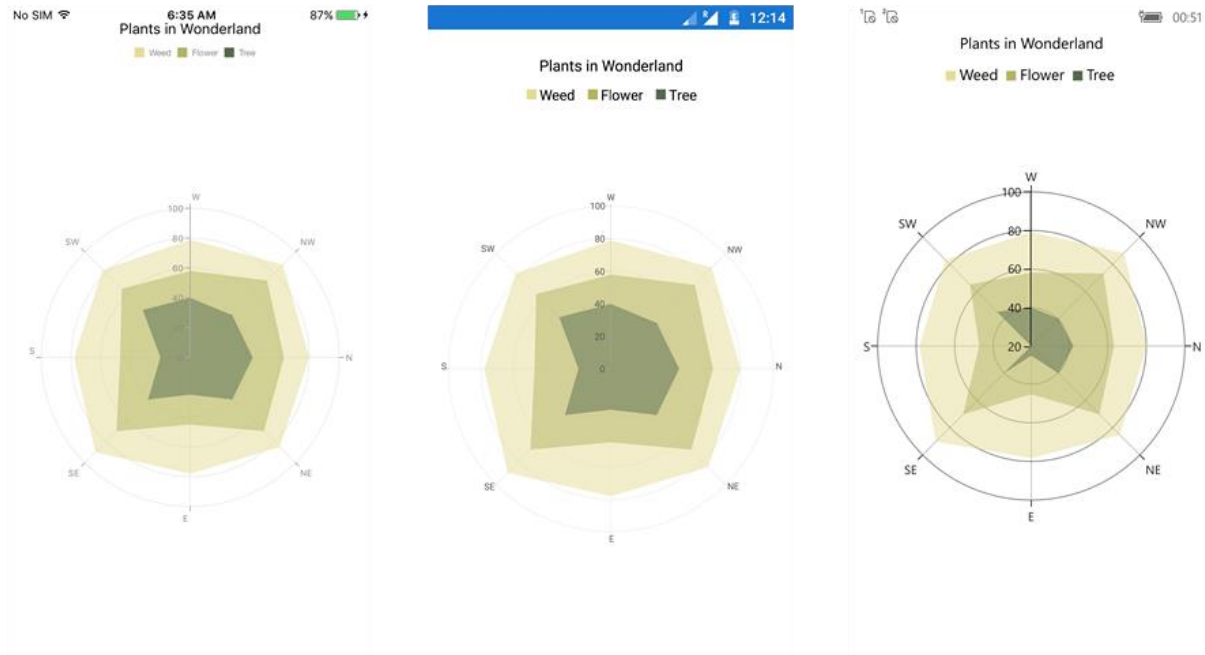
The start position of the polar series can be set by using [PolarAngle](#) property of axis. Default value of [PolarAngle](#) property is [Rotate270](#). [PolarAngle](#) property can be set for primary axis, secondary axis, or both axes.

### **XML**

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis PolarAngle = "Rotate0"/>
</chart:SfChart.PrimaryAxis >
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis/>
</chart:SfChart.SecondaryAxis >
```

### **C#**

```
chart.PrimaryAxis = new CategoryAxis(){ PolarAngle =
ChartPolarAngle.Rotate0 };
chart.SecondaryAxis = new NumericalAxis();
```



*Polar start angle for secondary axis*

#### **XML**

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis/>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis PolarAngle="Rotate0"/>
</chart:SfChart.SecondaryAxis>
```

#### **C#**

```
chart.PrimaryAxis = new CategoryAxis();
chart.SecondaryAxis = new NumericalAxis() { PolarAngle =
ChartPolarAngle.Rotate0 };
```



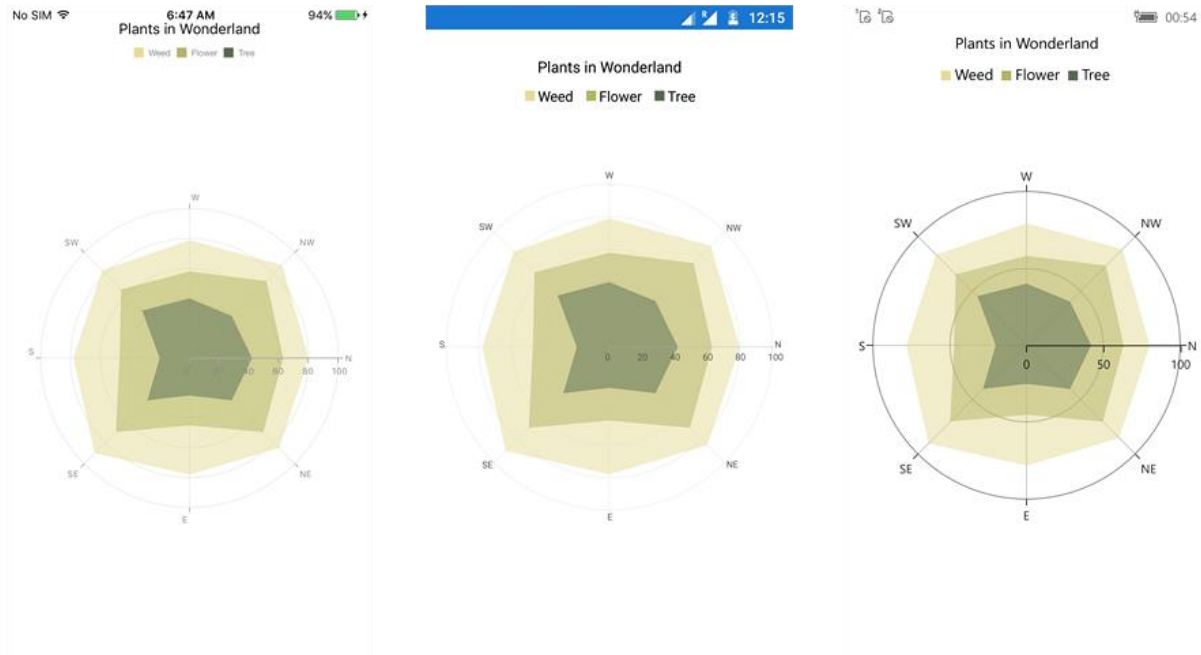
*Polar start angle for both axis*

#### **XML**

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis PolarAngle = "Rotate0" />
</chart:SfChart.PrimaryAxis >
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis PolarAngle = "Rotate0" />
</chart:SfChart.SecondaryAxis >
```

#### **C#**

```
chart.PrimaryAxis = new CategoryAxis() { PolarAngle =
ChartPolarAngle.Rotate0 };
chart.SecondaryAxis = new NumericalAxis() { PolarAngle =
ChartPolarAngle.Rotate0 };
```



### Pie Chart

To render a pie chart, create an instance of [PieSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the pie segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [DataMarkerPosition](#) - used to change the position of data marker at [Inside](#), [Outside](#) or [OutsideExtended](#).
- [ConnectorLinePosition](#) - used to change the position of the connector line at [Auto](#) and [Center](#).

**Note:** ConnectorLinePosition provides better alignment to the straight connector lines with outside extended label position for the minimum number of data points.

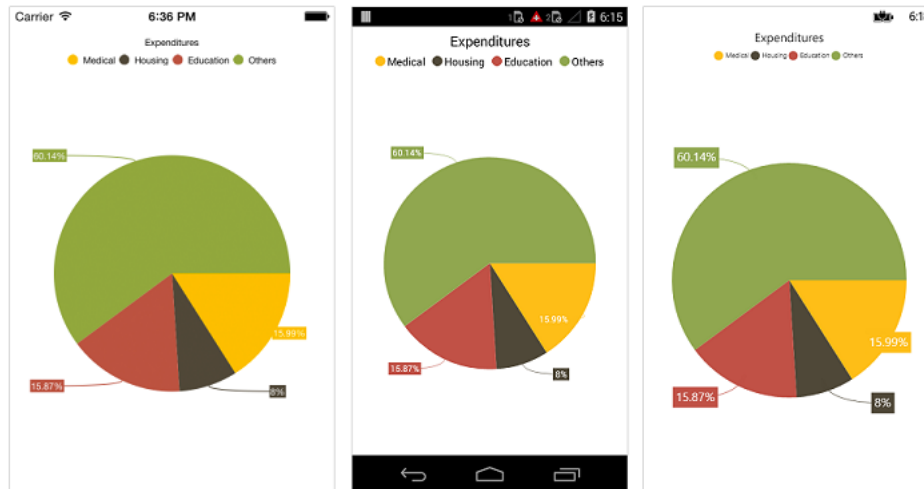
### XML

```
<chart:SfChart>
...
<chart:PieSeries ItemsSource="{Binding Data}" XBindingPath="Expense"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
PieSeries pieSeries = new PieSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value"
```

```
};
chart.Series.Add(pieSeries);
```



### Changing the pie size

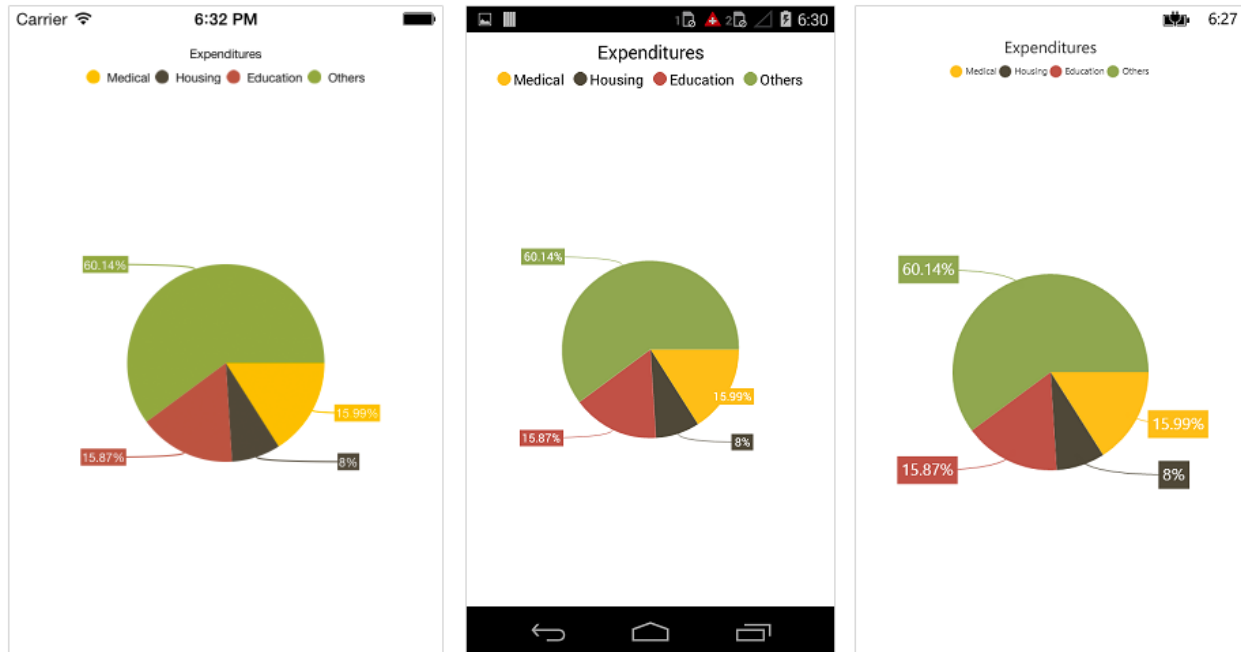
You can use [CircularCoefficient](#) property to change the diameter of the pie chart with respect to the plot area. It ranges from 0 to 1 and the default value is 0.8.

### XML

```
<chart:PieSeries CircularCoefficient="0.5"
ItemsSource="{Binding Data}"
XBindingPath="Expense"
YBindingPath="Value"/>
```

### C#

```
PieSeries pieSeries = new PieSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    CircularCoefficient = 0.5
};
```



### Exploding a pie segment

You can explode a pie segment using [ExplodeIndex](#) property and specify the explode radius using [ExplodeRadius](#) property of [PieSeries](#).

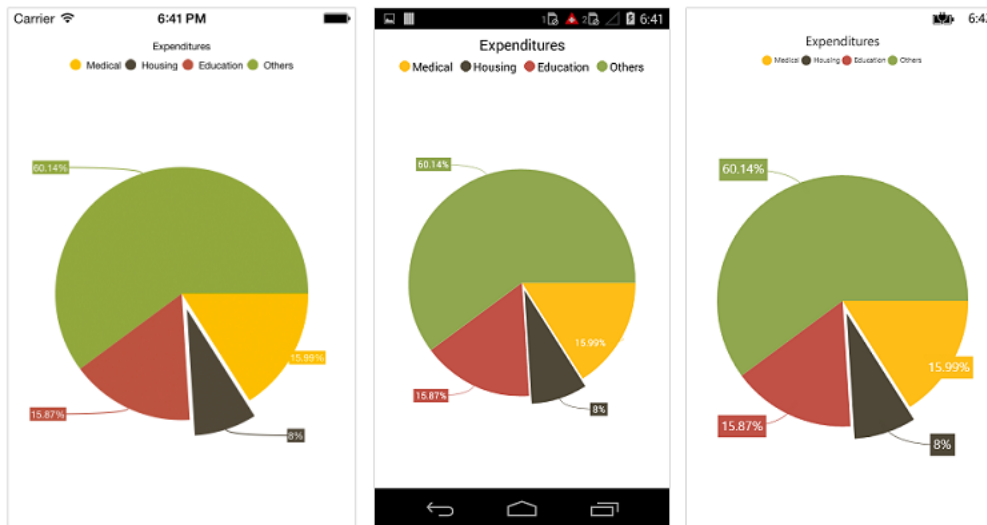
### XML

```
<chart:PieSeries ExplodeIndex="1"
ItemsSource="{Binding Data}"
XBindingPath="Expense"
YBindingPath="Value"/>
```

### C#

```
PieSeries pieSeries = new PieSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    ExplodeIndex = 1
};
```

Also, the segments can be exploded by touch using [ExplodeOnTouch](#) property of [PieSeries](#). Default value of this property is false.



### Exploding all the segments

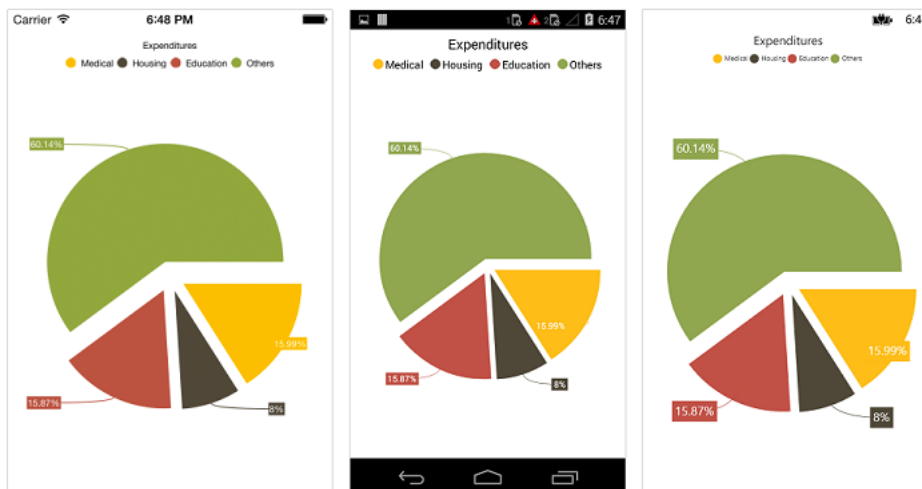
Using [ExplodeAll](#) property of [PieSeries](#), you can explode all the pie segments.

### XML

```
<chart:PieSeries ExplodeAll="True" ItemsSource="{Binding Data}"
  XBindingPath="Expense"
  YBindingPath="Value"/>
```

### C#

```
PieSeries pieSeries = new PieSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    ExplodeAll = true
};
```





### Sector of Pie

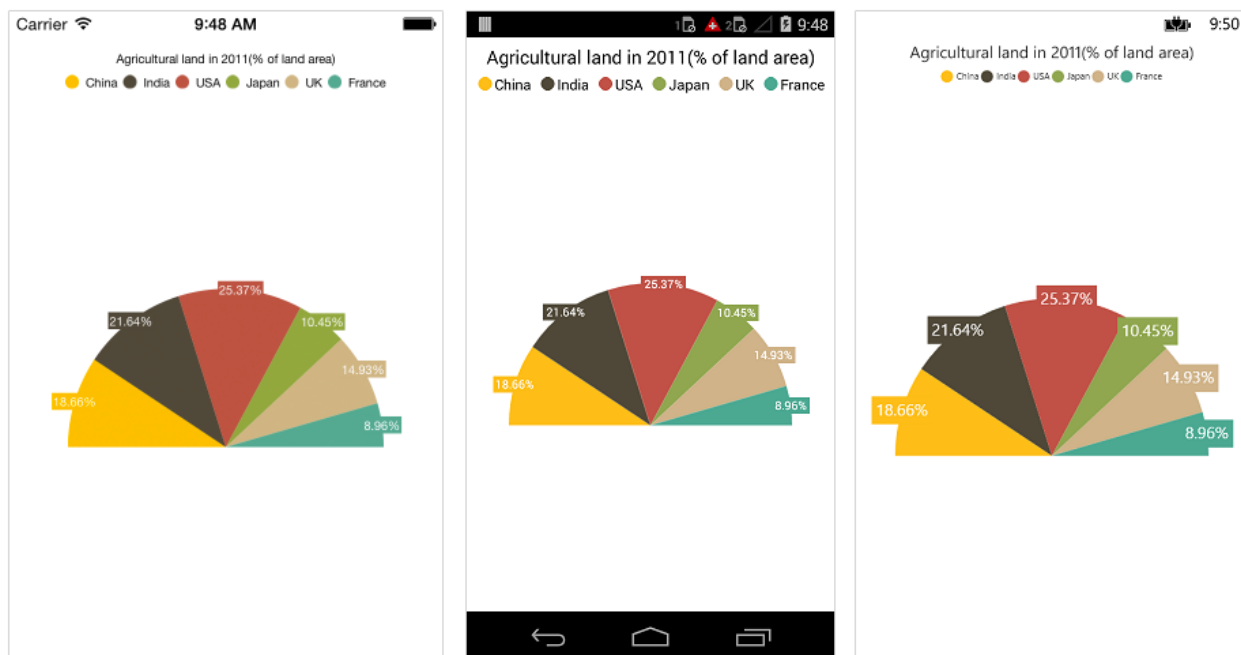
SfChart allows you to render all the data points/segments in semi-pie, quarter-pie or in any sector using [StartAngle](#) and [EndAngle](#) properties.

### XML

```
<chart:PieSeries StartAngle="180"
EndAngle="360"
ItemsSource="{Binding Data}"
XBindingPath="Expense"
YBindingPath="Value"/>
```

### C#

```
PieSeries pieSeries = new PieSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    StartAngle = 180,
    EndAngle = 360
};
```



### Group small data points into "others"

The small segments in the pie chart can be grouped into "others" category using the [GroupTo](#) and [GroupMode](#) properties of PieSeries. The [GroupMode](#) property is used to specify the grouping type based on slice [Angle](#), actual data point value, or [Percentage](#), and the [GroupTo](#) property is used to set the limit to group data points into a single slice. The grouped segment is labeled as "Others" in legend and toggled as any other segment. The default value of the [GroupTo](#) property is [double.NaN], and [GroupMode](#) property is Value.

## Doughnut Chart

To render a doughnut chart, create an instance of [DoughnutSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the doughnut segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.
- [DataMarkerPosition](#) - used to change the position of data marker at [Inside](#), [Outside](#) or [OutsideExtended](#).
- [ConnectorLinePosition](#) - used to change the position of the connector line at [Auto](#) and [Center](#).

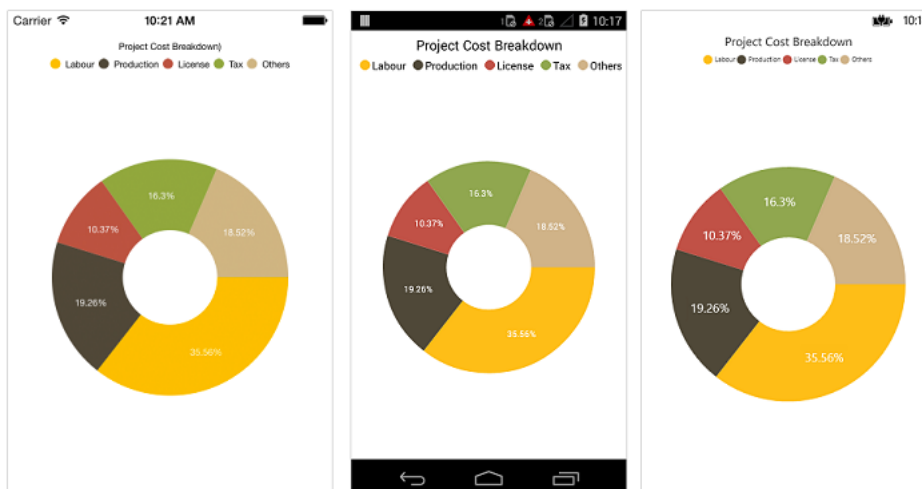
**Note:** ConnectorLinePosition provides better alignment to the straight connector lines with outside extended label position for the minimum number of data points.

## XML

```
<chart:SfChart>
...
<chart:DoughnutSeries ItemsSource="{Binding Data}"
XBindingPath="Expense"
YBindingPath="Value"/>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
DoughnutSeries doughnutSeries = new DoughnutSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value"
};
chart.Series.Add(doughnutSeries);
```



### Stacked doughnut

Doughnut segments can be separated as individual circles using the [IsStackedDoughnut](#) property. The following properties are used to customize the stacked doughnut chart:

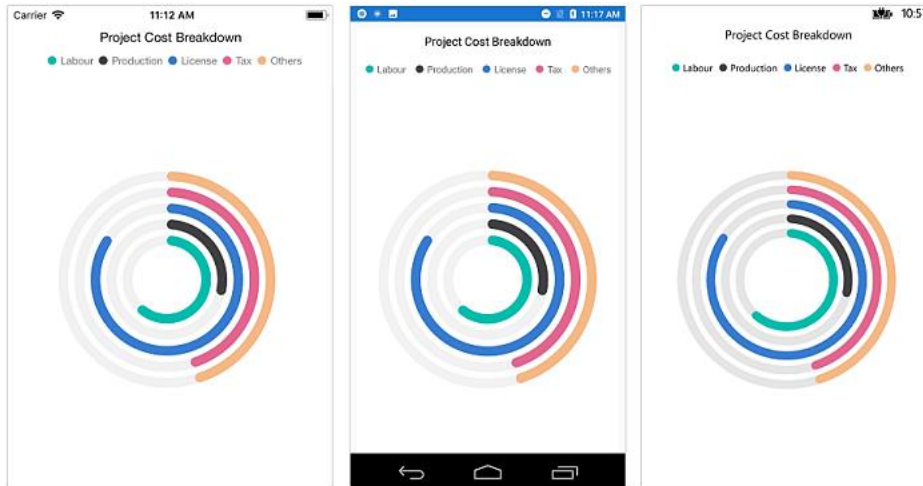
- [CapStyle](#) - Specifies the shape of the start and end points of the circular segment. The supported values are `BothFlat`, `BothCurve`, `StartCurve`, and `EndCurve`. The default value of the this property is `BothFlat`.
- [Spacing](#) - Changes the spacing between two individual segments. The default value of spacing is 0, and the value ranges from 0 to 1. Here, 1 and 0 correspond to 100% and 0% of the available space, respectively.
- [MaximumValue](#) - Represents the entire span of an individual circle. The default value of the this property is `double.NaN`.
- [TrackColor](#) - Changes the color of the track area.
- [TrackBorderColor](#) - Changes the color of the track border.
- [TrackBorderWidth](#) - Changes the width of the track border.

### XML

```
<chart:SfChart>
...
<chart:DoughnutSeries ItemsSource="{Binding Data}"
XBindingPath="Expense"
YBindingPath="Value"
IsStackedDoughnut="true"
CapStyle="BothCurve"
Spacing="0.4"
MaximumValue="100" />
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
DoughnutSeries doughnutSeries = new DoughnutSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    IsStackedDoughnut = true,
    CapStyle = DoughnutCapStyle.BothCurve,
    Spacing = 0.4,
    MaximumValue = 100
};
chart.Series.Add(doughnutSeries);
```



### Changing Doughnut inner radius

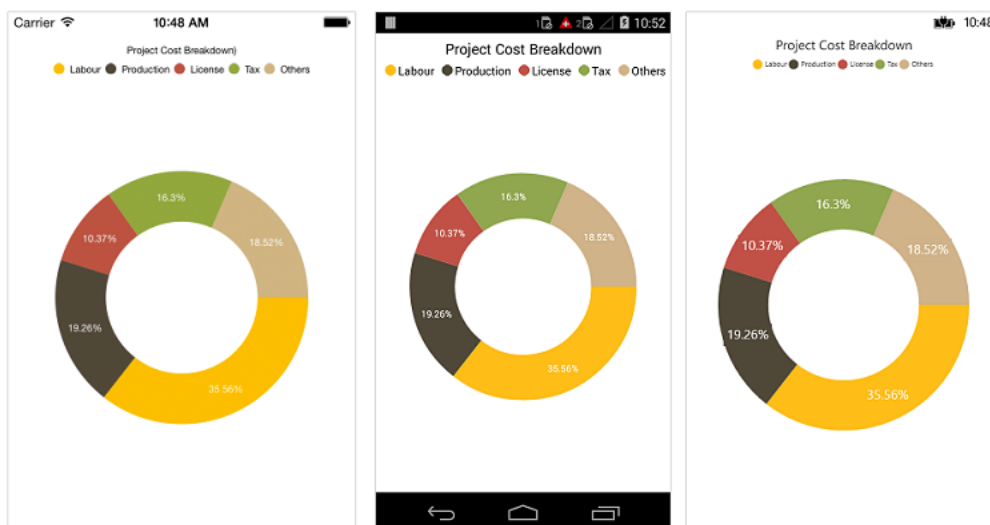
You can change the doughnut chart inner radius using [DoughnutCoefficient](#) with respect to the plot area. It ranges from 0 to 1 and the default value is 0.4.

### XML

```
<chart:DoughnutSeries DoughnutCoefficient="0.6"
ItemsSource="{Binding Data}"
XBindingPath="Expense"
YBindingPath="Value" />
```

### C#

```
DoughnutSeries doughnutSeries = new DoughnutSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    DoughnutCoefficient = 0.6
};
```



*Changing the doughnut size*

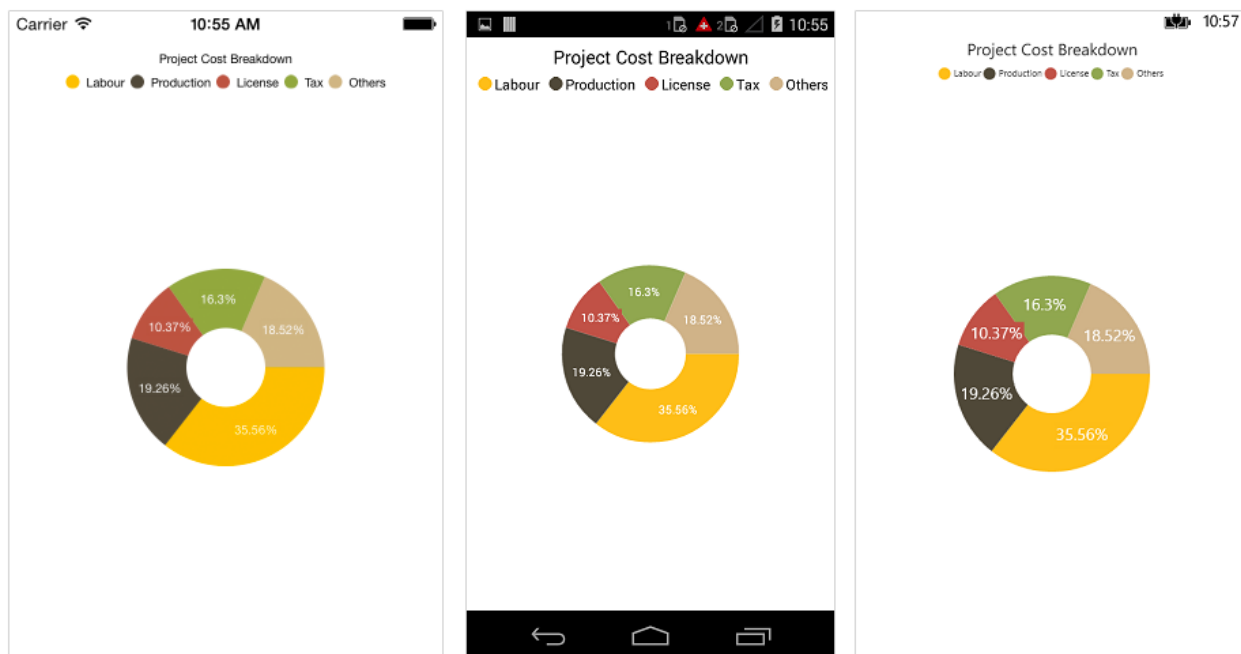
You can use the [CircularCoefficient](#) property to change the diameter of the doughnut chart with respect to the plot area. It ranges from 0 to 1 and the default value is 0.8.

**XML**

```
<chart:DoughnutSeries CircularCoefficient="0.5"
  ItemsSource="{Binding Data}"
  XBindingPath="Expense"
  YBindingPath="Value"/>
```

**C#**

```
DoughnutSeries doughnutSeries = new DoughnutSeries()
{
  ItemsSource = Data,
  XBindingPath = "Expense",
  YBindingPath = "Value",
  CircularCoefficient = 0.5
};
```

*Exploding a doughnut segment*

Exploding a specific doughnut segment, you have to set the index to be exploded using [ExplodeIndex](#) property of the series.

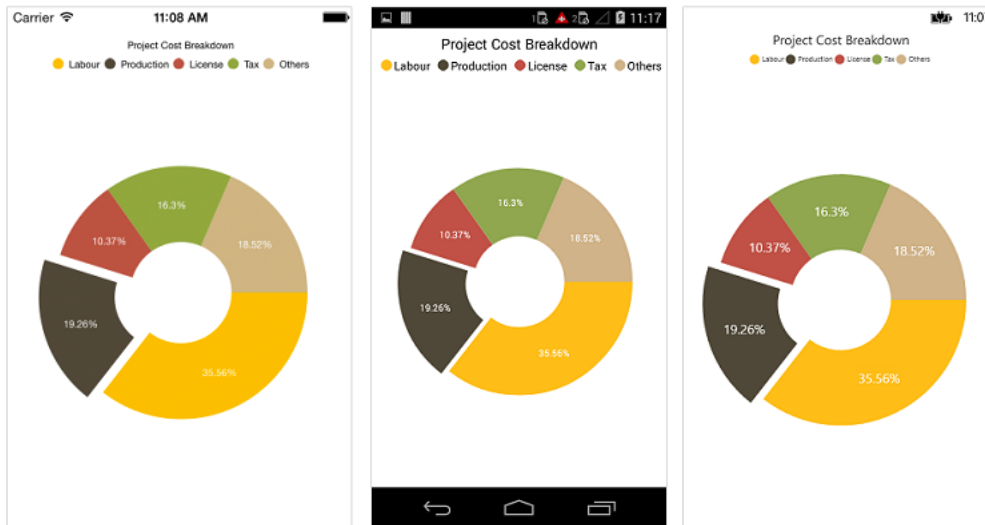
**XML**

```
<chart:DoughnutSeries ExplodeIndex="1" ItemsSource="{Binding Data}"
  XBindingPath="Expense"
  YBindingPath="Value" />
```

**C#**

```
DoughnutSeries doughnutSeries = new DoughnutSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    ExplodeIndex = 1
};
```

Also, the segments can be exploded by touch using [ExplodeOnTouch](#) property of [DoughnutSeries](#). Default value of this property is false.



#### *Exploding all the segments*

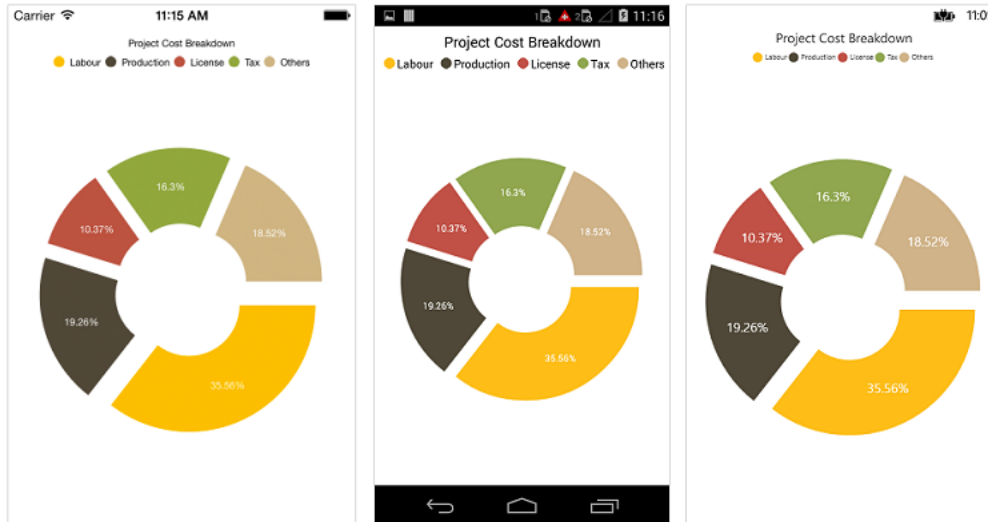
To explode all the segments, you have to enable [ExplodeAll](#) property of the series.

#### **XML**

```
<chart:DoughnutSeries ExplodeAll="True" ItemsSource="{Binding Data}"
    XBindingPath="Expense"
    YBindingPath="Value" />
```

#### **C#**

```
DoughnutSeries doughnutSeries = new DoughnutSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    ExplodeAll = true
};
```



### Sector of Doughnut

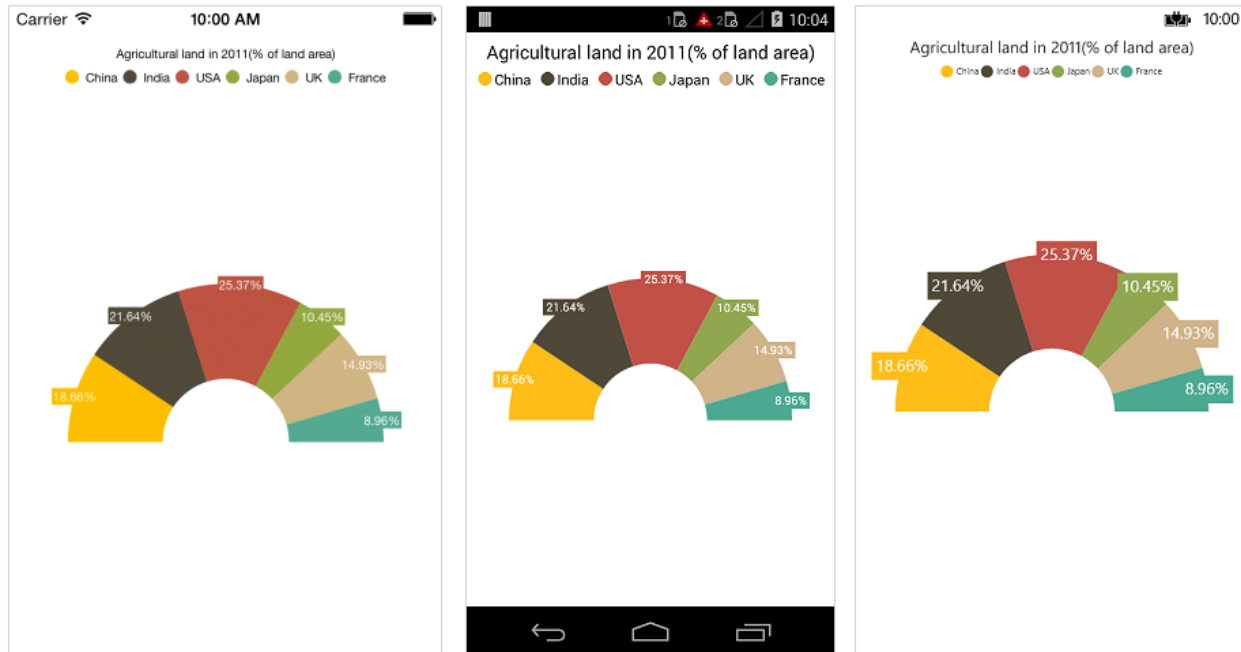
[SfChart](#) allows you to render all the data points/segments in semi-doughnut, quarter- doughnut or in any sector using [StartAngle](#) and [EndAngle](#) properties.

### XML

```
<chart:DoughnutSeries StartAngle="180" EndAngle="360" ItemsSource="{Binding
Data}"
XBindingPath="Expense" YBindingPath="Value" />
```

### C#

```
DoughnutSeries doughnutSeries = new DoughnutSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    StartAngle = 180,
    EndAngle = 360
};
```



#### Group small data points into "others"

The small segments in the doughnut chart can be grouped into "others" category using the [GroupTo](#) and [GroupMode](#) properties of [DoughnutSeries](#). The [GroupMode](#) property is used to specify the grouping type based on slice angle, actual data point value, or percentage, and the [GroupTo](#) property is used to set the limit to group data points into a single slice. The grouped segment is labeled as "Others" in legend and toggled as any other segment. The default value of the [GroupTo](#) property is [double.NAN], and [GroupMode](#) property is Value.

#### Add view to the center of doughnut chart

Any view can be added to the center of doughnut chart using the [CenterView](#) property of [DoughnutSeries](#). The binding context of the [CenterView](#) will be the respective [DoughnutSeries](#).

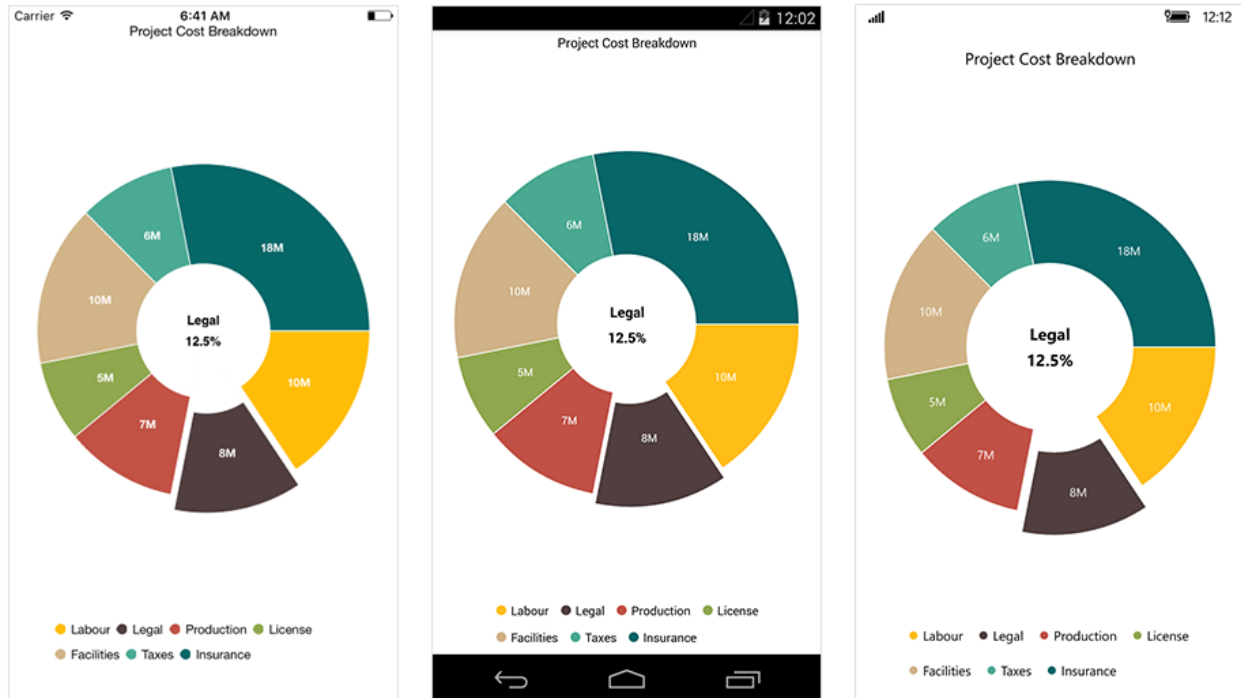
#### XML

```
<chart:DoughnutSeries>
...
<chart:DoughnutSeries.CenterView>
<StackLayout HorizontalOptions = "FillAndExpand"
VerticalOptions = "FillAndExpand">
...
</StackLayout>
</chart:DoughnutSeries.CenterView>
...
</chart:DoughnutSeries>
```

#### C#

```
DoughnutSeries doughnutSeries = new DoughnutSeries()
{
...
}
doughnutSeries.CenterView = new Label() { Text = "CenterView" };
```





### InnerRadius

The [InnerRadius](#) property of [DoughnutSeries](#) is used to get only the inner radius. Using this [InnerRadius](#) value, you can provide [CentreView](#) for series to avoid the view from being cropped outside the series.

### Pyramid Chart

To render a pyramid chart, create an instance of [PyramidSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the pyramid segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.

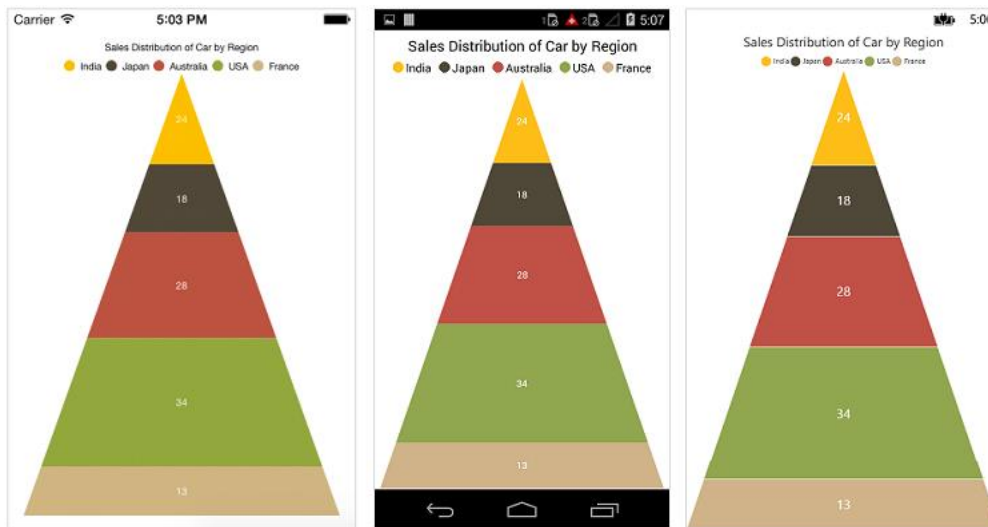
### XML

```
<chart:SfChart>
...
<chart:PyramidSeries ItemsSource="{Binding Data}" XBindingPath="Country"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
PyramidSeries pyramidSeries = new PyramidSeries()
{
    ItemsSource = Data,
    XBindingPath = "Country",
    YBindingPath = "Value"
};
```

```
chart.Series.Add(pyramidSeries);
```



### Pyramid Mode

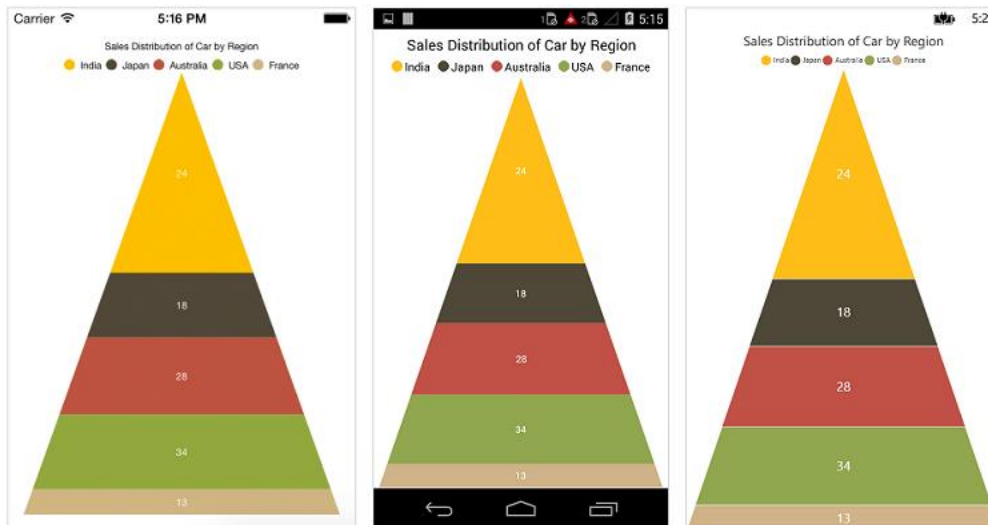
You can render the pyramid series as linear or surface mode. In linear mode, height of the pyramid segment is based on the Y value and in surface mode, area of the pyramid segment is based on the Y values. The default value of [PyramidMode](#) property is [Linear](#).

### XML

```
<chart:PyramidSeries ItemsSource="{Binding Data}" PyramidMode="Surface"
XBindingPath="Country"
YBindingPath="Value"/>
```

### C#

```
PyramidSeries pyramidSeries = new PyramidSeries ()
{
    ItemsSource = Data,
    XBindingPath = "Country",
    YBindingPath = "Value",
    PyramidMode = ChartPyramidMode.Surface
};
```



### *Gap between the segments*

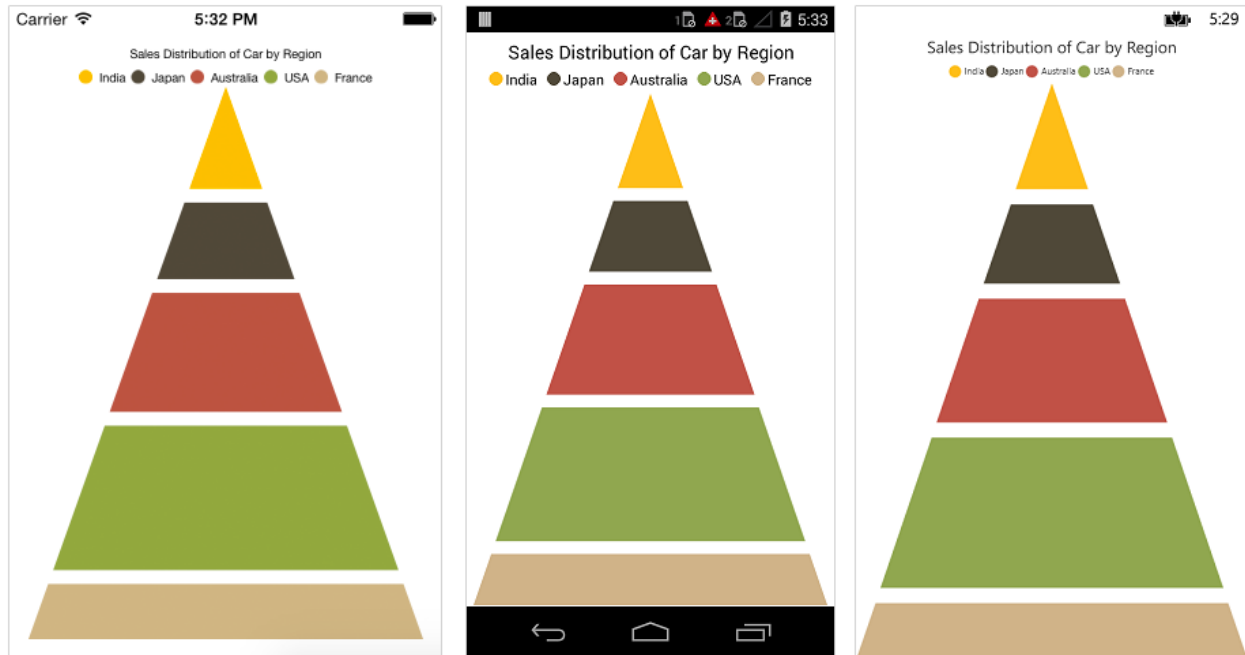
You can control the gap between the two segments using [GapRatio](#) property. Its ranges from 0 to 1.

### **XML**

```
<chart:PyramidSeries ItemsSource="{Binding Data}" GapRatio="0.1"
XBindingPath="Country"
YBindingPath="Value"/>
```

### **C#**

```
PyramidSeries pyramidSeries = new PyramidSeries()
{
    ItemsSource = Data,
    XBindingPath = "Country",
    YBindingPath = "Value",
    GapRatio = 0.1
};
```



#### *Exploding a pyramid segment*

You can explode a pyramid segment using [ExplodeIndex](#) property, and [ExplodeOffset](#) property is used to specify the exploded segment's distance.

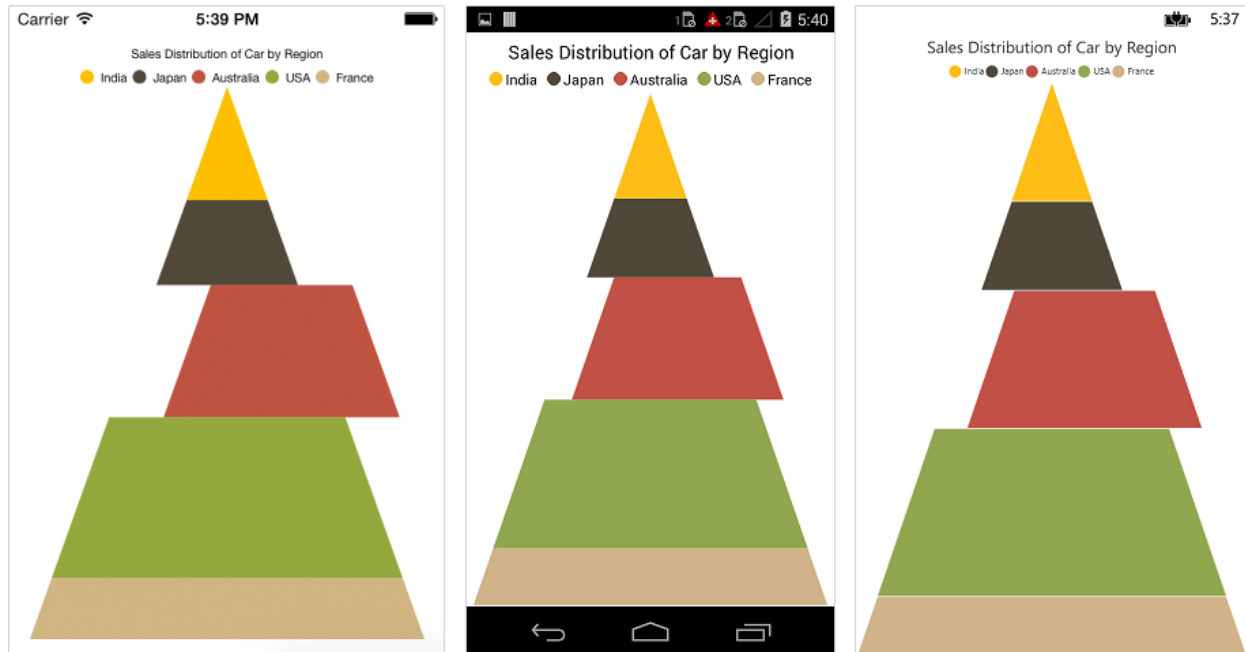
#### **XML**

```
<chart:PyramidSeries ItemsSource="{Binding Data}" ExplodeIndex="2"
XBindingPath="Country"
YBindingPath="Value" />
```

#### **C#**

```
PyramidSeries pyramidSeries = new PyramidSeries()
{
    ItemsSource = Data,
    XBindingPath = "Country",
    YBindingPath = "Value",
    ExplodeIndex = 2
};
```

Also, the segments can be exploded by touch using [ExplodeOnTouch](#) property of [PyramidSeries](#). Default value of this property is false.



### Funnel Chart

To render a funnel chart, create an instance of [FunnelSeries](#) and add to the [Series](#) collection property of [SfChart](#). You can use the following properties to customize the funnel segment appearance.

- [Color](#) – used to change the color of the series.
- [Opacity](#) - used to control the transparency of the chart series.
- [StrokeWidth](#) – used to change the stroke width of the series.
- [StrokeColor](#) – used to change the stroke color of the series.

### XML

```
<chart:SfChart>
...
<chart:FunnelSeries ItemsSource="{Binding Data}" XBindingPath="Status"
YBindingPath="Value"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
FunnelSeries funnelSeries = new FunnelSeries()
{
    ItemsSource = Data,
    XBindingPath = "Status",
    YBindingPath = "Value"
};
chart.Series.Add(funnelSeries);
```



### *Gap between the segments*

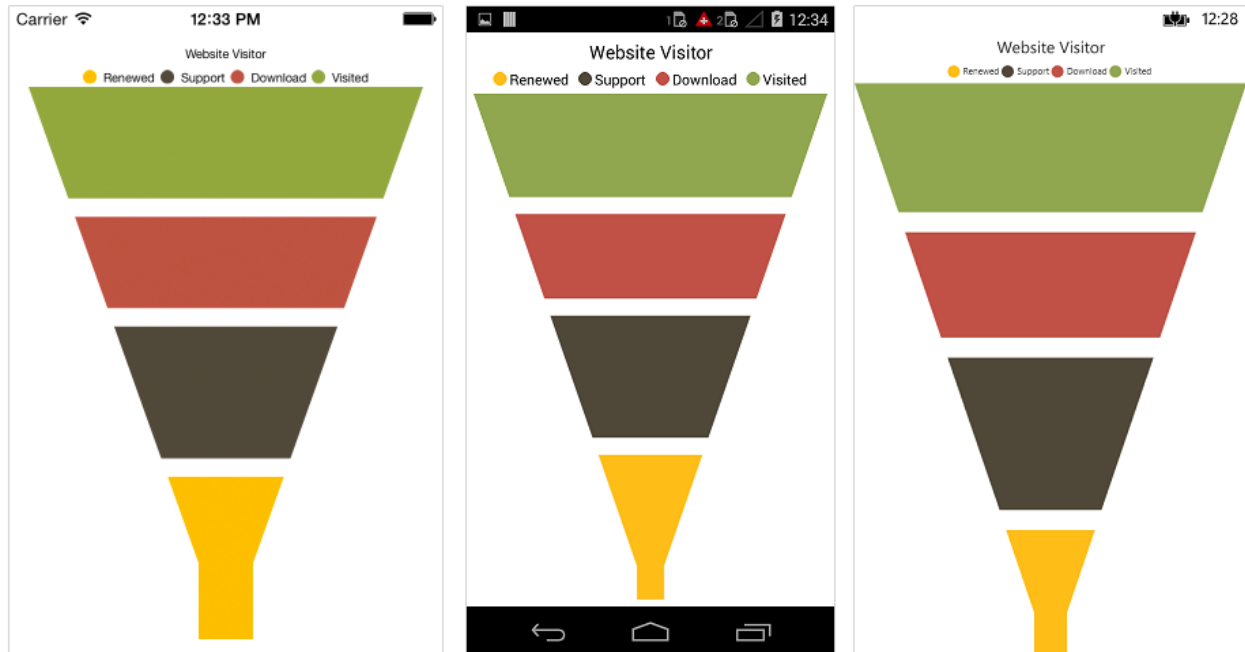
You can control the gap between the two segments using [GapRatio](#) property. Its ranges from 0 to 1.

### **XML**

```
<chart:FunnelSeries ItemsSource="{Binding Data}" GapRatio="0.1"
XBindingPath="Year"
YBindingPath="Value"/>
```

### **C#**

```
FunnelSeries funnelSeries = new FunnelSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    YBindingPath = "Value",
    GapRatio = 0.1
};
```



#### Exploding a funnel segment

You can explode a pyramid segment using [ExplodeIndex](#) property and [ExplodeOffset](#) property is used to specify the exploded segment's distance.

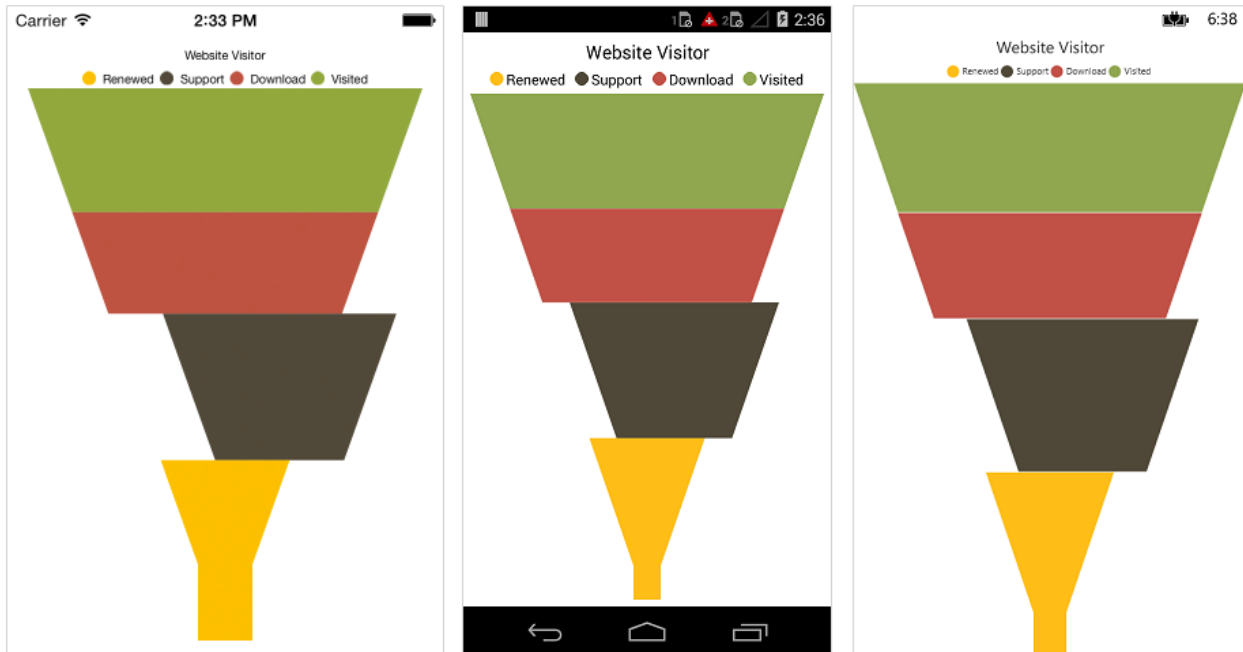
#### XML

```
<chart:FunnelSeries ItemsSource="{Binding Data}" ExplodeIndex="1"
XBindingPath="Status"
YBindingPath="Value" />
```

#### C#

```
FunnelSeries funnelSeries = new FunnelSeries()
{
    ItemsSource = Data,
    XBindingPath = "Status",
    YBindingPath = "Value",
    ExplodeIndex = 1
};
```

Also, the segments can be exploded by touch using [ExplodeOnTouch](#) property of [FunnelSeries](#). Default value of this property is false.



#### *Changing the minimum width of the funnel*

You can change the minimum width of the funnel neck using [MinWidth](#) property of [FunnelSeries](#). Default value of minWidth is 40.

#### **XML**

```
<chart:FunnelSeries ItemsSource="{Binding Data}" MinWidth="20"
XBindingPath="Year"
YBindingPath="Value"/>
```

#### **C#**

```
FunnelSeries funnelSeries = new FunnelSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    YBindingPath = "Value",
    MinWidth = 20
};
```





### Waterfall Chart

[WaterfallSeries](#) clarifies the cumulative effect of a set of provided positive and negative values. The series is represented by a rectangle and a connector between the rectangles.

- [SummaryBindingPath](#) – Gets or sets the string value that indicates the sum of previous segments in series.
- [SummarySegmentColor](#) – Changes the color of summary segment in series.
- [NegativeSegmentColor](#) – Changes the color of negative segment in series.
- [AllowAutoSum](#) – Enables or disables the segment that has been drawn based on the sum value of previous segments. By default, the value of this property is true. When disabling this property, it renders the segment by using the y-value of provided ItemsSource collection.
- [ShowConnectorLine](#) – Enables or disables the connector line of series. By default, value of this property is true.
- [ConnectorLineStyle](#) – Customizes the appearance of connector line style.

### XML

```
<chart:WaterfallSeries ItemsSource="{Binding RevenueDetails}"
XBindingPath="Category"
YBindingPath="Value"
AllowAutoSum="True"
SummaryBindingPath="IsSummary"
NegativeSegmentColor="#F14C72"
SummarySegmentColor="#8C8C8C"
ShowConnectorLine="True">
</chart:WaterfallSeries>
```

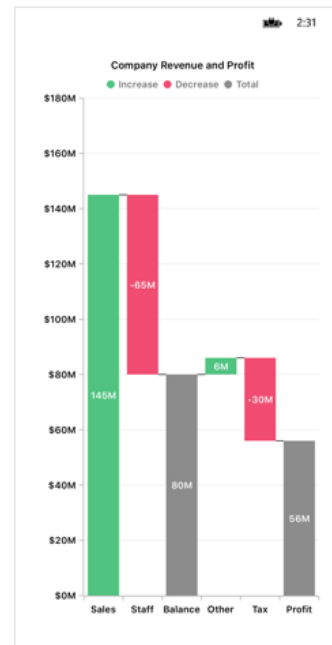
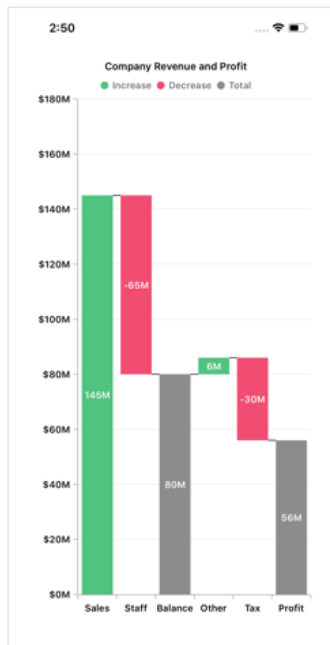
### C#

```
WaterfallSeries waterfallSeries = new WaterfallSeries()
{
    ItemsSource = RevenueDetails,
    XBindingPath = "Category",
    YBindingPath = "Value",
    AllowAutoSum = true,
```

```

SummaryBindingPath = "IsSummary",
NegativeSegmentColor = Color.FromHex("#F14C72"),
SummarySegmentColor = Color.FromHex("#8C8C8C"),
ShowConnectorLine = true
};
chart.Series.Add(waterfallSeries);

```



## Data Markers

Data markers are used to provide information about the data points to the user. You can add a shape and label to adorn each data point. This can be enabled using following code snippet,

### XML

```

<chart:LineSeries>
<chart:LineSeries.DataMarker>
<chart:ChartDataMarker/>
</chart:LineSeries.DataMarker>
</chart:LineSeries>

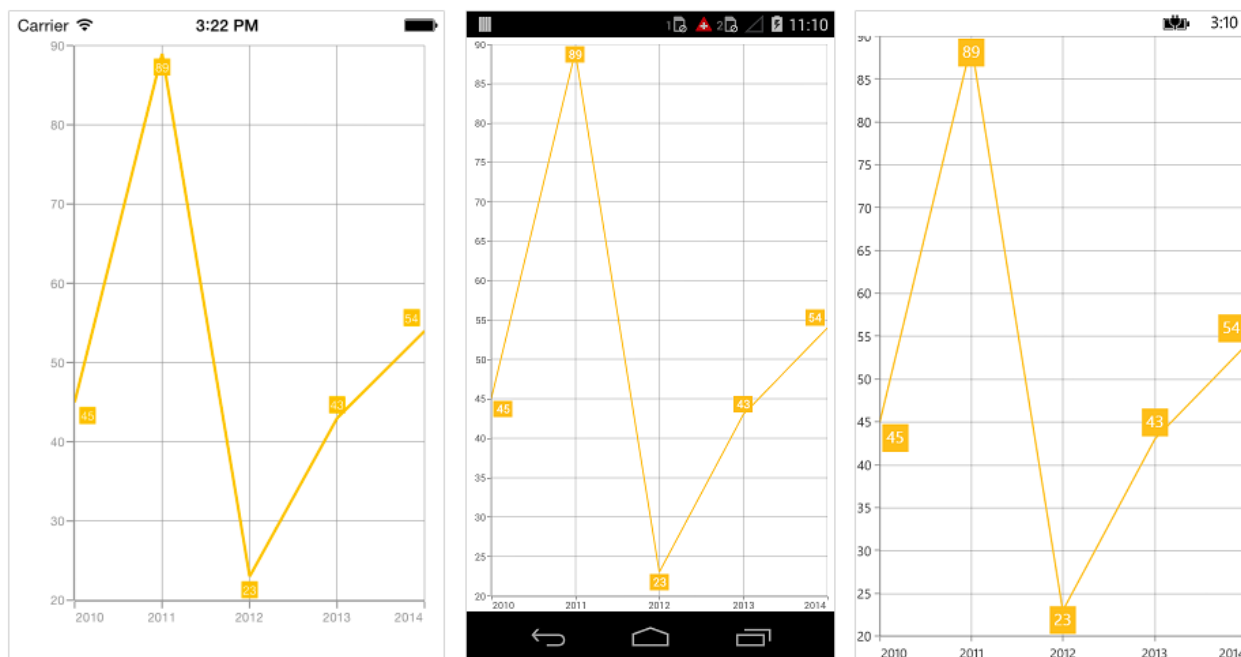
```

### C#

```

lineSeries.DataMarker = new ChartDataMarker();

```



### Customizing labels

Data labels are enabled by default but you can also change the visibility of the labels using [ShowLabel](#) property of [ChartDataMarker](#). The label appearance can be customized using [LabelStyle](#) property.

- [TextColor](#) – used to change the color of the label.
- [BackgroundColor](#) – used to change the label background color.
- [BorderColor](#) – used to change the border color.
- [BorderThickness](#) – used to change the thickness of the border.
- [Font](#) – used to change the text size, font family and font weight.
- [Margin](#) – used to change the margin size for labels.
- [Angle](#) – used to rotate the labels.
- [LabelPadding](#) – used to move the data label in the respective direction. For example, the positive labels in column series will be moved upwards and negative labels in column series will be moved downwards.

Following code snippet illustrates the customization of label and its background,

### XML

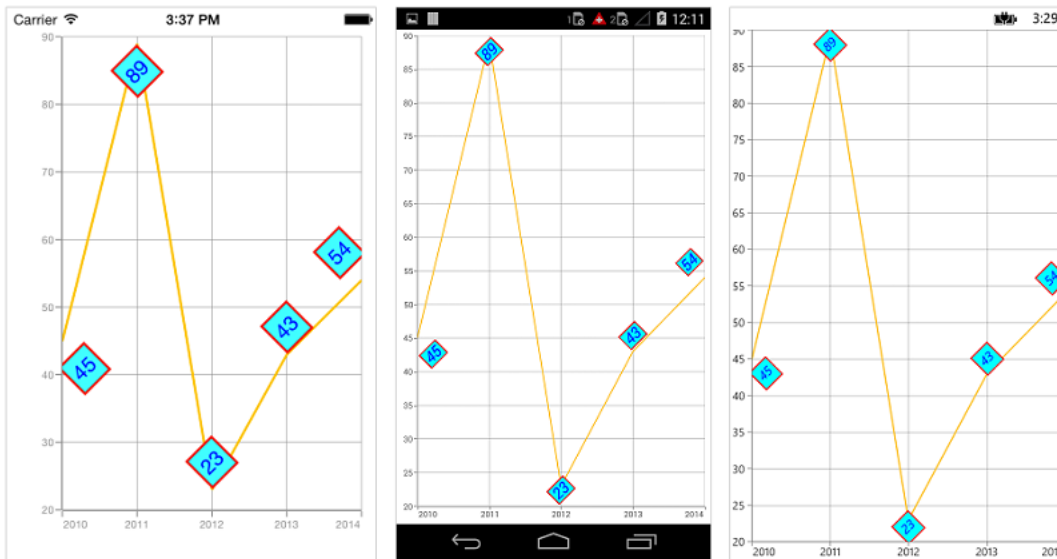
```
<chart:LineSeries.DataMarker>
  <chart:ChartDataMarker ShowLabel="True">
    <chart:ChartDataMarker.LabelStyle>
      <chart:DataMarkerLabelStyle TextColor="Blue"
        BorderColor="Red"
        BorderThickness="2"
        BackgroundColor="Aqua"
        Angle="315"
        Margin="5"
        Font="Italic,18"/>
    </chart:ChartDataMarker.LabelStyle>
  </chart:ChartDataMarker>
</chart:LineSeries.DataMarker>
```

**C#**

```

lineSeries.DataMarker = new ChartDataMarker();
lineSeries.DataMarker.ShowLabel = true;
lineSeries.DataMarker.LabelStyle.TextColor = Color.Blue;
lineSeries.DataMarker.LabelStyle.BorderColor = Color.Red;
lineSeries.DataMarker.LabelStyle.BorderThickness = 2;
lineSeries.DataMarker.LabelStyle.BackgroundColor = Color.Aqua;
lineSeries.DataMarker.LabelStyle.Angle = 315;
lineSeries.DataMarker.LabelStyle.Margin = 5;
lineSeries.DataMarker.LabelStyle.Font = Font.SystemFontOfSize(18,
FontAttributes.Italic);

```

**Formatting label content**

You can customize the content of the label using [LabelContent](#) property. Following are the two options that are supported now,

- [Percentage](#) – This will show the percentage value of corresponding data point Y value, this is often used in pie, doughnut, funnel and pyramid series types.
- [YValue](#) – This will show the corresponding Y value.

**XML**

```

<chart:PieSeries>
<chart:PieSeries.DataMarker>
<chart:ChartDataMarker LabelContent="Percentage"/>
</chart:PieSeries.DataMarker>
</chart:PieSeries>

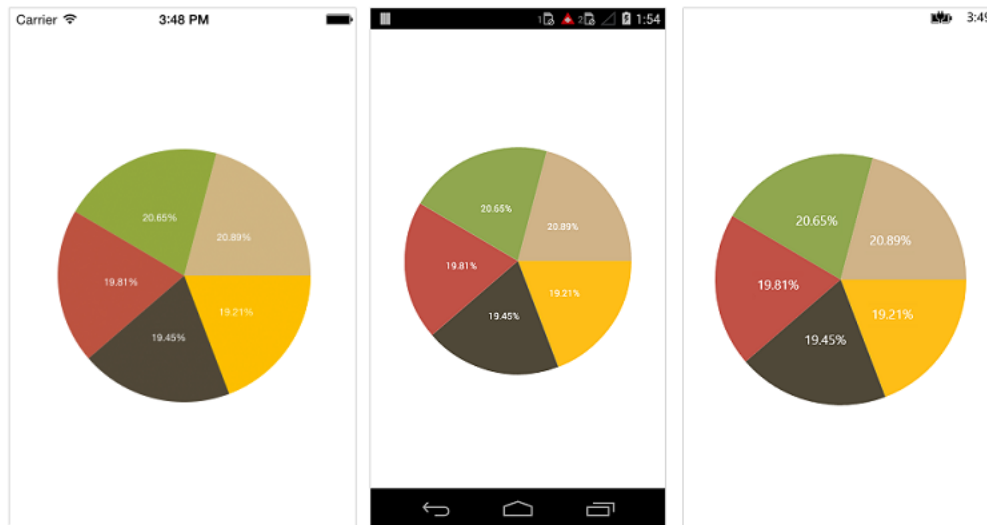
```

**C#**

```

pieSeries.DataMarker.LabelContent = LabelContent.Percentage;

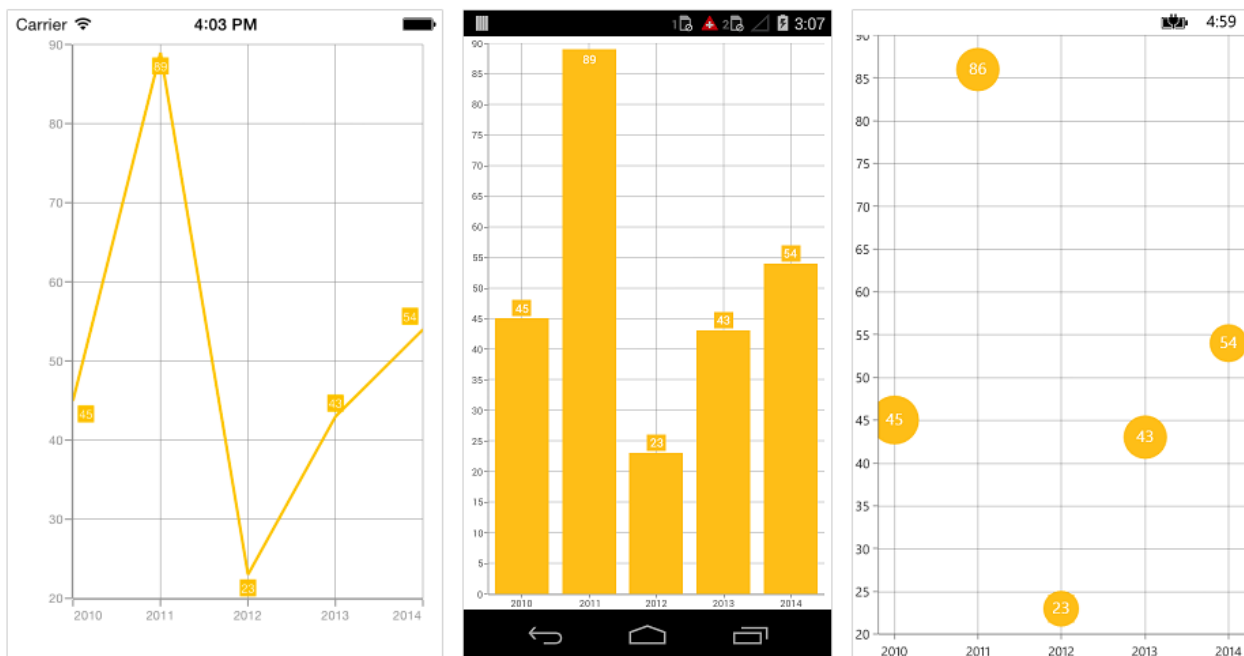
```



### Label position

The [LabelPosition](#) property is used to position the data marker labels at [Center](#), [Inner](#) and [Outer](#) position of the actual data point position. By default, labels are positioned based on the series types for better readability. You can move the labels horizontally and vertically using [OffsetX](#) and [OffsetY](#) properties respectively.

The following screenshot illustrates the default position of data marker labels,



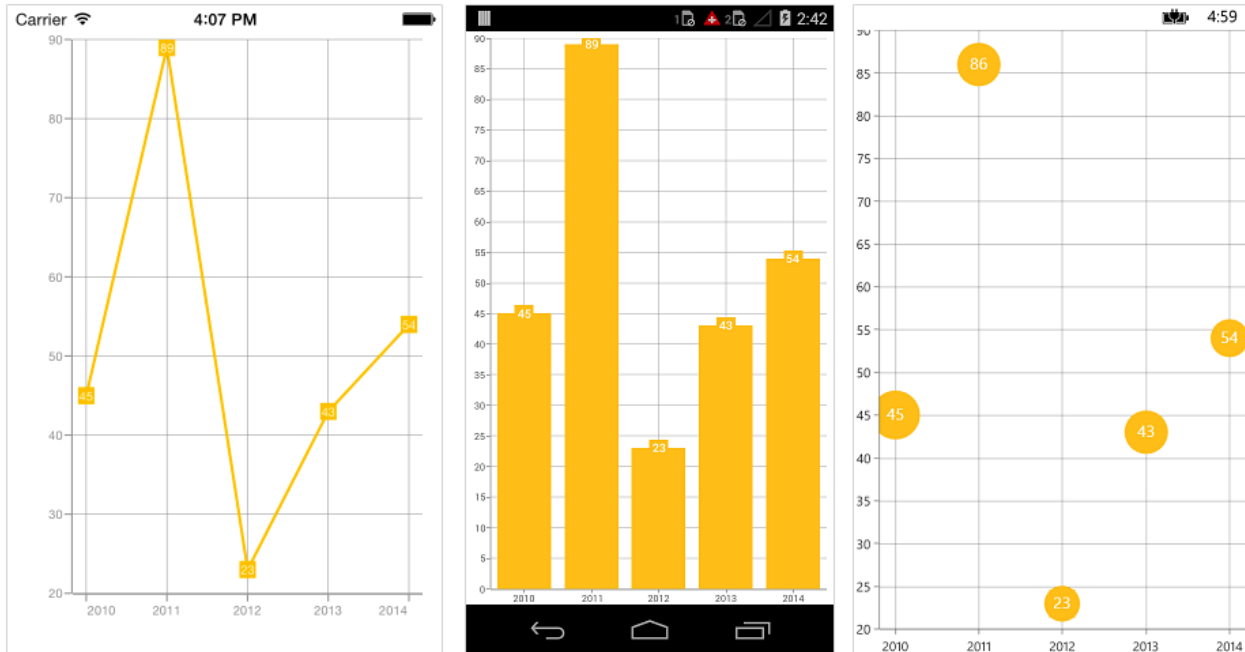
The following code sample illustrates the center position of data marker labels,

### XML

```
<chart:ChartDataMarker>
  <chart:ChartDataMarker.LabelStyle>
    <chart:DataMarkerLabelStyle LabelPosition="Center"/>
  </chart:ChartDataMarker.LabelStyle>
</chart:ChartDataMarker>
```

**C#**

```
series.DataMarker.LabelStyle.LabelPosition = DataMarkerLabelPosition.Center;
```



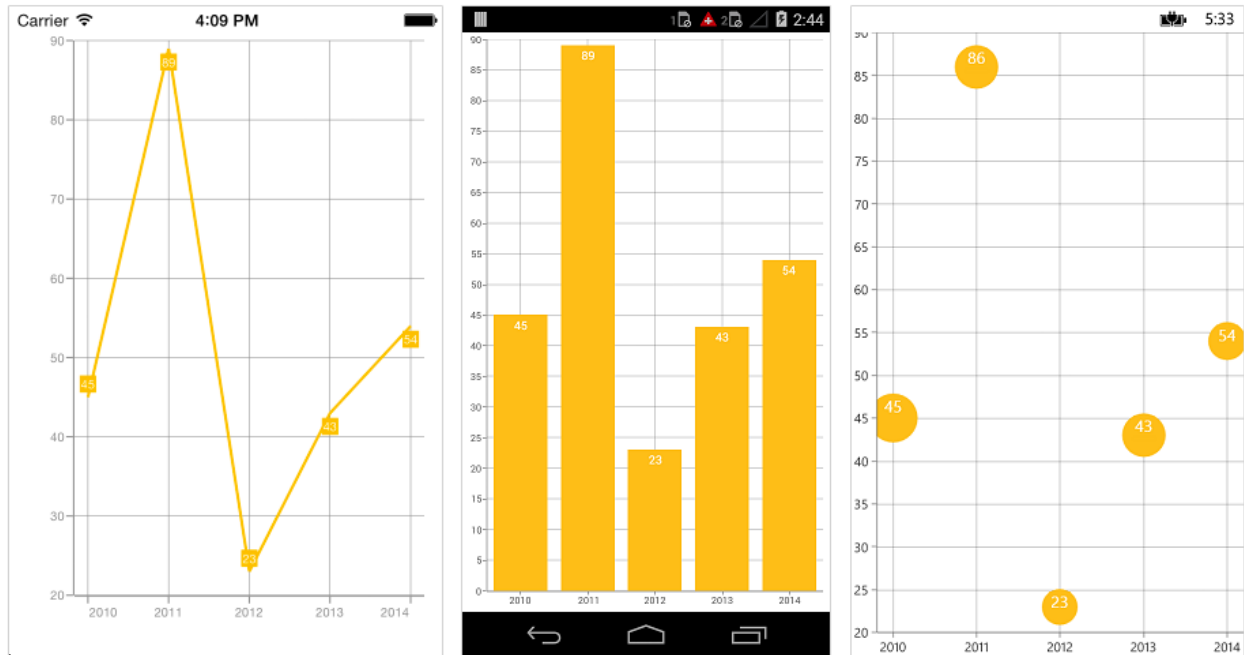
The following code sample illustrates the Inner position of data marker labels,

**XML**

```
<chart:ChartDataMarker>
  <chart:ChartDataMarker.LabelStyle>
    <chart:DataMarkerLabelStyle LabelPosition="Inner"/>
  </chart:ChartDataMarker.LabelStyle>
</chart:ChartDataMarker>
```

**C#**

```
series.DataMarker.LabelStyle.LabelPosition = DataMarkerLabelPosition.Inner;
```



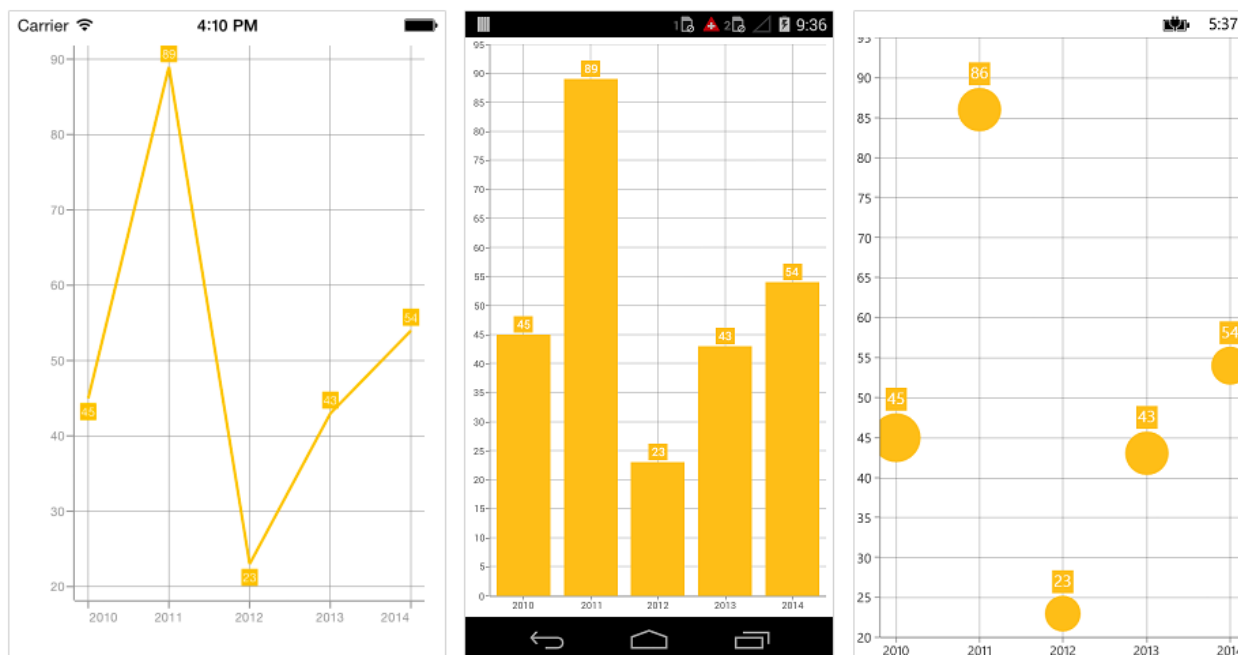
The following code sample illustrates the outer position of data marker labels,

#### XML

```
<chart:ChartDataMarker>
  <chart:ChartDataMarker.LabelStyle>
    <chart:DataMarkerLabelStyle LabelPosition="Outer"/>
  </chart:ChartDataMarker.LabelStyle>
</chart:ChartDataMarker>
```

#### C#

```
series.DataMarker.LabelStyle.LabelPosition = DataMarkerLabelPosition.Outer;
```



### Smart labels

This feature is used to arrange the data marker labels smartly and avoid the intersection when there is overlapping of labels. The property [EnableSmartLabels](#) in [CircularSeries](#), is used to arrange the data marker labels smartly. By default, it is false, we need to enable this property.

The following code sample illustrates how to enable the smart labels.

### XML

```
<chart:SfChart.Series>
  <chart:PieSeries ItemsSource="{Binding Data}"
    XBindingPath="Expense"
    YBindingPath="Value"
    StartAngle="75"
    EndAngle="435"
    EnableSmartLabels="True"
    ConnectorLineType="Bezier"
    DataMarkerPosition="OutsideExtended">
    <chart:PieSeries.DataMarker>
      <chart:ChartDataMarker />
    </chart:PieSeries.DataMarker>
  </chart:PieSeries>
</chart:SfChart.Series>
```

### C#

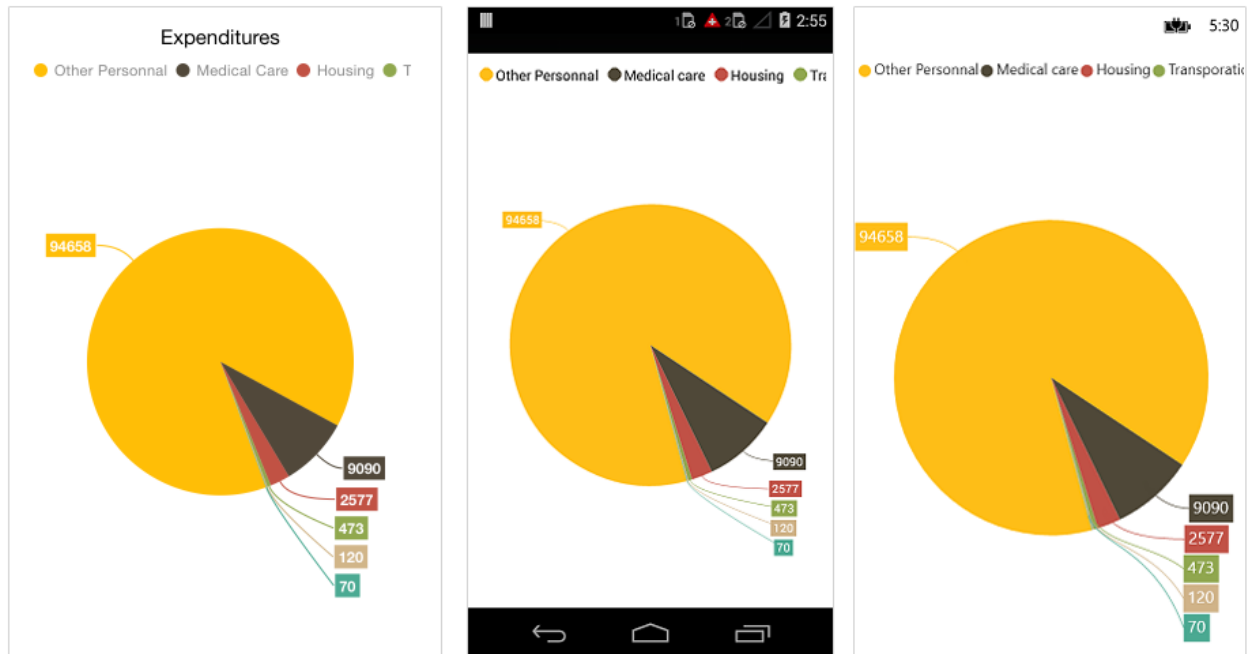
```
SfChart chart = new SfChart();
...
PieSeries pieSeries = new PieSeries()
{
    ItemsSource = Data,
    XBindingPath = "Expense",
    YBindingPath = "Value",
    EnableSmartLabels = true,
```



```

DataMarkerPosition = CircularSeriesDataMarkerPosition.OutsideExtended,
ConnectorLineType= ConnectorLineType.Bezier,
StartAngle=75,
EndAngle=435,
DataMarker=new ChartDataMarker(),
};
chart.Series.Add(pieSeries);

```



### Customizing marker shapes

Shapes can be added to chart data marker by setting the [ShowMarker](#) property to `true`. There are different shapes you can set to the chart using [MarkerType](#) property such as [Triangle](#), [Circle](#), [Diamond](#) etc. Following properties are used to customize marker appearance,

- [MarkerWidth](#) - used to change the width of the marker
- [MarkerHeight](#) - used to change the height of the marker
- [MarkerColor](#) - used to change the color of the marker
- [MarkerBorderColor](#) - used to change the border color of the shape
- [MarkerBorderWidth](#) – used to change the marker border thickness

The following code example shows how to enable marker and specify its types,

### XML

```

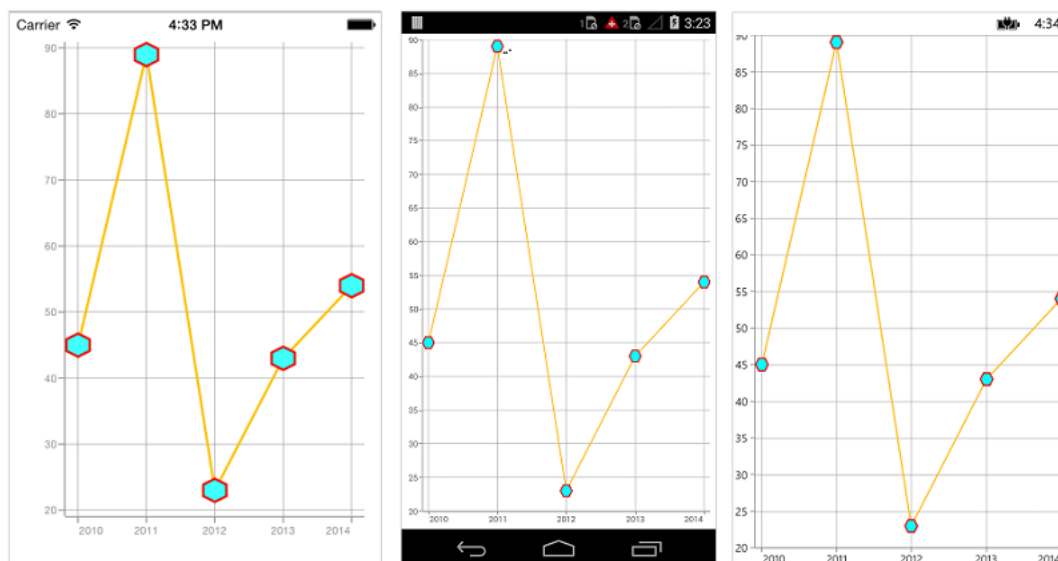
<chart:LineSeries.DataMarker>
<chart:ChartDataMarker ShowLabel="False"
ShowMarker="True"
MarkerType="Hexagon"
MarkerWidth="20"
MarkerHeight="20"
MarkerColor="Aqua"
MarkerBorderColor="Red"
MarkerBorderWidth="2"/>

```

```
</chart:LineSeries.DataMarker>
```

### C#

```
lineSeries.DataMarker = new ChartDataMarker();
lineSeries.DataMarker.ShowLabel = false;
lineSeries.DataMarker.ShowMarker = true;
lineSeries.DataMarker.MarkerType = DataMarkerType.Hexagon;
lineSeries.DataMarker.MarkerWidth = 20;
lineSeries.DataMarker.MarkerHeight = 20;
lineSeries.DataMarker.MarkerColor = Color.Aqua;
lineSeries.DataMarker.MarkerBorderColor = Color.Red;
lineSeries.DataMarker.MarkerBorderWidth = 2;
```



### Apply series color

The [UseSeriesPalette](#) property is used to apply the series color to background color of data marker labels. The default value of this property is true.

### XML

```
<chart:ColumnSeries>
  <chart:ColumnSeries.DataMarker>
    <chart:ChartDataMarker UseSeriesPalette="False"/>
  </chart:ColumnSeries.DataMarker>
</chart:ColumnSeries>
```

### C#

```
columnSeries.DataMarker = new ChartDataMarker();
columnSeries.DataMarker.UseSeriesPalette = false;
```

### Connector line

This feature is used to connect label and data point using a line. It can be enabled for any chart types but this is often used with Pie and Doughnut chart types. The [ConnectorLineStyle](#) property used to customize the connector line.

- [StrokeColor](#) – used to change the color of the line
- [StrokeWidth](#) – used to change the stroke thickness of the line
- [StrokeDashArray](#) – used to set the dashes for the line
- [ConnectorHeight](#) - used to set the height of the line.
- [ConnectorRotationAngle](#) - used to set the rotation angle of the line.

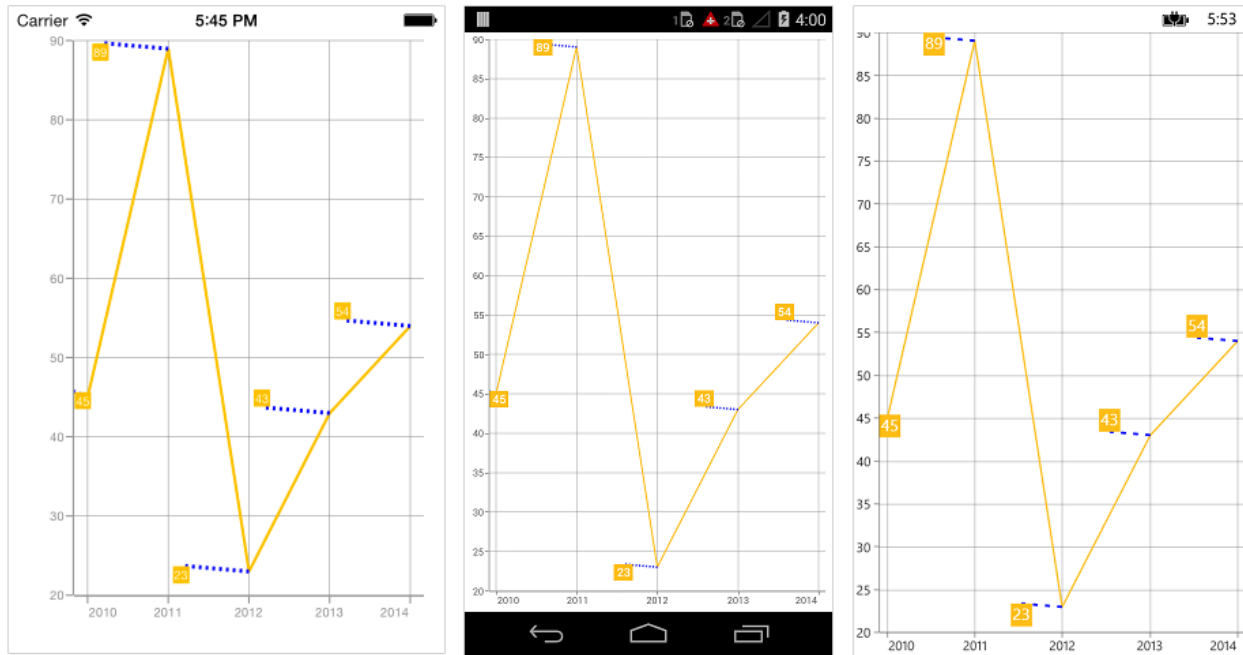
The following code illustrates how to specify the connector height and its angle,

#### XML

```
<chart:LineSeries.DataMarker>
<chart:ChartDataMarker>
<chart:ChartDataMarker.ConnectorLineStyle>
<chart:ConnectorLineStyle ConnectorHeight="50"
ConnectorRotationAngle="175"
StrokeColor="Blue"
StrokeWidth="3"/>
</chart:ChartDataMarker.ConnectorLineStyle>
</chart:ChartDataMarker>
</chart:LineSeries.DataMarker>
```

#### C#

```
lineSeries.DataMarker.ConnectorLineStyle.ConnectorHeight = 50;
lineSeries.DataMarker.ConnectorLineStyle.ConnectorRotationAngle = 175;
lineSeries.DataMarker.ConnectorLineStyle.StrokeColor = Color.Blue;
lineSeries.DataMarker.ConnectorLineStyle.StrokeWidth = 3;
lineSeries.DataMarker.ConnectorLineStyle.StrokeDashArray = new double[2] {
2, 3 };
```



**Note:** For Pie and Doughnut series, you can set different connector line types such as [Line](#), [StraightLine](#) and [Bezier](#) curve using the [ConnectorType](#) property of pie and doughnut series.

### Label template

You can customize the appearance of the data marker label with your own template using the [LabelTemplate](#) property of [ChartDataMarker](#).

**Note:** The BindingContext of template is the corresponding underlying model provided in the items source of chart series. You can also bind the corresponding [ChartDataMarkerLabel](#) class object using the [LabelContent](#) property set as [DataMarkerLabel](#).

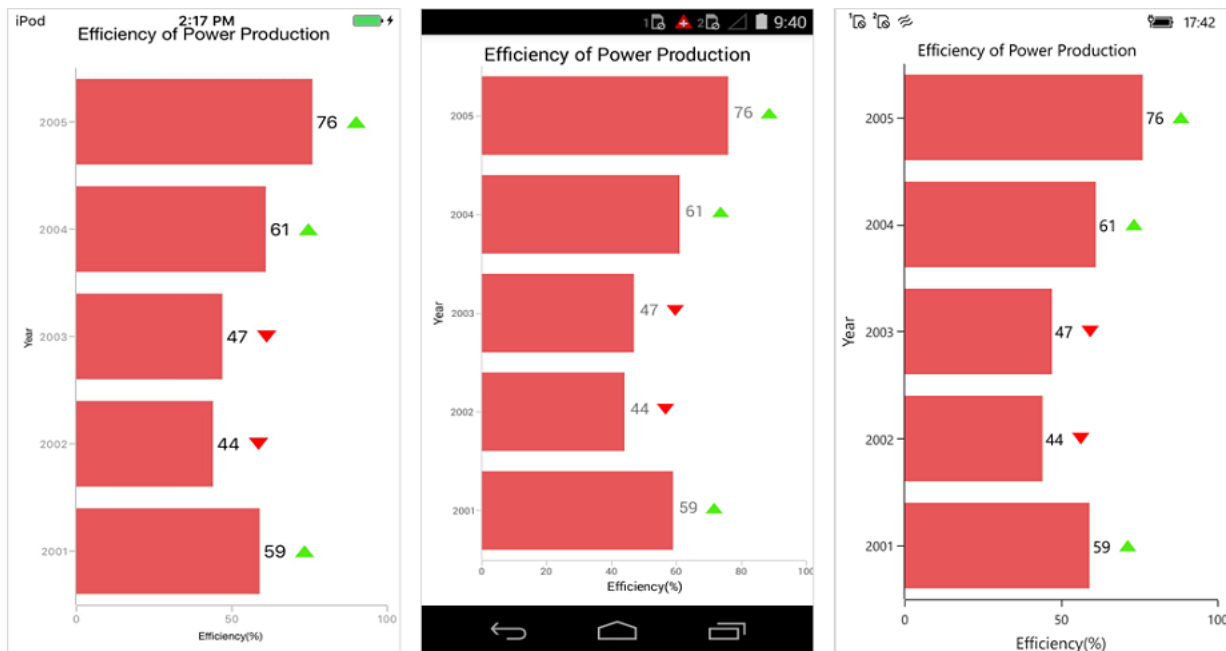
### XML

```
<ContentPage.Resources>
  <ResourceDictionary>
    <DataTemplate x:Key="dataMarkerTemplate">
      <StackLayout Orientation="Horizontal">
        <Label Text="{Binding Value}" VerticalOptions="Center" FontSize = "15"/>
        <Image Source="Down.jpg" WidthRequest="30" HeightRequest="30"/>
      </StackLayout>
    </DataTemplate>
  </ResourceDictionary>
</ContentPage.Resources>
<chart:SfChart.Series>
  <chart:BarSeries ItemsSource="{Binding Data}" XBindingPath="Name"
    YBindingPath="Value">
    <chart:BarSeries.DataMarker>
      <chart:ChartDataMarker ShowLabel="True" LabelTemplate="{StaticResource
        dataMarkerTemplate}">
      <chart:ChartDataMarker.LabelStyle>
        <chart:DataMarkerLabelStyle LabelPosition="Outer" />
      </chart:ChartDataMarker.LabelStyle>
    </chart:ChartDataMarker>
  </chart:BarSeries.DataMarker>
</chart:BarSeries.DataMarker>
```

```
</chart:BarSeries>  
</chart:SfChart.Series>
```

## C#

```
SfChart chart = new SfChart ();  
...  
var barSeries = new BarSeries();  
barSeries.Color = Color.FromRgb(231, 87, 89);  
barSeries.ItemsSource = Data;  
barSeries.XBindingPath = "Name";  
barSeries.YBindingPath = "Value";  
barSeries.DataMarker = new ChartDataMarker();  
barSeries.DataMarker.ShowLabel = true;  
barSeries.DataMarker.LabelStyle.LabelPosition =  
DataMarkerLabelPosition.Outer;  
DataTemplate dataMarkerTemplate = new DataTemplate(() =>  
{  
    StackLayout stack = new StackLayout();  
    stack.Orientation = StackOrientation.Horizontal;  
    Label label = new Label();  
    label.SetBinding(Label.TextProperty, "Value");  
    label.FontSize = 15;  
    label.VerticalOptions = LayoutOptions.Center;  
    Image image = new Image();  
    image.Source = "Down.jpg";  
    image.WidthRequest = 30;  
    image.HeightRequest = 30;  
    stack.Children.Add(label);  
    stack.Children.Add(image);  
    return stack;  
});  
barSeries.DataMarker.LabelTemplate = dataMarkerTemplate;  
chart.Series.Add(barSeries);
```



## Event

### [DataMarkerLabelCreated](#)

The [DataMarkerLabelCreated](#) event occurs when the data marker label is created. This argument contains object of the [ChartDataMarkerLabel](#). The following properties are available in [DataMarkerLabel](#) to customize the appearance of data markers based on condition.

- [Label](#) – Gets or sets the text of data marker.
- [BackgroundColor](#) – Gets the background color of data marker label.
- [Index](#) – Gets the data point index of data marker label.
- [XPosition](#) – Gets the x-position of data marker label.
- [YPosition](#) – Gets the y-position of data marker label.
- [LabelStyle](#) – Gets or sets the label style to customize the appearance of individual data marker label.
- [MarkerWidth](#) – Gets or sets the marker width.
- [MarkerHeight](#) – Gets or sets the marker height.
- [MarkerBorderWidth](#) – Gets or sets the border width of marker symbol.
- [MarkerBorderColor](#) – Gets or sets the border color of marker symbol.
- [MarkerColor](#) – Gets or sets the marker color.
- [MarkerType](#) – Gets or sets the shape type of marker. The available shapes are ellipse, diamond, hexagon, cross, HorizontalLine, VerticalLine, InvertedTriangle, triangle, pentagon, plus, and square.
- [Data](#) - Gets the underlying data of data marker label.

## Legend

The [Legend](#) contains list of chart series/data points in chart. The information provided in each legend item helps to identify the corresponding data series in chart.

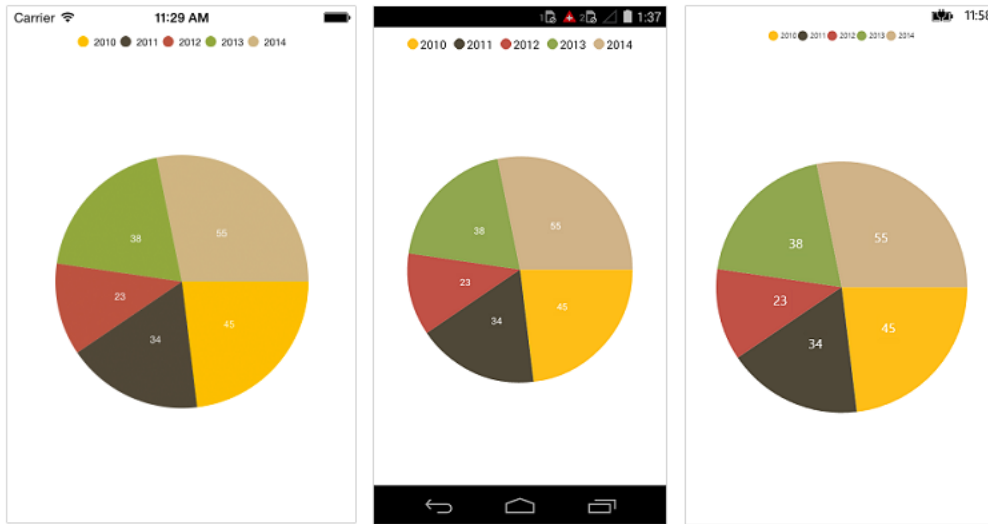
The following code example shows how to enable legend in chart.

### **XML**

```
<chart:SfChart>
<chart:SfChart.Legend>
<chart:ChartLegend/>
</chart:SfChart.Legend>
</chart:SfChart>
```

**C#**

```
chart.Legend = new ChartLegend();
```

**Customizing labels**

The [Label](#) property of [ChartSeries](#) is used to define the label for the corresponding series legend item. The appearance of the label can be customized using the [LabelStyle](#) property.

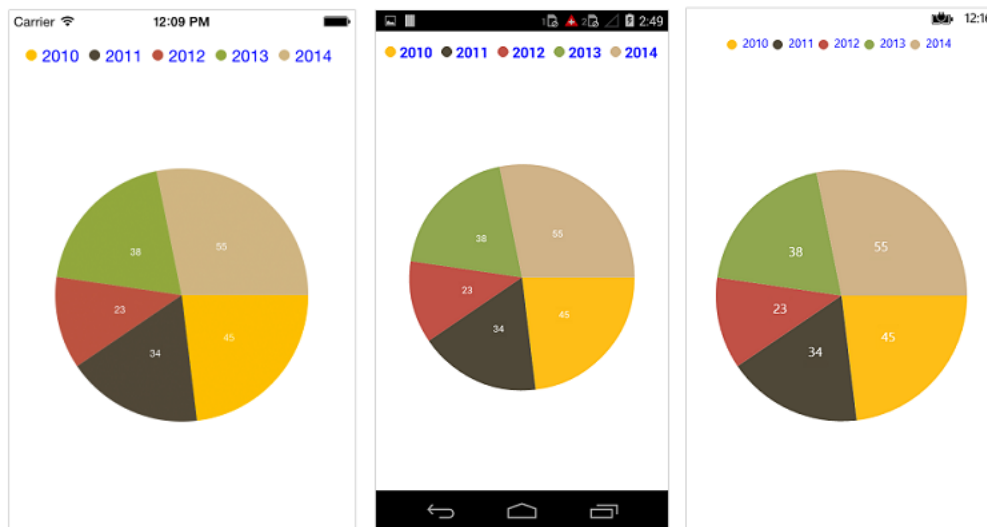
- [TextColor](#) – used to change the color of the label.
- [Font](#) – used to change the text size, font family, and font weight.
- [Margin](#) - used to change the margin size of labels.

**XML**

```
<chart:SfChart.Legend>
<chart:ChartLegend>
<chart:ChartLegend.LabelStyle>
<chart:ChartLegendLabelStyle TextColor="Blue" Margin="5" Font="Bold,18"/>
</chart:ChartLegend.LabelStyle>
</chart:ChartLegend>
</chart:SfChart.Legend>
```

**C#**

```
chart.Legend = new ChartLegend();
chart.Legend.LabelStyle.TextColor = Color.Blue;
chart.Legend.LabelStyle.Font = Font.SystemFontOfSize(18,
FontAttributes.Bold);
chart.Legend.LabelStyle.Margin = 5;
```



### Legend icons

The legend icons are enabled by default. However, you can control its visibility using the [IsIconVisible](#) property. The icon type also can be specified using the [LegendIcon](#) property such as [Rectangle](#), [Circle](#) and [Diamond](#), etc., The [IconWidth](#) and [IconHeight](#) properties are used to adjust the width and height of the legend icons, respectively.

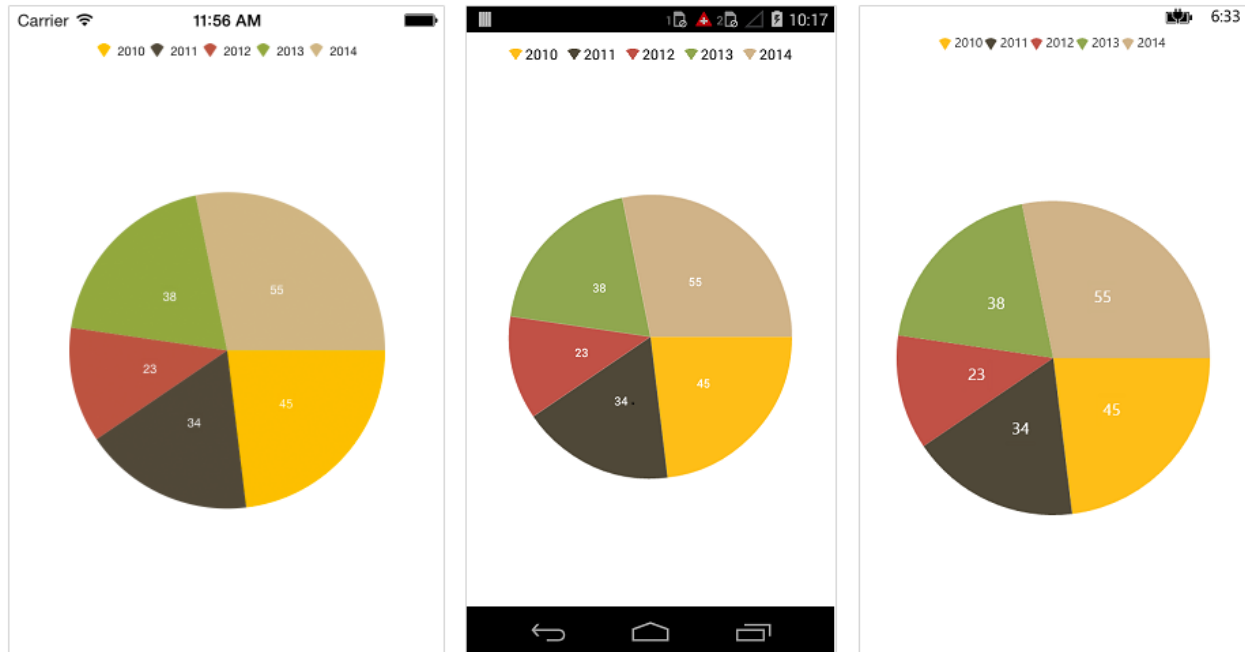
### XML

```
<chart:SfChart>
  <chart:SfChart.Legend>
    <chart:ChartLegend IsIconVisible="True" IconHeight="20" IconWidth="20"/>
  </chart:SfChart.Legend>
  <chart:PieSeries ItemsSource="{Binding Data1}" LegendIcon="SeriesType"/>
</chart:SfChart>
```

### C#

```
chart.Legend = new ChartLegend();
chart.Legend.IconHeight = 20;
chart.Legend.IconWidth = 20;
chart.Legend.IsIconVisible = true;
pieSeries.LegendIcon = ChartLegendIcon.SeriesType;
```





### Legend title

The following properties are used to define and customize the [Title](#) of legend.

- [Text](#) – used to change the text of the title.
- [TextColor](#) – used to change the color of the title text.
- [Font](#) – used to change the text size, font family, and font weight of the title. (This is deprecated API. Use [FontSize](#), [FontFamily](#), and [FontAttributes](#) properties instead of this.)
- [FontFamily](#) - used to change the font family for the legend label.
- [FontAttributes](#) - used to change the font style for the legend label.
- [FontSize](#) - used to change the font size for the legend label.
- [Margin](#) – used to change the margin size of title.
- [TextAlignment](#) – used to change the alignment of the title text; it can be start, end, or center.
- [BackgroundColor](#) – used to change the title background color.
- [BorderColor](#) – used to change the border color.
- [BorderWidth](#) – used to adjust the border width of title.

### XML

```
<chart:SfChart.Legend>
<chart:ChartLegend>
<chart:ChartLegend.Title >
<chart:ChartTitle Text="Years" TextColor="Maroon" TextAlignment="Center"
BackgroundColor="Silver" BorderWidth="3" BorderColor="Blue" Font="Bold,20"/>
</chart:ChartTitle>
</chart:ChartLegend.Title>
</chart:ChartLegend>
</chart:SfChart.Legend>
```

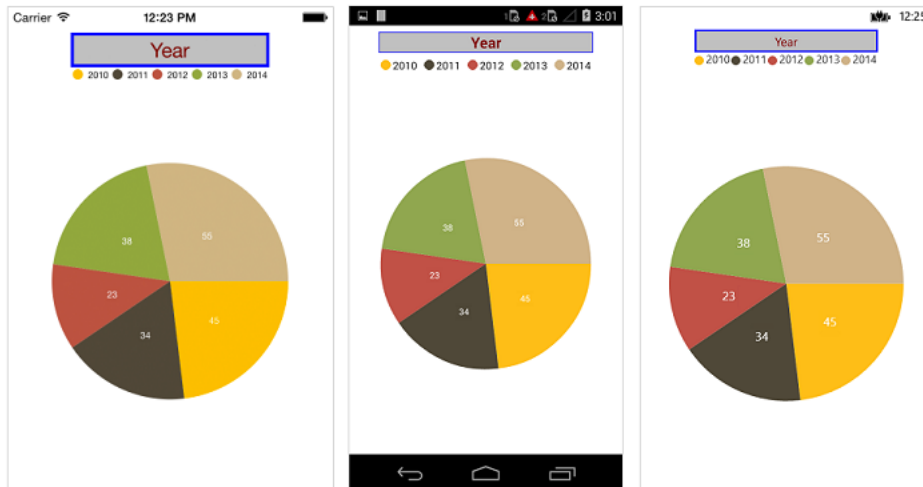
### C#

```
chart.Legend = new ChartLegend();
```

```

chart.Legend.Title.Text = "Year";
chart.Legend.Title.TextColor = Color.Maroon;
chart.Legend.Title.Font = Font.SystemFontOfSize(20, FontAttributes.Bold);
chart.Legend.Title.TextAlignment = TextAlignment.Center;
chart.Legend.Title.BackgroundColor = Color.Silver;
chart.Legend.Title.BorderWidth = 3;
chart.Legend.Title.BorderColor = Color.Blue;

```



### Toggle the series visibility

You can control the visibility of the series by tapping the legend item. You can enable this feature by enabling the [ToggleSeriesVisibility](#) property.

### XML

```

<chart:SfChart>
  <chart:SfChart.Legend>
    <chart:ChartLegend ToggleSeriesVisibility="True"/>
  </chart:SfChart.Legend>
</chart:SfChart>

```

### C#

```

chart.Legend = new ChartLegend();
chart.Legend.ToggleSeriesVisibility = true;

```

### Legend visibility

The [IsVisible](#) property of `ChartLegend` is used to toggle the visibility of legend.

### XML

```

<chart:SfChart>
  <chart:SfChart.Legend>
    <chart:ChartLegend IsVisible="False"/>
  </chart:SfChart.Legend>
</chart:SfChart>

```

### C#

```
chart.Legend = new ChartLegend();  
chart.Legend.IsVisible = false;
```

### Legend item visibility

You can control the visibility of a particular series' legend item using the [IsVisibleOnLegend](#) property of series. The default value of the [IsVisibleOnLegend](#) property is true.

#### XML

```
<chart:ColumnSeries ItemsSource="{Binding ColumnData}" XBindingPath="Name"  
YBindingPath="Value" IsVisibleOnLegend="true" >  
</chart:ColumnSeries>
```

#### C#

```
ColumnSeries column = new ColumnSeries();  
column.XBindingPath = "Name";  
column.YBindingPath = "Value";  
column.ItemsSource = viewModel.ColumnData;  
column.IsVisibleOnLegend = true;
```

### Item margin

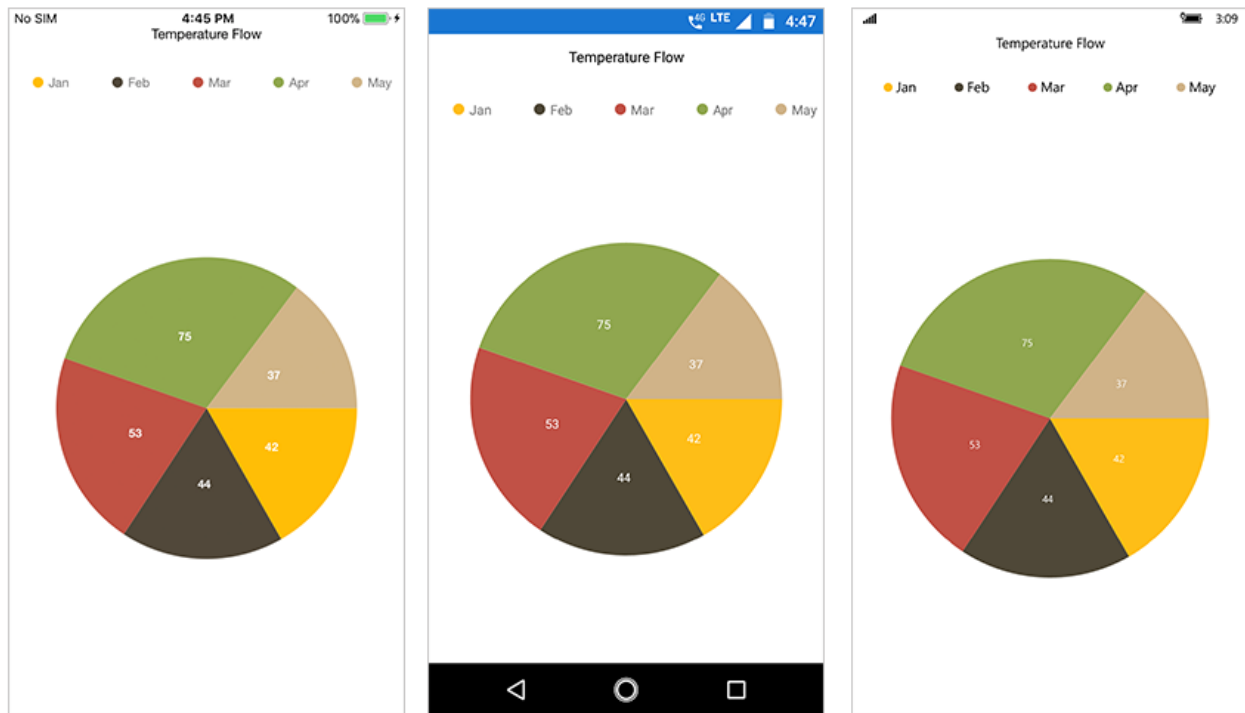
The [ItemMargin](#) property is used to set the spacing between the legend items.

#### XML

```
<chart:SfChart.Legend>  
<chart:ChartLegend ItemMargin="20"/>  
</chart:SfChart.Legend>
```

#### C#

```
chart.Legend = new ChartLegend();  
chart.Legend.ItemMargin = 20;
```



### Legend wrap

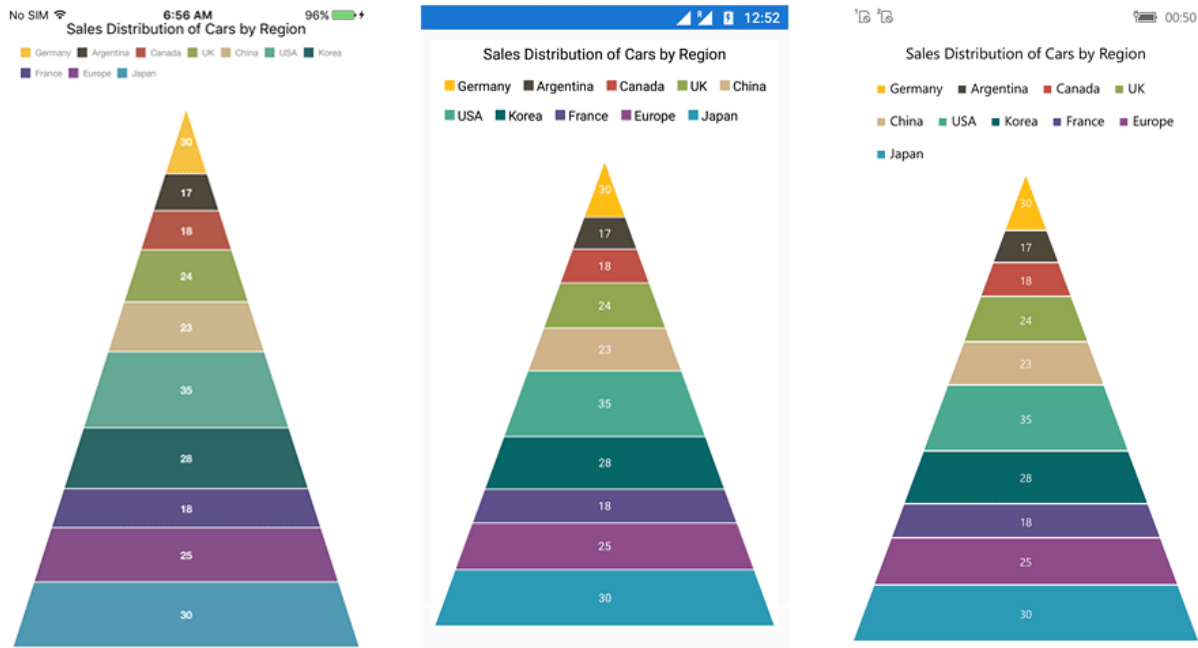
The legend items can be placed in multiple rows using the [OverflowMode](#) property if size of the total legend exceeds the available size. The default value of the [OverflowMode](#) property is [Scroll](#).

### XML

```
<chart:SfChart.Legend>
<chart:ChartLegend OverflowMode="Wrap" />
</chart:SfChart.Legend>
```

### C#

```
chart.Legend = new ChartLegend()
{
    OverflowMode = ChartLegendOverflowMode.Wrap
};
```



### Legend width

The legend width can be specified using the [MaxWidth](#) property. This property works only when the [OverflowMode](#) is [Wrap](#). The default value of the [MaxWidth](#) property is double.NAN.

### XML

```
<chart:SfChart>
  <chart:SfChart.Legend>
    <chart:ChartLegend OverflowMode = "Wrap" MaxWidth = "750" />
  </chart:SfChart.Legend>
</chart:SfChart>
```

### C#

```
chart.Legend = new ChartLegend()
{
    OverflowMode = ChartLegendOverflowMode.Wrap,
    MaxWidth = 750
};
```



### Positioning the legend

You can position the legend anywhere inside the chart. The following properties are used to customize the position of legend:

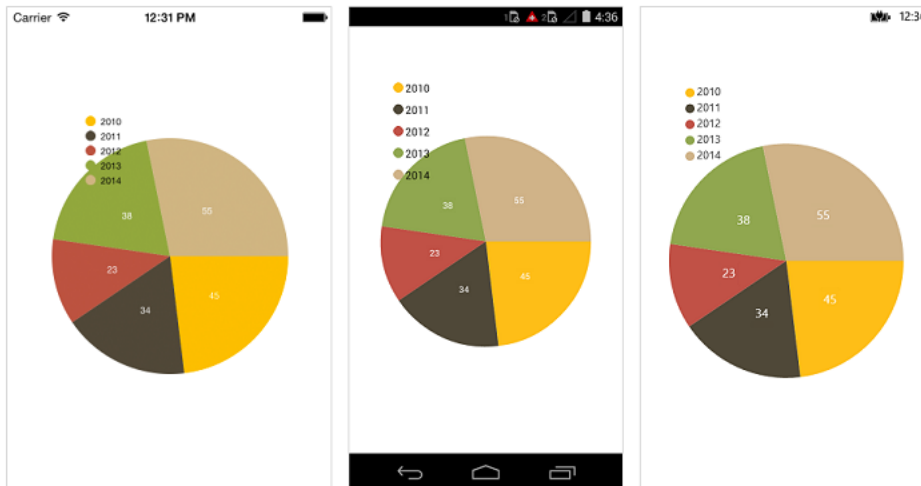
- [DockPosition](#) – used to position the legend relatively. The available options are [Left](#), [Right](#), [Top](#), [Bottom](#), and [Floating](#). If the DockPosition is Floating, you can position the legend using the x and y-coordinates.
- [OffsetX](#) – used to move the legend on x-coordinate by the given offset value.
- [OffsetY](#) - used to move the legend on y-coordinate by the given offset value.
- [Orientation](#) - used to change the orientation of the legend, the default value is Auto, orientation of the legend items will be changed based on its dock position. Also, you can manually set [Horizontal](#) or [Vertical](#).

### XML

```
<chart:SfChart>
  <chart:SfChart.Legend>
    <chart:ChartLegend DockPosition="Floating" OffsetX="70" OffsetY="90"
      Orientation="Vertical">
    </chart:SfChart.Legend>
  </chart:SfChart>
```

### C#

```
chart.Legend = new ChartLegend();
chart.Legend.DockPosition = LegendPlacement.Floating;
chart.Legend.Orientation = ChartOrientation.Vertical;
chart.Legend.OffsetX = 70;
chart.Legend.OffsetY = 90;
```



Populate the data based legend items for all series

The [Series](#) property of [ChartLegend](#) is used to populate the legend items based on the data points which are present in the assigned series.

The following code example shows how to enable datapoint-based legend for Cartesian series.

#### XML

```
<chart:SfChart>
...
<chart:ChartLegend x:Name="chartLegend" Series="{Binding Source={
x:Reference Series}}"/>
...
<chart:SfChart.Series>
<chart:ColumnSeries x:Name="Series" ItemsSource="{Binding CategoryData}"
XBindingPath="Name" YBindingPath="Value"/>
...
</chart:SfChart>
```

#### C#

```
...
ColumnSeries series = new ColumnSeries()
{
    ItemsSource = model.CategoryData,
    XBindingPath = "XValue",
    YBindingPath = "YValue",
};
Chart.Legend = new ChartLegend();
Chart.Legend.Series = series;
Chart.Series.Add(series);
...
```

#### ItemTemplate

You can customize the appearance of legend items with your template by using [ItemTemplate](#) property of [ChartLegend](#).

**Note:** The BindingContext of the template is the corresponding underlying legend item that provided in the ChartLegendItem class.

### XML

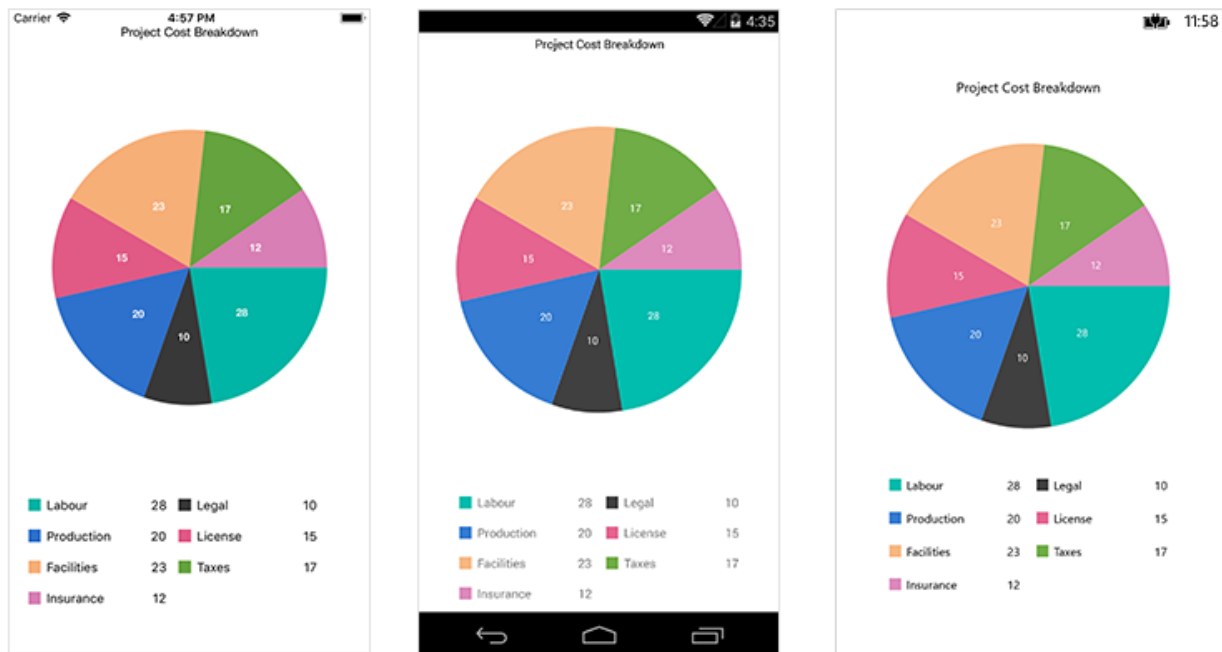
```
<chart:PieSeries ItemsSource="{Binding Data}" XBindingPath="Name"
YBindingPath="Value">
....
</chart:PieSeries>
<chart:SfChart.Legend>
....
<chart:ChartLegend.ItemTemplate>
<DataTemplate>
<StackLayout Orientation="Horizontal" WidthRequest="143">
<BoxView Color="{Binding IconColor}" HorizontalOptions="Center"
VerticalOptions="Center" HeightRequest="13" WidthRequest="13" />
<Label FontSize="13" VerticalTextAlignment="Center" Text="{Binding
DataPoint.Name}"/>
<Label HorizontalTextAlignment="End" VerticalTextAlignment="Center"
HorizontalOptions="EndAndExpand" FontSize="13" Text="{Binding
DataPoint.Value}"/>
</StackLayout>
</DataTemplate>
</chart:ChartLegend.ItemTemplate>
</chart:SfChart.Legend>
```

### C#

```
ChartLegend legend = new ChartLegend();
chart.Legend = legend;
legend.ItemTemplate = new DataTemplate ( () =>
{
    StackLayout stack = new StackLayout()
    {
        Orientation = StackOrientation.Horizontal, WidthRequest = 143
    };
    BoxView boxView = new BoxView()
    {
        HorizontalOptions = LayoutOptions.Center, VerticalOptions =
        LayoutOptions.Center, WidthRequest = 13, HeightRequest = 13
    };
    boxView.SetBinding(BoxView.BackgroundColorProperty, "IconColor");
    Label name = new Label()
    {
        VerticalTextAlignment = TextAlignment.Center, FontSize = 13
    };
    name.SetBinding(Label.TextProperty, "DataPoint.Name");
    Label value = new Label()
    {
        HorizontalTextAlignment = TextAlignment.End, VerticalTextAlignment =
        TextAlignment.Center, FontSize = 13
    };
    value.HorizontalOptions = LayoutOptions.EndAndExpand;
    value.SetBinding(Label.TextProperty, "DataPoint.Value");
    stack.Children.Add(boxView);
    stack.Children.Add(name);
}
```



```
stack.Children.Add(value);
return stack;
});
```



## Event

### LegendItemClicked

The [LegendItemClicked](#) event is triggered when the chart legend item is clicked. This argument contains the following information.

- [LegendItem](#) – Used to customize the label and appearance of individual legend item.

### LegendItemCreated

The [LegendItemCreated](#) event is triggered when the chart legend item is created. This argument contains the following information.

- [LegendItem](#) – Used to customize the label and appearance of individual legend item.

You can customize the legend item using following properties of [ChartLegendItem](#):

- [Label](#) – Get or sets the legend item label.
- [LabelStyle](#) – Customizes the appearance of legend labels. The properties listed in [customizing label](#) can be customized using the [LabelStyle](#) property.
- [IconColor](#) – Gets or sets the legend icon color.
- [Index](#) – Gets the legend item index.
- [IsEnabled](#) – Gets the visibility of the series if the series is the type of [CartesianSeries](#) and get the visibility of the data point if the series is type of [AccumulationSeries](#).
- [DataPoint](#) – Gets the legend item data point for accumulation series only.
- [Series](#) – Gets respective chart series.

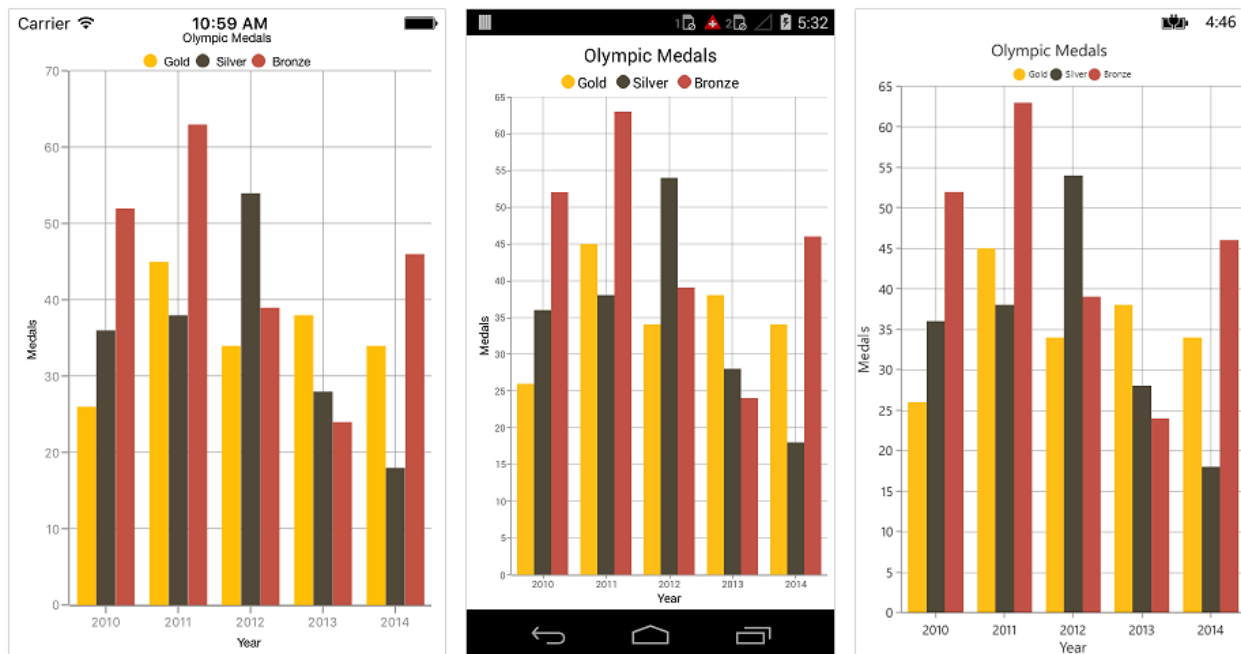
## Color Palette

### Apply palette for Chart

[ColorModel](#) property of [SfChart](#) is used to define the colors for each series. [ColorModel](#) contains the following color palettes.

### Predefined Palettes

Currently, Chart supports only [Metro](#) palette and it is the default palette for [SfChart](#). The following screenshot shows the default appearance of multiple series.



### Custom Palette

Chart will use the colors from [CustomBrushes](#) property if [ColorModel.Palette](#) is set to [Custom](#).

Following code illustrates how to set the custom colors.

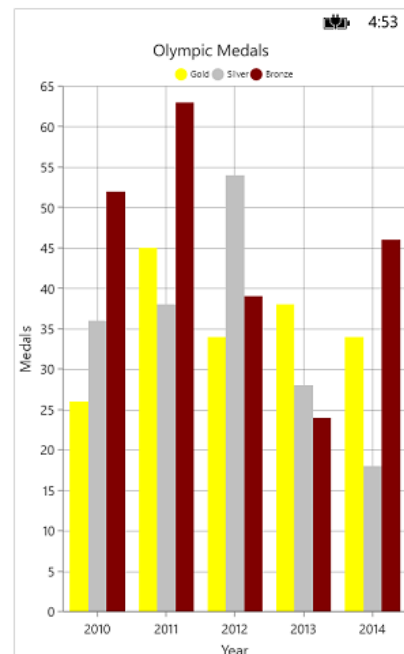
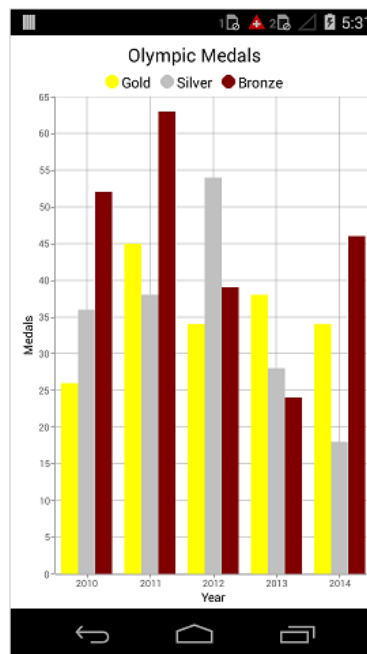
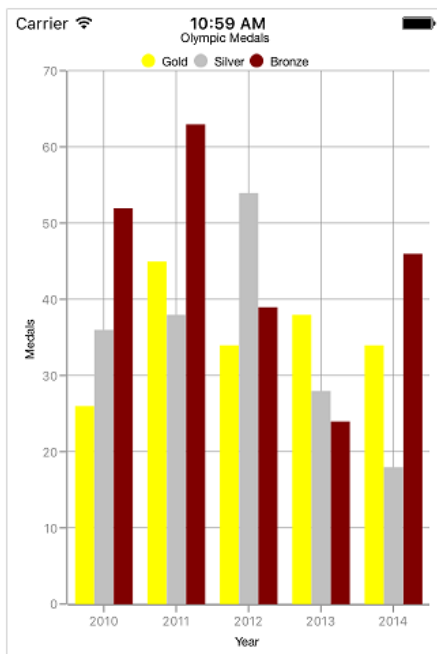
### XML

```
<ContentPage.Resources>
  <ResourceDictionary>
    <chart:ChartColorCollection x:Key="Colors">
      <Color>Red</Color>
      <Color>Gray</Color>
      <Color>Blue</Color>
      <Color>Maroon</Color>
      <Color>Pink</Color>
    </chart:ChartColorCollection>
  </ResourceDictionary>
</ContentPage.Resources>
<chart:SfChart>
  <chart:SfChart.ColorModel>
    <chart:ChartColorModel Palette="Custom" CustomBrushes="{StaticResource
Colors}"/>
  </chart:SfChart.ColorModel>
  ...
</chart:SfChart>
```

```
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart()
{
    ColorModel = new ChartColorModel()
    {
        Palette = ChartColorPalette.Custom,
        CustomBrushes = new ChartColorCollection()
        {
            Color.Red,
            Color.Gray,
            Color.Blue,
            Color.Maroon,
            Color.Pink,
        }
    },
    ...
};
```



## None Palette

[None](#) palette will not apply any color to the series. So in order to define the color for any series, you can use the [Color](#) property or the [ColorModel](#) property of [ChartSeries](#) (The ColorModel of Series will be explained later in this document).

## Apply palette for Series

[ColorModel](#) property of [ChartSeries](#) is used to define the colors for each data point. Following palettes are used to define the colors.

## Predefined Palettes

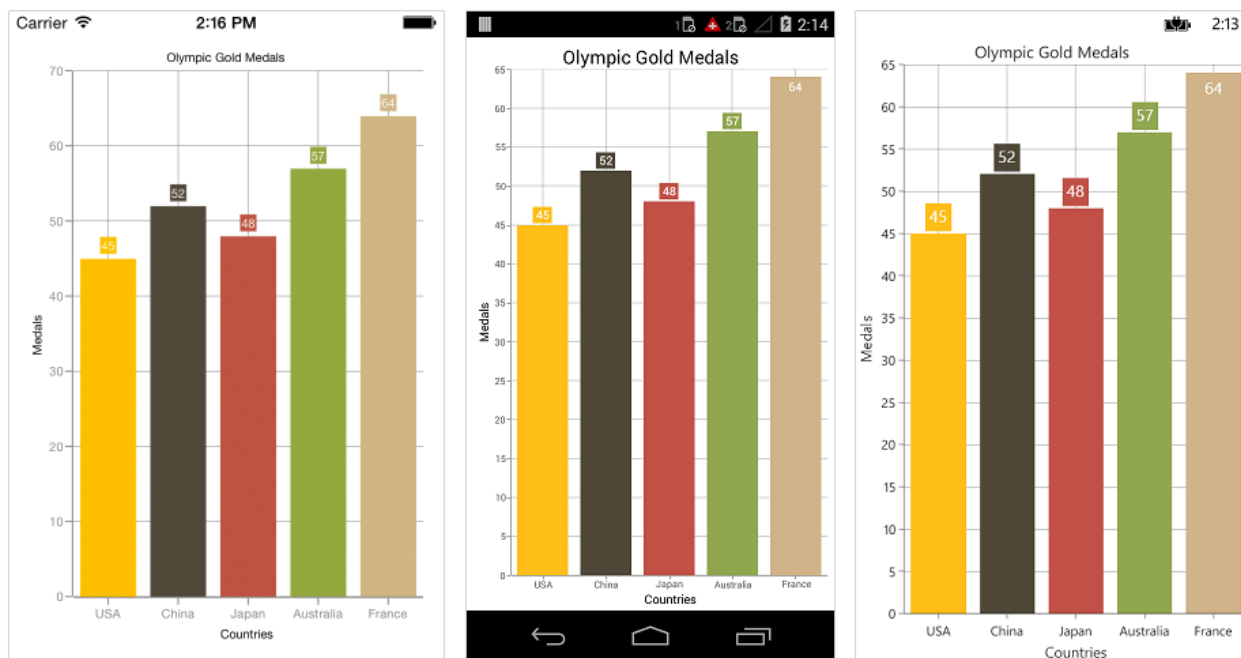
Currently, Chart supports only [Metro](#) palette.

### XML

```
<chart:SfChart>
...
<chart:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Country"
YBindingPath="Value">
<chart:ColumnSeries.ColorModel>
<chart:ChartColorModel Palette="Metro"/>
</chart:ColumnSeries.ColorModel>
</chart:ColumnSeries>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
ColumnSeries columnSeries = new ColumnSeries() {
    ItemsSource = Data,
    XBindingPath = "Country",
    YBindingPath = "Value"
};
columnSeries.ColorModel.Palette = ChartColorPalette.Metro;
chart.Series.Add(columnSeries);
```



### Custom Palette

Series will use the colors from [CustomBrushes](#) property if the [ColorModel.Palette](#) property of series is set to [Custom](#).

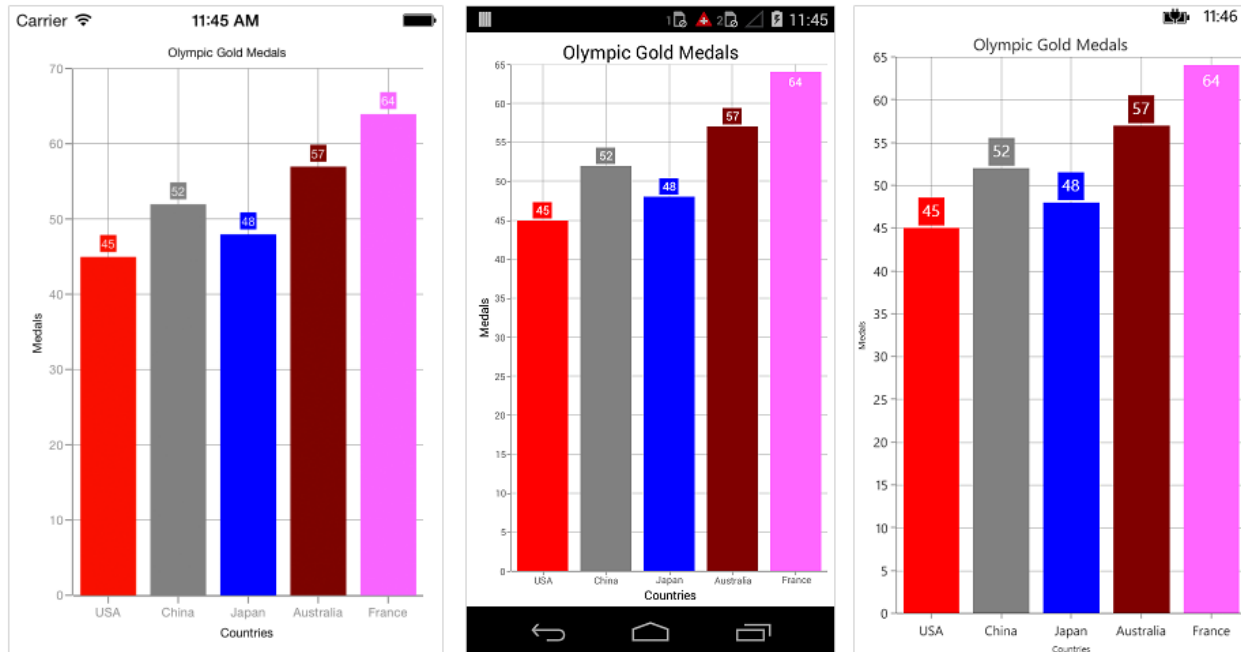
Following code illustrates how to set the custom colors.

### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<chart:ChartColorCollection x:Key="Colors">
<Color>Red</Color>
<Color>Gray</Color>
<Color>Blue</Color>
<Color>Maroon</Color>
<Color>Pink</Color>
</chart:ChartColorCollection>
</ResourceDictionary>
</ContentPage.Resources>
<chart:SfChart>
...
<chart:SfChart.Series>
<chart:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Name"
YBindingPath="Height">
<chart:ColumnSeries.ColorModel>
<chart:ChartColorModel Palette="Custom" CustomBrushes="{StaticResource
Colors}"/>
</chart:ColumnSeries.ColorModel>
</chart:ColumnSeries>
</chart:SfChart.Series>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart()
{
...
Series =
{
new ColumnSeries()
{
ItemsSource = viewModel.Data,
XBindingPath = "Name",
YBindingPath = "Height",
ColorModel = new ChartColorModel()
{
Palette = ChartColorPalette.Custom,
CustomBrushes = new ChartColorCollection()
{
Color.Red,
Color.Gray,
Color.Blue,
Color.Maroon,
Color.Pink,
}
}
}
}
};
```



### None Palette

[None](#) palette will not apply any color to the data points. So in order to define the color for the data points, you can use the [Color](#) property of [ChartSeries](#).

### Gradient Colors

The [CustomGradientColors](#) property of [ChartColorModel](#) is used to define the gradient colors, the colors from this property is used for series or chart if the [Palette](#) property of [ChartColorModel](#) is [Custom](#) and the [CustomBrushes](#) property is null. The following properties are used to define the gradient color for the chart.

The [StartPoint](#) and [EndPoint](#) properties of [ChartGradientColor](#) is used to configure the direction of gradient color, [GradientStops](#) property is used to set the color based on the offset.

The [Color](#) and [Offset](#) properties of [ChartGradientStop](#) is used to configure the color and offset position of each color.

Following code snippets and screenshot illustrates how to apply the gradient color to the chart series.

### XML

```
<chart:ColumnSeries.ColorModel>
<chart:ChartColorModel Palette="Custom">
<chart:ChartColorModel.CustomGradientColors>
<chart:ChartGradientColor StartPoint="0.5,1" EndPoint="0.5, 0">
<chart:ChartGradientColor.GradientStops>
<chart:ChartGradientStop Color="#FFE7C7" Offset= "0"/>
<chart:ChartGradientStop Color="#FCB69F" Offset= "1"/>
</chart:ChartGradientColor.GradientStops>
</chart:ChartGradientColor>
<chart:ChartGradientColor StartPoint="0.5,1" EndPoint="0.5, 0">
<chart:ChartGradientColor.GradientStops>
<chart:ChartGradientStop Color="#DCFA97" Offset= "0"/>
<chart:ChartGradientStop Color="#96E6A1" Offset= "1"/>
</chart:ChartGradientColor.GradientStops>
</chart:ChartGradientColor>
```

```

</chart:ChartGradientColor>
<chart:ChartGradientColor StartPoint="0.5,1" EndPoint="0.5, 0">
<chart:ChartGradientColor.GradientStops>
<chart:ChartGradientStop Color="#DDD6F3" Offset= "0"/>
<chart:ChartGradientStop Color="#FAACA8" Offset= "1"/>
</chart:ChartGradientColor.GradientStops>
</chart:ChartGradientColor>
<chart:ChartGradientColor StartPoint="0.5,1" EndPoint="0.5, 0">
<chart:ChartGradientColor.GradientStops>
<chart:ChartGradientStop Color="#A8EAE" Offset= "0"/>
<chart:ChartGradientStop Color="#7BB0F9" Offset= "1"/>
</chart:ChartGradientColor.GradientStops>
</chart:ChartGradientColor>
</chart:ChartColorModel.CustomGradientColors>
</chart:ChartColorModel>
</chart:ColumnSeries.ColorModel>

```

## C#

```

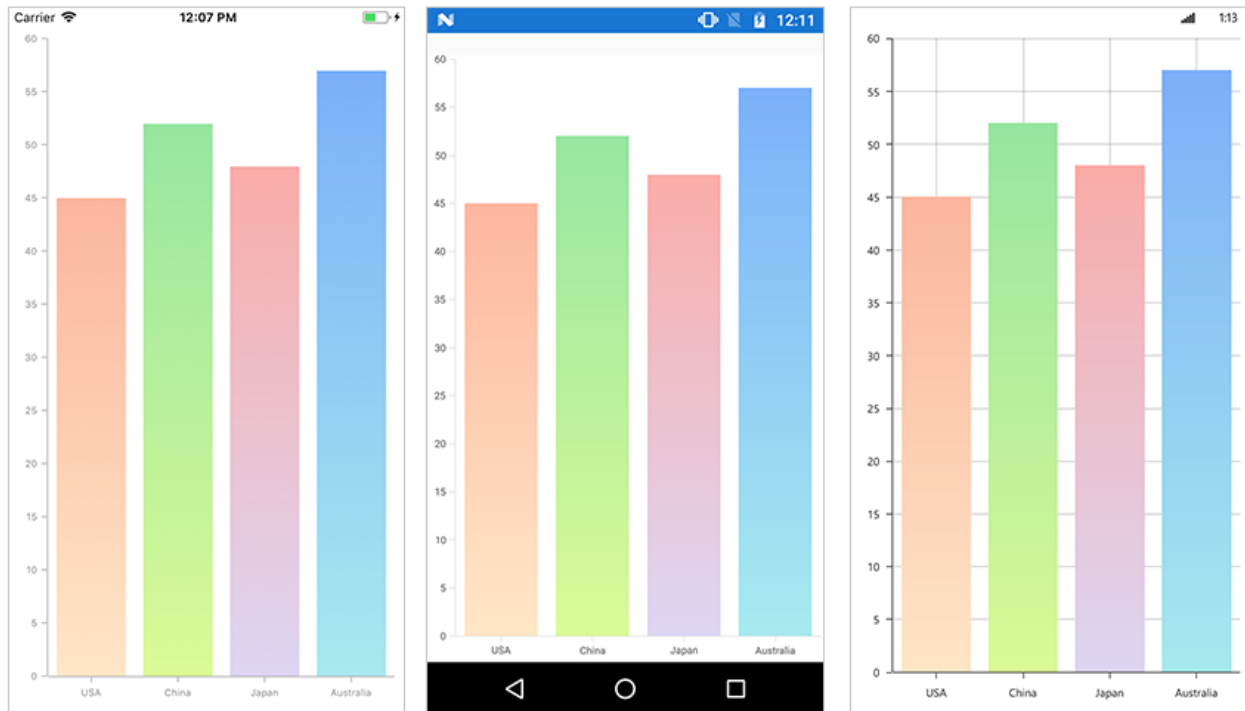
series.ColorModel.Palette = ChartColorPalette.Custom;
ChartGradientColor gradientColor1 = new ChartGradientColor() { StartPoint =
new Point(0.5, 1), EndPoint = new Point(0.5, 0) };
ChartGradientStop stop1 = new ChartGradientStop() { Color =
Color.FromHex("#FFE7C7"), Offset = 0 };
ChartGradientStop stop2 = new ChartGradientStop() { Color =
Color.FromHex("#FCB69F"), Offset = 1 };
gradientColor1.GradientStops.Add(stop1);
gradientColor1.GradientStops.Add(stop2);
ChartGradientColor gradientColor2 = new ChartGradientColor() { StartPoint =
new Point(0.5, 1), EndPoint = new Point(0.5, 0) };
ChartGradientStop stop21 = new ChartGradientStop() { Color =
Color.FromHex("#DCFA97"), Offset = 0 };
ChartGradientStop stop22 = new ChartGradientStop() { Color =
Color.FromHex("#96E6A1"), Offset = 1 };
gradientColor2.GradientStops.Add(stop21);
gradientColor2.GradientStops.Add(stop22);
ChartGradientColor gradientColor3 = new ChartGradientColor() { StartPoint =
new Point(0.5, 1), EndPoint = new Point(0.5, 0) };
ChartGradientStop stop31 = new ChartGradientStop() { Color =
Color.FromHex("#DDD6F3"), Offset = 0 };
ChartGradientStop stop32 = new ChartGradientStop() { Color =
Color.FromHex("#FAACA8"), Offset = 1 };
gradientColor3.GradientStops.Add(stop31);
gradientColor3.GradientStops.Add(stop32);
ChartGradientColor gradientColor4 = new ChartGradientColor() { StartPoint =
new Point(0.5, 1), EndPoint = new Point(0.5, 0) };
ChartGradientStop stop41 = new ChartGradientStop() { Color =
Color.FromHex("#A8EAE"), Offset = 0 };
ChartGradientStop stop42 = new ChartGradientStop() { Color =
Color.FromHex("#7BB0F9"), Offset = 1 };
gradientColor4.GradientStops.Add(stop41);
gradientColor4.GradientStops.Add(stop42);
ChartGradientColorCollection gradientColors = new
ChartGradientColorCollection()
{
gradientColor1,

```

```

gradientColor2,
gradientColor3,
gradientColor4
};
series.ColorModel.CustomGradientColors = gradientColors;

```



Following code snippet and screenshot illustrates how to apply the gradient color to the chart area.

### XML

```

<chart:SfChart.ColorModel>
  <chart:ChartColorModel Palette="Custom">
    <chart:ChartColorModel.CustomGradientColors>
      <chart:ChartGradientColor StartPoint="0.5,1" EndPoint="0.5,0">
        <chart:ChartGradientColor.GradientStops>
          <chart:ChartGradientStop Color="#FFE7C7" Offset="0"/>
          <chart:ChartGradientStop Color="#FCB69F" Offset="1"/>
        </chart:ChartGradientColor.GradientStops>
      </chart:ChartGradientColor>
    </chart:ChartColorModel.CustomGradientColors>
  </chart:ChartColorModel>
</chart:SfChart.ColorModel>

```

### C#

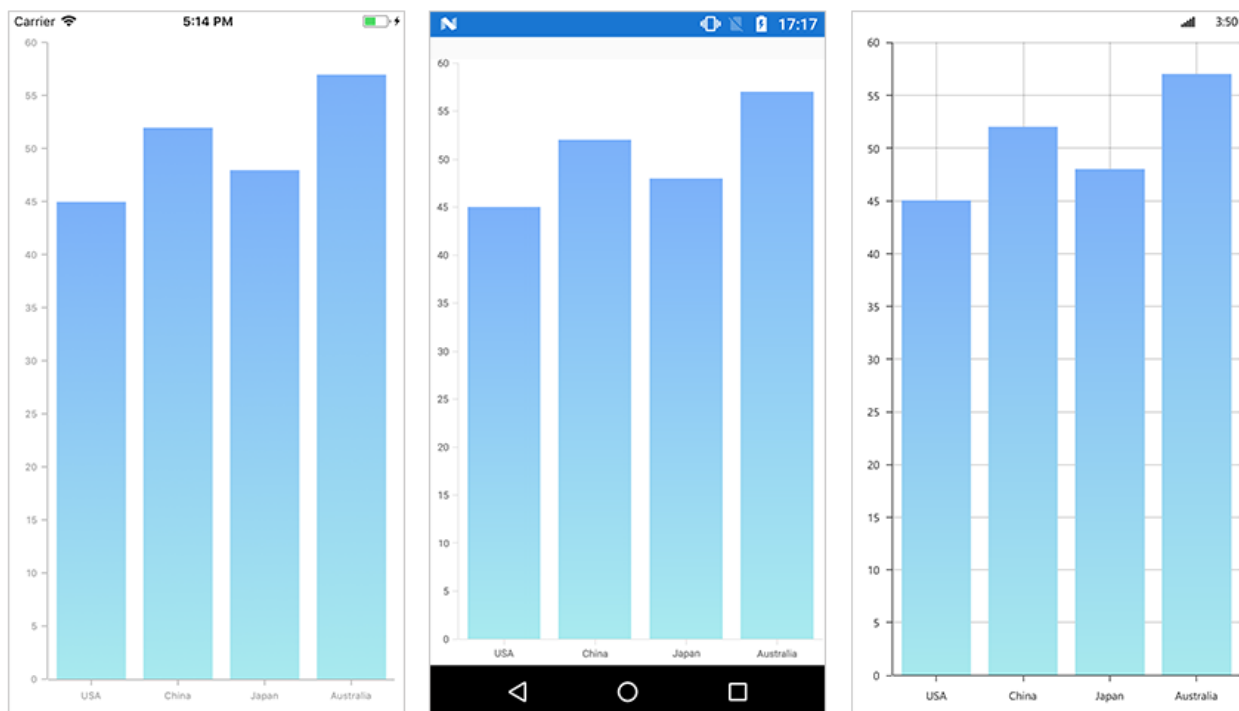
```

chart.ColorModel.Palette = ChartColorPalette.Custom;
ChartGradientColor gradientColor = new ChartGradientColor() { StartPoint =
new Point(0.5, 1), EndPoint = new Point(0.5, 0) };
ChartGradientStop stop1 = new ChartGradientStop() { Color =
Color.FromHex("#FFE7C7"), Offset = 0 };
ChartGradientStop stop2 = new ChartGradientStop() { Color =
Color.FromHex("#FCB69F"), Offset = 1 };

```



```
gradientColor.GradientStops.Add(stop1);
gradientColor.GradientStops.Add(stop2);
chart.ColorModel.CustomGradientColors.Add(gradientColor);
```



### Plotting Area Customization

[SfChart](#) provides the [AreaBackgroundColor](#), [AreaBorderColor](#), and [AreaBorderWidth](#) properties to customize the plot area.

The following code samples demonstrate the usage of these properties:

#### XML

```
<chart:SfChart
AreaBackgroundColor="Cyan"
AreaBorderColor="Gray"
AreaBorderWidth="3"/>
```

#### C#

```
SfChart chart = new SfChart();
chart.AreaBackgroundColor = Color.Cyan;
chart.AreaBorderColor = Color.Gray;
chart.AreaBorderWidth = 3;
```

### Zooming and Panning

#### Enable Zooming

Chart allows you to zoom in to view the data clearly. To enable this feature, you need to add an instance of [ChartZoomPanBehavior](#) to the [ChartBehaviors](#) collection property of [SfChart](#).

Following properties are used to configure the zooming feature,

- [EnableZooming](#) – used to enable/disable the pinch zooming. Default value is true.
- [EnableDirectionalZooming](#) - Enables or disables the pinch zooming based on pinch gesture direction. The default value of this property is false.
- [EnableDoubleTap](#) – when you enable this property, you can double tap on the chart to reset it to the original size or zoom in by one level.
- [EnableSelectionZooming](#) – when this property is set to true, you can double tap and drag to select a range on the chart to be zoomed in.
- [EnablePanning](#) – used to enable/disable the panning. Default value is true.
- [MaximumZoomLevel](#) - used to determine the maximum zoom level of the chart.
- [SelectionRectStrokeWidth](#) - used to change the stroke width of selection rectangle
- [SelectionRectStrokeColor](#) - used to change the stroke color of selection rectangle.
- [SelectionRectStrokeDashArray](#) - used to change the stroke dashes of selection rectangle.
- [SelectionRectFillColor](#) - used to change the stroke fill color of selection rectangle.

---

**Note:** EnableDirectionalZooming is not supported in the macOS platform.

---

Following code snippet illustrates how to enable zooming.

#### XML

```
<chart:SfChart>
<chart:SfChart.ChartBehaviors>
<chart:ChartZoomPanBehavior/>
</chart:SfChart.ChartBehaviors>
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart();
ChartZoomPanBehavior zoomPanBehavior = new ChartZoomPanBehavior();
chart.ChartBehaviors.Add(zoomPanBehavior);
```

Following code snippet illustrates how to enable the box selection zooming,

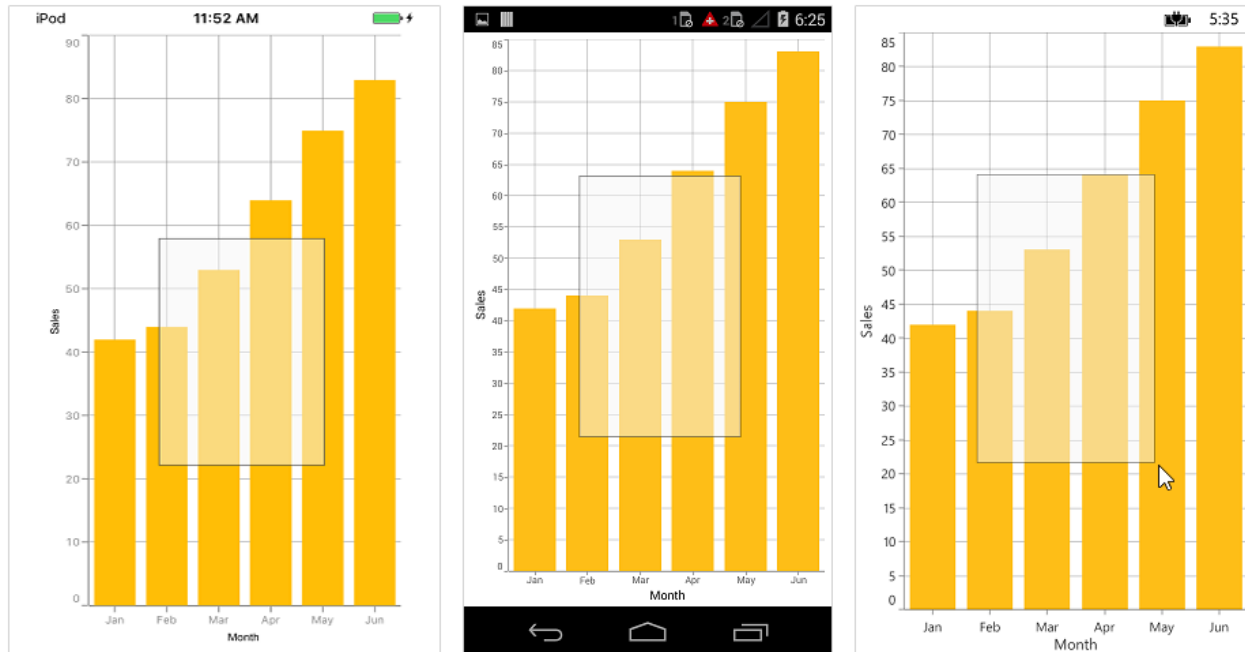
#### XML

```
<chart:SfChart>
<chart:SfChart.ChartBehaviors>
<chart:ChartZoomPanBehavior EnableSelectionZooming="True"/>
</chart:SfChart.ChartBehaviors>
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart();
ChartZoomPanBehavior zoomPanBehavior = new ChartZoomPanBehavior();
zoomPanBehavior.EnableSelectionZooming = true;
chart.ChartBehaviors.Add(zoomPanBehavior);
```

Following screenshot shows the box selection on chart area,



Following screenshot shows the zoomed area,



### Zoom Mode

The [ZoomMode](#) property specifies whether chart should be allowed to scale along horizontal axis or vertical axis or along both axis. The default value of [ZoomMode](#) is [XY](#) (both axis).

Following code example illustrates how to restrict the chart to be zoomed only along horizontal axis,

### XML

```
<chart:SfChart.ChartBehaviors>
  <chart:ChartZoomPanBehavior ZoomMode="X"/>
</chart:SfChart.ChartBehaviors>
```

**C#**

```
ChartZoomPanBehavior zoomPanBehavior = new ChartZoomPanBehavior();  
zoomPanBehavior.ZoomMode = ZoomMode.X;
```

**Auto Interval On Zooming**

[EnableAutoIntervalOnZooming](#) property determines the update of axis interval based on the current visible range while zooming the chart. Default value of this property is true. If this property is false, the nice interval will not be calculated for new range after zoom in and actual interval will be sustained.

**Events****ZoomStart**

The [ZoomStart](#) event is triggered when the user starts zooming the chart through pinch gesture, and this is a cancelable event. The argument contains the following information.

- [Axis](#) – the zoom start event will be triggered for all the axis in the Chart.
- [CurrentZoomFactor](#) – used to get the new zoom factor of the corresponding axis.
- [CurrentZoomPosition](#) – used to get the new zoom position of the corresponding axis.
- [Cancel](#) – used to set the value indicating whether the zooming should be canceled.

**ZoomDelta**

The [ZoomDelta](#) event is triggered while zooming, and this is a cancelable event. The argument contains the following information.

- [Axis](#) – Instance of the axis whose range is changed because of zooming. This event is triggered for each axis in the chart.
- [PreviousZoomFactor](#) – used to get the previous zoom factor of the axis.
- [PreviousZoomPosition](#) – used to get the previous zoom position of the axis.
- [CurrentZoomFactor](#) – used to get the current zoom factor of the axis.
- [CurrentZoomPosition](#) – used to get the current zoom position of the axis.
- [Cancel](#) – used to set the value indicating whether the zooming should be canceled.

### ZoomEnd

The [ZoomEnd](#) event is triggered after the zooming is stopped. The argument contains the following information.

- [Axis](#) – Instance of the axis whose range is changed because of zooming. This event is triggered for each axis in the chart.
- [CurrentZoomFactor](#) – the axis zoom factor after zoom.
- [CurrentZoomPosition](#) – the axis zoom position after zoom.

### SelectionZoomStart

The [SelectionZoomStart](#) event is triggered when the user starts the box selection zooming. The argument contains the following information.

- [ZoomRect](#) – used to get the initial bounds of the box selection.

### SelectionZoomDelta

The [SelectionZoomDelta](#) event is triggered while selecting a region to be zoomed, and this is a cancelable event. The argument contains the following information.

- [ZoomRect](#) – contains the bounds of the currently selected region.
- [Cancel](#) – used to set the value indicating whether the box selection zooming should be canceled.

### SelectionZoomEnd

The [SelectionZoomEnd](#) event is triggered after selection zooming ends. The argument contains the following information.

- [ZoomRect](#) – used to get the final bounds of the zoomed region.

### Scroll

The [Scroll](#) event is triggered while panning, and this is a cancelable event. The argument contains the following information.

- [Axis](#) – Instance of the axis whose range is changed while panning. This event is triggered for each axis in the chart.
- [ZoomPosition](#) – the current zoom position of the axis.
- [Cancel](#) – used to set a value indicating whether the scrolling should be canceled.

### ResetZoom

The [ResetZoom](#) event is triggered after the chart is reset on double tap. The argument contains the following information.

- [Axis](#) – Instance of the axis whose range is changed because of this event. This event is triggered for each axis in the chart.
- [PreviousZoomFactor](#) – used to get the previous zoom factor.
- [PreviousZoomPosition](#) – used to get the previous zoom position.

## Methods

Zooming and panning can be performed programmatically with the following methods:

### *ZoomIn*

[ZoomIn](#) method is used to increase the magnification of the plot area to view the data clearly.

#### **C#**

```
ChartZoomPanBehavior zoomPan = new ChartZoomPanBehavior();  
zoomPan.ZoomIn();
```

### *ZoomOut*

[ZoomOut](#) is used to decrease the magnification of the plot area to reset the default view.

#### **C#**

```
ChartZoomPanBehavior zoomPan = new ChartZoomPanBehavior();  
zoomPan.ZoomOut();
```

### *Zoom*

#### **Zoom(factor)**

[Zoom\(factor\)](#) method is used to change the zoom level by using zoom factor.

#### **C#**

```
ChartZoomPanBehavior zoomPan = new ChartZoomPanBehavior();  
zoomPan.Zoom(0.5f);
```

#### **Zoom(Rect)**

[Zoom\(Rect\)](#) method is used to zoom the chart for a given rectangle value. Create an instance of Rect by passing [Top](#), [Left](#), [Bottom](#), and [Right](#) positions and pass it to zoom method.

#### **C#**

```
ChartZoomPanBehavior zoomPan = new ChartZoomPanBehavior();  
zoomPan.Zoom(new Rect(10, 10, 200, 350));
```

#### **Zoom(chartAxis, cumulativeLevel, origin)**

[Zoom\(chartAxis, cumulativeLevel, origin\)](#) method is used to change the zoom level of given axis to the specified level and origin.

#### **C#**

```
ChartZoomPanBehavior zoomPan = new ChartZoomPanBehavior();  
zoomPan.Zoom(axis, 0.5f, 0.5f);
```

### *ZoomByRange*

#### **ZoomByRange(chartAxis, start, end)**

[ZoomByRange\(chartAxis, start, end\)](#) method is used to zoom the given axis to the given range.

#### **C#**

```
ChartZoomPanBehavior zoomPan = new ChartZoomPanBehavior();
zoomPan.ZoomByRange(axis, 20, 25);
```

### ZoomByRange(dateTimeAxis, start, end)

[ZoomByRange\(dateTimeAxis, start, end\)](#) method is used to zoom the given axis to the given date time range.

#### C#

```
ChartZoomPanBehavior zoomPan = new ChartZoomPanBehavior();
zoomPan.ZoomByRange(axis, new DateTime(2017, 3, 1), new DateTime(2017, 5, 1));
```

### [ZoomToFactor\(chartAxis, zoomPosition, zoomFactor\)](#)

[ZoomToFactor](#) method is used to change the zoom level by using zoom position and zoom factor. Zoom position value specifies the starting point of zooming, and zoom factor value specifies the level of zooming.

#### C#

```
ChartZoomPanBehavior zoomPan = new ChartZoomPanBehavior();
zoomPan.ZoomToFactor(axis, 0.5f, 0.5f);
```

### [Reset](#)

[Reset](#) method is used to return the plot area back to its original position after zooming.

#### C#

```
ChartZoomPanBehavior zoomPan = new ChartZoomPanBehavior();
zoomPan.Reset();
```

### [Override Methods](#)

The following override methods are available in ChartZoomPanBehavior to customize zooming actions.

- [OnScaleStart](#) - It gets called when the user start to zoom on the chart.
- [OnScaleDelta](#) - It gets called while performing the zoom action.
- [OnScaleEnd](#) - It gets called after end the zoom action.
- [OnScroll](#) - It gets called while scrolling the chart.

#### C#

```
public class ChartZoomPanBehaviorExt : ChartZoomPanBehavior
{
    protected override void OnScaleStart(float manipulationX, float
manipulationY, float scaleFactor)
    {
        base.OnScaleStart(manipulationX, manipulationY, scaleFactor);
    }
    protected override void OnScaleDelta(float manipulationX, float
manipulationY, float scaleFactor)
    {
        base.OnScaleDelta(manipulationX, manipulationY, scaleFactor);
    }
}
```

```
protected override void OnScaleEnd(float manipulationX, float manipulationY,
float scaleFactor)
{
    base.OnScaleEnd(manipulationX, manipulationY, scaleFactor);
}
protected override void OnScroll(float pointX, float pointY, float
distanceX, float distanceY)
{
    base.OnScroll(pointX, pointY, distanceX, distanceY);
}
}
```

## Selection

[SfChart](#) supports selection that enables you to select a segment in a series or series itself.

### Data Point Selection

You can select a data point by tapping on it. To enable the selection feature, set [EnableDataPointSelection](#) property as `true` for [Series](#).

### XML

```
<chart:ColumnSeries EnableDataPointSelection="True" ItemsSource="{Binding
Data}" XBindingPath="Month" YBindingPath="Value"/>
```

### C#

```
ColumnSeries columnSeries = new ColumnSeries()
{
    ItemsSource = Data,
    XBindingPath = "Month",
    YBindingPath = "Value"
};
columnSeries.EnableDataPointSelection = true;
```





Following properties are used to configure the selection feature,

- [SelectedDataPointIndex](#) – used to programmatically select a data point.
- [SelectedDataPointColor](#) – used to change the selected data point color.

### XML

```
<chart:ColumnSeries EnableDataPointSelection="True"
SelectedDataPointIndex="2" SelectedDataPointColor="Red" ItemsSource
="{Binding Data}" />
```

### C#

```
ColumnSeries columnSeries = new ColumnSeries();
columnSeries.SelectedDataPointIndex = 2;
columnSeries.SelectedDataPointColor = Color.Red;
```



**Note:** For Accumulation series like pie, doughnut, pyramid and funnel, when you select a data point, the corresponding legend item also will be selected.

### Series Selection

Series selection is used in case of multiple series when you want to highlight a particular series. Series Selection can be enabled by setting [EnableSeriesSelection](#) property to True. The [SeriesSelectionColor](#) property is used to set the color to highlight the series.

### XML

```
<chart:SfChart x:Name="Chart" EnableSeriesSelection="True">
    ...
</chart:SfChart>
<chart:SfChart.Series>
<chart:ColumnSeries x:Name="series1" ItemsSource="{Binding Data}"
XBindingPath="XValue" YBindingPath="YValue" Color="#b2ebe7">
</chart:ColumnSeries>
<chart:ColumnSeries x:Name="series2" ItemsSource="{Binding Data1}"
XBindingPath="XValue" YBindingPath="YValue" Color="#d3bee5">
</chart:ColumnSeries>
<chart:ColumnSeries x:Name="series2" ItemsSource="{Binding Data2}"
XBindingPath="XValue" YBindingPath="YValue" Color="#c0d8f0">
</chart:ColumnSeries>
</chart:SfChart.Series>
```

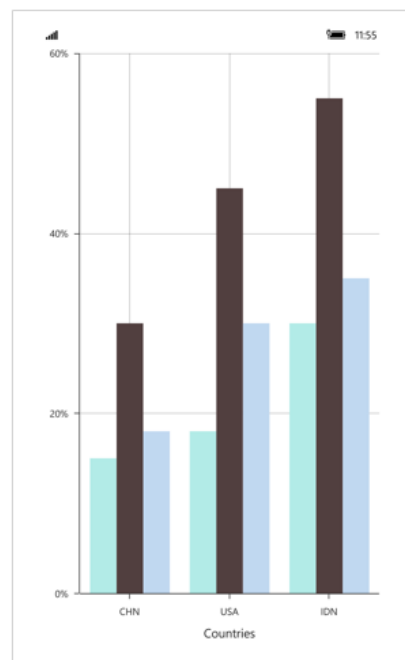
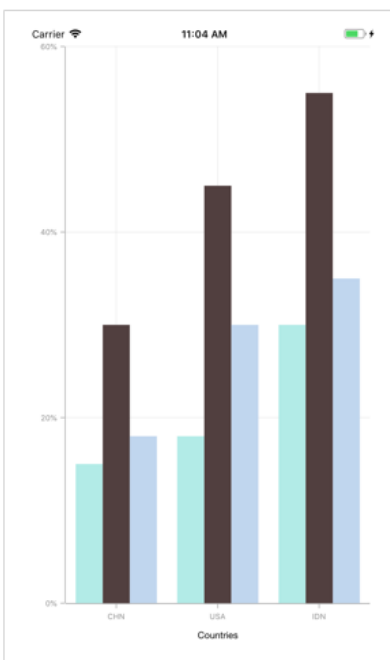
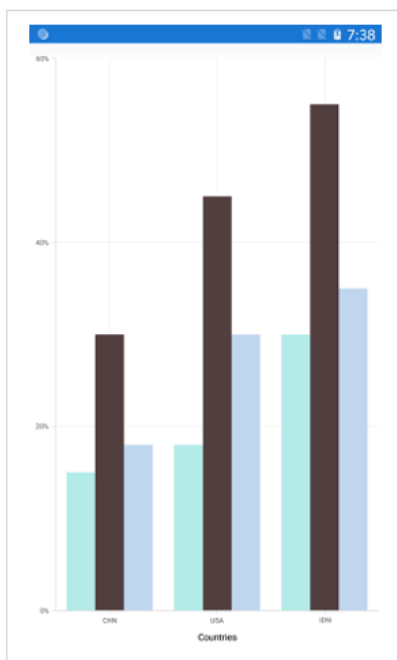
### C#

```
chart.EnableSeriesSelection = true;
ColumnSeries columnSeries = new ColumnSeries()
{
    ItemsSource = Data,
    XBindingPath = "XValue",
    YBindingPath = "YValue",
    Color = Color.FromRgb(178, 235, 231);
};
```

```

ColumnSeries columnSeries1 = new ColumnSeries()
{
    ItemsSource = Data1,
    XBindingPath = "XValue",
    YBindingPath = "YValue",
    Color = Color.FromRgb(211, 190, 229);
};
ColumnSeries columnSeries2 = new ColumnSeries()
{
    ItemsSource = Data2,
    XBindingPath = "XValue",
    YBindingPath = "YValue",
    Color = Color.FromRgb(192, 216, 240);
};

```



To set the series selection color,

### XML

```

<chart:SfChart x:Name="Chart" EnableSeriesSelection="True"
SeriesSelectionColor="Red">
...
</chart:SfChart>

```

### C#

```
chart.SeriesSelectionColor = Color.Red;
```

### Events

#### SelectionChanging

The [SelectionChanging](#) event is triggered before the data point is selected. You can restrict a data point from being selected, by canceling this event, by setting [Cancel](#) property in the event argument to true. The argument contains the following information,

- [SelectedSeries](#) – used to get the series of selected data point.
- [SelectedDataPointIndex](#) – used to get the selected data point index.
- [PreviousSelectedIndex](#) – used to get the previous selected data point index.
- [PreviousSelectedSeries](#) - used to get the previous selected series.
- [Cancel](#) – used to set the value indicating whether the selection should be canceled.

### SelectionChanged

The [SelectionChanged](#) event triggered after a data point is selected. The argument contains the following information,

- [SelectedSeries](#) - Gets the series of selected data point.
- [SelectedDataPointIndex](#) - Gets the selected data point index.
- [PreviousSelectedIndex](#) - Gets the previous selected data point index.
- [PreviousSelectedSeries](#) - Gets the previous selected series.

### Methods

#### OnSelectionChanging

The [OnSelectionChanging](#) method of chart selection behavior is used to perform the operations, before the data point is selected, by extending the [ChartSelectionBehavior](#) class. This method argument contains the following information:

- [SelectedSeries](#) - Gets the series of selected data point.
- [SelectedDataPointIndex](#) - Gets the selected data point index.
- [PreviousSelectedIndex](#) - Gets the previous selected data point index.
- [PreviousSelectedSeries](#) - Gets the previous selected series.
- [Cancel](#) -Sets the value that indicates whether the selection should be canceled.

### C#

```
public class ChartSelectionBehaviorExt : ChartSelectionBehavior
{
    protected override void OnSelectionChanging(ChartSelectionChangingEventArgs
args)
    {
        var selectedSeries = args.SelectedSeries;
        var dataPointIndex = args.SelectedDataPointIndex;
        var previousSelectedIndex = args.PreviousSelectedIndex;
        var previousSelectedSeries = args.PreviousSelectedSeries;
    }
}
```

#### OnSelectionChanged

The [OnSelectionChanged](#) method of the [ChartSelectionBehavior](#) is used to perform the operations after a data point is selected. This method argument contains the following information:

- [SelectedSeries](#) - Gets the series of selected data point.
- [SelectedDataPointIndex](#) - Gets the selected data point index.
- [PreviousSelectedIndex](#) - Gets the previous selected data point index.
- [PreviousSelectedSeries](#) - Gets the previous selected series.

**C#**

```
public class ChartSelectionBehaviorExt : ChartSelectionBehavior
{
    protected override void OnSelectionChanged(ChartSelectionEventArgs args)
    {
        var selectedSeries = args.SelectedSeries;
        var dataPointIndex = args.SelectedDataPointIndex;
        var previousSelectedIndex = args.PreviousSelectedIndex;
        var previousSelectedSeries = args.PreviousSelectedSeries;
    }
}
```

**Trackball**

Trackball feature displays the tooltip for the data points that are closer to the point where you touch on the chart area. This feature, especially, can be used instead of data label feature when you cannot show data labels for all data points due to space constraint. To enable this feature, add an instance of [ChartTrackballBehavior](#) to the [ChartBehaviors](#) collection property of [SfChart](#). Trackball will be activated once you long-press anywhere on the chart area. Once it is activated, it will appear in the UI and move based on your touch movement until you stop touching on the chart.

You can use the following properties to show/hide the line and labels.

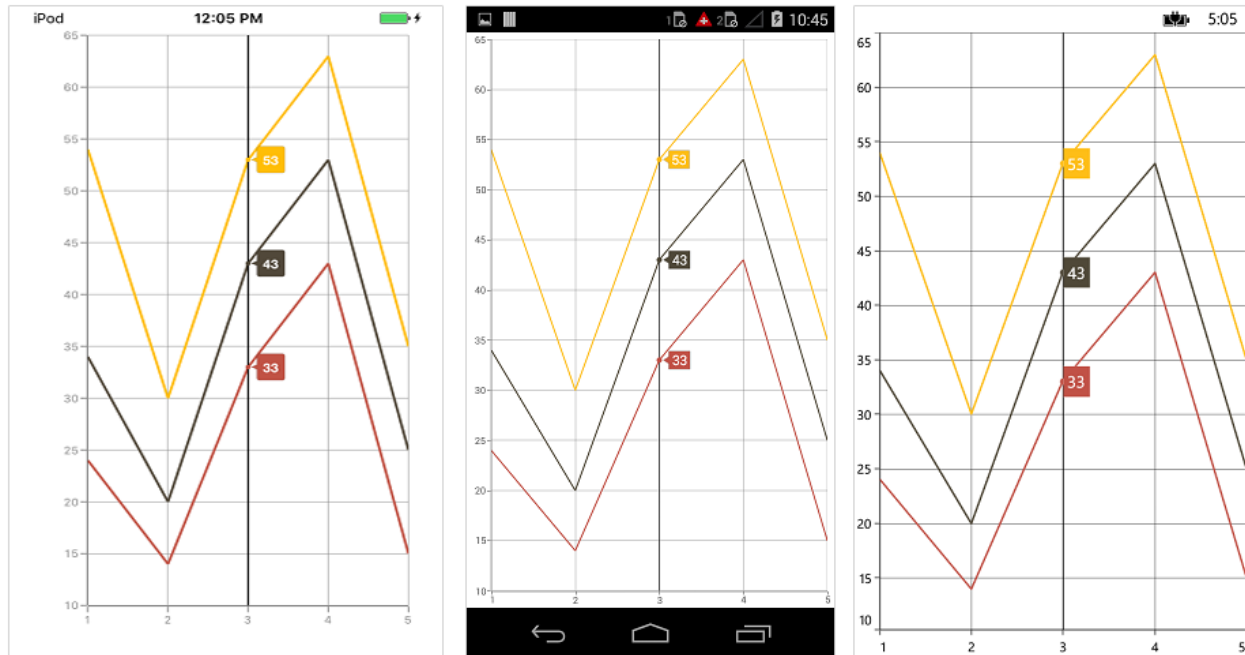
- [ShowLabel](#) – Shows/hides trackball label. Default value is true.
- [ShowLine](#) – Shows/hides the trackball line. Default value is true.

**XML**

```
<chart:SfChart>
...
<chart:SfChart.ChartBehaviors>
<chart:ChartTrackballBehavior ShowLabel="True" ShowLine="True"/>
</chart:SfChart.ChartBehaviors>
</chart:SfChart>
```

**C#**

```
SfChart chart = new SfChart();
...
ChartTrackballBehavior trackballBehavior = new ChartTrackballBehavior();
trackballBehavior.ShowLabel = true;
trackballBehavior.ShowLine = true;
chart.ChartBehaviors.Add(trackballBehavior);
```



### Label Display Mode

[TrackballLabelDisplayMode](#) property is used to specify whether to display label for all the data points along the vertical line or display only single label. Following are the two options you can set to this property,

- [FloatAllPoints](#) – Displays label for all the data points along the vertical line.
- [NearestPoint](#) – Displays label for single data point that is nearer to the touch contact position.
- [GroupAllPoints](#) – Displays label for all the data points grouped and positioned at the top of the chart area.

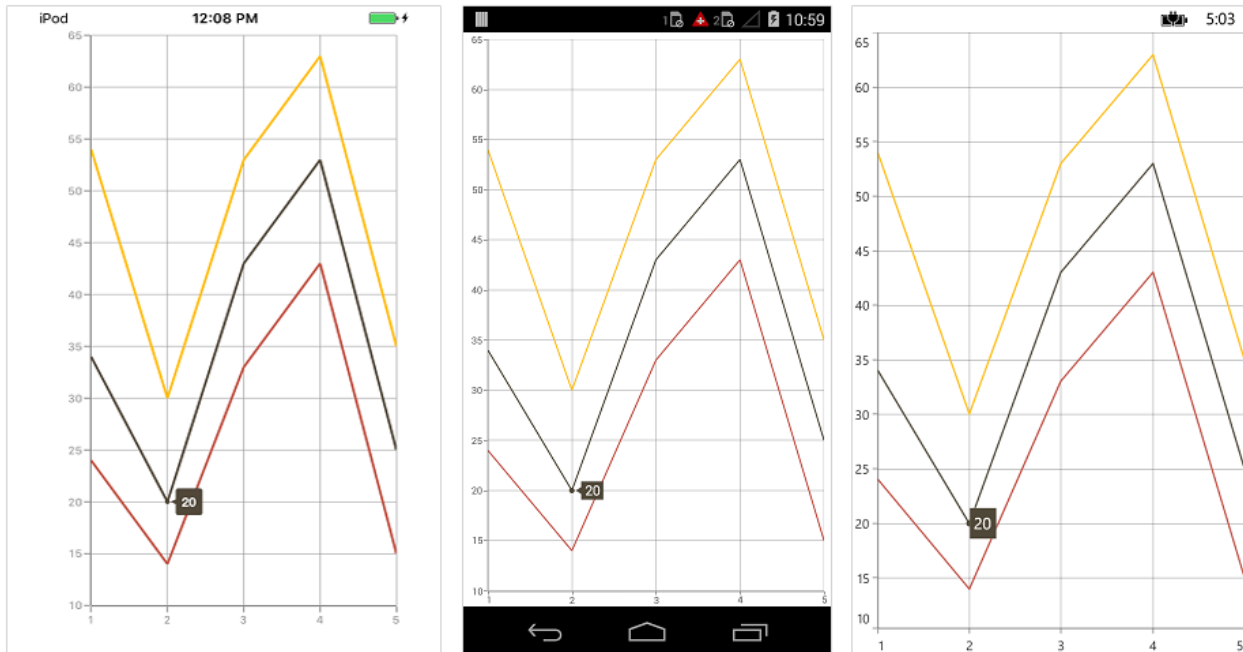
### XML

```
<chart:ChartTrackballBehavior LabelDisplayMode="NearestPoint"
ShowLine="False"/>
```

### C#

```
ChartTrackballBehavior trackballBehavior = new ChartTrackballBehavior();
trackballBehavior.ShowLine = false;
trackballBehavior.LabelDisplayMode = TrackballLabelDisplayMode.NearestPoint;
```

In the following screenshot, trackball label is shown for only single data point,



### Activation mode

The [ActivationMode](#) property is used to restrict the visibility of trackball based on the touch actions. The default value of this property is [ChartTrackballActivationMode.LongPress](#).

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

- [LongPress](#) – Activates trackball only when performing the long press action.
- [TouchMove](#) – Activates trackball only when performing touch move action.
- [None](#) – Hides the visibility of trackball when setting activation mode to [None](#). It will be activated when calling the [Show](#) method.

---

**Note:** The default value of [ActivationMode](#) property is [ChartTrackballActivationMode.LongPress](#) for Android and iOS platform and default value for MacOS and UWP platform is [ChartTrackballActivationMode.TouchMove](#).

---

### Customizing appearance

#### Customize Trackball Labels

The [LabelStyle](#) property provides options to customize the trackball labels.

- [BorderColor](#) – used to change the label border color.
- [BackgroundColor](#) – used to change the label background color.
- [BorderThickness](#) – used to change label border thickness.
- [TextColor](#) – used to change the text color.
- [Font](#) – used to change label font size, family and weight.

### XML

```
<chart:SfChart>
...
</chart:SfChart>
<chart:SfChart.ChartBehaviors>
```

```

<chart:ChartTrackballBehavior>
<chart:ChartTrackballBehavior.LabelStyle>
<chart:ChartTrackballLabelStyle BorderColor="Maroon" BackgroundColor="Aqua"
BorderThickness="2" TextColor="Red" Font="Italic,18"/>
</chart:ChartTrackballBehavior.LabelStyle>
</chart:ChartTrackballBehavior>
</chart:SfChart.ChartBehaviors>
</chart:SfChart>

```

## C#

```

SfChart chart = new SfChart();
...
ChartTrackballBehavior trackBallBehavior = new ChartTrackballBehavior();
trackBallBehavior.LabelStyle.BorderColor = Color.Maroon;
trackBallBehavior.LabelStyle.BorderThickness = 2;
trackBallBehavior.LabelStyle.Font = Font.SystemFontOfSize(18,
FontAttributes.Italic);
trackBallBehavior.LabelStyle.BackgroundColor = Color.Aqua;
trackBallBehavior.LabelStyle.TextColor = Color.Red;
chart.ChartBehaviors.Add(trackballBehavior);

```

## Customize Trackball Marker

The [MarkerStyle](#) property provides options to customize the trackball markers.

Following properties are used to customize the trackball marker.

- [ShowMarker](#) – used to enable / disable the marker. Default value is true.
- [BorderColor](#) – used to change the marker border color.
- [Color](#) – used to change the marker background color.
- [BorderWidth](#) – used to change the width of the marker border.
- [Width](#) – used to change the width of the marker.
- [Height](#) – used to change the height of the marker.

## XML

```

<chart:SfChart HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
...
<chart:SfChart.ChartBehaviors>
<chart:ChartTrackballBehavior>
<chart:ChartTrackballBehavior.MarkerStyle>
<chart:ChartTrackballMarkerStyle BorderColor="Purple" ShowMarker="True"
BorderWidth="1" Width="8" Height="8" Color="Green"/>
</chart:ChartTrackballBehavior.MarkerStyle>
</chart:ChartTrackballBehavior>
</chart:SfChart.ChartBehaviors>
</chart:SfChart>

```

## C#

```

SfChart chart = new SfChart();
...

```



```

ChartTrackballBehavior trackBallBehavior = new ChartTrackballBehavior();
trackBallBehavior.MarkerStyle.BorderWidth = 1;
trackBallBehavior.MarkerStyle.BorderColor = Color.Purple;
trackBallBehavior.MarkerStyle.Width = 8;
trackBallBehavior.MarkerStyle.Height = 8;
trackBallBehavior.MarkerStyle.Color = Color.Green;
trackBallBehavior.MarkerStyle.ShowMarker = true;
chart.ChartBehaviors.Add(trackballBehavior);

```

### Customize Trackball Line

The [LineStyle](#) property provides options to customize the trackball line.

- [ShowLine](#) – used to enable / disable the line. Default value is true.
- [StrokeWidth](#) – used to change the stroke width of the line.
- [StrokeColor](#) – used to change the stroke color of the line.
- [StrokeDashArray](#) – Specifies the dashes to be applied on the line.

### XML

```

<chart:SfChart HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
...
<chart:SfChart.ChartBehaviors>
<chart:ChartTrackballBehavior>
<chart:ChartTrackballBehavior.LineStyle>
<chart:ChartLineStyle StrokeColor="Blue" StrokeWidth="2"/>
</chart:ChartTrackballBehavior.LineStyle>
</chart:ChartTrackballBehavior>
</chart:SfChart.ChartBehaviors>
</chart:SfChart>

```

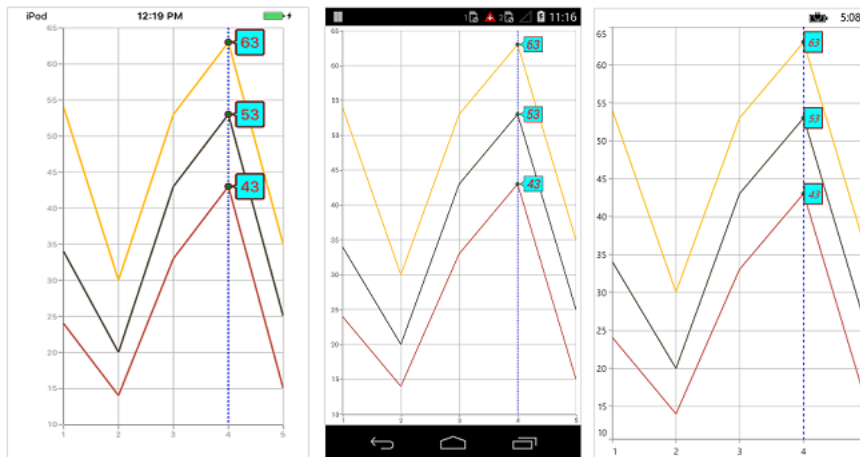
### C#

```

SfChart chart = new SfChart();
...
ChartTrackballBehavior trackBallBehavior = new ChartTrackballBehavior();
trackBallBehavior.ShowLine = true;
trackBallBehavior.LineStyle.StrokeWidth = 2;
trackBallBehavior.LineStyle.StrokeColor = Color.Blue;
trackBallBehavior.LineStyle.StrokeDashArray = new double[2] { 2, 3 };
chart.ChartBehaviors.Add(trackballBehavior);

```

Following screenshot illustrates the customization of trackball elements.



Show/hide the trackball label in axis

This feature is used to highlight the respective axis label when the trackball is moving across the axis.

[ChartAxis.ShowTrackballInfo](#) property is used show/hide the trackball label of the axis.

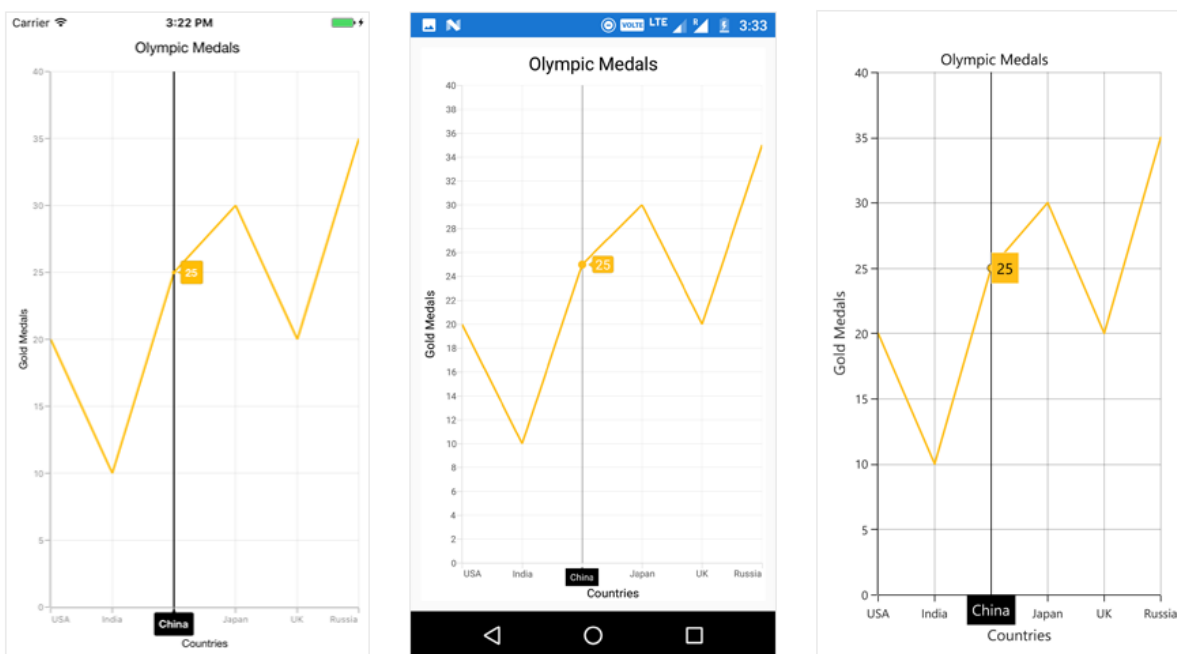
[ChartAxis.TrackballLabelStyle](#) property is used to customize its appearance. Default value of [ChartAxis.ShowTrackballInfo](#) is `False`.

### XML

```
<chart:SfChart.PrimaryAxis >
<chart:CategoryAxis ShowTrackballInfo = "true" />
</chart:SfChart.PrimaryAxis >
```

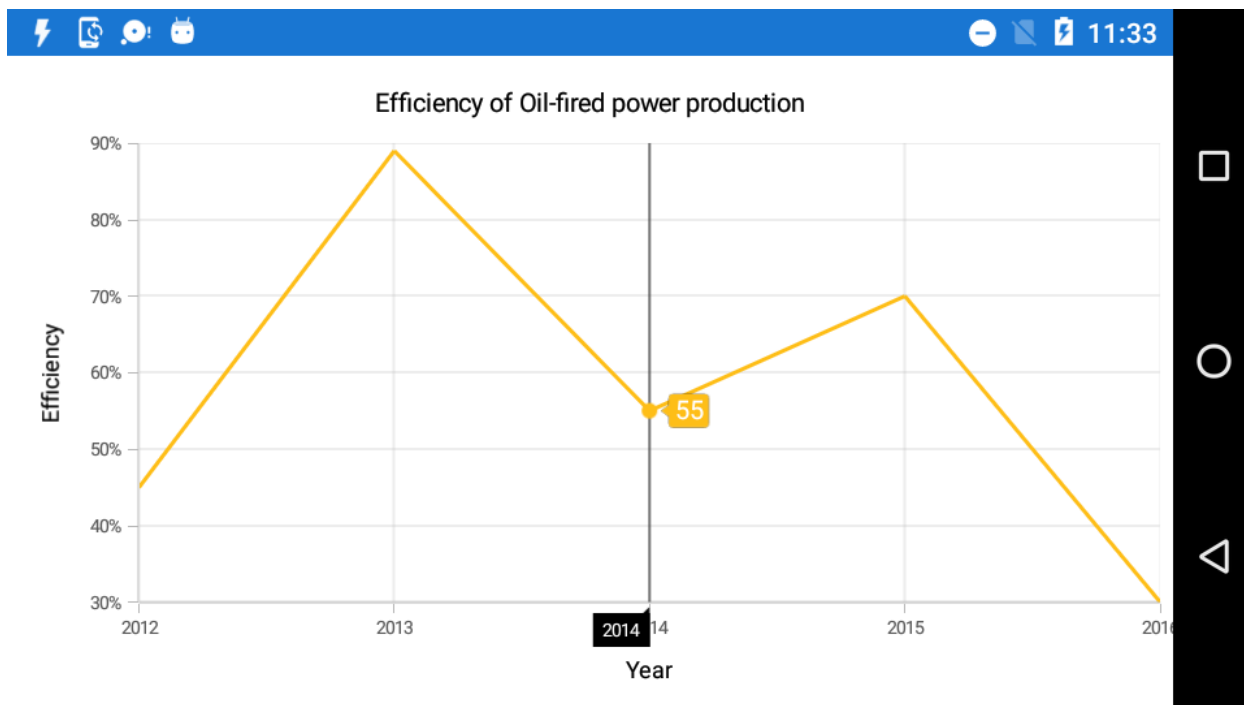
### C#

```
chart.PrimaryAxis.ShowTrackballInfo = true;
```



### Axis label alignment

The position of trackball's axis label can be changed using the [AxisLabelAlignment](#) property of [ChartTrackballAxisLabelStyle](#). The following options are available in [AxisLabelAlignment](#).



\* [Far](#) - The label will be positioned below the tick in vertical axis and right of the tick in horizontal axis. \*

[Near](#) - The label will be positioned above the tick in vertical axis and left of the tick in horizontal axis. \*

[Center](#) - The label will be positioned at the center of tick. This is the default value.

The following code snippet and screenshot demonstrate the placement of label at the left to tick line.

#### XML

```
<chart:SfChart>
...
<chart:CategoryAxis.TrackballLabelStyle>
<chart:ChartTrackballAxisLabelStyle AxisLabelAlignment="Near"/>
</chart:CategoryAxis.TrackballLabelStyle>
...
</chart:SfChart>
```

#### C#

```
primaryAxis.TrackballLabelStyle.AxisLabelAlignment =
ChartLabelAlignment.Near;
```

![Label alignment support for trackball axis label in Xamarin.Forms

Chart](trackball\_images/AxisLabelAlignment-Near.png)

### Show/hide the series label

This feature is used to show/hide the trackball label of the series by using

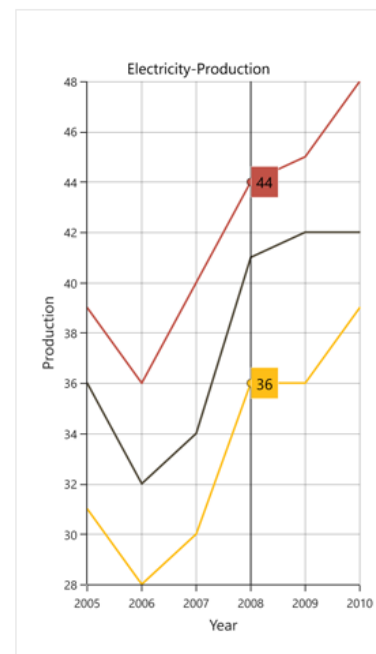
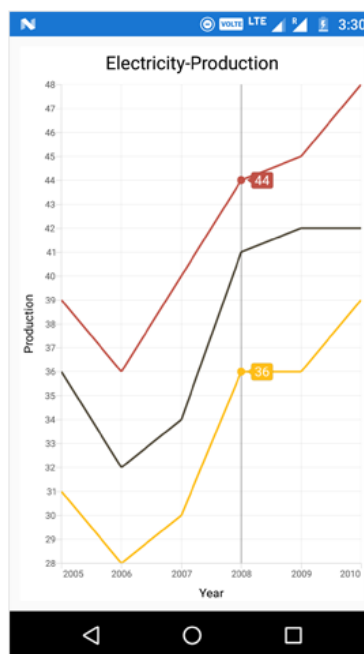
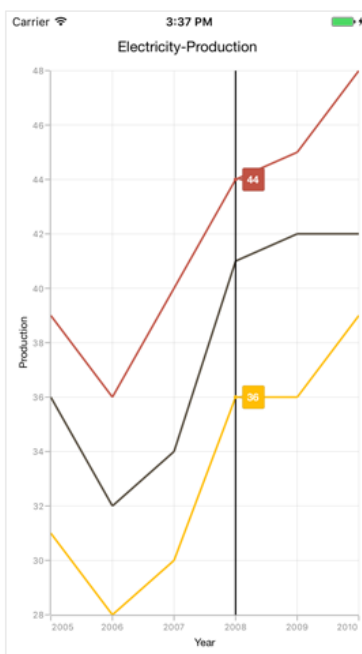
[CartesianSeries.ShowTrackballInfo](#) property. Default value of [CartesianSeries.ShowTrackballInfo](#) property is `True`.

## XML

```
<chart:SfChart>
...
<chart:LineSeries ItemsSource="{Binding Data}" XBindingPath="Year"
YBindingPath="Value" ShowTrackballInfo="false" />
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
LineSeries lineSeries = new LineSeries()
{
    ItemsSource = Data,
    XBindingPath = "Year",
    YBindingPath = "Value",
    ShowTrackballInfo = false
};
chart.Series.Add(lineSeries);
```



## Label Template

You can customize the appearance of the Trackball label with your own template by using [TrackballLabelTemplate](#) property of [ChartSeries](#).

## XML

```
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="trackballTemplate">
<StackLayout Orientation="Horizontal">
<Label Text="{Binding Value}" TextColor="White" FontSize="15"
VerticalTextAlignment="Center"/>
```

```

<Image Source ="grain.jpg" WidthRequest="30" HeightRequest="30"/>
</StackLayout>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<chart:SfChart.Series>
<chart:LineSeries TrackballLabelTemplate="{StaticResource
trackballTemplate}" ItemsSource="{Binding Data1}" XBindingPath="Name"
YBindingPath="Value"/>
<chart:LineSeries TrackballLabelTemplate="{StaticResource
trackballTemplate}" ItemsSource="{Binding Data2}" XBindingPath="Name"
YBindingPath="Value"/>
</chart:SfChart.Series>
<chart:SfChart.ChartBehaviors>
<chart:ChartTrackballBehavior/>
</chart:SfChart.ChartBehaviors>

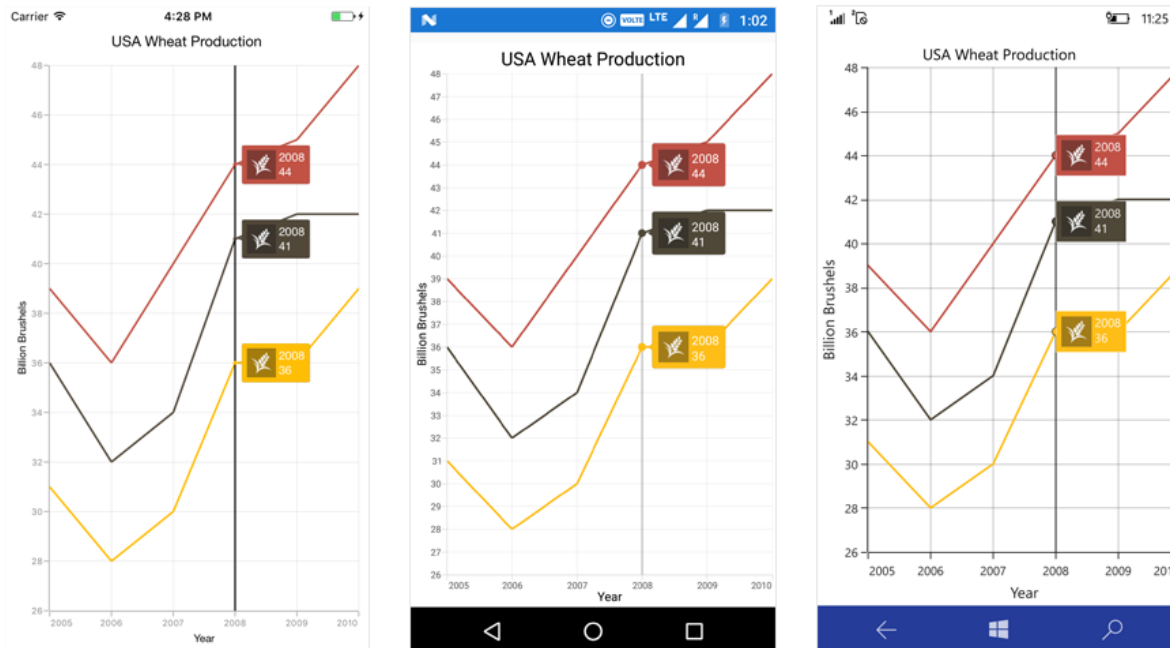
```

## C#

```

SfChart chart = new SfChart ();
...
var lineSeries1 = new LineSeries();
lineSeries1.ItemsSource = Data1;
lineSeries1.XBindingPath = "Name";
lineSeries1.YBindingPath = "Value";
var lineSeries2 = new LineSeries();
lineSeries2.ItemsSource = Data2;
lineSeries2.XBindingPath = "Name";
lineSeries2.YBindingPath = "Value";
DataTemplate trackBallTemplate = new DataTemplate(() =>
{
    StackLayout stack = new StackLayout();
    stack.Orientation = StackOrientation.Horizontal;
    Label label = new Label();
    label.SetBinding(Label.TextProperty, "Value");
    label.FontSize = 15;
    label.VerticalTextAlignment = TextAlignment.Center;
    label.TextColor = Color.White;
    Image image = new Image();
    image.Source = "grain.jpg";
    image.WidthRequest = 30;
    image.HeightRequest = 30;
    stack.Children.Add(label);
    stack.Children.Add(image);
    return stack;
});
lineSeries1.TrackballLabelTemplate = trackBallTemplate;
lineSeries2.TrackballLabelTemplate = trackBallTemplate;
chart.Series.Add(lineSeries1);
chart.Series.Add(lineSeries2);
chart.ChartBehaviors.Add(new ChartTrackballBehavior());

```



### Customize the Axis Label with DataTemplate

Customize the appearance of axis label of trackball using [TrackballLabelTemplate](#) property of [ChartAxis](#).

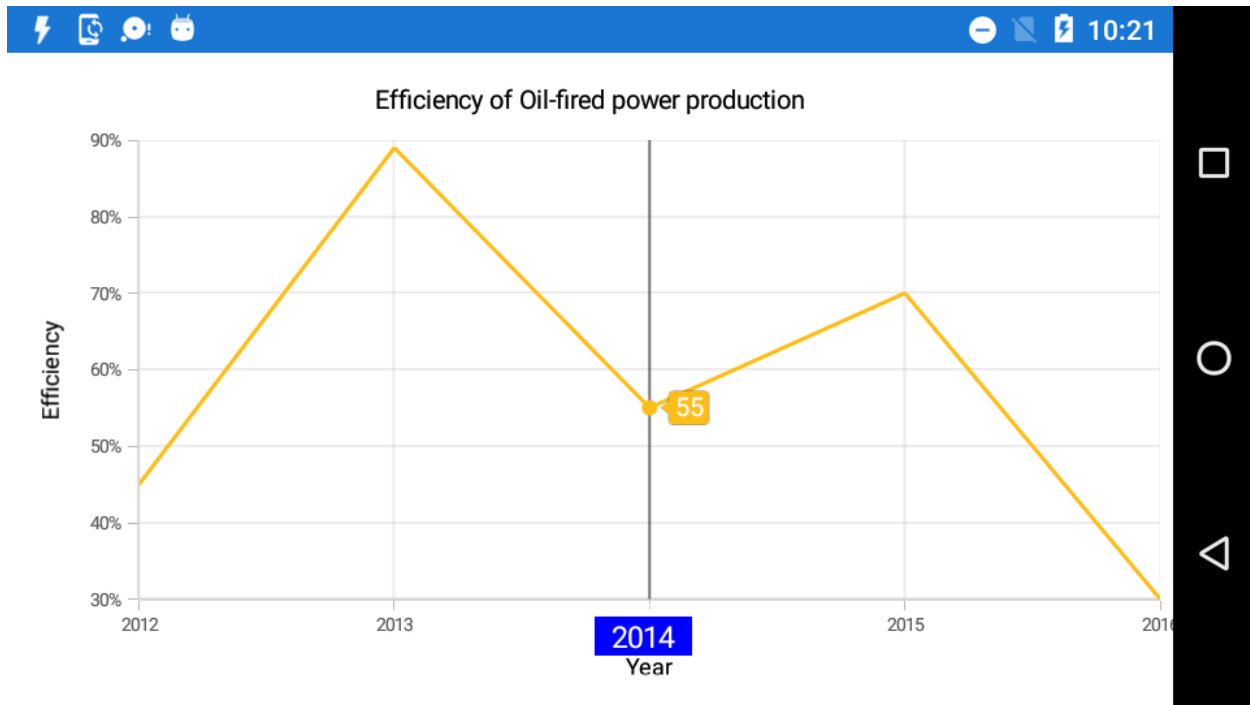
#### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="axisLabelTemplate">
<Label WidthRequest="50" HeightRequest="20" HorizontalTextAlignment="Center"
BackgroundColor="Blue" Text="{Binding }" TextColor="White" FontSize="15" />
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis ShowTrackballInfo="True"
TrackballLabelTemplate="{StaticResource axisLabelTemplate}">
<chart:CategoryAxis.TrackballLabelStyle>
<chart:ChartTrackballAxisLabelStyle BackgroundColor="Transparent"/>
</chart:CategoryAxis.TrackballLabelStyle>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.ChartBehaviors>
<chart:ChartTrackballBehavior />
</chart:SfChart.ChartBehaviors>
```

#### C#

```
Chart.PrimaryAxis = new CategoryAxis();
Chart.PrimaryAxis.ShowTrackballInfo = true;
Chart.PrimaryAxis.TrackballLabelStyle.BackgroundColor = Color.Transparent;
DataTemplate axisLabelTemplate = new DataTemplate(() =>
{
    Label label = new Label();
    label.SetBinding(Label.TextProperty, ".");
```

```
label.FontSize = 15;
label.HorizontalTextAlignment = TextAlignment.Center;
label.TextColor = Color.White;
label.BackgroundColor = Color.Blue;
label.WidthRequest = 50;
label.HeightRequest = 20;
return label;
});
Chart.PrimaryAxis.TrackballLabelTemplate = axisLabelTemplate;
Chart.ChartBehaviors.Add(new ChartTrackballBehavior());
```



## Methods

### Show method

The [Show](#) method is used to activate the trackball at the specified location.

#### C#

```
trackball.Show(pointX, pointY);
```

### Hide method

The [Hide](#) method is used to hide the trackball programmatically.

#### C#

```
trackball.Hide();
```

### HitTest method

The [HitTest](#) method is used to check whether the point is in trackball or not.

#### C#

```
[C#]  
trackball.HitTest(pointX, pointY);
```

## Events

### *TrackballCreated*

The [TrackballCreated](#) event occurs when the trackball moves from one data point to another. This argument contains object of [ChartPointsInfo](#). The following properties are available in ChartPointInfo class to customize the appearance of trackball label based on condition.

- [Label](#) - Gets or sets the text of trackball label.
- [IsVisible](#) - Gets or sets the visibility of trackball.
- [Series](#) - Gets the series of the data point in which trackball is activated.
- [LabelStyle](#) - Customizes the appearance of trackball label.
- [DataPoint](#) - Gets the respective underlying object of the data in which trackball is activated.
- [DataPointIndex](#) - Gets the index of the selected data point.
- [XPosition](#) - Gets the x-position of trackball label.
- [YPosition](#) - Gets the y-position of trackball label.
- [BackgroundColor](#) - Gets the default background color of trackball label.

## Tooltip

[SfChart](#) provides tooltip support for all series. It is used to show information about the segment, when you tap on the segment. To enable the tooltip, you need to set [EnableTooltip](#) property as `true`.

### XML

```
<chart:SfChart.Series>  
<chart:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Month"  
YBindingPath="Value" EnableTooltip="True"/>  
</chart:SfChart.Series>
```

### C#

```
ColumnSeries column = new ColumnSeries ();  
column.XBindingPath = "Month";  
column.YBindingPath = "Value";  
column.ItemsSource = Data;  
column.EnableTooltip = true;  
chart.Series.Add(column);
```





### Customizing appearance

You can customize the tooltip label. For customizing, you need to add an instance of [ChartTooltipBehavior](#) to the [ChartBehaviors](#) collection property of [SfChart](#). Following properties are used to customize the tooltip label which are available in [ChartTooltipBehavior](#).

- [BorderColor](#) – used to change the label border color
- [BorderWidth](#) – used to change the label border width
- [BackgroundColor](#) – used to change the label background color
- [Margin](#) – used to change label border thickness
- [TextColor](#) – used to change the text color
- [Font](#) – used to change the label font size, family, and weight. (This is deprecated API. Use [FontSize](#), [FontFamily](#), and [FontAttributes](#) properties instead of this.)
- [FontFamily](#) – used to change the font family for the tooltip text.
- [FontAttributes](#) – used to change the font style for the tooltip text.
- [FontSize](#) – used to change the font size for the tooltip text.
- [LabelFormat](#) – used to format the label
- [Duration](#) – used to set the visible duration of label
- [OffsetX](#) – used to move the label horizontally
- [OffsetY](#) – used to move the label vertically

### XML

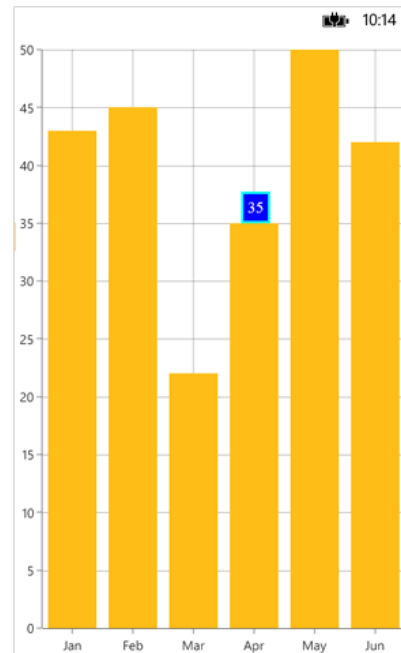
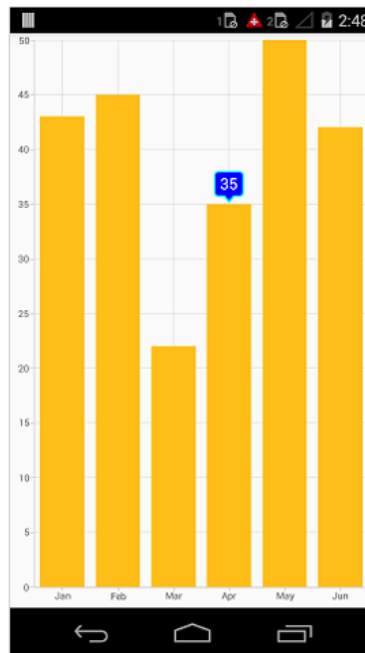
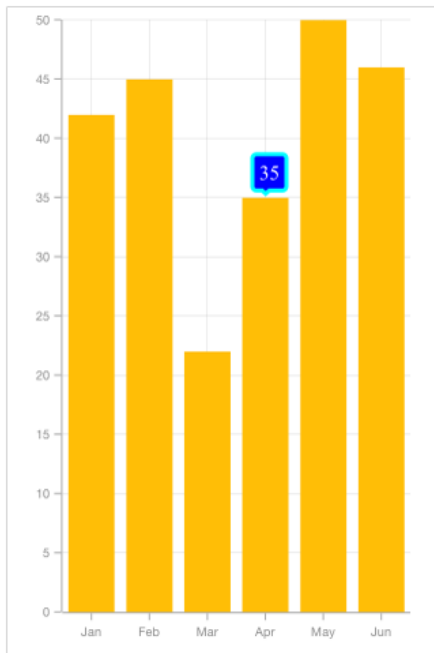
```
<chart:SfChart.ChartBehaviors>
  <chart:ChartTooltipBehavior BackgroundColor="Blue" BorderWidth="3"
    BorderColor="Aqua" TextColor="White" Margin="5" Duration="10" Font="Times
    New Roman, 15"/>
</chart:ChartTooltipBehavior>
</chart:SfChart.ChartBehaviors>
```

### C#

```

SfChart chart = new SfChart();
...
ChartTooltipBehavior tool = new ChartTooltipBehavior();
tool.BackgroundColor = Color.Blue;
tool.BorderWidth = 3;
tool.BorderColor = Color.Aqua;
tool.TextColor = Color.White;
tool.Margin = new Thickness(5, 5, 5, 5);
tool.Duration = 10;
tool.Font = Font.OfSize("Times New Roman", 15);
chart.ChartBehaviors.Add(tool);

```



### Tooltip Template

You can customize the appearance of the tooltip with your own template by using the [TooltipTemplate](#) property of [Series](#). The BindingContext in the data template will be the respective underlying object from ItemsSource.

### XML

```

<chart:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Month"
YBindingPath="Value" EnableTooltip="True">
  <chart:ColumnSeries.TooltipTemplate>
    <DataTemplate>
      <StackLayout Orientation="Vertical">
        <StackLayout Orientation="Horizontal">
          <Label Text="Month : " />
          <Label Text="{Binding Month}" />
        </StackLayout>
        <StackLayout Orientation="Horizontal">
          <Label Text="Value : " />
          <Label Text="{Binding Value}" />
        </StackLayout>
      </StackLayout>
    </DataTemplate>
  </chart:ColumnSeries.TooltipTemplate>
</chart:ColumnSeries>

```

```
</DataTemplate>
</chart:ColumnSeries.TooltipTemplate>
</chart:ColumnSeries>
<chart:SfChart.ChartBehaviors>
<chart:ChartTooltipBehavior BorderWidth="3" BorderColor="Maroon"/>
</chart:SfChart.ChartBehaviors>
```

## C#

```
ColumnSeries column = new ColumnSeries();
column.ItemsSource = viewModel.Data;
column.XBindingPath = "Month";
column.YBindingPath = "Value";
column.EnableTooltip = true;
ChartTooltipBehavior tooltip = new ChartTooltipBehavior();
tooltip.BorderColor = Color.Maroon;
tooltip.BorderWidth = 3;
chart.ChartBehaviors.Add(tooltip);
DataTemplate template = new DataTemplate(() =>
{
    StackLayout stack = new StackLayout() { Orientation =
    StackOrientation.Vertical };
    StackLayout first = new StackLayout() { Orientation =
    StackOrientation.Horizontal };
    Label label = new Label() { Text = "Month:" };
    Label xValue = new Label();
    xValue.SetBinding(Label.TextProperty, new Binding("Month"));
    first.Children.Add(label);
    first.Children.Add(xValue);
    StackLayout second = new StackLayout() { Orientation =
    StackOrientation.Horizontal };
    Label label1 = new Label() { Text = "Value:" };
    Label yValue = new Label();
    yValue.SetBinding(Label.TextProperty, "Value");
    second.Children.Add(label1);
    second.Children.Add(yValue);
    stack.Children.Add(first);
    stack.Children.Add(second);
    return stack;
});
column.TooltipTemplate = template;
chart.Series.Add(column);
```



## Methods

You can show or hide the chart tooltip programmatically by using the show or hide method.

### Show method

The [Show](#) method is used to activate the tooltip at the specified location.

## XML

```
<Button Text="Show tooltip" Clicked="ShowTooltip" />
. . .
<chart:SfChart.ChartBehaviors>
<chart:ChartTooltipBehavior x:Name="tooltipBehavior" />
</chart:SfChart.ChartBehaviors>
```

## C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
    private void ShowTooltip(object sender, EventArgs e)
    {
        //pointX - determines the x position of tooltip, pointY - determines the y
        position of tooltip and bool value determines whether the tooltip should be
        animated while displaying.
        tooltipBehavior.Show(pointX, pointY, true);
    }
}
```

---

**Note:** The tooltip will be activated at the specified location only if there is any data point under the specified location.

---

#### Hide method

The [Hide](#) method is used to hide the tooltip programmatically.

#### C#

```
//The argument determines whether the tooltip should be animated while hiding.
tooltip.Hide(true);
```

## Strip Lines

### What is strip line?

Strip lines are used to shade the different ranges in plot area in different colors to improve the readability of the chart. You can annotate it with text to indicate what that particular region indicates. You can also enable the strip lines to be drawn repeatedly at regular intervals – this will be useful when you want to mark an event that occurs recursively along the timeline of the chart.

### How to add strip lines?

Strip line is classified into **NumericalStripLine** and **DateTimeStripLine** based on the type of input you provide to draw the strip line. Since strip lines are drawn based on the axis, you have to add strip line instance using the [StripLines](#) property of the respective axis. You can also add multiple [StripLines](#) to an axis.

Following properties are used to configure the strip line.

- [Start](#) – used to change the [Start](#) position of the strip line.
- [Width](#) – used to change how long strip line should expand.
- [WidthType](#) - used to change the date time unit of the value specified in the width property. The values can be [Year](#), [Month](#), [Day](#), [Hour](#), [Minute](#), [Second](#) and [Millisecond](#).
- [Text](#) – used to change the text of the strip line.
- [FillColor](#) – used to change the fill color of the strip line.
- [StrokeWidth](#) – used to change the stroke width of the strip line.
- [StrokeColor](#) – used to change the stroke color of the strip line.
- [IsVisible](#) - used to change the visibility of the strip line in chart axis.
- [IsPixelWidth](#) - used to specify the unit type for [Width](#) property, whether it will be screen point or chart value.

### Numerical StripLine

[NumericalStripLine](#) are used to draw strip lines for [NumericalAxis](#) and [CategoryAxis](#). To add a strip line, create an instance of [NumericalStripLine](#) and add to the [StripLines](#) collection property of the respective axis.

#### XML

```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis Maximum="52" Minimum="28">
<chart:NumericalAxis>
<chart:NumericalStripLine Start="36" Width ="8" Text="Average Temperature"
FillColor="#F4C762"/>
```

```

</chart:NumericalAxis.StripLines>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>

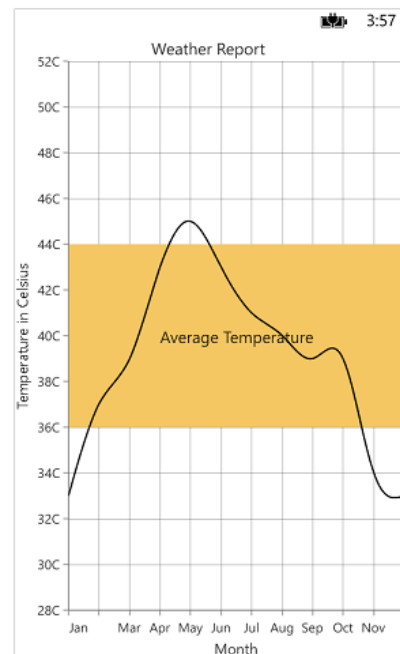
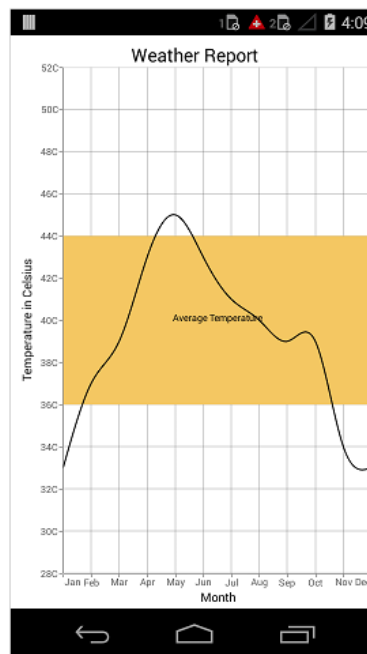
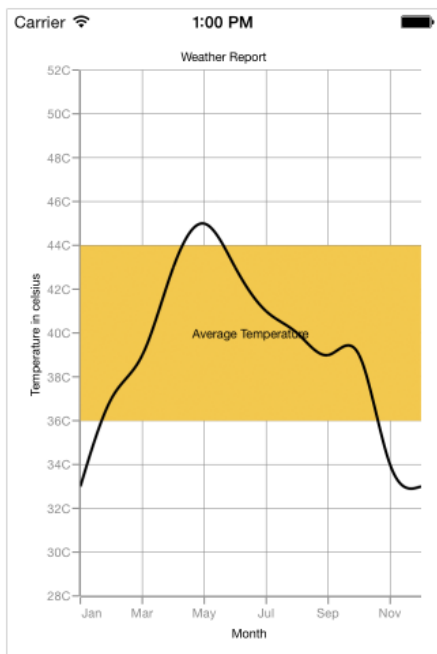
```

## C#

```

NumericalAxis numericalAxis = new NumericalAxis()
{
    Minimum=28,
    Maximum=52
};
chart.SecondaryAxis = numericalAxis;
NumericalStripLine stripLine1 = new NumericalStripLine()
{
    Start = 36,
    Width = 8,
    Text = "Average Temperature",
    FillColor = Color.FromHex("#F4C762")
};
numericalAxis.StripLines.Add(stripLine1);

```



## DateTime StripLine

As the name indicates, [DateTimeStripLine](#) are used to draw strip lines for [DateTimeAxis](#). To add a strip line for [DateTimeAxis](#), create an instance of [DateTimeStripLine](#) and add to the [StripLines](#) collection property of [DateTimeAxis](#).

## XML

```

Namespace:
xmlns:sys="clr-namespace:System;assembly=mscorlib"
...
<chart:SfChart.PrimaryAxis >

```

```

<chart:DateTimeAxis>
<chart:DateTimeAxis.StripLines>
<chart:DateTimeStripLine WidthType="Month" Width="3" Text="Quarter-2"
FillColor="#3FAA38">
<chart:DateTimeStripLine.Start>
<sys:DateTime x:FactoryMethod="Parse">
<x:Arguments>
<x:String>Apr 01 2000</x:String>
</x:Arguments>
</sys:DateTime>
</chart:DateTimeStripLine.Start>
</chart:DateTimeStripLine >
</chart:DateTimeAxis.StripLines>
</chart:DateTimeAxis>
</chart:SfChart.PrimaryAxis>

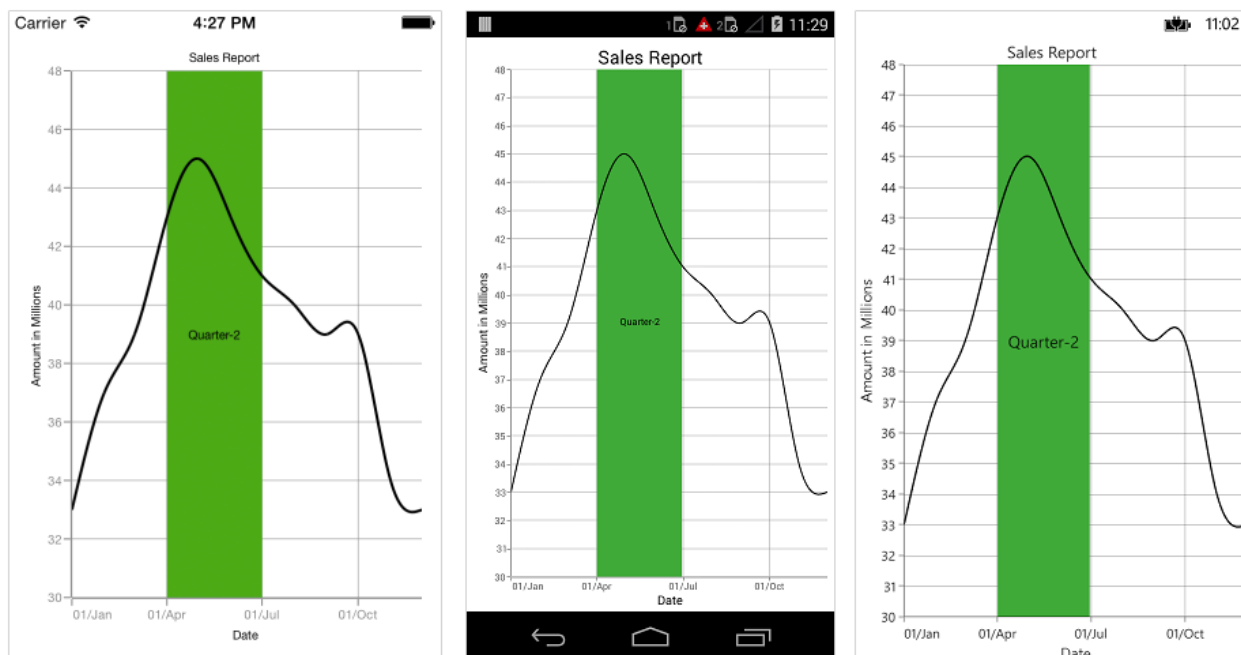
```

**C#**

```

DateTimeAxis dateTimeAxis = new DateTimeAxis()
{
    Interval = 3,
    IntervalType = DateTimeIntervalType.Months,
    Minimum = new DateTime(2000, 01, 01),
    Maximum = new DateTime(2000, 12, 30)
};
chart.PrimaryAxis = dateTimeAxis;
DateTimeStripLine stripLine = new DateTimeStripLine()
{
    Start = new DateTime(2000, 04, 01),
    WidthType = DateTimeComponent.Month,
    Width = 3,
    Text = "Quarter-2",
    FillColor = Color.FromHex("#3FAA38")
};
dateTimeAxis.StripLines.Add(stripLine);

```



### Strip Line Recurrence

This feature is used to enable the strip lines to be drawn repeatedly at the regular intervals – this will be useful when you want to mark an event that occurs recursively along the timeline of the chart. Following properties are used to configure this feature.

- [RepeatEvery](#) – used to change the frequency of the strip line being repeated.
- [RepeatUntil](#) – specifies the end value at which point strip line has to stop repeating.

Following code snippet and screenshot demonstrates this feature by highlighting weekends.

### XML

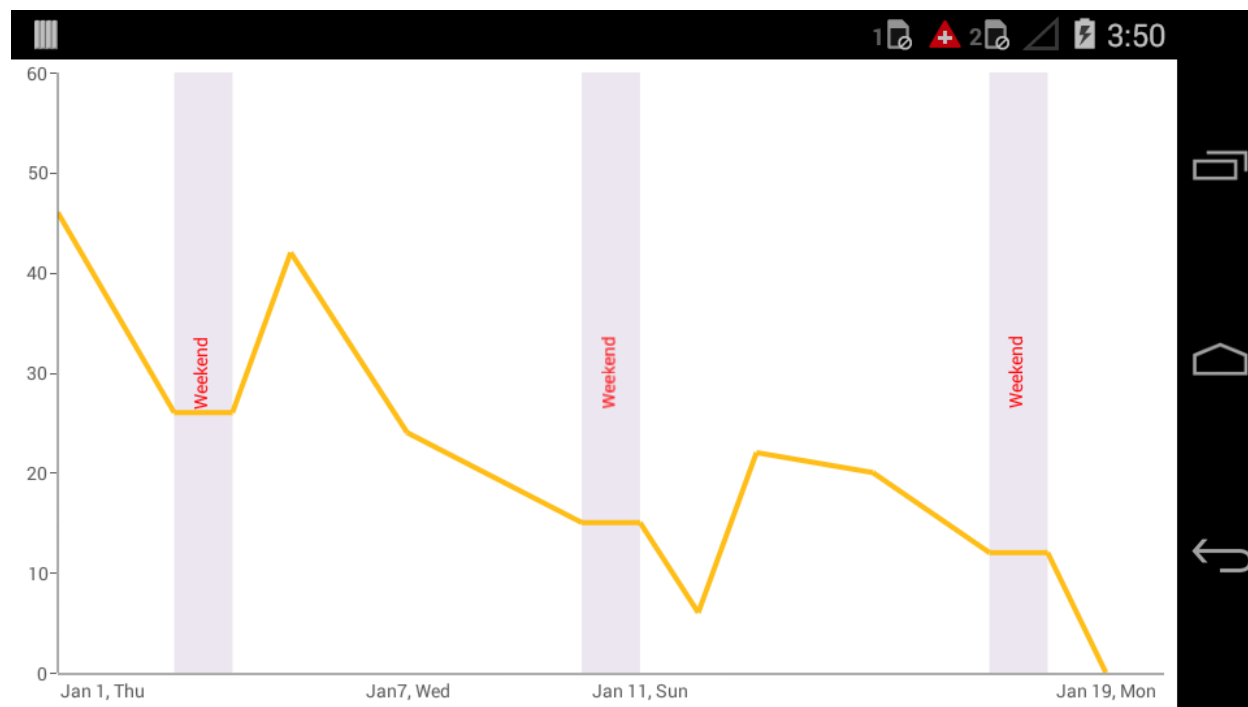
```
<chart:SfChart.PrimaryAxis>
<chart:NumericalAxis>
<chart:NumericalAxis.StripLines>
<chart:NumericalStripLine Start="6" Width="1" RepeatEvery="7"
Text="Weekend" FillColor="#ECE6F1">
<chart:NumericalStripLine.LabelStyle>
<chart:ChartStripLineLabelStyle Angle="270" TextColor="Red"/>
</chart:NumericalStripLine.LabelStyle>
</chart:NumericalStripLine>
</chart:NumericalAxis.StripLines>
</chart:NumericalAxis>
</chart:SfChart.PrimaryAxis>
```

### C#

```
NumericalAxis numericalAxis = new NumericalAxis()
{
    EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Shift
};
sfChart.PrimaryAxis = numericalAxis;
NumericalStripLine stripLine = new NumericalStripLine();
```



```
stripLine.FillColor = Color.FromHex("FFECE6F1");
stripLine.Start = 6;
stripLine.Width = 1;
stripLine.RepeatEvery = 7;
stripLine.Text = "Weekend";
stripLine.LabelStyle.Angle = 270;
stripLine.LabelStyle.TextColor = Color.Red;
numericalAxis.StripLines.Add(stripLine);
```



### Customize Text

The [LabelStyle](#) property provide options to customize the font-family, color, size and font-weight of strip line text. Following are the options available,

- [TextColor](#) – used to change the color of the text.
- [BackgroundColor](#) – used to change the label background color.
- [BorderColor](#) – used to change the border color.
- [BorderThickness](#) – used to change the thickness of the border.
- [Font](#) – used to change the text size, font family and font weight.
- [Margin](#) - used to change the margin size for text.
- [Angle](#) – used to rotate the text.
- [HorizontalAlignment](#) – used to change the horizontal alignment of text.
- [VerticalAlignment](#) - used to change the vertical alignment of text.

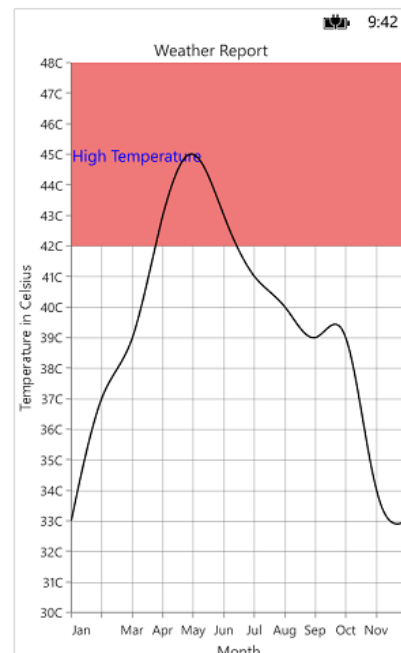
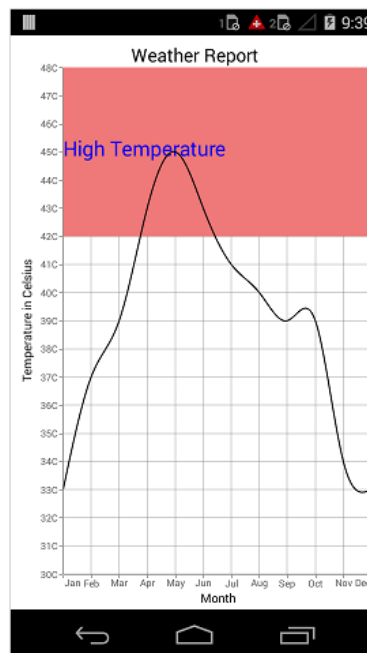
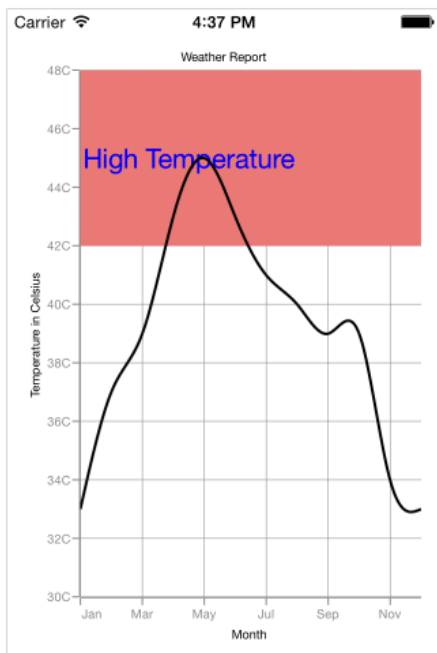
### XML

```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis Maximum="30" Minimum="48">
<chart:NumericalAxis.StripLines>
```

```
<chart:NumericalStripLine Start="42" Width="6" Text="High Temperature"
FillColor="#EF7878">
<chart:NumericalStripLine.LabelStyle>
<chart:ChartStripLineLabelStyle HorizontalAlignment="Near"
VerticalAlignment="Center" TextColor="Blue" Font="20"/>
</chart:NumericalStripLine.LabelStyle>
</chart:NumericalStripLine>
</chart:NumericalAxis.StripLines>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

**C#**

```
NumericalAxis numericalAxis = new NumericalAxis()
{
    Minimum=30,
    Maximum=48
};
chart.SecondaryAxis = numericalAxis;
chart.SecondaryAxis.Title.Text = "Temperature in Celsius";
NumericalStripLine stripLine = new NumericalStripLine()
{
    Start = 42,
    Width = 6,
    Text = "High Temperature",
    FillColor = Color.FromHex("#EF7878")
};
stripLine.LabelStyle.TextColor = Color.Blue;
stripLine.LabelStyle.Font = Font.SystemFontOfSize(20);
stripLine.LabelStyle.HorizontalAlignment = ChartLabelAlignment.Near;
stripLine.LabelStyle.VerticalAlignment = ChartLabelAlignment.Center;
numericalAxis.StripLines.Add(stripLine);
```



### Segmented StripLine

Typically, if you draw a strip line for a vertical axis, the height of the strip line is determined by the [Start](#) and [Width](#) properties and width of the strip line is equivalent to the width of its associated horizontal axis i.e., strip line is drawn horizontally to the entire stretch of its associated horizontal axis. Similarly, for horizontal axis, width is determined by [Start](#) and [Width](#) properties, and vertically, it is drawn to the entire stretch of the associated vertical axis.

Suppose, you want to draw a strip line that should not stretch along its associated axis, you have to set [SegmentStart](#) and [SegmentEnd](#) properties. Values provided in these two properties correspond to its associated axis specified by [SegmentAxisName](#) property. Finally, you need to set [IsSegmented](#) property to true to indicate that strip line should be drawn as a segment.

- [IsSegmented](#) – Used to enable / disable the segmented strip line.
- [SegmentStart](#) – Used to change the segment start value. Value correspond to associated axis.
- [SegmentEnd](#) – Used to change the segment end value. Value correspond to associated axis.
- [SegmentAxisName](#) – Used to specify the name of the associated axis name.
- [Name](#) - Used to specify the unique name for the axis.

---

**Note:** You can set the double or DateTime value for SegmentStart and SegmentEnd properties based on the associated axis type.

---

Following code snippet shows how to set the segment start and end value if the associated axis type is numerical.

#### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis EdgeLabelsDrawingMode="Shift" Interval="3">
<chart:CategoryAxis.StripLines>
<chart:NumericalStripLine IsSegmented="True" SegmentAxisName="Amount"
SegmentStart="40" SegmentEnd="46" Start="3" Width ="3" Text="Quarter-2"
FillColor="#EF7878"/>
</chart:CategoryAxis.StripLines>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis Minimum="30" Maximum="48" Name="Amount"/>
</chart:SfChart.SecondaryAxis>
```

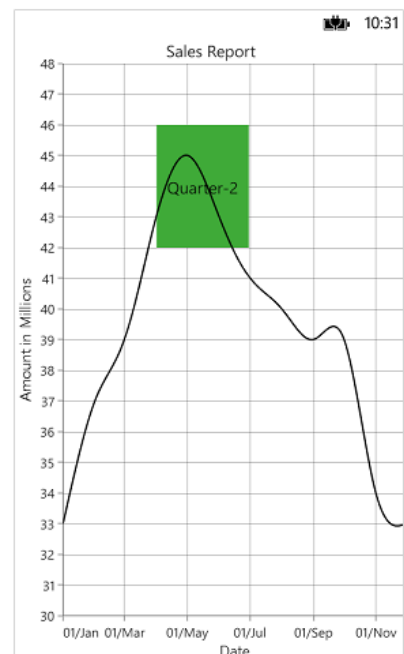
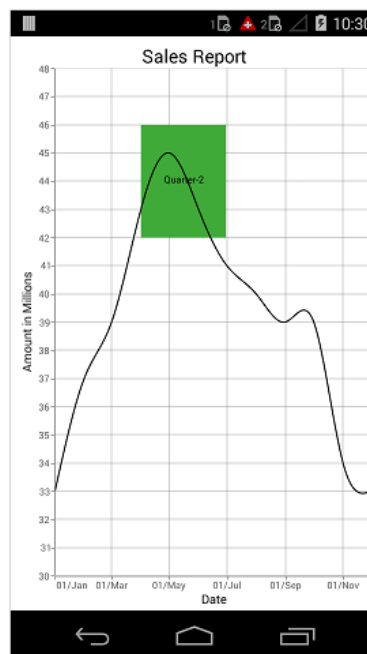
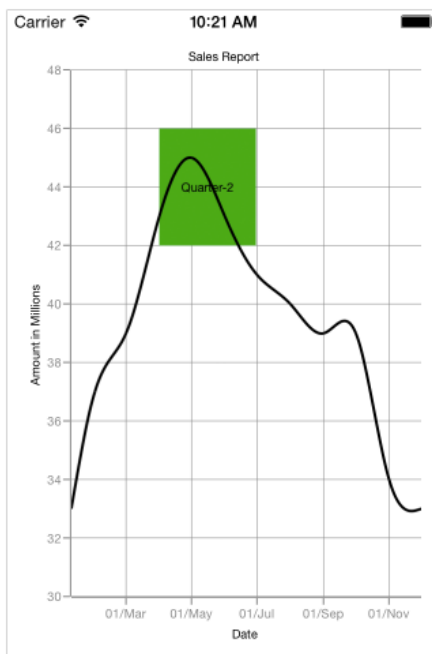
#### C#

```
NumericalStripLine stripLine = new NumericalStripLine()
{
    Start = 3,
    Width = 3,
    Text = "Quarter-2",
    FillColor = Color.FromHex("#EF7878"),
    IsSegmented = true,
    SegmentAxisName = "Amount",
    SegmentStart = 40,
    SegmentEnd = 46
};
CategoryAxis categoryAxis = new CategoryAxis()
{
```

```

EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Shift,
Interval = 3
};
NumericalAxis numericalAxis = new NumericalAxis()
{
    Minimum = 30,
    Maximum = 48,
    Name = "Amount"
};
categoryAxis.StripLines.Add(stripLine);
chart.PrimaryAxis = categoryAxis;
chart.SecondaryAxis = numericalAxis;

```



Following code snippet shows how to set the segment start and end value if the associated axis type is date time.

### XML

```

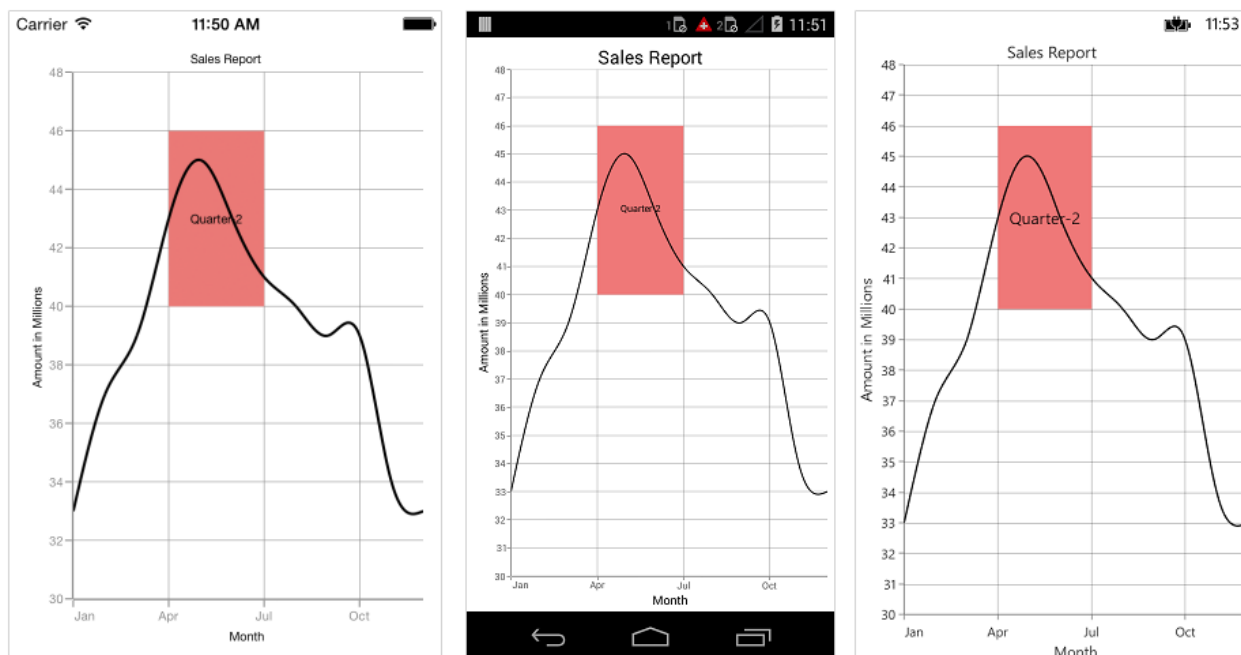
<chart:SfChart.PrimaryAxis>
<chart:DateTimeAxis EdgeLabelsDrawingMode="Shift" Name="Period"/>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis>
<chart:NumericalAxis.StripLines>
<chart:NumericalStripLine IsSegmented="True" SegmentAxisName="Period"
Start="42" Width="4" Text="Quarter-2" FillColor="#3FAA38">
<chart:NumericalStripLine.SegmentStart>
<sys:DateTime x:FactoryMethod="Parse">
<x:Arguments>
<x:String>Jan 1 2000</x:String>
</x:Arguments>
</sys:DateTime>
</chart:NumericalStripLine.SegmentStart>
<chart:NumericalStripLine.SegmentEnd>

```

```
<sys:DateTime x:FactoryMethod="Parse">
<x:Arguments>
<x:String>Jun 30 2000</x:String>
</x:Arguments>
</sys:DateTime>
</chart:NumericalStripLine.SegmentEnd>
</chart:NumericalStripLine>
</chart:NumericalAxis.StripLines>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

## C#

```
NumericalStripLine stripLine = new NumericalStripLine()
{
    Start = 42,
    Width = 4,
    Text = "Quarter-2",
    FillColor = Color.FromHex("#3FAA38"),
    IsSegmented = true,
    SegmentAxisName = "Period",
    SegmentStart = new DateTime(2000, 4, 1),
    SegmentEnd = new DateTime(2000, 6, 30)
};
chart.PrimaryAxis = new DateTimeAxis()
{
    EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Shift,
    Name = "Period"
};
NumericalAxis numericalAxis = new NumericalAxis()
{
    Minimum = 30,
    Maximum = 48
};
chart.SecondaryAxis = numericalAxis;
numericalAxis.StripLines.Add(stripLine);
```



## Performance

Following are the key points that can be used to boost the performance of the chart when there is a need to plot high volume data.

- When there are large number of points to load in line series, you can use [FastLineSeries](#) series instead of [LineSeries](#). To render a fast line chart, create an instance of [FastLineSeries](#) and add to the [Series](#) collection property of [SfChart](#).

## XML

```
<chart:SfChart>
...
<chart:FastLineSeries ItemsSource="{Binding Data}" XBindingPath="XValue"
YBindingPath="YValue"/>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
FastLineSeries fastLineSeries = new FastLineSeries()
{
    ItemsSource = Data,
    XBindingPath = "XValue",
    YBindingPath = "YValue"
};
chart.Series.Add(fastLineSeries);
```

**Note:** If you have minimal set of data points, the recommended approach is to use normal line series to visualize those data using line chart. Because the normal line series has provisions to customize the color and shape of individual line.

- Instead of enabling data markers and labels when there are large number of data points, you can use [Trackball](#) to view the point information.
- The default stroke width of the [FastLineSeries](#) is 2, reducing this to 1 will improve the performance. Refer the following code snippet to configure the stroke width of FastLineSeries.

### XML

```
<chart:SfChart>
...
<chart:FastLineSeries ItemsSource="{Binding Data}" XBindingPath="XValue"
YBindingPath="YValue" StrokeWidth="1"/>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
FastLineSeries fastLineSeries = new FastLineSeries()
{
    ItemsSource = Data,
    XBindingPath = "XValue",
    YBindingPath = "YValue",
    StrokeWidth = 1
};
chart.Series.Add(fastLineSeries);
```

- When the underlying data object implements [INotifyPropertyChanged](#), you need to enable the [ListenPropertyChange](#) property of the series, to make the chart listen to the property changes of your data object. However enabling this property registers [PropertyChanged](#) event of every object in the data source. Due to this, chart's loading time is affected when there are a large number of points. By default, [ListenPropertyChange](#) is set to false in order to avoid the event registration unnecessarily.
- Whenever there is a new data point is added to the [ItemsSource](#) property of [ChartSeries](#), the chart will be refreshed with new data point if the [ItemsSource](#) property contains [ObservableCollection](#). In order to avoid the chart rendering for each update in [ItemsSource](#), you can suspend the chart using [SuspendSeriesNotification](#) method of chart and the [ResumeSeriesNotification](#) should be called once the required data points are added to the collection and the chart should be refreshed with data points that have been added between these two method calls.

### C#

```
Chart.SuspendSeriesNotification();
// ...
// Add the data points to ItemsSource property.
// ...
Chart.ResumeSeriesNotification();
```

Similarly, you can use [SuspendNotification](#) and [ResumeNotification](#) methods of [ChartSeries](#) to suspend and resume the update of the respective series.

## C#

```
series.SuspendNotification();  
// ...  
// Add the data points to ItemsSource property.  
// ...  
series.ResumeNotification();
```

## Annotation

[SfChart](#) supports annotations which allows you to mark the specific area of interest in the chart area. You can add text, images, and custom views.

The following annotations are supported in [SfChart](#):

- Text annotation
- Shape annotation
- View annotation

## Adding Annotations

You can create an instance for any type of annotations and it can be added to [ChartAnnotations](#) collection. Here for an instance, the [EllipseAnnotation](#) is added.

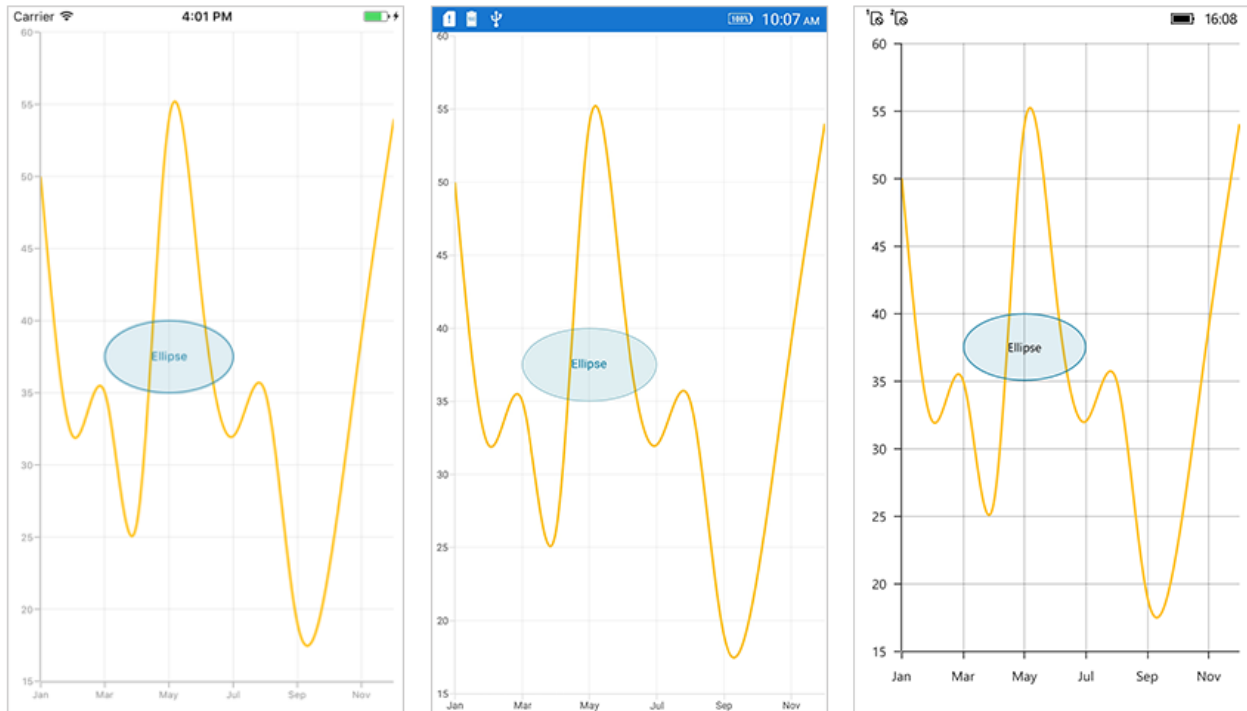
## XML

```
<chart:SfChart>  
...  
<chart:SfChart.ChartAnnotations>  
<chart:EllipseAnnotation X1="2" Y1="35" X2="6" Y2="40" Text="Ellipse" />  
</chart:SfChart.ChartAnnotations>  
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();  
...  
EllipseAnnotation annotation = new EllipseAnnotation()  
{  
    X1 = 2,  
    Y1 = 35,  
    X2 = 6,  
    Y2 = 40,  
    Text = "Ellipse"  
};  
chart.ChartAnnotations.Add(annotation);
```





### Positioning the annotation

Annotations can be positioned in plot area based on [X1](#) and [Y1](#) properties. For shape annotations, specify [X2](#) and [Y2](#) properties, if needed. The [X](#) and [Y](#) values can be specified with axis units or pixel units, and these can be identified by using [CoordinateUnit](#) property.

#### Positioning based on *CoordinateUnit* as axis

To position the annotation based on axis, set the [X1](#) and [Y1](#), [X2](#) and [Y2](#) properties based on axis range values, if needed, set the [CoordinateUnit](#) value as [Axis](#).

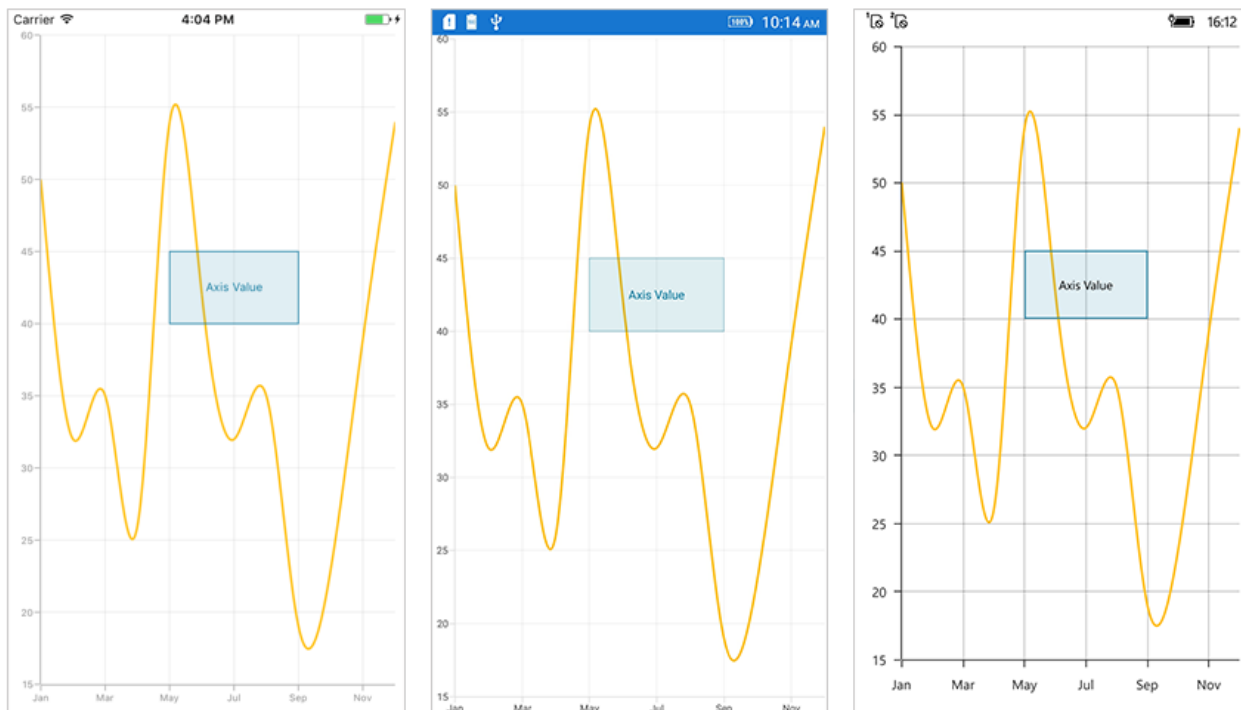
### XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:RectangleAnnotation X1="4" Y1="40" X2="8" Y2="45" Text="Axis Value"
CoordinateUnit="Axis" />
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
RectangleAnnotation annotation = new RectangleAnnotation()
{
    X1 = 4,
    Y1 = 40,
    X2 = 8,
    Y2 = 45,
    Text = "Axis Value",
    CoordinateUnit = ChartCoordinateUnit.Axis
}
```

```
};
chart.ChartAnnotations.Add(annotation);
```



#### Positioning based on *CoordinateUnit* as pixels

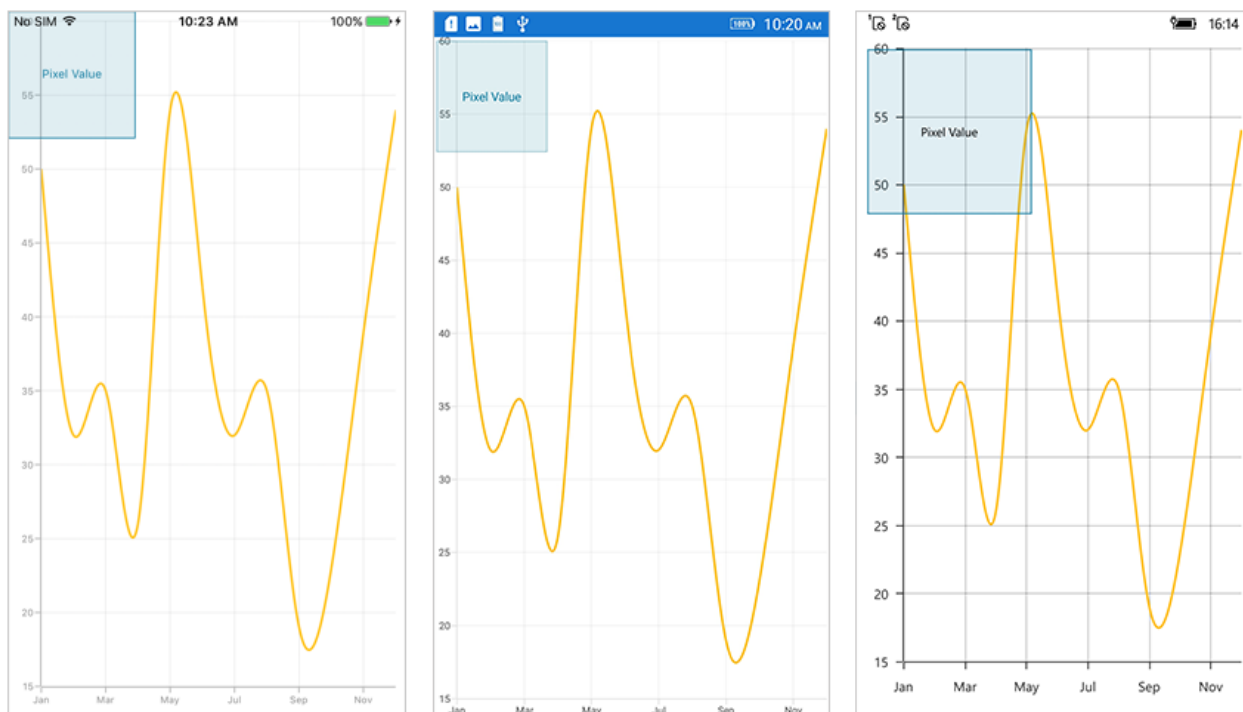
To position the annotation based on the pixel values, set the [CoordinateUnit](#) value as [Pixels](#), and the pixel values in [X1](#) and [Y1](#), [X2](#) and [Y2](#) properties of annotation are shown in the following code snippet,

#### XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:RectangleAnnotation X1="1" Y1="1" X2="150" Y2="150" Text="Pixel
Value" CoordinateUnit="Pixels" />
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart();
...
RectangleAnnotation annotation = new RectangleAnnotation()
{
    X1 = 1,
    Y1 = 1,
    X2 = 150,
    Y2 = 150,
    Text = "Pixel Value",
    CoordinateUnit = ChartCoordinateUnit.Pixels
};
chart.ChartAnnotations.Add(annotation);
```



### Adding annotation for multiple axes

When there are multiple axes, annotation also can be added for a particular axis by using the [XAxisName](#) and [YAxisName](#) properties. It can be shown in the below code snippet,

#### XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:EllipseAnnotation X1="4" Y1="30" X2="8" Y2="35" YAxisName="Yaxis"/>
</chart:SfChart.ChartAnnotations>
<chart:SfChart.Series>
<chart:SplineSeries ItemsSource="{Binding Data}" XBindingPath="Name"
YBindingPath="Value">
<chart:SplineSeries.YAxis>
<chart:NumericalAxis Name="Yaxis" OpposedPosition="true"/>
</chart:SplineSeries.YAxis>
</chart:SplineSeries>
</chart:SfChart.Series>
</chart:SfChart>
```

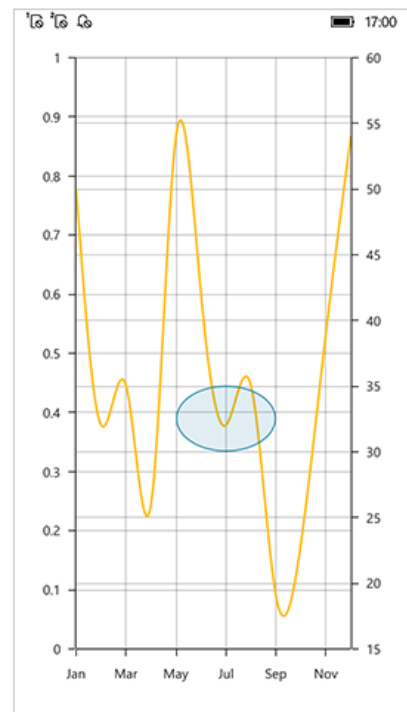
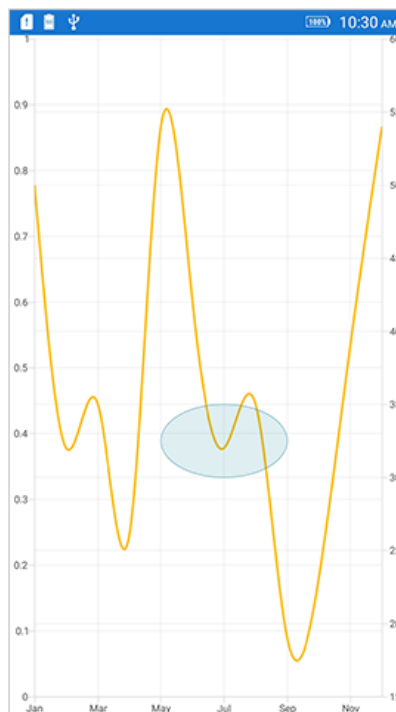
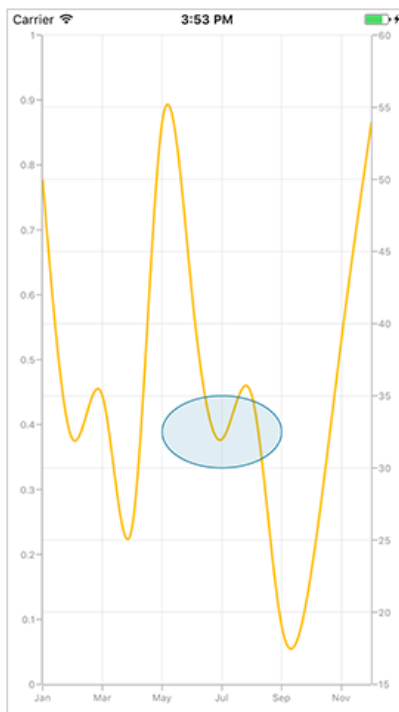
#### C#

```
SfChart chart = new SfChart();
...
EllipseAnnotation annotation = new EllipseAnnotation()
{
    X1 = 4,
    Y1 = 30,
    X2 = 8,
```

```

Y2 = 35,
YAxisName = "YAxis"
};
chart.ChartAnnotations.Add(annotation);
SplineSeries series = new SplineSeries()
{
    ItemsSource = model.Data,
    XBindingPath = "Name",
    YBindingPath = "Value",
    YAxis = new NumericalAxis()
    {
        OpposedPosition = true,
        Name = "YAxis"
    }
};
chart.Series.Add(series);

```



### Text annotation

The [TextAnnotation](#) is used to add simple text with the help of [Text](#) property in specific points over the [Chart](#) area.

### XML

```

<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:TextAnnotation X1="7" Y1="35" Text="August" />
</chart:SfChart.ChartAnnotations>
</chart:SfChart>

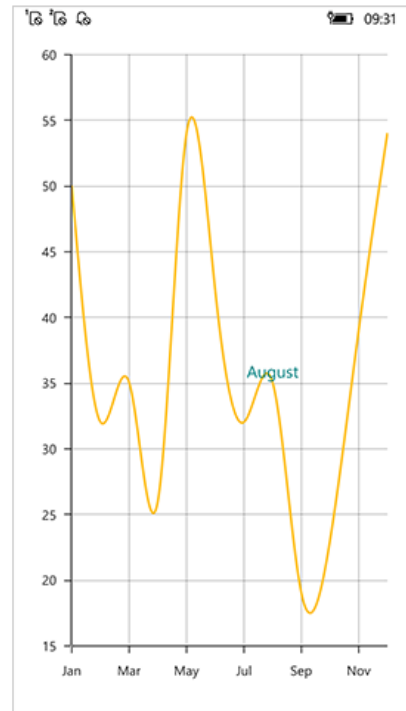
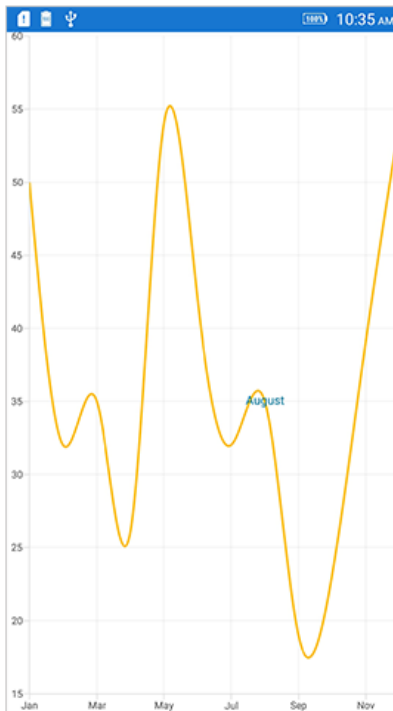
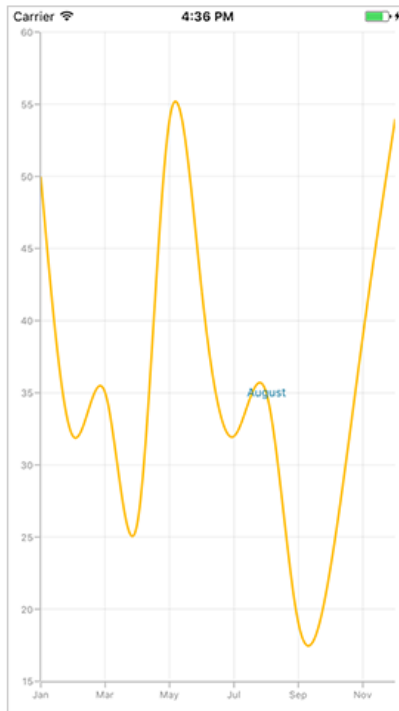
```

### C#

```

SfChart chart = new SfChart();
...
TextAnnotation annotation = new TextAnnotation()
{
    X1 = 7,
    Y1 = 35,
    Text = "August"
};
chart.ChartAnnotations.Add(annotation);

```



### Customizing text annotation

The [TextAnnotation](#) can be customized by using the [LabelStyle](#) property. The following properties are used to customize the text:

- [TextColor](#) - Used to change the text color.
- [BackgroundColor](#) - Used to change the background color of the text.
- [Margin](#) - Used to set the margin for text.
- [BorderThickness](#) - Used to change the text border thickness.
- [BorderColor](#) - Used to change the border color of the text.
- [Font](#) - Used to change text font size, family and weight.
- [HorizontalTextAlignment](#) - Used to align the text horizontally at the [Start](#), [Center](#) and [End](#).
- [VerticalTextAlignment](#) - Used to align the text vertically at the [Start](#), [Center](#) and [End](#).

### XML

```

<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:TextAnnotation X1="7" Y1="35" Text="August">

```

```

<chart:TextAnnotation.LabelStyle>
<chart:ChartAnnotationLabelStyle Margin="5" Font="Italic,16"
BorderColor="Red" BorderThickness="2" BackgroundColor="Teal"
TextColor="White" VerticalTextAlignment="Start"/>
</chart:TextAnnotation.LabelStyle>
</chart:TextAnnotation>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>

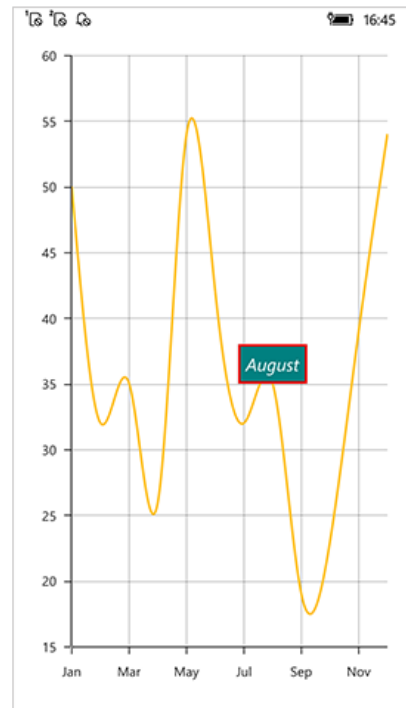
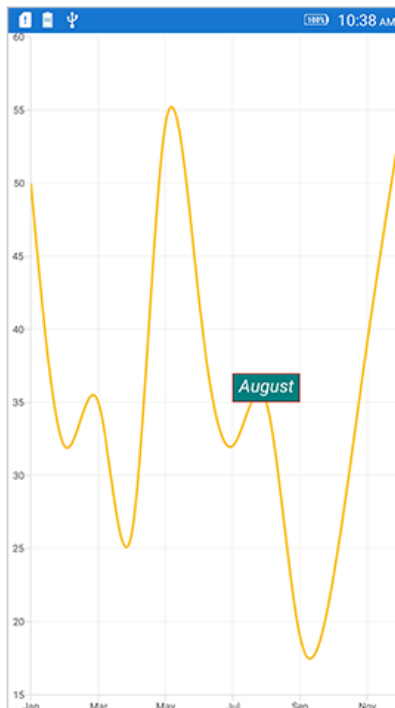
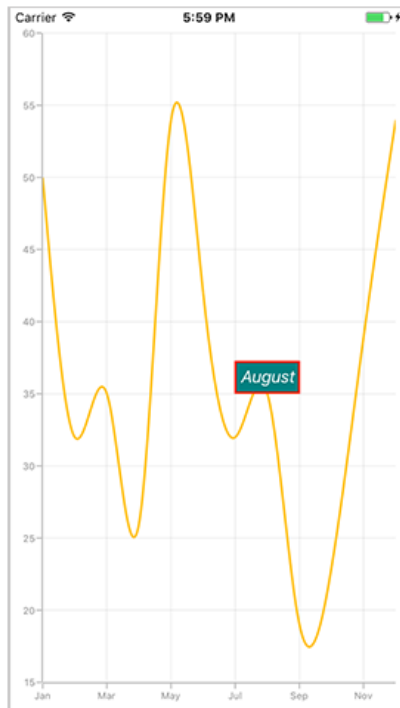
```

## C#

```

SfChart chart = new SfChart();
...
TextAnnotation annotation = new TextAnnotation()
{
    X1 = 7,
    Y1 = 35,
    Text = "August"
};
annotation.LabelStyle = new ChartAnnotationLabelStyle()
{
    Margin = new Thickness(5),
    Font = Font.SystemFontOfSize(16, FontAttributes.Italic),
    BorderColor = Color.Red,
    BorderThickness = new Thickness(2),
    BackgroundColor = Color.Teal,
    TextColor = Color.White,
    VerticalTextAlignment = ChartAnnotationAlignment.Start
};
chart.ChartAnnotations.Add(annotation);

```



### Shape annotation

The [ShapeAnnotation](#) allows you to add annotations in the form of shapes such as rectangle, ellipse, horizontal line, vertical line, etc., at the specific area of interest in the chart area.

- [RectangleAnnotation](#) - Used to draw a rectangle over the chart area.
- [EllipseAnnotation](#) - Used to draw a circle or an ellipse over the chart area.
- [LineAnnotation](#) - Used to draw a line over the chart area.
- [VerticalLineAnnotation](#) - Used to draw a vertical line across the chart area.
- [HorizontalLineAnnotation](#) - Used to draw a horizontal line across the chart area.

The following APIs are commonly used in all [ShapeAnnotation](#):

- [X2](#) - Represents the X2 coordinate of the shape annotation.
- [Y2](#) - Represents the Y2 coordinate of the shape annotation.
- [FillColor](#) - Represents the inside background color of the shape annotation.
- [StrokeColor](#) - Represents the stroke color of the shape annotation.
- [StrokeWidth](#) - Represents the stroke width of the shape annotation.
- [StrokeDashArray](#) - Represents the stroke dashes of the shape annotation.
- [Text](#) - Represents the annotation text of the shape annotation.
- [LabelStyle](#) - Represents the style for customizing the annotation text of shape annotation.

### Rectangle annotation

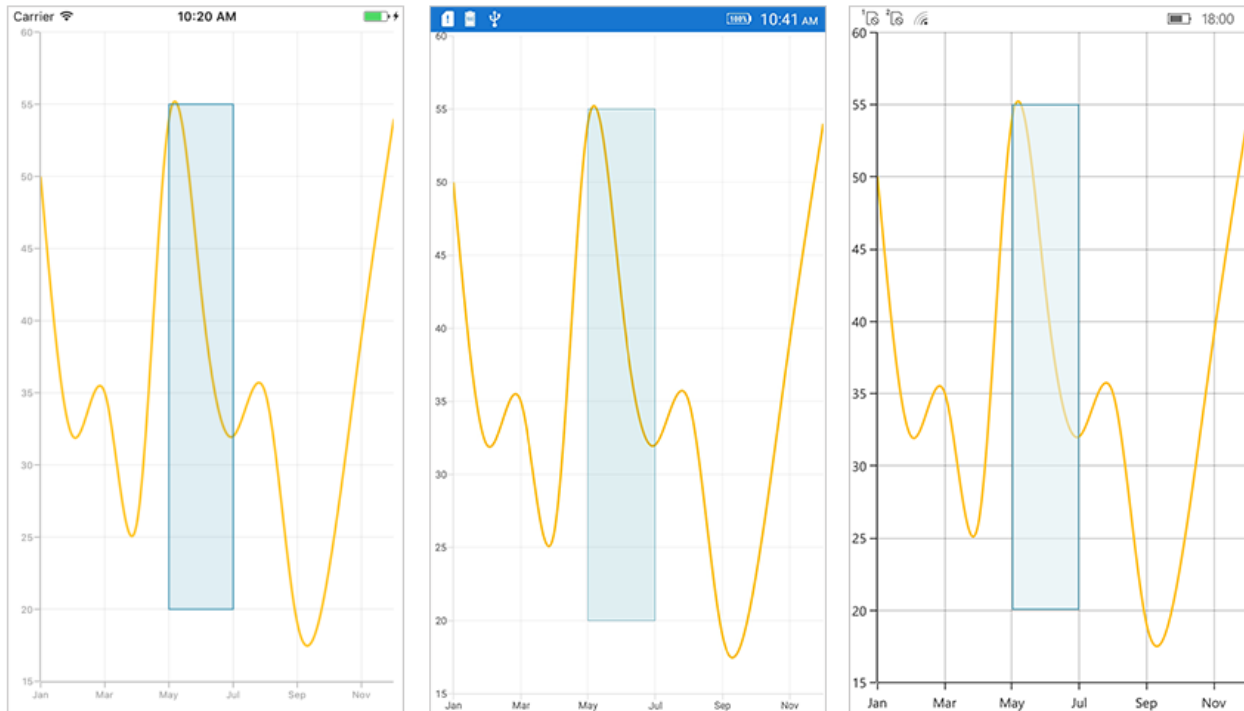
The [RectangleAnnotation](#) is used to draw a rectangle or a square in specific points over the chart area.

#### XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:RectangleAnnotation X1="4" Y1="20" X2="6" Y2="55"/>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart();
...
RectangleAnnotation annotation = new RectangleAnnotation()
{
    X1 = 4,
    Y1 = 20,
    X2 = 6,
    Y2 = 55,
};
chart.ChartAnnotations.Add(annotation);
```



### Ellipse annotation

The [EllipseAnnotation](#) is used to draw an oval or a circle in specific points over the chart area. You can also specify the height and width of [EllipseAnnotation](#) by using the [Height](#) and [Width](#) properties respectively.

### XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:EllipseAnnotation X1="6" Y1="32" Height="30" Width="30"/>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
EllipseAnnotation annotation = new EllipseAnnotation()
{
    X1 = 6,
    Y1 = 32,
    Height = 30,
    Width = 30
};
chart.ChartAnnotations.Add(annotation);
```

**Note:** When X2 and Y2 properties of [EllipseAnnotation](#) are set, the [Height](#) and [Width](#) properties do not work.



### Line annotation

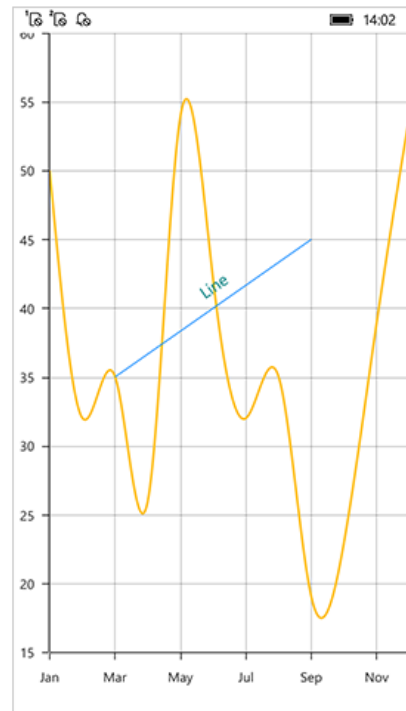
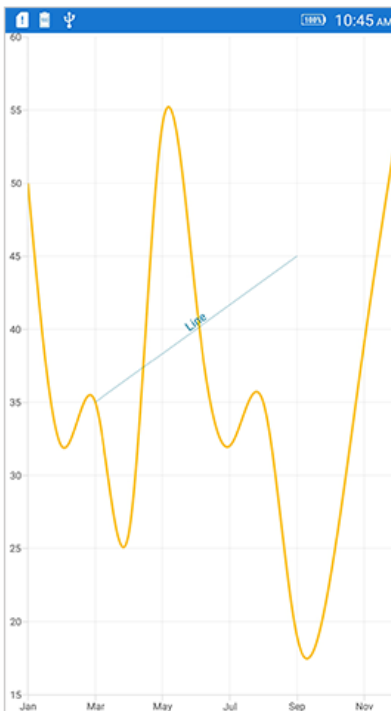
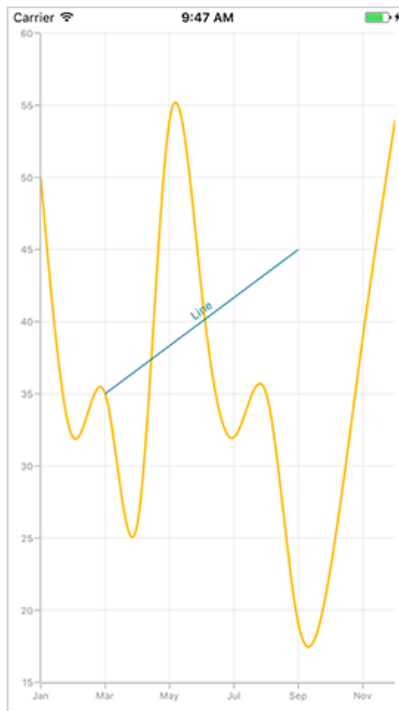
The [LineAnnotation](#) is used to draw a line in specific points over the chart area.

#### XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:LineAnnotation X1="2" Y1="35" X2="8" Y2="45" Text="Line"/>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart();
...
LineAnnotation annotation = new LineAnnotation()
{
    X1 = 2,
    Y1 = 35,
    X2 = 8,
    Y2 = 45,
    Text = "Line"
};
chart.ChartAnnotations.Add(annotation);
```



#### Adding arrow to line annotation

To display single headed arrow, set the [LineCap](#) property to [Arrow](#). The default value of the [LineCap](#) property is [None](#).

#### XML

```

<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:LineAnnotation X1="2" Y1="40" X2="10" Y2="40" LineCap="Arrow"/>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>

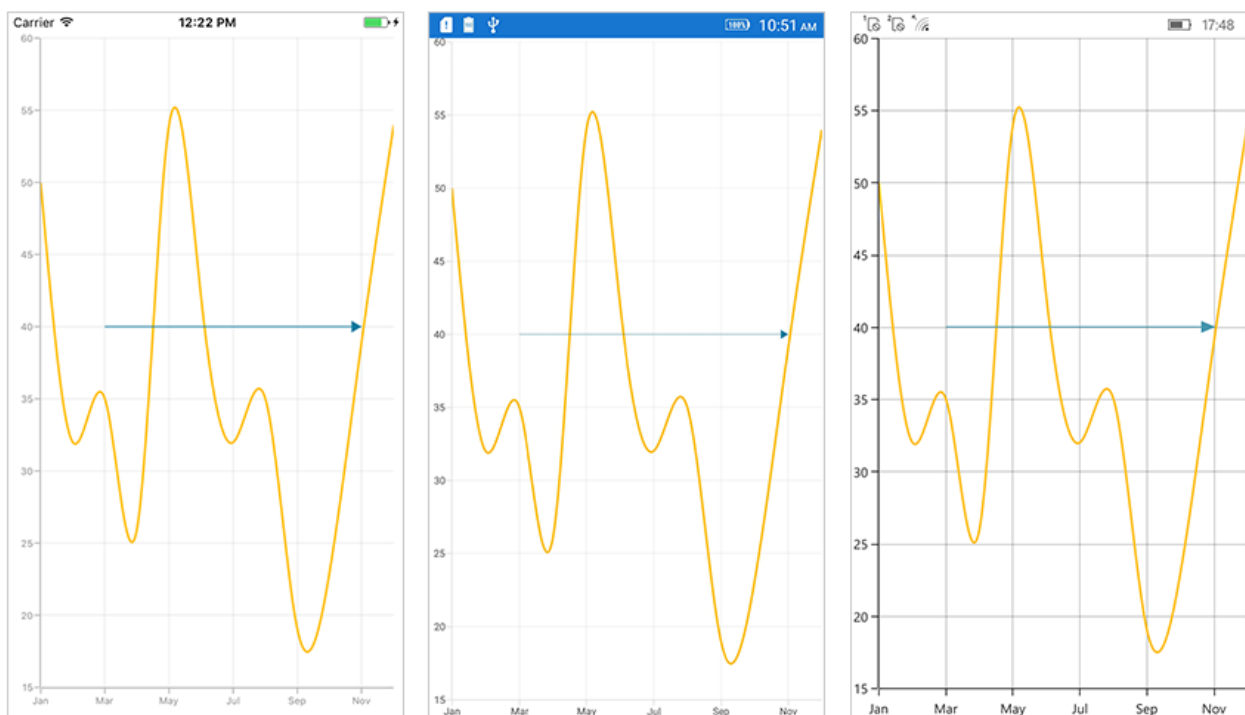
```

## C#

```

SfChart chart = new SfChart();
...
LineAnnotation annotation = new LineAnnotation()
{
    X1 = 2,
    Y1 = 40,
    X2 = 10,
    Y2 = 40,
    LineCap = ChartLineCap.Arrow
};
chart.ChartAnnotations.Add(annotation);

```



### Vertical and Horizontal line annotation

The [VerticalLineAnnotation](#) and [HorizontalLineAnnotation](#) are used to draw vertical and horizontal lines in specific points over the chart area.

## XML

```

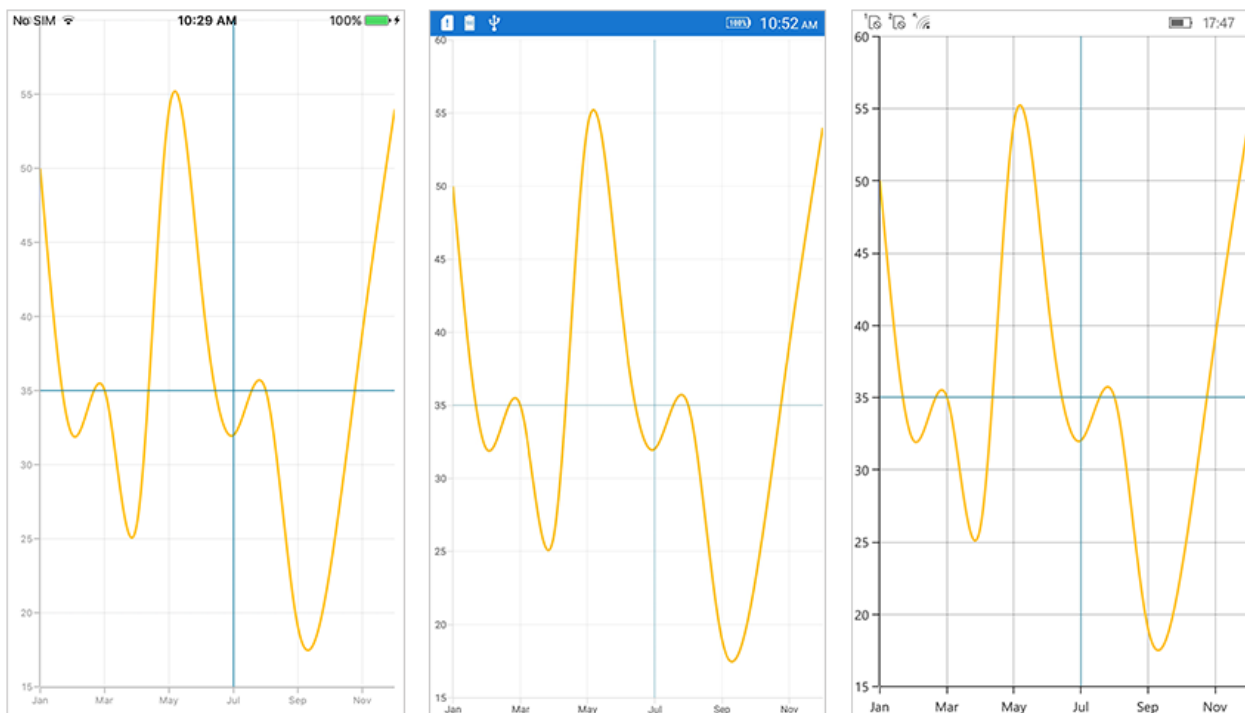
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:VerticalLineAnnotation X1="6" />

```

```
<chart:HorizontalLineAnnotation Y1="35" />
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
VerticalLineAnnotation vertical = new VerticalLineAnnotation()
{
    X1 = 6,
};
chart.ChartAnnotations.Add(vertical);
HorizontalLineAnnotation horizontal = new HorizontalLineAnnotation()
{
    Y1 = 35
};
chart.ChartAnnotations.Add(horizontal);
```



## Displaying axis label for vertical and horizontal line annotations

The [VerticalLineAnnotation](#) and [HorizontalLineAnnotation](#) display the axis labels in which the line is placed. This feature can be enabled by setting the [ShowAxisLabel](#) property to `true` as shown in the below code snippet,

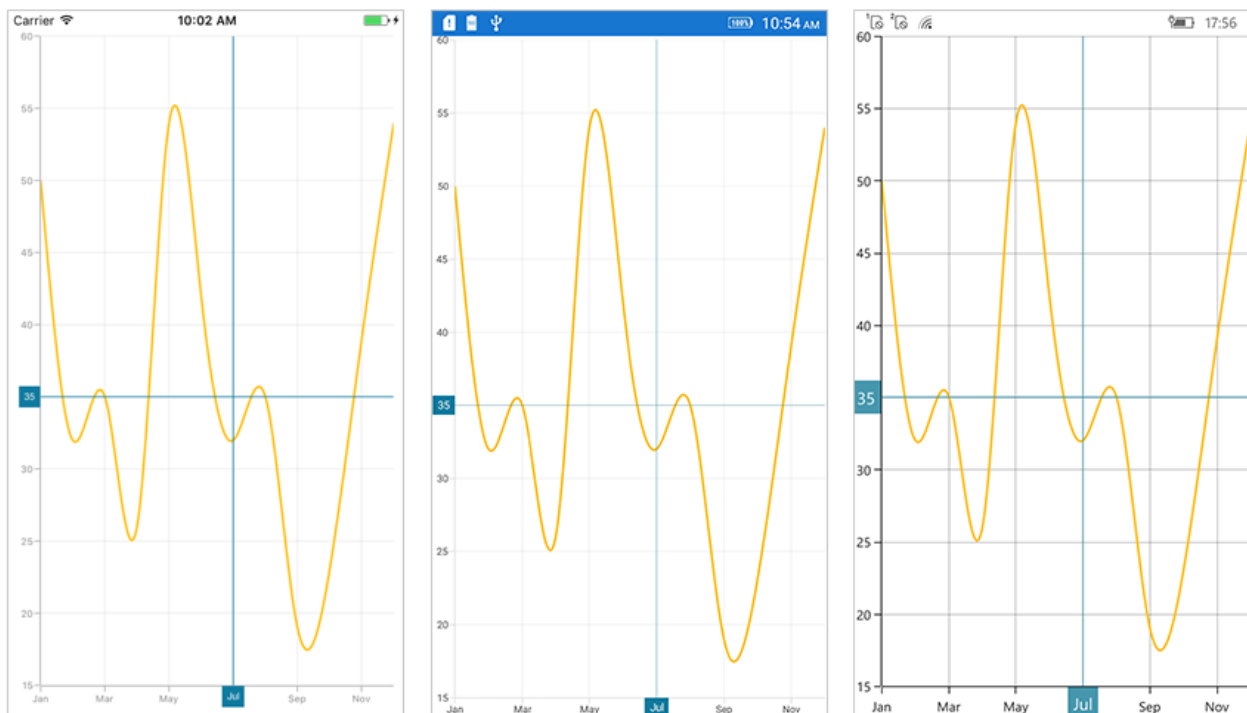
## XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:VerticalLineAnnotation X1="6" ShowAxisLabel="true"/>
```

```
<chart:HorizontalLineAnnotation Y1="35" ShowAxisLabel="true"/>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart();
...
VerticalLineAnnotation vertical = new VerticalLineAnnotation()
{
    X1 = 6,
    ShowAxisLabel = true
};
chart.ChartAnnotations.Add(vertical);
HorizontalLineAnnotation horizontal = new HorizontalLineAnnotation()
{
    Y1 = 35,
    ShowAxisLabel = true
};
chart.ChartAnnotations.Add(horizontal);
```



## Customizing axis label

The default appearance of the axis label also can be customized by using the [AxisLabelStyle](#) property. The following properties are used to customize the axis label:

- [TextColor](#) - Used to change the text color.
- [BackgroundColor](#) - Used to change the background color of the text.
- [Margin](#) - Used to set the margin for text.
- [BorderThickness](#) - Used to change the text border thickness.

- [BorderColor](#) - Used to change the border color of the text.
- [Font](#) - Used to change text font size, family, and weight.

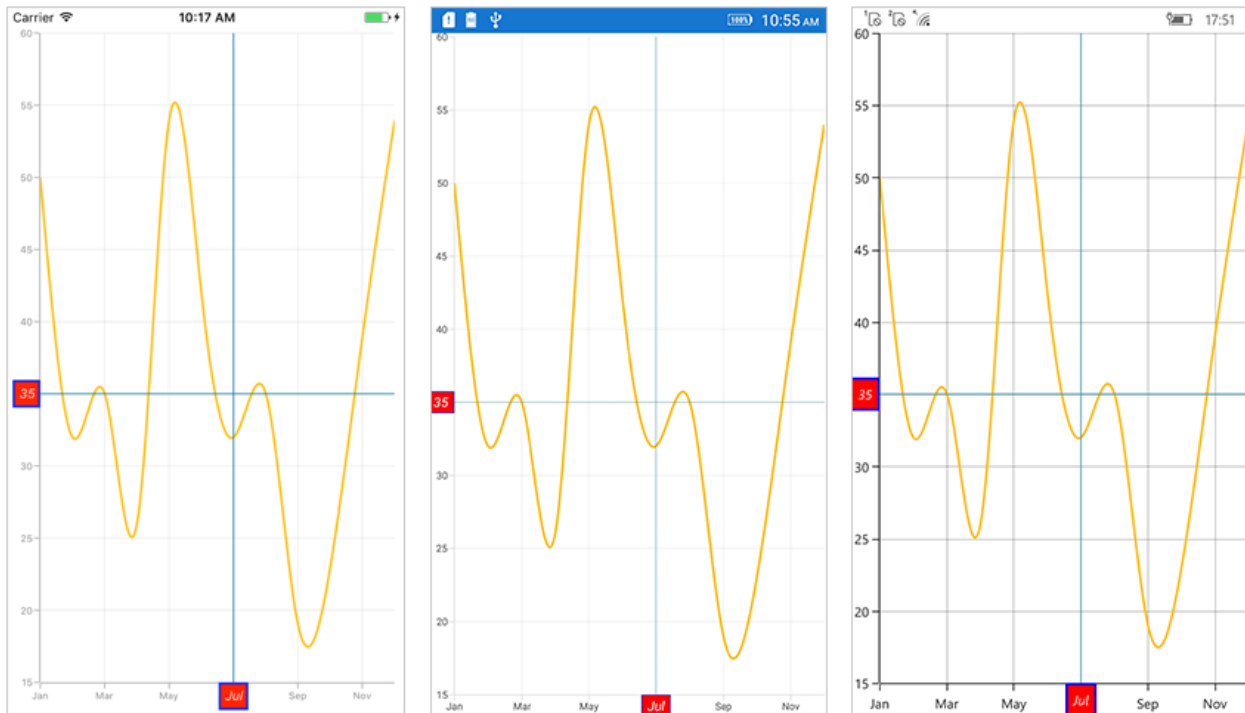
### XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:VerticalLineAnnotation X1="6" ShowAxisLabel="true">
<chart:VerticalLineAnnotation.AxisLabelStyle>
<chart:ChartLabelStyle Margin="5" Font="Italic,12" BorderColor="Blue"
BorderThickness="2" BackgroundColor="Red" TextColor="White"/>
</chart:VerticalLineAnnotation.AxisLabelStyle>
</chart:VerticalLineAnnotation>
<chart:HorizontalLineAnnotation Y1="35" ShowAxisLabel="true">
<chart:HorizontalLineAnnotation.AxisLabelStyle>
<chart:ChartLabelStyle Margin="5" Font="Italic,12" BorderColor="Blue"
BorderThickness="2" BackgroundColor="Red" TextColor="White"/>
</chart:HorizontalLineAnnotation.AxisLabelStyle>
</chart:HorizontalLineAnnotation>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
VerticalLineAnnotation vertical = new VerticalLineAnnotation()
{
    X1 = 6,
    ShowAxisLabel = true
};
vertical.AxisLabelStyle = new ChartLabelStyle()
{
    Margin = new Thickness(5),
    Font = Font.SystemFontOfSize(12, FontAttributes.Italic),
    BorderColor = Color.Blue,
    BorderThickness = new Thickness(2),
    BackgroundColor = Color.Red,
    TextColor = Color.White
};
chart.ChartAnnotations.Add(vertical);
HorizontalLineAnnotation horizontal = new HorizontalLineAnnotation()
{
    Y1 = 35,
    ShowAxisLabel = true
};
horizontal.AxisLabelStyle = new ChartLabelStyle()
{
    Margin = new Thickness(5),
    Font = Font.SystemFontOfSize(12, FontAttributes.Italic),
    BorderColor = Color.Blue,
    BorderThickness = new Thickness(2),
    BackgroundColor = Color.Red,
    TextColor = Color.White
};
```

```
chart.ChartAnnotations.Add(horizontal);
```



### Adding arrow to vertical and horizontal line annotations

To display the single headed arrow, set the [LineCap](#) property to `Arrow`. The default value of the [LineCap](#) property is `None`.

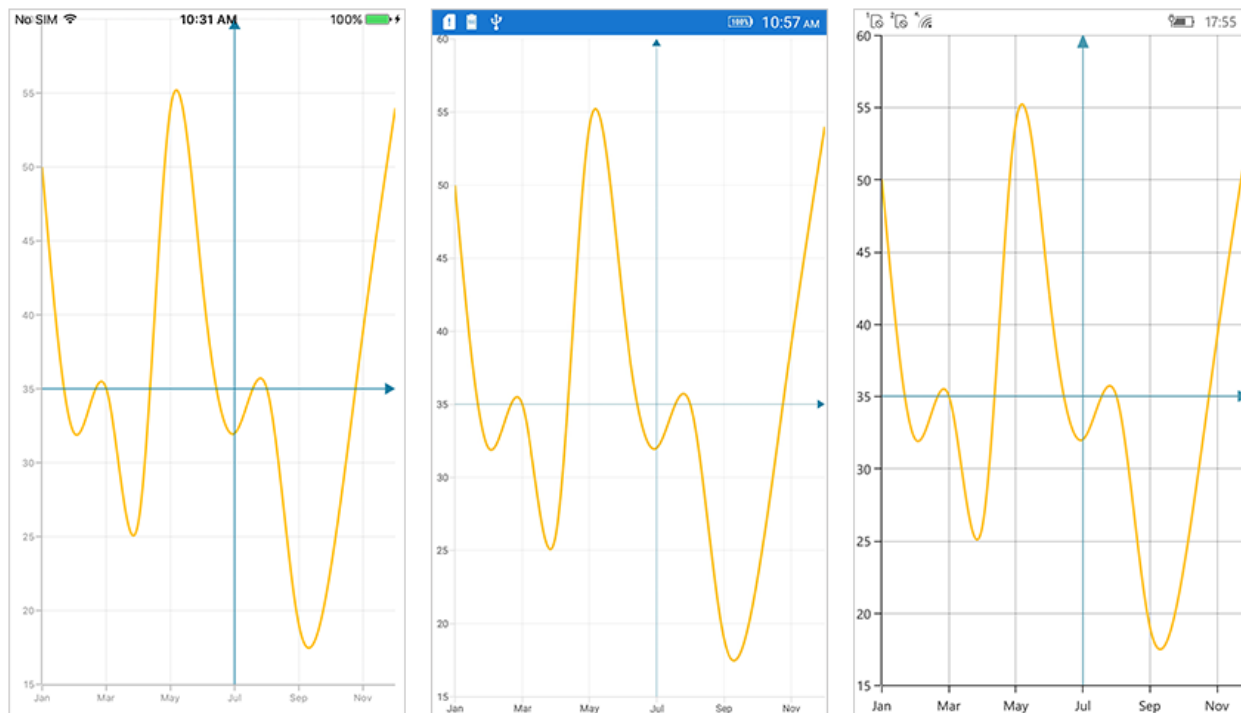
### XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart: VerticalLineAnnotation X1="6" LineCap="Arrow"/>
<chart:HorizontalLineAnnotation Y1="35" LineCap="Arrow"/>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
VerticalLineAnnotation vertical = new VerticalLineAnnotation()
{
    X1 = 6,
    LineCap = ChartLineCap.Arrow
};
chart.ChartAnnotations.Add(vertical);
HorizontalLineAnnotation horizontal = new HorizontalLineAnnotation()
{
    Y1 = 35,
    LineCap = ChartLineCap.Arrow
};
```

```
chart.ChartAnnotations.Add(horizontal);
```



#### *Adding text in shape annotation*

For all the shape annotations, the text can be displayed by using the [Text](#) property.

#### **Customizing text in shape annotation**

The [Text](#) in shape annotation also can be customized by using the [LabelStyle](#) property. The following properties are used to customize the text:

- [TextColor](#) - Used to change the text color.
- [BackgroundColor](#) - Used to change the background color of the text.
- [Margin](#) - Used to set the margin for text.
- [BorderThickness](#) - Used to change the text border thickness.
- [BorderColor](#) - Used to change the text border color.
- [Font](#) - Used to change text font size, family, and weight.
- [HorizontalTextAlignment](#) - Used to align the text horizontally.
- [VerticalTextAlignment](#) - Used to align the text vertically.

#### **XML**

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:EllipseAnnotation X1="2" Y1="30" X2="6" Y2="35" Text="Ellipse" >
<chart:EllipseAnnotation.LabelStyle>
<chart:ChartAnnotationLabelStyle Margin="5" Font="Italic,16"
BorderColor="Red" BorderThickness="2" BackgroundColor="Blue"
TextColor="White"/>
</chart:EllipseAnnotation.LabelStyle>
```

```

</chart:EllipseAnnotation>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>

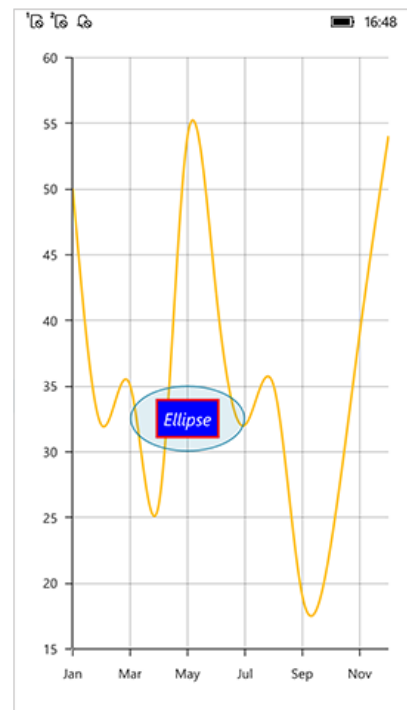
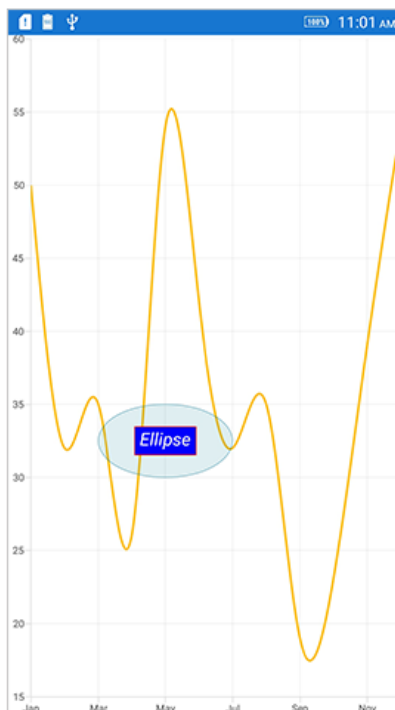
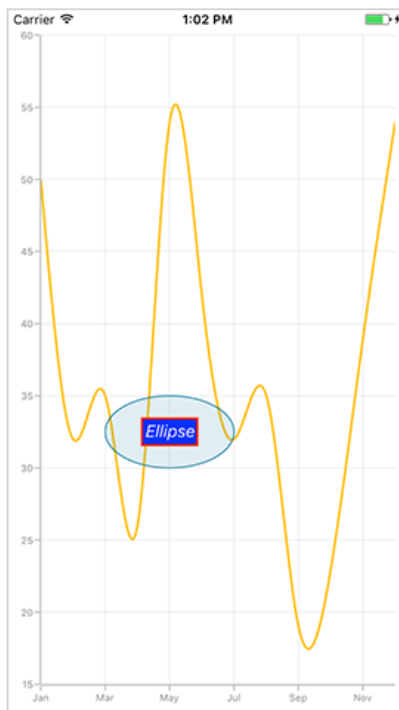
```

## C#

```

SfChart chart = new SfChart();
...
EllipseAnnotation annotation = new EllipseAnnotation()
{
    X1 = 2,
    Y1 = 30,
    X2 = 6,
    Y2 = 35,
    Text = "Ellipse"
};
annotation.LabelStyle = new ChartAnnotationLabelStyle()
{
    Margin = new Thickness(5),
    Font = Font.SystemFontOfSize(16, FontAttributes.Italic),
    BorderColor = Color.Red,
    BorderThickness = new Thickness(2),
    BackgroundColor = Color.Blue,
    TextColor = Color.White,
};
chart.ChartAnnotations.Add(annotation);

```





### View annotation

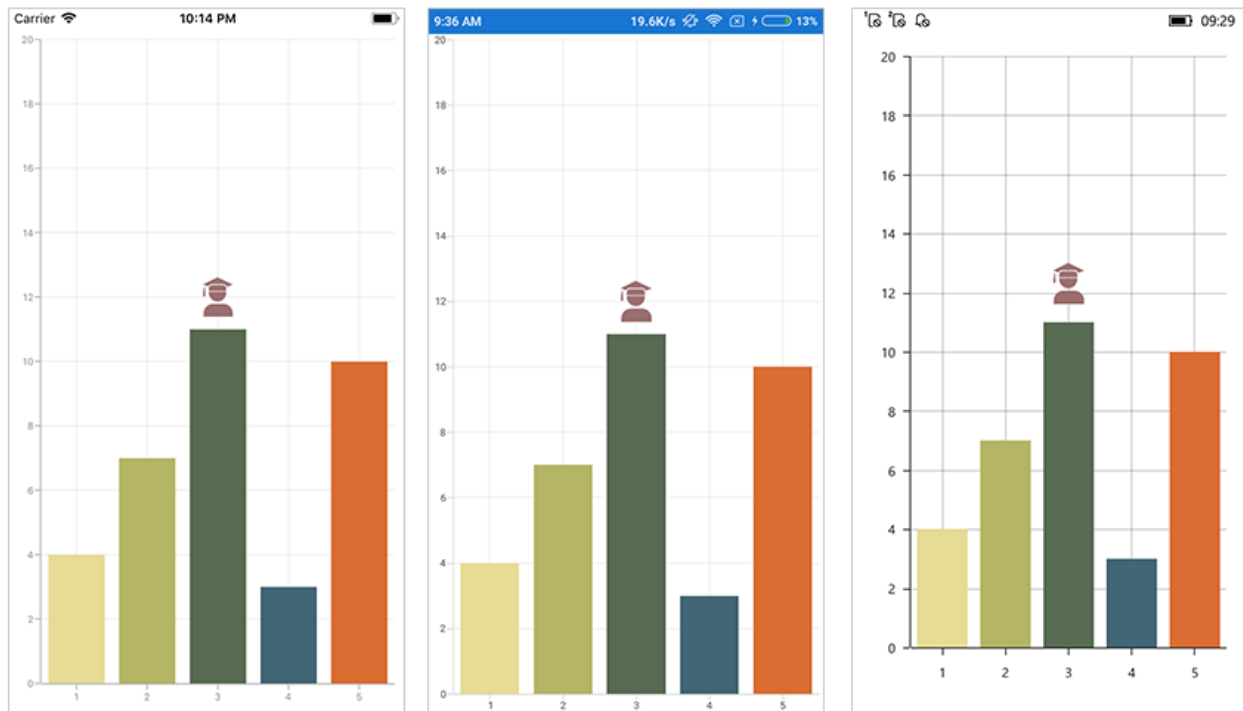
The [ViewAnnotation](#) allows you to add annotations in the form of own custom view with the help of [View](#) property at the specific area of interest in the chart area. The [ViewAnnotation](#) also can be aligned by using the [VerticalAlignment](#) and [HorizontalAlignment](#) properties.

### XML

```
<chart:SfChart>
...
<chart:SfChart.ChartAnnotations>
<chart:ViewAnnotation X1="3" Y1="12" VerticalAlignment="Start" >
<chart:ViewAnnotation.View>
<StackLayout>
<Image Source="Graduate.png" WidthRequest="70" HeightRequest="70" />
</StackLayout>
</chart:ViewAnnotation.View>
</chart:ViewAnnotation>
</chart:SfChart.ChartAnnotations>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
ViewAnnotation annotation = new ViewAnnotation()
{
    X1 = 3,
    Y1 = 12,
    VerticalAlignment = ChartAnnotationAlignment.Start
};
StackLayout layout = new StackLayout();
Image image = new Image();
image.Source = "Graduate.png";
image.HeightRequest = 70;
image.WidthRequest = 70;
layout.Children.Add(image);
annotation.View = layout;
chart.ChartAnnotations.Add(annotation);
```



## Event

### AnnotationClicked

The [AnnotationClicked](#) event is triggered when the user has clicked the annotation. The argument contains the following information.

- [Annotation](#) – used to get the instance of annotation which is clicked.
- [X](#) – used to get the x position of touch point on annotation.
- [Y](#) – used to get the y position of touch point on annotation..

## Get the touch position in annotation

Following are the override methods that are available in annotation to send the information about touch interactions.

- [OnTouchDown](#) – occurs when touch down inside the annotation.
- [OnTouchMove](#) – occurs while moving the finger or mouse inside the annotation.
- [OnTouchUp](#) – occurs when touch up inside the annotation.

## C#

```
public class TextAnnotationExt : TextAnnotation
{
    protected override void OnTouchDown(float pointX, float pointY)
    {
        base.OnTouchDown(pointX, pointY);
    }
    protected override void OnTouchMove(float pointX, float pointY)
    {
        base.OnTouchMove(pointX, pointY);
    }
}
```

```
protected override void OnTouchUp(float pointX, float pointY)
{
    base.OnTouchUp(pointX, pointY);
}
```

## Technical Indicators

The different types of technical indicators available in chart are follows:

- [AverageTrueIndicator](#)
- [SimpleMovingAverageIndicator](#)
- [RSITechnicalIndicator](#)
- [AccumulationDistributionIndicator](#)
- [MomentumIndicator](#)
- [StochasticIndicator](#)
- [ExponentialMovingAverageIndicator](#)
- [TriangularMovingAverageIndicator](#)
- [BollingerBandIndicator](#)
- [MACDIndicator](#)

Adding technical indicators to chart

The following section illustrates how to add technical indicators to chart.

### Initializing indicator

Create an instance for any technical indicator, and add it to the [TechnicalIndicators](#) collection.

Here, for an instance, the [AccumulationDistributionIndicator](#) is added.

### XML

```
<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:AccumulationDistributionIndicator />
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
...
AccumulationDistributionIndicator indicator= new
AccumulationDistributionIndicator();
chart.TechnicalIndicators.Add(indicator);
```

### Binding data

Set the [ItemsSource](#) and binding paths ([Open](#), [High](#), [Low](#), [Close](#), [XBindingPath](#) and [Trigger](#)) to fetch the values from model.

### XML

```

<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:AccumulationDistributionIndicator ItemsSource="{Binding
TechnicalIndicatorData}" XBindingPath="XValue" Open="Open" High="High"
Low="Low" Close="Close" Volume="Volume"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>

```

**C#**

```

SfChart chart = new SfChart()
{
...
TechnicalIndicators =
{
new AccumulationDistributionIndicator()
{
ItemsSource = viewModel.TechnicalIndicatorData,
XBindingPath = "XValue",
Open = "Open",
High = "High",
Low = "Low",
Close = "Close"
}
}
};

```

**Binding the items source of chart series**

By setting [Name](#) property of [FinancialSeriesBase](#) to the [SeriesName](#) property of [FinancialTechnicalIndicator](#) you can bind the items source of chart series to technical indicators, including x and y axis.

**XML**

```

<chart:SfChart>
...
<chart:SfChart.Series>
<chart:HiLoOpenCloseSeries Name="OHLC" ItemsSource="{Binding
TechnicalIndicatorData}" XBindingPath="XValue" Open="Open" High="High"
Low="Low" Close="Close"/>
</chart:SfChart.Series>
<chart:SfChart.TechnicalIndicators>
<chart:AccumulationDistributionIndicator SeriesName="OHLC"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>

```

**C#**

```

SfChart chart = new SfChart()
{
...
Series =
{

```

```

new HiLoOpenCloseSeries()
{
    ItemsSource = viewModel.TechnicalIndicatorData,
    XBindingPath = "XValue",
    Open = "Open",
    High = "High",
    Low = "Low",
    Close = "Close",
    Name = "OHLC"
},
TechnicalIndicators =
{
    new AccumulationDistributionIndicator()
    {
        SeriesName = "OHLC"
    }
}
};

```

Technical indicators have the below properties as common;

- [Period](#) - used to indicates the moving average period.
- [SignalLineColor](#) - used to defines the color for the respective indicator line.
- [StrokeWidth](#) - used to change the stroke width of the indicator.

### Adding axis

The [XAxis](#) and [YAxis](#) properties of technical indicators are used to set the x and y-axes.

You can define the axis using the following code example.

### XML

```

<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:AccumulationDistributionIndicator SeriesName="OHLC">
<chart:AccumulationDistributionIndicator.XAxis>
<chart:NumericalAxis/>
</chart:AccumulationDistributionIndicator.XAxis>
</chart:AccumulationDistributionIndicator>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>

```

### C#

```

SfChart chart = new SfChart()
{
    ...
    TechnicalIndicators =
    {
        new AccumulationDistributionIndicator()
        {
            SeriesName = "OHLC",

```

```
XAxis = new NumericalAxis()
}
}
};
```

## Animation

[SfChart](#) provides animation support for technical indicators. Technical indicators will be animated whenever the ItemsSource changes. Animation can be enabled by setting the [EnableAnimation](#) property to true. You can also control the duration of the animation using the [AnimationDuration](#) property.

## XML

```
<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:AccumulationDistributionIndicator SeriesName="OHLC"
EnableAnimation="True" AnimationDuration="0.8"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart()
{
...
TechnicalIndicators =
{
new AccumulationDistributionIndicator()
{
SeriesName = "OHLC",
EnableAnimation = true,
AnimationDuration = 0.8
}
}
};
```

## Indicator visibility

The [IsVisible](#) property of [FinancialTechnicalIndicator](#) is used to controls the visibility of the indicator.

## XML

```
<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:AccumulationDistributionIndicator SeriesName="OHLC"
IsVisible="False"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart()
{
...
};
```

```
TechnicalIndicators =
{
    new AccumulationDistributionIndicator()
    {
        SeriesName = "OHLC",
        IsVisible = false
    }
}
};
```

### Average true range indicator

ATR indicator is a technical analysis volatility indicator. This indicator does not provide an indication of price trend; simply the degree of price volatility. The average true range is an N-day smoothed moving average (SMMA) of the true range values.

You can define the [AverageTrueIndicator](#) using the following code example.

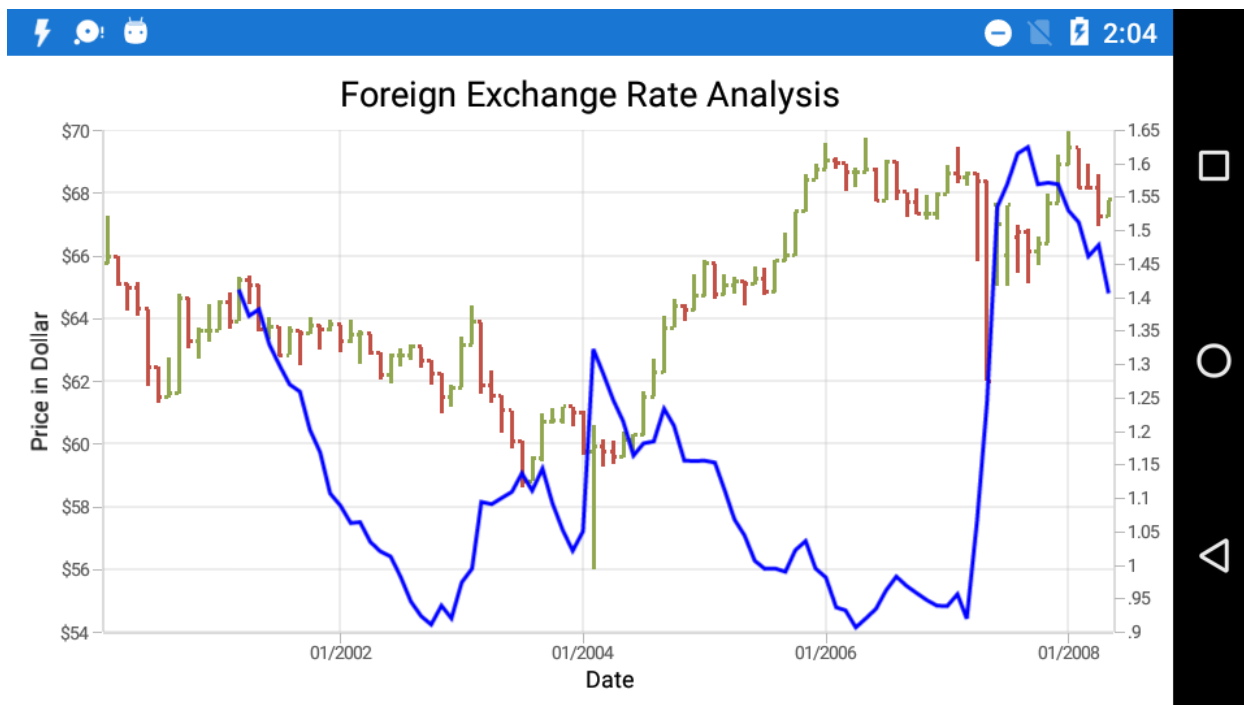
### XML

```
<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:AverageTrueIndicator ItemsSource="{Binding TechnicalIndicatorData}"
Period="14" SignalLineColor="Blue" XBindingPath="XValue" High="High"
Low="Low" Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

### C#

```
SfChart chart = new SfChart()
{
    ...
    TechnicalIndicators =
    {
        new AverageTrueIndicator()
        {
            Period = 14,
            SignalLineColor = Color.Blue,
            ItemsSource = viewModel.TechnicalIndicatorData,
            XBindingPath = "XValue",
            Open = "Open",
            High = "High",
            Low = "Low",
            Close = "Close",
        }
    }
};
```

The following screenshot illustrates an ATR indicator.



### Simple moving average indicator

A SMA indicator is a simple, arithmetic moving average that is calculated by adding the closing price for number of time periods and dividing the total value by the number of time periods.

The following code example demonstrates the usage of [SimpleMovingAverageIndicator](#).

#### XML

```
<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:SimpleMovingAverageIndicator ItemsSource="{Binding
TechnicalIndicatorData}" Period="14" SignalLineColor="Blue"
XBindingPath="XValue" High="High" Low="Low" Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart()
{
...
TechnicalIndicators =
{
new SimpleMovingAverageIndicator()
{
Period = 14,
SignalLineColor = Color.Blue,
ItemsSource = viewModel.TechnicalIndicatorData,
XBindingPath = "XValue",
Open = "Open",
High = "High",
Low = "Low",
```

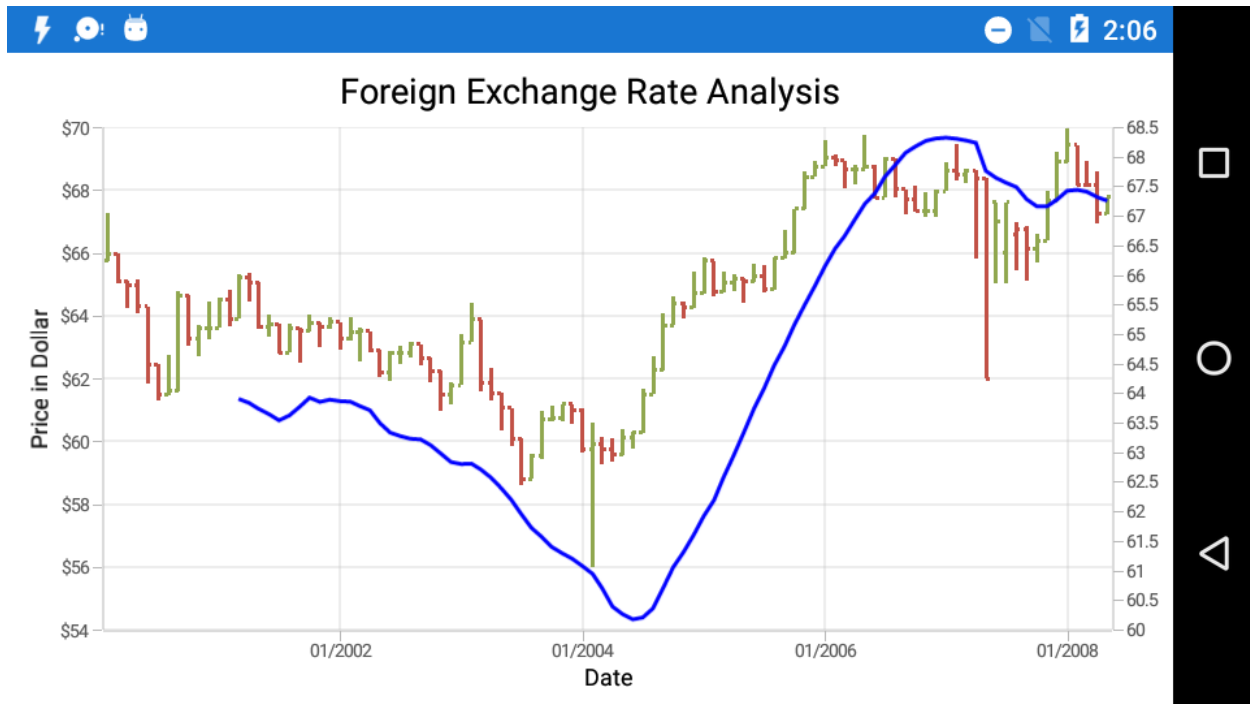


```

Close = "Close",
}
}
};

```

The following screenshot illustrates an SMA indicator.



### Relative strength index indicator

The RSI indicator has additional two lines other than signal line; they indicate the overbought and oversold region.

The [UpperLineColor](#) property is used to define the color for the line that indicates overbought region, and the [LowerLineColor](#) property is used to define the color for the line that indicates oversold region.

To define the [RSITechnicalIndicator](#), use the following code example.

#### XML

```

<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:RSIIndicator ItemsSource="{Binding TechnicalIndicatorData}"
Period="14" SignalLineColor="Blue" UpperLineColor="Teal"
LowerLineColor="Red" XBindingPath="XValue" High="High" Low="Low" Open="Open"
Close="Close"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>

```

#### C#

```

SfChart chart = new SfChart()
{

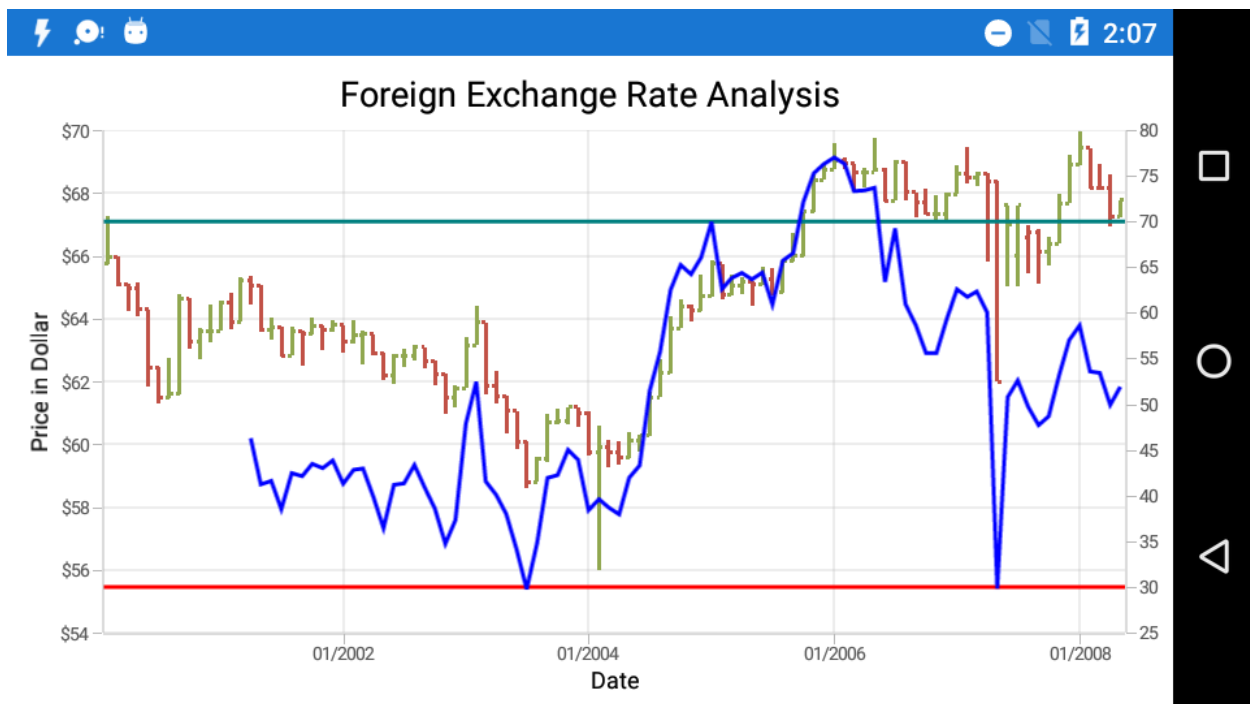
```

```

...
TechnicalIndicators =
{
    new RSIIndicator()
    {
        Period = 14,
        SignalLineColor = Color.Blue,
        UpperLineColor = Color.Teal,
        LowerLineColor = Color.Red,
        ItemsSource = viewModel.TechnicalIndicatorData,
        XBindingPath = "XValue",
        Open = "Open",
        High = "High",
        Low = "Low",
        Close = "Close",
    }
}
};

```

The following screenshot illustrates an RSI technical indicator.



#### Accumulation distribution indicator

Accumulation distribution indicator is a volume-based indicator designed to measure the accumulative flow of money into and out of a security. It requires [Volume](#) property additionally with the data source to calculate the signal line.

The following code example helps you to add [AccumulationDistributionIndicator](#).

#### XML

```

<chart:SfChart>
...

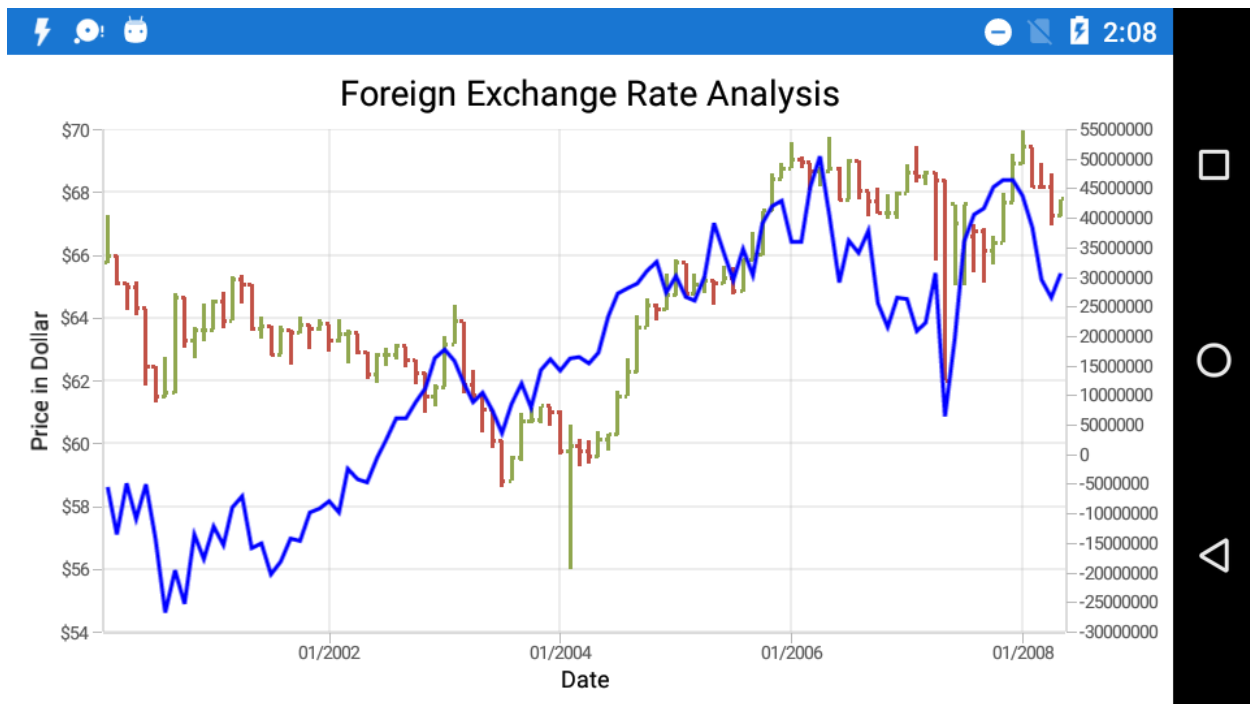
```

```
<chart:SfChart.TechnicalIndicators>
<chart:AccumulationDistributionIndicator ItemsSource="{Binding
TechnicalIndicatorData}" SignalLineColor="Blue" XBindingPath="XValue"
Open="Open" High="High" Low="Low" Close="Close" Volume="Volume"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart()
{
    ...
    TechnicalIndicators =
    {
        new AccumulationDistributionIndicator()
        {
            SignalLineColor = Color.Blue,
            ItemsSource = viewModel.TechnicalIndicatorData,
            XBindingPath = "XValue",
            Open = "Open",
            High = "High",
            Low = "Low",
            Close = "Close",
            Volume = "Volume"
        }
    }
};
```

The following screenshot illustrates an accumulation distribution indicator.



### Momentum indicator

This indicator is having two lines, momentum line and center line. The [SignalLineColor](#) property and [UpperLineColor](#) property is used to define the color for the momentum and center line respectively.

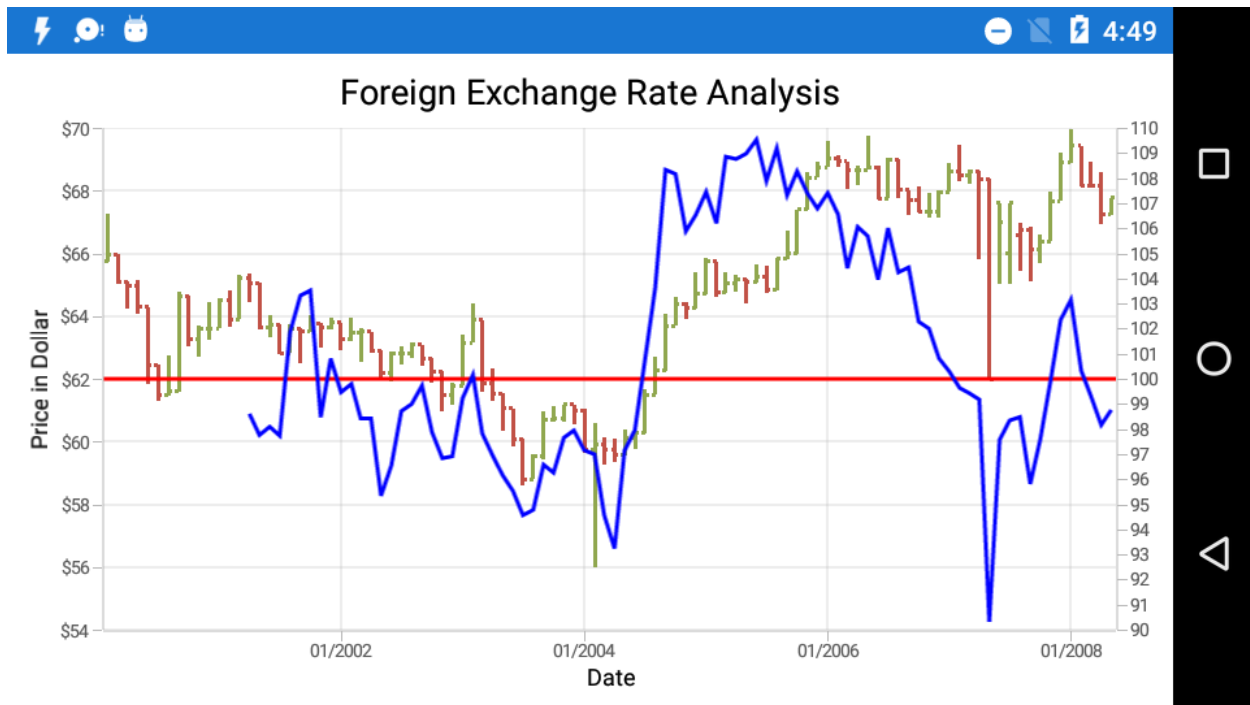
You can define [MomentumIndicator](#) using the following code example.

#### XML

```
<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:MomentumIndicator ItemsSource="{Binding TechnicalIndicatorData}"
Period="14" SignalLineColor="Blue" UpperLineColor="Red"
XBindingPath="XValue" High="High" Low="Low" Open="Open" Close="Close">
</chart:MomentumIndicator>
</chart:SfChart.TechnicalIndicators>
...
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart()
{
...
TechnicalIndicators =
{
new MomentumIndicator()
{
Period = 14,
SignalLineColor = Color.Blue,
UpperLineColor = Color.Red,
ItemsSource = viewModel.TechnicalIndicatorData,
XBindingPath = "XValue",
Open = "Open",
High = "High",
Low = "Low",
Close = "Close",
}
}
};
```



### Stochastic indicator

This indicator is used to measure the range and momentum of price movements. It contains [KPeriod](#) and [DPeriod](#) property defining the 'K' percentage and 'D' percentage respectively. No signal line in this indicator.

The [UpperLineColor](#), [LowerLineColor](#) and [PeriodLineColor](#) property are used to define the color for the Stochastic indicator lines.

You can define [StochasticIndicator](#) using the following code example.

### XML

```
<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:StochasticIndicator ItemsSource="{Binding TechnicalIndicatorData}"
Period="14" SignalLineColor="Yellow" UpperLineColor="Red"
LowerLineColor="Teal" PeriodLineColor="DarkBlue" KPeriod="8" DPeriod="5"
XBindingPath="XValue" High="High" Low="Low" Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

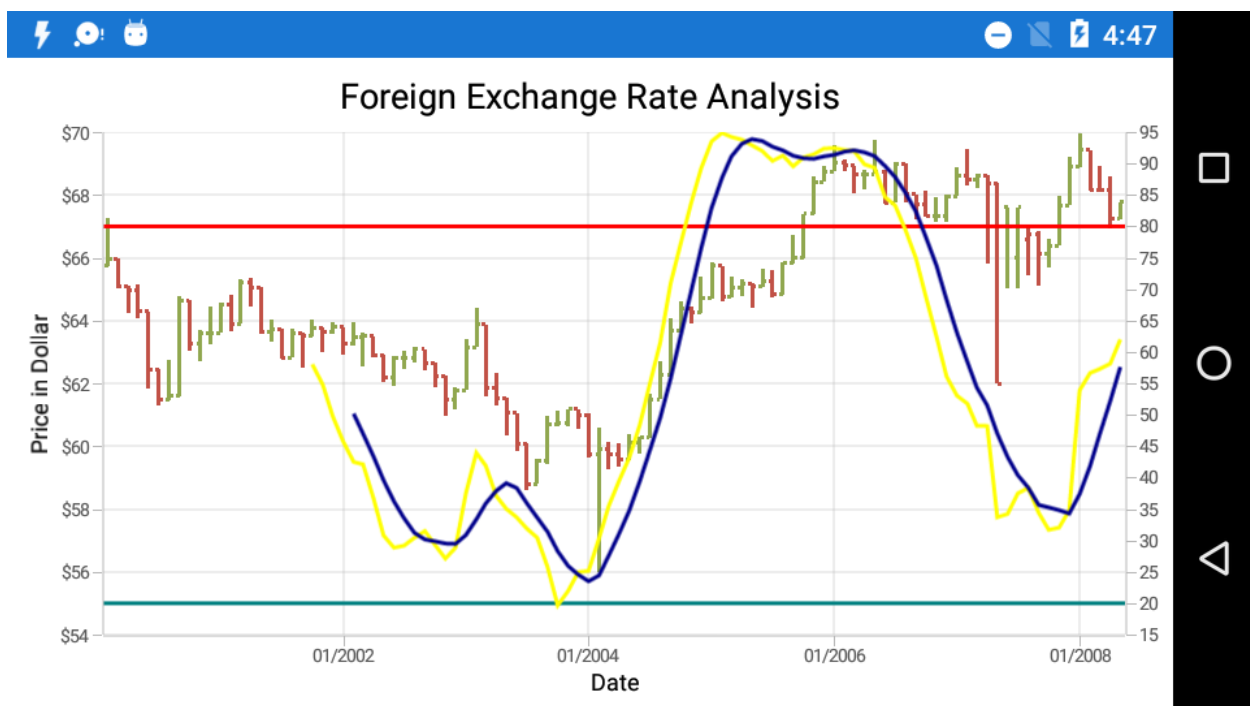
### C#

```
SfChart chart = new SfChart()
{
...
TechnicalIndicators =
{
new StochasticIndicator()
{
Period = 14,
SignalLineColor = Color.Yellow,
```

```

UpperLineColor = Color.Red,
LowerLineColor = Color.Teal,
PeriodLineColor = Color.DarkBlue,
KPeriod = 8,
DPeriod = 5,
ItemsSource = viewModel.TechnicalIndicatorData,
XBindingPath = "XValue",
Open = "Open",
High = "High",
Low = "Low",
Close = "Close",
}
}
};

```



Exponential moving average indicator

The [ExponentialMovingAverageIndicator](#) is similar to [SimpleMovingAverageIndicator](#) and this can be defined using the following code examples.

#### XML

```

<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:ExponentialMovingAverageIndicator ItemsSource="{Binding
TechnicalIndicatorData}" Period="14" SignalLineColor="Blue"
XBindingPath="XValue" High="High" Low="Low" Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>

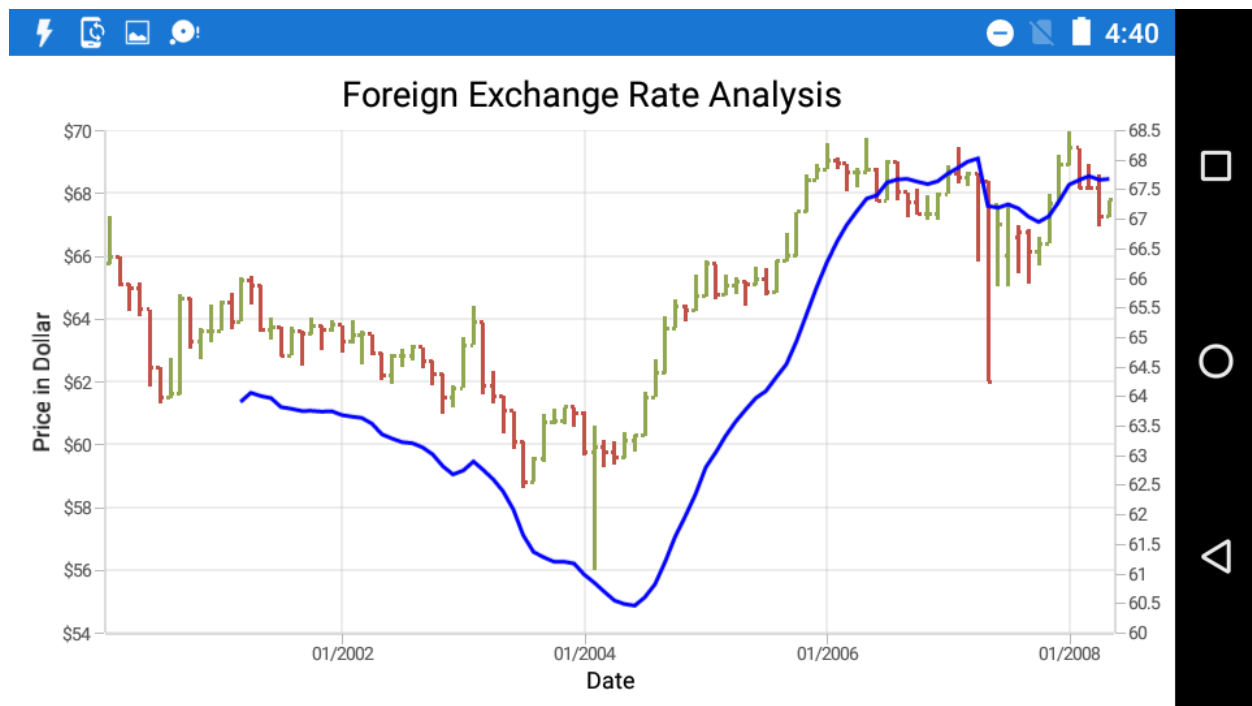
```

#### C#

```

SfChart chart = new SfChart()
{
    ...
    TechnicalIndicators =
    {
        new ExponentialMovingAverageIndicator()
        {
            Period = 14,
            SignalLineColor = Color.Blue,
            ItemsSource = viewModel.TechnicalIndicatorData,
            XBindingPath = "XValue",
            Open = "Open",
            High = "High",
            Low = "Low",
            Close = "Close",
        }
    }
};

```



### Triangular moving average indicator

A Triangular moving average is simply a double-smoothed simple moving average of data calculated over a period of time where the middle portion of the data has more weight.

The [TriangularMovingAverageIndicator](#) can be defined as in the following code example.

### XML

```

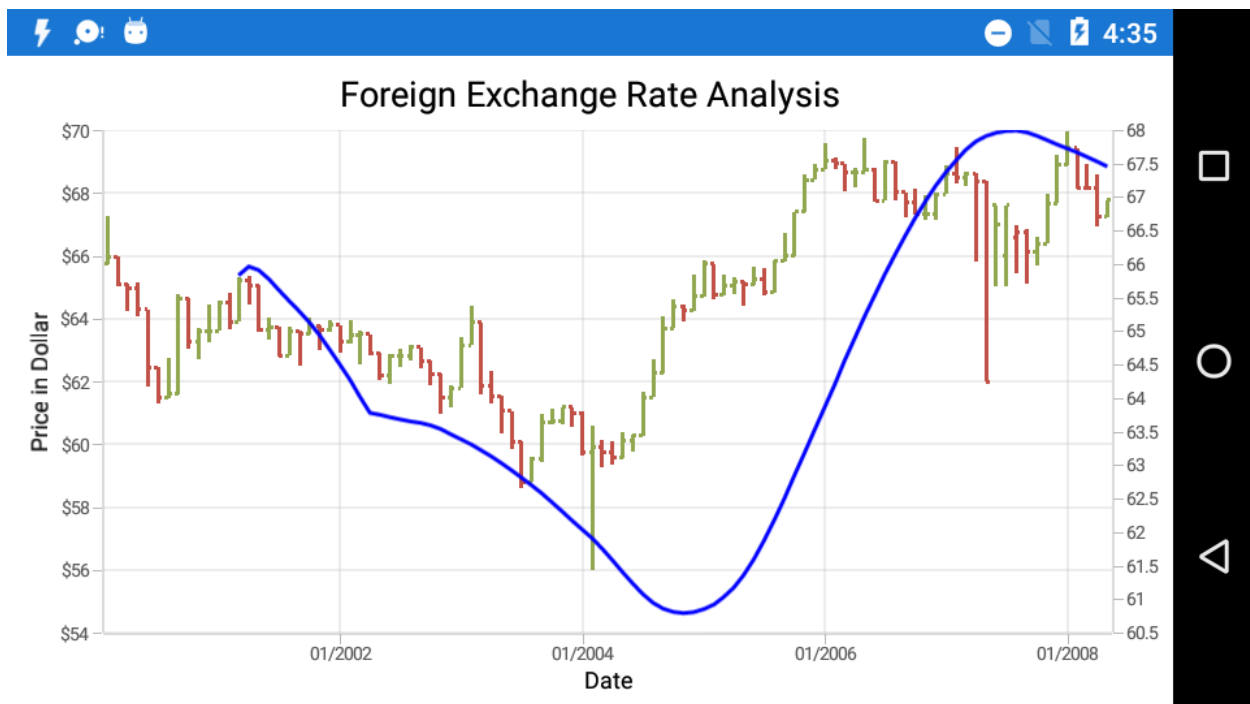
<chart:SfChart>
    ...
    <chart:SfChart.TechnicalIndicators>

```

```
<chart:TriangularMovingAverageIndicator ItemsSource="{Binding
TechnicalIndicatorData}" Period="14" SignalLineColor="Blue"
XBindingPath="XValue" High="High" Low="Low" Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

## C#

```
SfChart chart = new SfChart()
{
    ...
    TechnicalIndicators =
    {
        new TriangularMovingAverageIndicator()
        {
            Period = 14,
            SignalLineColor = Color.Blue,
            ItemsSource = viewModel.TechnicalIndicatorData,
            XBindingPath = "XValue",
            Open = "Open",
            High = "High",
            Low = "Low",
            Close = "Close",
        }
    }
};
```



## Bollinger band indicator

This indicator also having [UpperLineColor](#), [LowerLineColor](#) and [SignalLineColor](#) property for defining the brushes for the indicator lines.



Also, we can specify standard deviation values for BollingerBand indicator using [StandardDeviation](#) property.

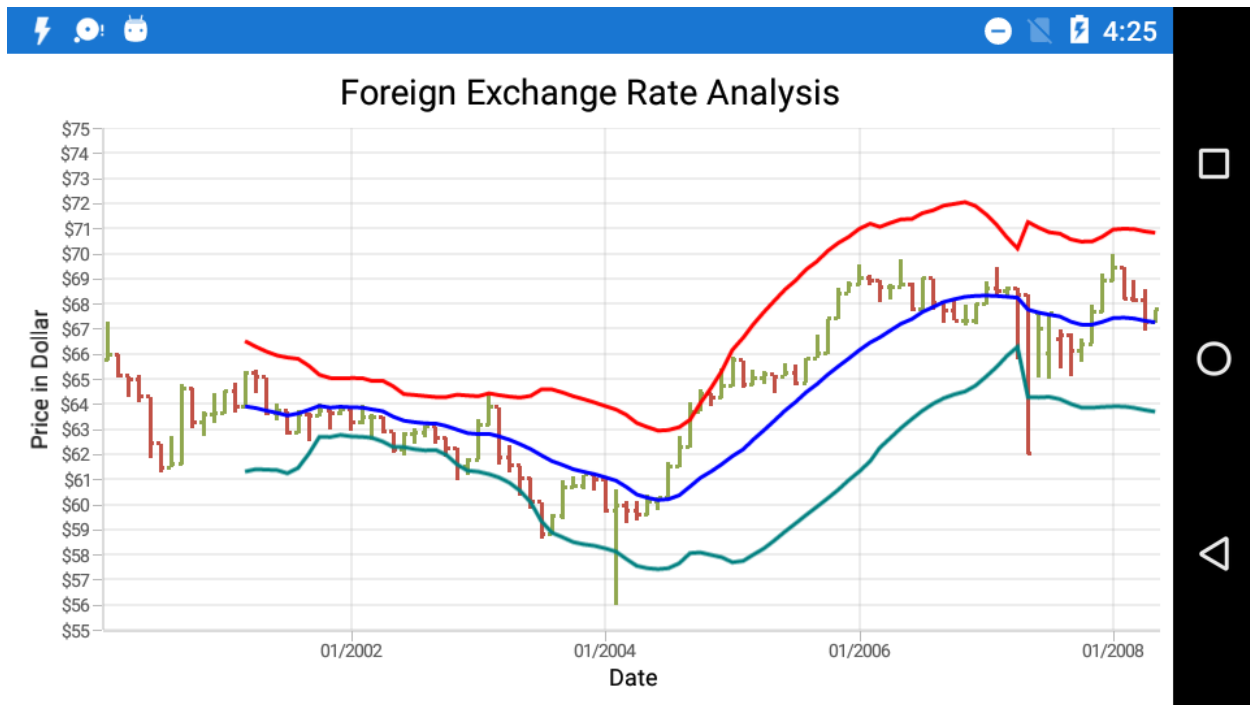
You can define the [BollingerBandIndicator](#) using the following code example.

#### **XML**

```
<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:BollingerBandIndicator ItemsSource="{Binding TechnicalIndicatorData}"
Period="14" SignalLineColor="Blue" UpperLineColor="Red"
LowerLineColor="Teal" XBindingPath="XValue" High="High" Low="Low"
Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

#### **C#**

```
SfChart chart = new SfChart()
{
...
TechnicalIndicators =
{
new BollingerBandIndicator()
{
Period = 14,
SignalLineColor = Color.Blue,
UpperLineColor = Color.Red,
LowerLineColor = Color.Teal,
ItemsSource = viewModel.TechnicalIndicatorData,
XBindingPath = "XValue",
Open = "Open",
High = "High",
Low = "Low",
Close = "Close",
}
}
};
```



### MACD indicator

This is mostly using indicator having [ShortPeriod](#) and [LongPeriod](#) for defining the motion of the indicator.

Also you can draw [Line](#), [Histogram](#) MACD or [Both](#) using the [MACDType](#) property, which defines the type of MACD to be drawn.

The [MACDLineColor](#) property is used to define the color for the MACD line and the [HistogramLineColor](#) property is used to define the color for the MACD histogram.

You can specify the MACD indicator using the following code example.

### XML

```
<chart:SfChart>
...
<chart:SfChart.TechnicalIndicators>
<chart:MACDIndicator ItemsSource="{Binding TechnicalIndicatorData}"
MACDType="Both" ShortPeriod="2" LongPeriod="10" Trigger="14"
SignalLineColor="Blue" HistogramColor="LightSkyBlue" MACDLineColor="Orange"
XBindingPath="XValue" High="High" Low="Low" Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>
</chart:SfChart>
```

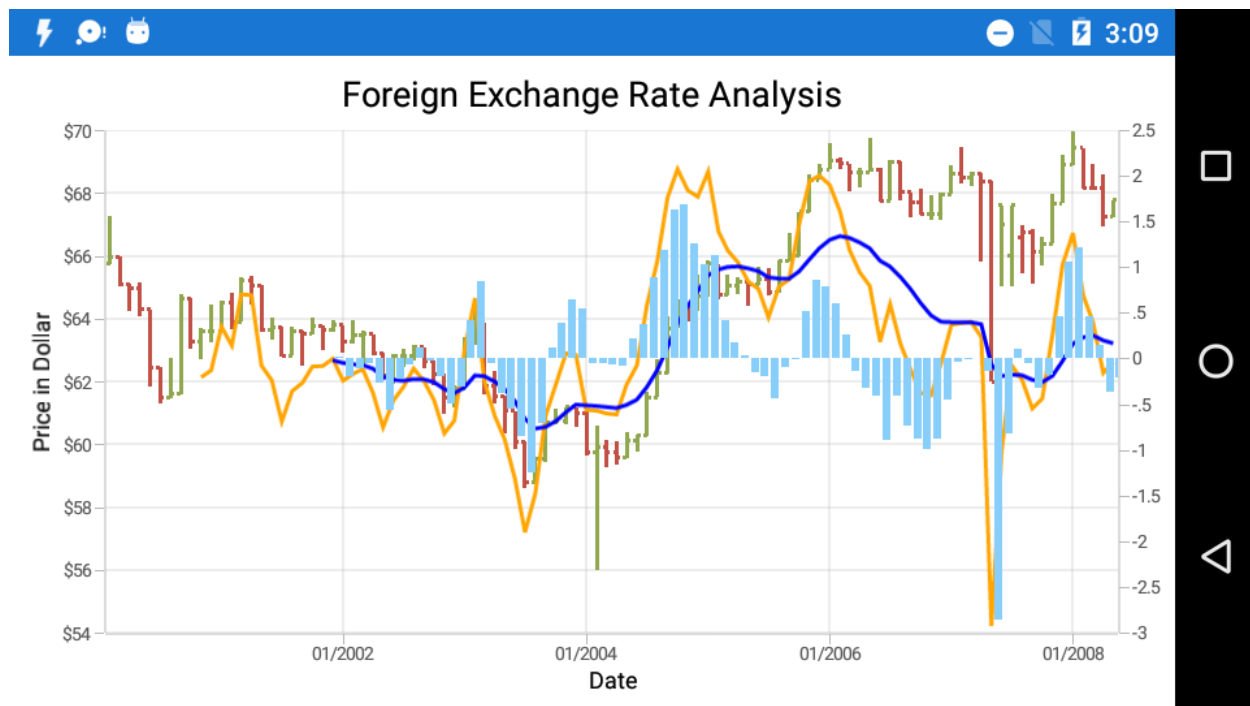
### C#

```
SfChart chart = new SfChart()
{
...
TechnicalIndicators =
{
new MACDIndicator()
{
```

```

MACDType = MACDType.Both,
ShortPeriod = 2,
LongPeriod = 10,
Trigger = 14,
SignalLineColor = Color.Blue,
HistogramColor = Color.LightSkyBlue,
MACDLineColor = Color.Orange,
ItemsSource = viewModel.TechnicalIndicatorData,
XBindingPath = "XValue",
Open = "Open",
High = "High",
Low = "Low",
Close = "Close",
}
}
};

```



## Exporting

### Export as an image

You can export the Chart as a JPG image using [SaveAsImage](#) method of [SfChart](#).

### C#

```

SfChart chart = new SfChart();
...
chart.SaveAsImage("ChartSample.jpg");

```

The exported image will be saved in the different location across the platforms.

**Android** – The image will be saved inside the Pictures directory.

**iOS** – The image will be saved inside the “Photos/Album” directory.

**Windows Phone** – The image will be saved inside the “/Pictures/ Saved Pictures” directory.

---

**Note:** In order to save the image in Android and Windows Phone, you have to enable the permission to write the file in device storage.

---

### Get the stream of Chart

'[SfChart](#)' contains the following methods to get the chart stream.

#### GetStream

The [GetStream](#) method of SfChart is used to get the chart as stream. The output stream can be passed as an input of any other components which accept the stream such as pdf, excel, word etc. This method is only applicable for Android and iOS platforms only.

#### C#

```
SfChart chart = new SfChart();  
...  
chart.GetStream();
```

#### GetStreamAsync

The [GetStreamAsync](#) method of SfChart is used to get the chart as stream asynchronously.

---

**Note:** This method will work only when the SfChart view in UI.

---

The following code snippet demonstrates the usage of this method:

#### C#

```
SfChart chart = new SfChart();  
...  
var stream = await chart.GetStreamAsync();
```

### How to

Transform axis value to pixel value and vice-versa

[SfChart](#) offers two utility methods to transform the pixel to chart point and vice-versa.

- [ValueToPoint\(ChartAxis axis, double value\)](#) - Converts the data point value to screen point.
- [PointToValue\(ChartAxis axis, Point point\)](#) - Converts the screen point to chart value.

#### C#

```
double xValue = Chart.PointToValue(Chart.PrimaryAxis, screenPoint);  
double yValue = Chart.PointToValue(Chart.SecondaryAxis, screenPoint);  
double chartPointX = Chart.ValueToPoint(Chart.PrimaryAxis, xValue);  
double chartPointY = Chart.ValueToPoint(Chart.SecondaryAxis, yValue);
```

Use the [ValueToPoint](#) and [PointToValue](#) methods, which are available in [ChartAxis](#), to convert the screen point within the rendered area of the series.

---

**Note:** You can convert the actual axis value to 0 to 1 coefficient using the [ValueToCoefficient\(double value\)](#) and [CoefficientToValue\(double value\)](#) methods of [ChartAxis](#).

---

### Get the touch position in chart

ChartBehavior provides the following override methods to get the x and y positions when touch the [Chart](#).

- [OnTouchUp](#) - Called when touch up on the chart area with respective x and y position.
- [OnTouchMove](#) - Called when touch move on the chart area with respective x and y position.
- [OnTouchDown](#) - Called when touch down on the chart area with respective x and y position.
- [DoubleTap](#) - Called when double tap on the chart area with respective x and y position.

### C#

```
public class ChartTooltipBehaviorExt : ChartTooltipBehavior
{
    protected override void OnTouchUp(float pointX, float pointY)
    {
        base.OnTouchUp(pointX, pointY);
    }
    protected override void OnTouchMove(float pointX, float pointY)
    {
        base.OnTouchMove(pointX, pointY);
    }
    protected override void OnTouchDown(float pointX, float pointY)
    {
        base.OnTouchDown(pointX, pointY);
    }
    protected override void DoubleTap(float pointX, float pointY)
    {
        base.DoubleTap(pointX, pointY);
    }
}
```

### Get the data point collection based on region

[CartesianSeries](#) provides the following methods to get a collection of data under a particular region.

- [GetDataPoints\(Rectangle rect\)](#) - Gets the collection of data that fall inside the given rectangle region.
- [GetDataPoints\(double startX, double endX, double startY, double endY\)](#) - Gets the collection of data from the given ChartAxis visible range.

### C#

```
List<object> dataPoints = Series.GetDataPoints(rectangle);
or
List<object> dataPoints = Series.GetDataPoints(startX, endX, startY, endY);
```

**Note:** You can get the visible plotting region of the series in the chart using [SeriesBounds](#) property in run time.

### Adding duplicate axis in SfChart

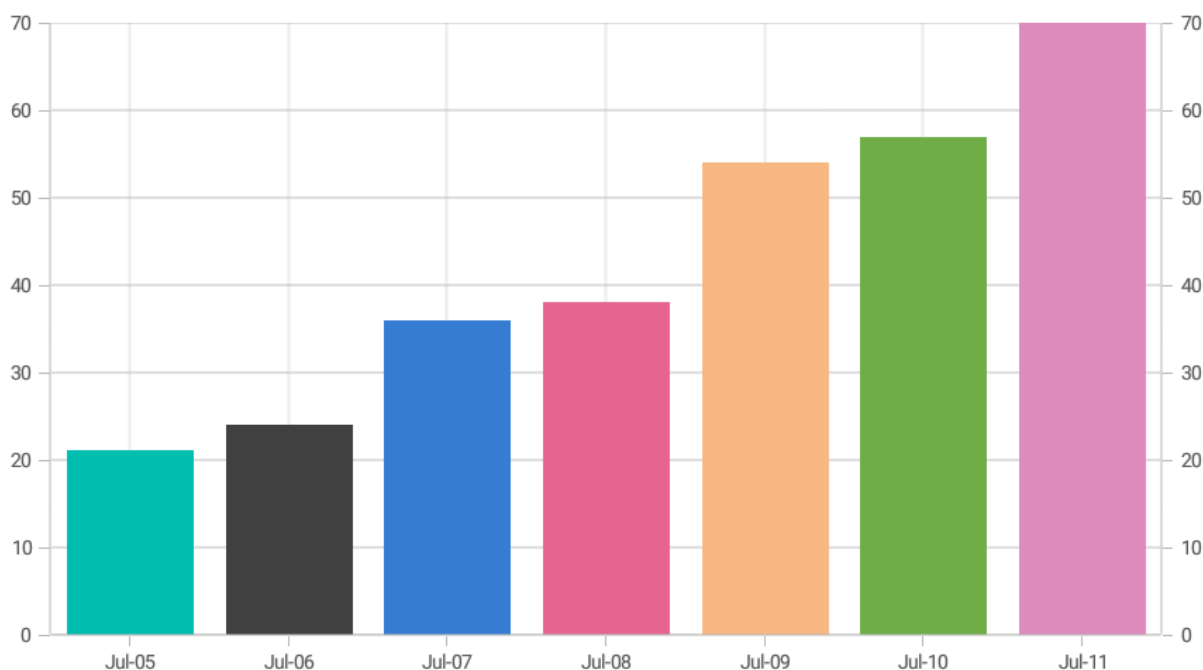
Duplicate axis can be added in the [SfChart](#) using the chart [Axes](#) collection property. The axis added in the [Axes](#) collection will be aligned to the horizontal position by default. The axis position can be changed

using the [IsVertical](#) bool property of [ChartAxis](#). When the [IsVertical](#) property is set true, the axis will be placed vertically and vice versa.

The following code sample demonstrates this.

### C#

```
SfChart chart = new SfChart();  
...  
chart.Axes.Add(new NumericalAxis()  
{  
    Minimum = 0,  
    Maximum = 70,  
    IsVertical = true,  
    OpposedPosition = true  
});
```



### Note:

- The [ChartAxis](#) added in the [Axes](#) collection will not be removed until removing it from the [Axes](#) collection.
- The [Axes](#) collection does not support the clear method.
- Same axis cannot be added more than once in [Axes](#); only distinct axis will be added to the [Axes](#) collection.

### Customize the SfChart padding

By default, padding is applied to all the sides of chart to avoid the cropping of axis labels and leaving some space between nearby views and chart. However, it can be removed or changed using the [ChartPadding](#) property of [SfChart](#).

The following code example shows how to customize the padding of chart.

#### XML

```
<chart:SfChart x:Name="Chart" ChartPadding ="5,5,5,5">
...
</chart:SfChart>
```

#### C#

```
...
SfChart chart = new SfChart()
{
    ChartPadding = new Thickness(5)
};
...
```

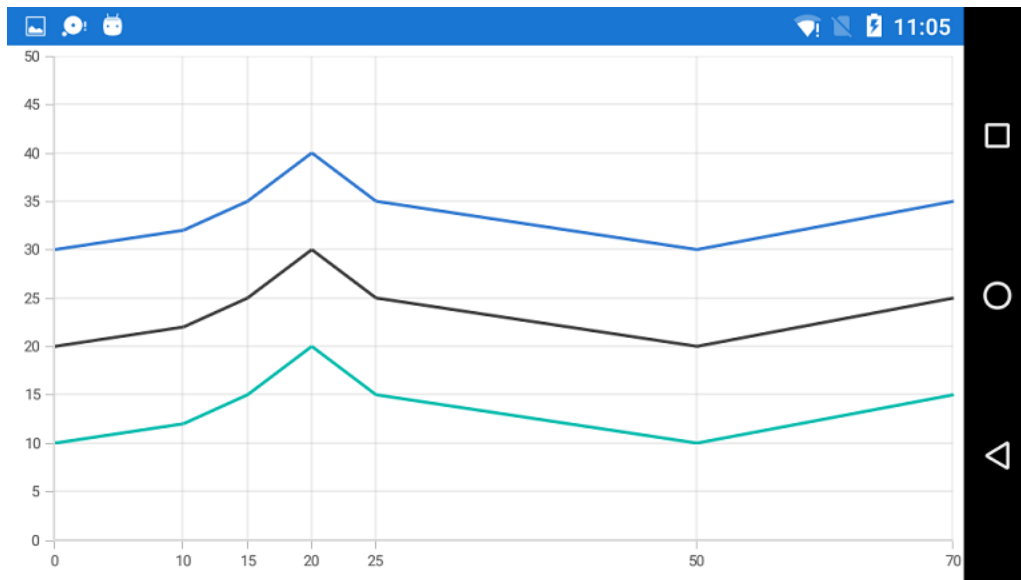
#### Adding custom labels to SfChart axis

To add a custom label to chart axis, write a class derived from `NumericalAxis` class. You need to override the [OnCreatedLabels](#) method, which will be called whenever new labels are generated, and add, remove, or modify the labels using the [VisibleLabels](#) property.

The following code sample demonstrates this.

#### C#

```
public class NumericalAxisExt : NumericalAxis
{
    protected override void OnCreateLabels()
    {
        base.OnCreateLabels();
        //Using VisibleLabels collection you can define your custom labels
        VisibleLabels.Clear();
        ViewModel viewModel = BindingContext as ViewModel;
        for (int i = 0; i < viewModel.Data.Count; i++)
        {
            var data = viewModel.Data[i];
            VisibleLabels.Add(new ChartAxisLabel(data.XValue, data.XValue.ToString()));
        }
    }
}
```

**Note:**

- This is applicable for all types of axis.
- Labels are rendered only if the label position presents within the visible range.
- The labels should be created only if users call the base of [OnCreateLabels](#).

### Localization

You can localize [SfChart](#) in all the platforms by adding a .resx file in a .NET Standard project alone. The following steps describe how to localize SfChart in a project and you can download the complete sample from this [link](#).

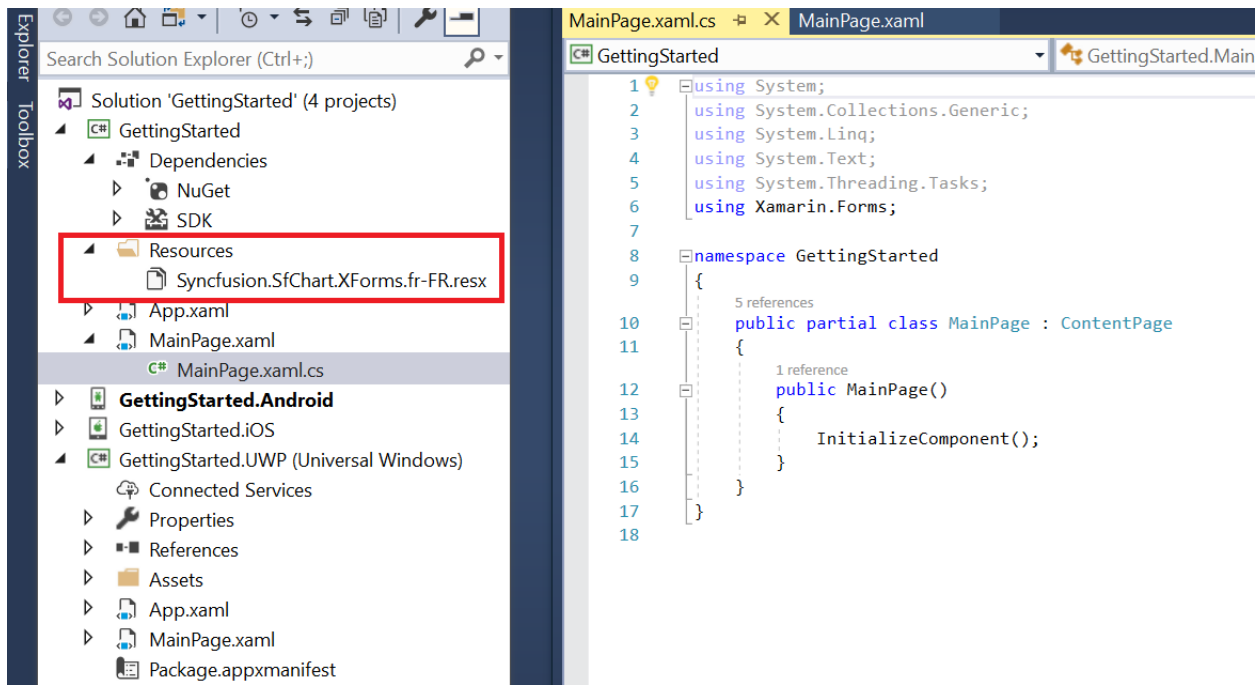
---

**Note:** Here, the resources have been already created for some cultures and shared them on [Syncfusion GitHub](#) for your convenience.

---

1. Add a new folder in the .NET Standard project named Resources.
2. Add resource files for the languages you wish to support, and set their Build Action to EmbeddedResource. The name of the resource file should be \$name of the Syncfusion component\$+\$language code\$.resx. For example, if you add a resource file for the French culture, add the Syncfusion.SfChart.XForms.fr-FR.resx file to Resources folder as illustrated in the following screenshot.





3. Provide the French values for each key in the respective .resx files. Here, “Close” and “High” are the keys, and “Fermer” and “Haute” are their respective French values.

### XML

```
<data name="Close" xml:space="preserve">
<value>Fermer</value>
</data>
<data name="High" xml:space="preserve">
<value>Haute</value>
</data>
```

4. Set the resource manager to '[ChartResourceManager.Manager](#)' as demonstrated in the following code to get the resource manager from the users. For more details, refer [Localization](#).

### C#

```
ChartResourceManager.Manager = new
ResourceManager("GettingStarted.Resources.Syncfusion.SfChart.XForms",
Application.Current.GetType().Assembly);
```

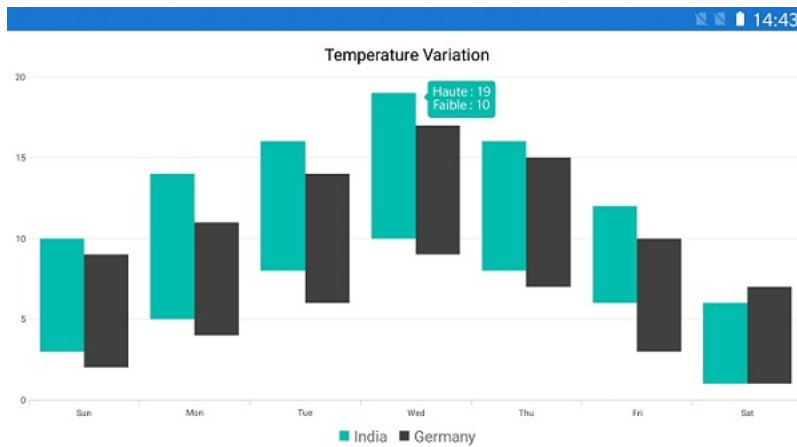
### Localize at application level

You can also localize the text at application-level regardless of the language selected on the device. The following platform-specific codes are needed to localize the text at application-level. Use the [DependencyServices](#) to set this from .NET Standard project.

### C#

```
//For Android and iOS,
```

```
Thread.CurrentThread.CurrentUICulture = new CultureInfo("fr-FR");  
//For UWP,  
CultureInfo.CurrentUICulture = new CultureInfo("fr-FR");
```



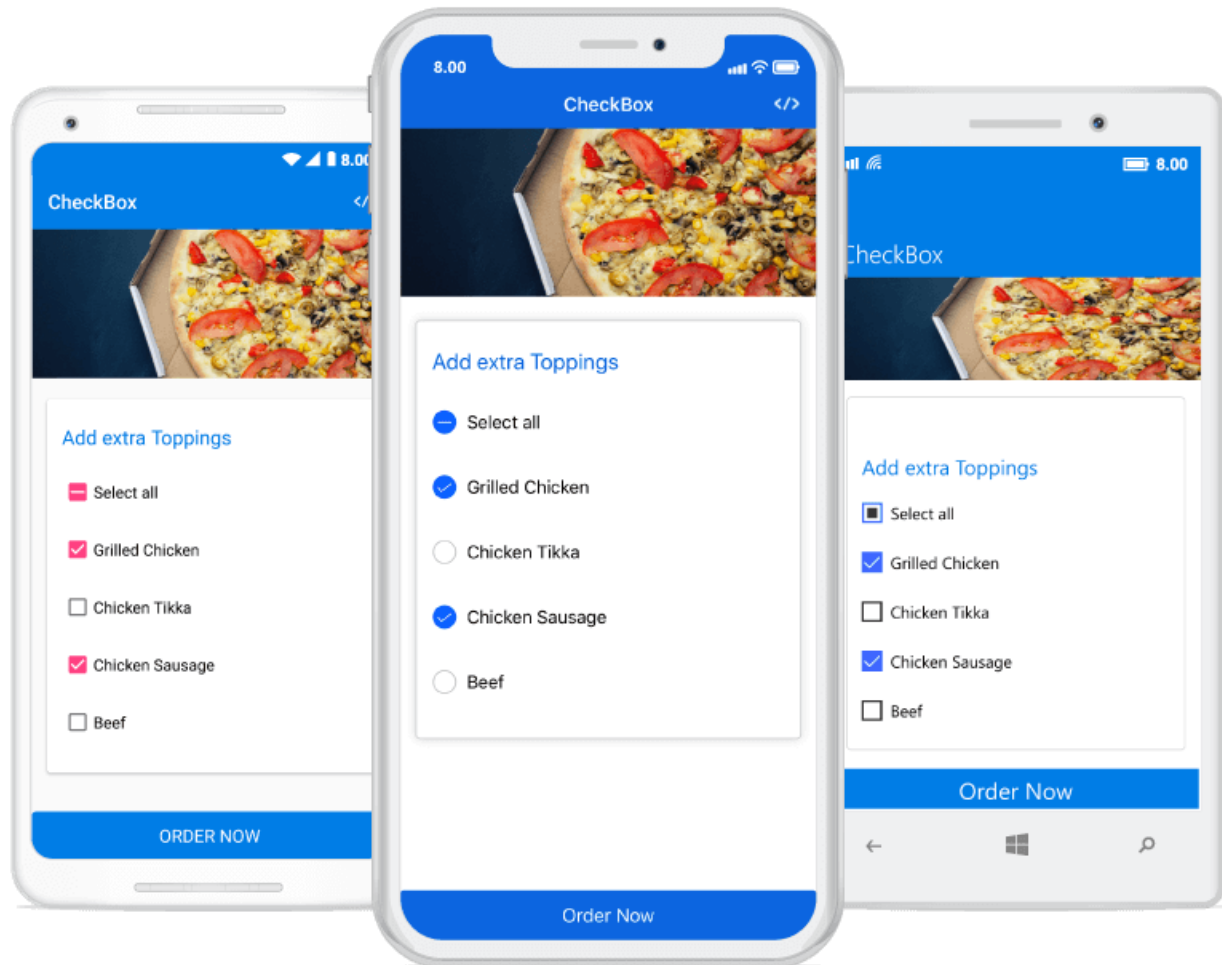
## SfCheckBox

### Overview

The check box is a selection control that allows users to select one or more options from a set. The three states of check box are checked, unchecked and indeterminate.

### Key features

- Supports three states.
- Allow users to select and clear the control by tapping.
- Supports check box color, shape and label text customization.



## Getting Started

This section explains the steps required to configure the `SfCheckBox` control in a real-time scenario and provides a walk-through on some of the customization features available in `SfCheckBox` control.

### Adding `SfCheckBox` reference

You can add `SfCheckBox` reference using one of the following methods:

#### Method 1: Adding `SfCheckBox` reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Buttons). To add `SfCheckBox` to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Buttons](https://www.nuget.org/packages/Syncfusion.Xamarin.Buttons), and then install it.

![Xamarin.Forms CheckBox NuGet](Images/Adding SfCheckBox reference.png)

#### Note:

- Install the same version of `SfCheckBox` NuGet in all the projects.
- In addition, you need to install the `[Syncfusion.Xamarin.Buttons.WPF]()` package for Xamarin.Forms WPF platform only.

#### Method 2: Adding `SfCheckBox` reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfCheckBox control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfCheckBox assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Buttons.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
WPF	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.WPF.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.WPF.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** If you are adding the references from toolbox, this step is not needed.

#### *Additional step for iOS*

To launch SfCheckBox in iOS, call the SfCheckBoxRenderer.Init() in FinishedLaunching overridden method of AppDelegate class in iOS Project, as demonstrated in the following code example.

**C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    LoadApplication(new App());
    SfCheckBoxRenderer.Init();
    return base.FinishedLaunching(app, options);
}
```

*Additional step for UWP*

This step is required only if the application is deployed in Release mode with .NET native tool chain enabled. It is for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the SfCheckBox assembly at OnLaunched overridden method of the App class in UWP project is the suggested work around, as demonstrated in the following code example.

**C#**

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    .....
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies that your app uses
    assembliesToInclude.Add(typeof(SfCheckBoxRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    .....
}
```

*Additional step for WPF*

To launch the check box in WPF, call the SfCheckBoxRenderer.Init() method in the MainWindow constructor of the MainWindow class after the Xamarin.Forms framework has been initialized and before the LoadApplication method is called as demonstrated in the following code sample.

**C#**

```
public partial class MainWindow : FormsApplicationPage
{
    public MainWindow()
    {
        InitializeComponent();
        Forms.Init();
        Syncfusion.XForms.WPF.Buttons.SfCheckBoxRenderer.Init();
        LoadApplication(new App());
    }
}
```

### Create a Simple SfCheckBox

The **SfCheckBox** control is configured entirely in C# code or by using XAML markup. The following steps explain how to create a **SfCheckBox** and configure its elements.

*Add namespace for referred assemblies*

#### **XML**

```
xmlns:syncfusion="clr-  
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
```

#### **C#**

```
using Syncfusion.XForms.Buttons;
```

*Refer SfCheckBox control with declared suffix name for Namespace*

#### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:local="clr-namespace:GettingStarted"  
xmlns:syncfusion="clr-  
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"  
x:Class="GettingStarted.MainPage">  
  <ContentPage.Content>  
    <StackLayout>  
      <syncfusion:SfCheckBox x:Name="checkBox"/>  
    </StackLayout>  
  </ContentPage.Content>  
</ContentPage>
```

#### **C#**

```
using Syncfusion.XForms.Buttons;  
using Xamarin.Forms;  
namespace GettingStarted  
{  
  public partial class MainPage : ContentPage  
  {  
    public MainPage()  
    {  
      InitializeComponent();  
      StackLayout stackLayout = new StackLayout();  
      SfCheckBox checkBox = new SfCheckBox();  
      stackLayout.Children.Add(checkbox);  
      this.Content = stackLayout;  
    }  
  }  
}
```

### Setting caption

The check box caption can be defined using the **Text** property of **SfCheckBox**. This caption normally describes the meaning of the check box and it displays next to check box.

**XML**

```
<syncfusion:SfCheckBox x:Name="checkBox" IsChecked="True" Text="CheckBox"/>
```

**C#**

```
SfCheckBox checkBox = new SfCheckBox();  
checkBox.IsChecked = true;  
checkBox.Text = "CheckBox";
```



CheckBox

This demo can be downloaded from this [link](#).

[Change the check box state](#)

The three visual states of **SfCheckBox** are:

- Checked
- Unchecked
- Indeterminate



CheckBox



CheckBox



CheckBox

You can change the state of the check box using the `IsChecked` property of `SfCheckBox`. In checked state, a tick mark is added to the visualization of check box.

State	Property	Value
checked	<code>IsChecked</code>	true
unchecked	<code>IsChecked</code>	false
indeterminate	<code>IsChecked</code>	null

**Note:** For the check box, to report the indeterminate state, set the `IsThreeState` property to true.

Check box can be used as a single or as a group. A single check box mostly used for a binary yes/no choice, such as "Remember me?", login scenario, or a terms of service agreement.

#### XML

```
<syncfusion:SfCheckBox x:Name="checkBox" Text="I agree to the terms of
services for this site" IsChecked="True"/>
```

#### C#

```
SfCheckBox checkBox = new SfCheckBox();
checkBox.Text = "I agree to the terms of services for this site";
checkBox.IsChecked = true;
```



I agree to the terms of services for this site

Multiple check boxes can be used as a group for multi-select scenarios in which a user chooses one or more items from the group of choices that are not mutually exclusive.

#### XML

```
<Label x:Name="label" Text="Pizza Toppings" />
<syncfusion:SfCheckBox x:Name="pepperoni" Text="Pepperoni"/>
<syncfusion:SfCheckBox x:Name="beef" Text="Beef" IsChecked="True"/>
<syncfusion:SfCheckBox x:Name="mushroom" Text="Mushrooms"/>
<syncfusion:SfCheckBox x:Name="onion" Text="Onions" IsChecked="True"/>
```

#### C#

```
Label label = new Label();
label.Text = "Pizza Toppings";
SfCheckBox pepperoni= new SfCheckBox();
pepperoni.Text = "Pepperoni";
SfCheckBox beef= new SfCheckBox();
beef.Text = "Beef";
beef.IsChecked = true;
```



```
SfCheckBox mushroom = new SfCheckBox();  
mushroom.Text = "Mushrooms";  
SfCheckBox onion = new SfCheckBox();  
onion.Text = "Pepperoni";  
onion.IsChecked = true;
```

## Pizza Toppings

☐ Pepperoni

☒ Beef

☐ Mushrooms

☒ Onions

This demo can be downloaded from this [link](#).

### Indeterminate

The **SfCheckBox** allows an indeterminate state in addition to the checked and unchecked state. The indeterminate state of the check box is enabled by setting the **IsThreeState** property of the control to **True**.

---

**Note:** When the **IsThreeState** property is set to **False** and **IsChecked** property is set to **null** then the check box will be in unchecked state.

---

The indeterminate state is used when a group of sub-choices has both checked and unchecked states. In the following example, the "Select all" checkbox has the **IsThreeState** property set to **true**. The "Select all" checkbox is checked if all child elements are checked, unchecked if all the child elements are unchecked, and indeterminate otherwise.

### XML

```
<syncfusion:SfCheckBox x:Name="selectAll" Text="Select All"  
IsChecked="{x:Null}" StateChanged="SelectAll_StateChanged"/>  
<syncfusion:SfCheckBox x:Name="pepperoni" Text="Pepperoni"  
StateChanged="CheckBox_StateChanged"/>
```

```

<syncfusion:SfCheckBox x:Name="beef" Text="Beef" IsChecked="True"
StateChanged="CheckBox_StateChanged"/>
<syncfusion:SfCheckBox x:Name="mushroom" Text="Mushrooms"
StateChanged="CheckBox_StateChanged"/>
<syncfusion:SfCheckBox x:Name="onion" Text="Onions" IsChecked="True"
StateChanged="CheckBox_StateChanged"/>
bool skip = false;
private void SelectAll_StateChanged(object sender, StateChangedEventArgs e)
{
    if (!skip)
    {
        skip = true;
        pepperoni.IsChecked = beef.IsChecked = mushroom.IsChecked = onion.IsChecked
        = e.IsChecked;
        skip = false;
    }
}
private void CheckBox_StateChanged(object sender, StateChangedEventArgs e)
{
    if (!skip)
    {
        skip = true;
        if (pepperoni.IsChecked.Value && beef.IsChecked.Value &&
            mushroom.IsChecked.Value && onion.IsChecked.Value)
            selectAll.IsChecked = true;
        else if (!pepperoni.IsChecked.Value && !beef.IsChecked.Value &&
            !mushroom.IsChecked.Value && !onion.IsChecked.Value)
            selectAll.IsChecked = false;
        else
            selectAll.IsChecked = null;
        skip = false;
    }
}

```

**C#**

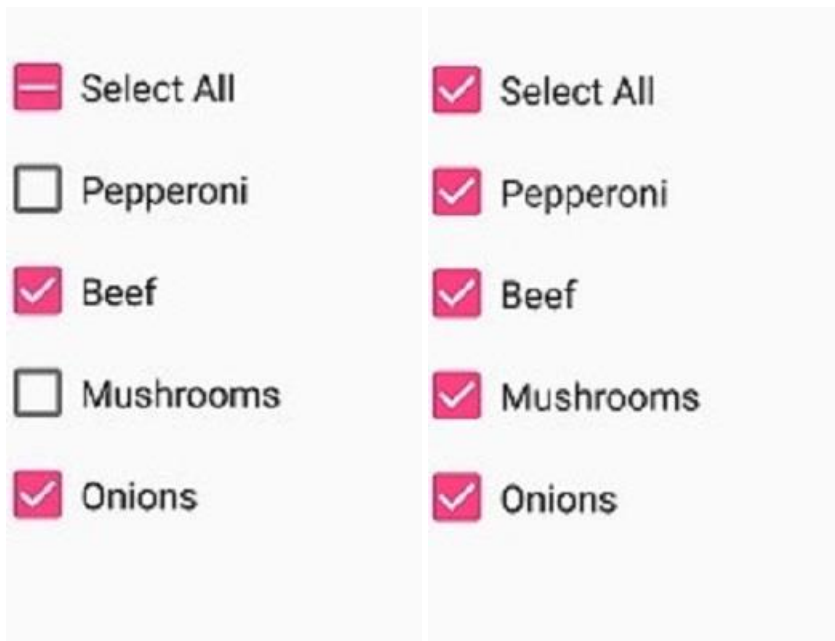
```

bool skip = false;
SfCheckBox selectAll, pepperoni, beef, mushroom, onion;
selectAll = new SfCheckBox();
selectAll.StateChanged += SelectAll_StateChanged;
selectAll.Text = "Select All";
pepperoni = new SfCheckBox();
pepperoni.StateChanged += CheckBox_StateChanged;
pepperoni.Text = "Pepperoni";
beef = new SfCheckBox();
beef.StateChanged += CheckBox_StateChanged;
beef.Text = "Beef";
beef.IsChecked = true;
mushroom = new SfCheckBox();
mushroom.StateChanged += CheckBox_StateChanged;
mushroom.Text = "Mushrooms";
onion = new SfCheckBox();
onion.StateChanged += CheckBox_StateChanged;
onion.Text = "Onions";
onion.IsChecked = true;
private void SelectAll_StateChanged(object sender, StateChangedEventArgs e)

```

```
{
    if (!skip)
    {
        skip = true;
        pepperoni.IsChecked = beef.IsChecked = mushroom.IsChecked = onion.IsChecked
        = e.IsChecked;
        skip = false;
    }
}

private void CheckBox_StateChanged(object sender, StateChangedEventArgs e)
{
    if (!skip)
    {
        skip = true;
        if (pepperoni.IsChecked.Value && beef.IsChecked.Value &&
            mushroom.IsChecked.Value && onion.IsChecked.Value)
            selectAll.IsChecked = true;
        else if (!pepperoni.IsChecked.Value && !beef.IsChecked.Value &&
            !mushroom.IsChecked.Value && !onion.IsChecked.Value)
            selectAll.IsChecked = false;
        else
            selectAll.IsChecked = null;
        skip = false;
    }
}
```



This demo can be downloaded from this [link](#).

## Visual Customization

### Customizing shape

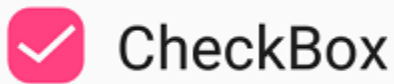
The check box shape can be customized using the `CornerRadius` property. This property specifies uniform radius value for every corner of the check box.

**XML**

```
<syncfusion:SfCheckBox x:Name="checkBox" Text="CheckBox" IsChecked="True"
    CornerRadius="5.0"/>
```

**C#**

```
SfCheckBox checkBox = new SfCheckBox();
checkBox.Text = "CheckBox";
checkBox.IsChecked = true;
checkBox.CornerRadius = 5.0f;
```

**Customizing state color**

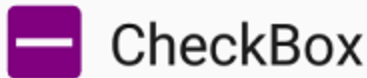
The default state colors can be customized using the `CheckedColor` and `UncheckedColor` properties. The checked/indeterminate state color is updated to the `CheckedColor` property value when the state is changed to the checked/indeterminate. The unchecked state color is updated to `UncheckedColor` property value when the state is changed to unchecked.

**XML**

```
<syncfusion:SfCheckBox x:Name="check" Text="CheckBox" IsChecked="True"
    CheckedColor="Green"/>
<syncfusion:SfCheckBox x:Name="uncheck" Text="CheckBox"
    UncheckedColor="Violet"/>
<syncfusion:SfCheckBox x:Name="indeterminate " Text="CheckBox"
    IsThreeState="True" IsChecked="{x:Null}" CheckedColor="Purple"/>
```

**C#**

```
SfCheckBox check= new SfCheckBox();
check.Text = "CheckBox";
check.IsChecked = true;
check.CheckedColor = Color.Green;
SfCheckBox uncheck = new SfCheckBox();
uncheck.Text = "CheckBox";
uncheck.UncheckedColor = Color.Violet;
SfCheckBox indeterminate = new SfCheckBox();
indeterminate.IsChecked = null;
indeterminate.IsThreeState = true;
indeterminate.Text = "CheckBox";
indeterminate.CheckedColor = Color.Purple;
```



### Setting caption text appearance

You can customize the display text appearance of the **SfCheckBox** control using the following properties:

- **TextColor**: Changes the color of the text.
- **HorizontalTextAlignment**: Changes the horizontal alignment of the caption text.
- **FontFamily**: Changes the font family of the caption text.
- **FontAttributes**: Sets font attributes(bold/italic/none) of the caption text.
- **FontSize**: Sets font size of the caption text.

### XML

```
<syncfusion:SfCheckBox x:Name="caption" Text="CheckBox" IsChecked="True"
TextColor="Violet" HorizontalTextAlignment="Center" FontFamily="Arial"
FontAttributes="Bold" FontSize="20"/>
```

### C#

```
SfCheckBox caption = new SfCheckBox();
caption.IsChecked = true;
caption.Text = "CheckBox";
caption.TextColor = Color.Violet;
caption.HorizontalTextAlignment = TextAlignment.Center;
caption.FontFamily = "Arial";
caption.FontAttributes = FontAttributes.Bold;
caption.FontSize = 20;
```

**CheckBox**

### LineBreakMode

The **LineBreakMode** allows you to wrap or truncate the text. The default value of this property is **NoWrap**. The following other options are available in **LineBreakMode**:

- **NoWrap** - Avoids the text wrap.
- **WordWrap** - Wraps the text by words.
- **CharacterWrap** - Wraps the text by character.
- **HeadTruncation** - Truncates the text at the start.
- **MiddleTruncation** - Truncates the text at the center.
- **TailTruncation** - Truncates the text at the end.

### TickColor Customization

The **TickColor** property customizes the color of the tick in SfCheckBox.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="CheckBoxCustomization.checkbox">
<ContentPage.Content>
<StackLayout>
<syncfusion:SfCheckBox x:Name="checkBox" IsChecked="True"
CheckedColor="Aqua" TickColor="Fuchsia" Text="CheckBox" />
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace CheckBoxCustomization
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout();
SfCheckBox checkBox = new SfCheckBox();
checkBox.IsChecked = true;
checkBox.Text = "CheckBox";
checkBox.CheckedColor = Color.Aqua;
checkBox.TickColor = Color.Fuchsia;
stackLayout.Children.Add(checkBox);
this.Content = stackLayout;
}
```



**Note:** The `TickColor` is not applicable for Android Platform. The default value of `TickColor` is `[Color.White]`.

This demo can be downloaded from this [link](#).

### Visual States

The visual of `CheckBox` can be customized using `VisualStates`. The `SfCheckBox` control contains the following three visual states:

- Checked
- Unchecked
- Intermediate

### XML

```
<buttons:SfCheckBox Text="CheckBox">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="Checked">
        <VisualState.Setters>
          <Setter Property="TextColor" Value="Accent"/>
          <Setter Property="CheckedColor" Value="Accent"/>
        </VisualState.Setters>
      </VisualState>
      <VisualState x:Name="Unchecked">
        <VisualState.Setters>
          <Setter Property="TextColor" Value="#ea3737"/>
          <Setter Property="UncheckedColor" Value="#ea3737"/>
        </VisualState.Setters>
      </VisualState>
      <VisualState x:Name="Intermediate">
        <VisualState.Setters>
          <Setter Property="Text" Value="Intermediate State"/>
        </VisualState.Setters>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</buttons:SfCheckBox>
```

### C#

```
SfCheckBox checkBox = new SfCheckBox() { Text = "CheckBox" };
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState checkedState = new VisualState
{
    Name = "Checked"
};
checkedState.Setters.Add(new Setter { Property =
SfCheckBox.TextColorProperty, Value = Color.Accent });
checkedState.Setters.Add(new Setter { Property =
SfCheckBox.CheckedColorProperty, Value = Color.Accent });
VisualState uncheckedState = new VisualState
{
    Name = "Unchecked"
```

```

};
uncheckedState.Setters.Add(new Setter { Property =
SfCheckBox.TextColorProperty, Value = Color.FromHex("#ea3737") });
uncheckedState.Setters.Add(new Setter { Property =
SfCheckBox.UncheckedColorProperty, Value = Color.FromHex("#ea3737") });
VisualState intermediateState = new VisualState
{
    Name = "Intermediate"
};
intermediateState.Setters.Add(new Setter { Property =
SfCheckBox.TextProperty, Value = "Intermediate State" });
commonStateGroup.States.Add(checkedImageState);
commonStateGroup.States.Add(uncheckedState);
commonStateGroup.States.Add(intermediateState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(checkBox, visualStateGroupList);

```

**Checked visual state:****Unchecked visual state:****Intermediate visual state:****Event****StateChanged event**

Occurs when the value(state) of the `IsChecked` property is changed by either touching the check box or setting the value to the `IsChecked` property using XAML or C# code. The event arguments are of type `StateChangedEventArgs` and expose the following property:

- `IsChecked`: The new value(state) of the `IsChecked` property.

**XML**

```

<syncfusion:SfCheckBox x:Name="checkBox" Text="Unchecked State"
IsThreeState="True" StateChanged="CheckBox_StateChanged"/>
private void CheckBox_StateChanged(object sender, StateChangedEventArgs e)
{
    if (e.IsChecked.HasValue && e.IsChecked.Value)
    {

```



```
checkBox.Text = "Checked State";  
}  
else if(e.IsChecked.HasValue && !e.IsChecked.Value)  
{  
    checkBox.Text = "Unchecked State";  
}  
else  
{  
    checkBox.Text = "Indeterminate State";  
}  
}
```

## C#

```
SfCheckBox checkBox = new SfCheckBox();  
checkBox.Text = "Unchecked State";  
checkBox.IsThreeState = true;  
checkBox.StateChanged += CheckBox_StateChanged;  
private void CheckBox_StateChanged(object sender, StateChangedEventArgs e)  
{  
    if (e.IsChecked.HasValue && e.IsChecked.Value)  
    {  
        checkBox.Text = "Checked State";  
    }  
    else if(e.IsChecked.HasValue && !e.IsChecked.Value)  
    {  
        checkBox.Text = "Unchecked State";  
    }  
    else  
    {  
        checkBox.Text = "Indeterminate State";  
    }  
}
```

☐ Unchecked State

☒ Checked State

☐ Indeterminate State

This demo can be downloaded from this [link](#).

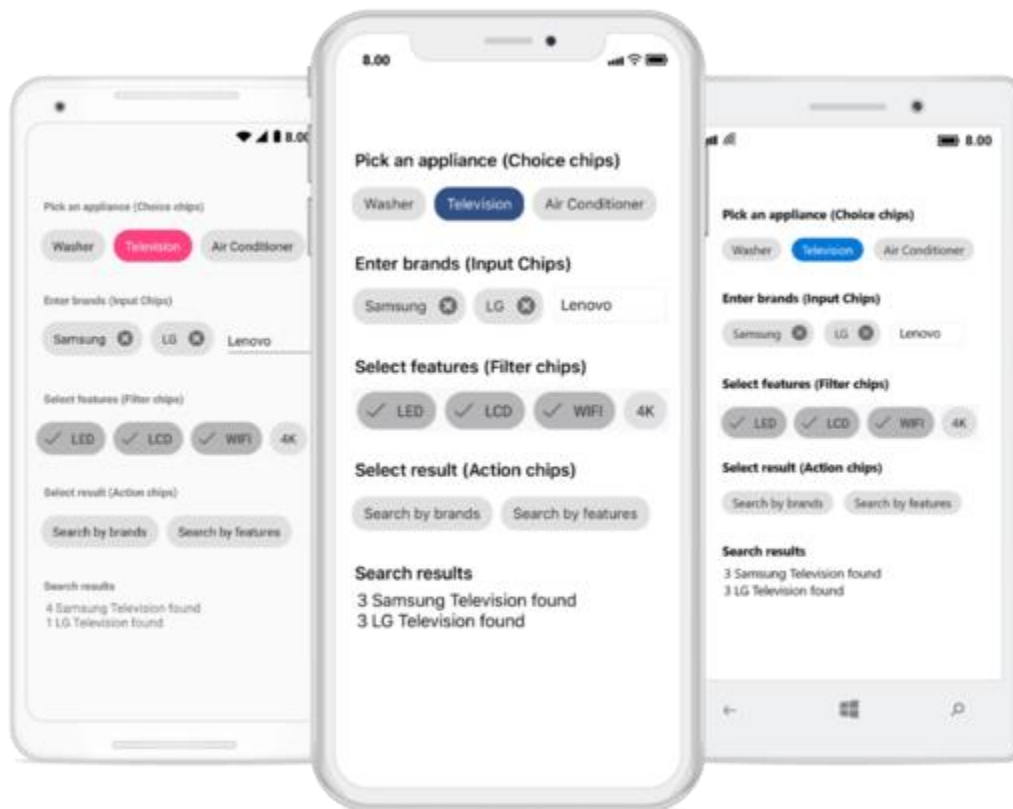
## Chips

### Overview

The chips control for Xamarin.Forms presents information in an interactive and customizable layout. It also arranges multiple chips in a user-preferred layout and groups them to make selections.

### Key Features

- Supports using StackLayout, FlexLayout, Grid, AbsoluteLayout, and RelativeLayout as a layout for adding and arranging chips.
- Provides options to choose from four different types: Input, Choice, Filter, and Action, each chip has different behavior as a group.
- Allows you make single and multiple selections in the Choice and Filter chips, respectively.
- Provides a way to add a view at the end of the group using the Input type chips.
- Has Command support for the chip group and individual chips in Action chips type.
- Provides user-friendly customization support to customize the corner radius, border color, border thickness, text color, background color, close button color, selection indicator color, etc.



### Getting started

This section explains the steps required to create chips and arrange them in a layout for performing action. This section covers only the minimal features that you needed to know to get started with the chips.

### Adding Chips reference

You can add Chips reference using one of the following methods:

#### Method 1: Adding Chips reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Buttons). To add Chips to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Buttons](https://www.nuget.org/packages/Syncfusion.Xamarin.Buttons), and then install it.

![Adding Chips reference from NuGet](images/action-type-chip/Adding Chips reference.png)

#### Note:

- Install the same version of Chips NuGet in all the projects.
- In addition, you need to install the [Syncfusion.Xamarin.Buttons.WPF]() package for Xamarin.Forms WPF platform only.

#### Method 2: Adding Chips reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the Chips control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding Chips assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Buttons.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
WPF	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.WPF.dll Syncfusion.Core.XForms.dll

	Syncfusion.Core.XForms.WPF.dll Syncfusion.Licensing.dll
--	--

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching an application on each platform with chips

To use the chips control inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and are discussed in the following sections:

#### iOS

**Note:** If you are adding the references from toolbox, this step is not needed.

#### For SfChip

To launch the **SfChip** in iOS, call the **SfChipRenderer.Init()** method in the **FinishedLaunching** overridden method of the **AppDelegate** class after the **Xamarin.Forms** framework has been initialized and before the **LoadApplication** method is called as demonstrated in the following code sample:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init();
    Syncfusion.XForms.iOS.Buttons.SfChipRenderer.Init();
    LoadApplication(new App());
    ...
}
```

#### For SfChipGroup

To launch the **SfChipGroup** in iOS, call the **SfChipGroupRenderer.Init()** method in the **FinishedLaunching** overridden method of the **AppDelegate** class after the **Xamarin.Forms** framework has been initialized and before the **LoadApplication** method is called as demonstrated in the following code sample:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init();
    Syncfusion.XForms.iOS.Buttons.SfChipGroupRenderer.Init();
    LoadApplication(new App());
    ...
}
```

### Universal Windows Platform (UWP)

#### For SfChip

To deploy the SfChip in Release mode, you need to initialize the SfChip assemblies in the App.xaml.cs file in the UWP project as shown in the following code sample.

#### C#

```
// In App.xaml.cs
using Syncfusion.XForms.UWP.Buttons;
using Syncfusion.XForms.UWP.Border;
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    if (rootFrame == null)
    {
        List<Assembly> assembliesToInclude = new List<Assembly>();
        assembliesToInclude.Add(typeof(SfButtonRenderer).GetTypeInfo().Assembly);
        assembliesToInclude.Add(typeof(SfChipRenderer).GetTypeInfo().Assembly);
        assembliesToInclude.Add(typeof(SfBorderRenderer).GetTypeInfo().Assembly);
        Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    }
    ...
}
```

#### For SfChipGroup

To deploy the SfChipGroup in Release mode, you need to initialize the SfChipGroup assemblies in the App.xaml.cs file in the UWP project as shown in the following code sample.

#### C#

```
// In App.xaml.cs
using Syncfusion.XForms.UWP.Buttons;
using Syncfusion.XForms.UWP.Border;
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    if (rootFrame == null)
    {
        List<Assembly> assembliesToInclude = new List<Assembly>();
        assembliesToInclude.Add(typeof(SfButtonRenderer).GetTypeInfo().Assembly);
        assembliesToInclude.Add(typeof(SfChipGroupRenderer).GetTypeInfo().Assembly);
        assembliesToInclude.Add(typeof(SfBorderRenderer).GetTypeInfo().Assembly);
        Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    }
    ...
}
```

### Windows Presentation Foundation (WPF)

To launch the chip group and chip in WPF, call the SfChipGroupRenderer.Init() and SfChipRenderer.Init() methods in the MainWindow constructor of the MainWindow class after the Xamarin.Forms framework has been initialized and before the LoadApplication method is called as demonstrated in the following code sample.

#### C#

```
public partial class MainWindow : FormsApplicationPage
{
    public MainWindow()
    {
        InitializeComponent();
        Forms.Init();
        Syncfusion.XForms.WPF.Border.SfBorderRenderer.Init();
        //For chip group
        Syncfusion.XForms.WPF.Buttons.SfChipGroupRenderer.Init();
        // For chip
        Syncfusion.XForms.WPF.Buttons.SfChipRenderer.Init();
        LoadApplication(new App());
    }
}
```

### Android

The Android platform does not require any additional configuration to render the chips control.

### Initialize chips

Import the [SfChipGroup](#) namespace in respective page.

### XML

```
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
```

### C#

```
using Syncfusion.XForms.Buttons;
```

Then initialize an empty SfChipGroup as shown in the following code:

### XML

```
<ContentPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Chips"
x:Class="Chips.GettingStarted">
    <ContentPage.Content>
        <Grid>
            <buttons:SfChipGroup/>
        </Grid>
    </ContentPage.Content>
</ContentPage>
```

### C#

```
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace Chips
{
    public partial class GettingStarted: ContentPage
```

```
{
public GettingStarted()
{
InitializeComponent();
Grid grid = new Grid();
grid.Children.Add(new SfChipGroup());
this.Content = grid;
}
}
}
```

### Set layout for the control

Any layout can be used to arrange the chips in the chips control using `ChipLayout` property. The chips control creates chip for each object and arranges chips in a `StackLayout` with horizontal orientation. Any layout can be used to arrange the chips in the chips control. In the following example, the `FlexLayout` has been used.

### XML

```
<ContentPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Chips"
x:Class="Chips.GettingStarted">
<ContentPage.Content>
<Grid>
<buttons:SfChipGroup>
<buttons:SfChipGroup.ChipLayout>
<FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
</buttons:SfChipGroup>
</Grid>
</ContentPage.Content>
</ContentPage >
```

### C#

```
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace Chips
{
public partial class GettingStarted: ContentPage
{
public GettingStarted()
{
InitializeComponent();
}
```

```
Grid grid = new Grid();
SfChipGroup chipGroup = new SfChipGroup();
grid.Children.Add(chipGroup);
FlexLayout layout = new FlexLayout()
{
    Direction = FlexDirection.Row,
    Wrap = FlexWrap.Wrap,
    HorizontalOptions = LayoutOptions.Start,
    VerticalOptions = LayoutOptions.Center,
    AlignContent = FlexAlignContent.Start,
    JustifyContent = FlexJustify.Start,
    AlignItems = FlexAlignItems.Start,
};
chipGroup.ChipLayout = layout;
this.Content = grid;
}
```

### Populating business objects

Now, define a simple data model of person with the name and image properties. Create a view model class and initialize a collection of persons as shown in the following code sample.

#### C#

```
namespace Chips
{
    //Model class for chips
    public class Person
    {
        public string Name
        {
            get;
            set;
        }
    }
}
```

#### C#

```
using System.Collections.ObjectModel;
using System.ComponentModel;
namespace Chips
{
    //View model class for chips
    public class ViewModel : INotifyPropertyChanged
    {
        private ObservableCollection<Person> employees;
        public ObservableCollection<Person> Employees
        {
            get
            {
                return employees;
            }
            set
            {
            }
        }
    }
}
```



```

{
    Employees = value;
    OnPropertyChanged("Employees");
}
}
public ViewModel()
{
    employees = new ObservableCollection<Person>();
    employees.Add(new Person() { Name = "John" });
    employees.Add(new Person() { Name = "James" });
    employees.Add(new Person() { Name = "Linda" });
    employees.Add(new Person() { Name = "Rose" });
    employees.Add(new Person() { Name = "Mark" });
}
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string property)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(property));
    }
}
}
}

```

Create an instance of ViewModel class, and then set it as the **BindingContext**. Bind the **ItemsSource** property with a collection, and then set the **DisplayMemberPath** property:

#### XML

```

<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:Chips"
    x:Class="Chips.GettingStarted">
    <ContentPage.BindingContext>
    <local:ViewModel x:Name="viewModel"/>
    </ContentPage.BindingContext>
    <ContentPage.Content>
    <Grid>
    <buttons:SfChipGroup
        ItemsSource="{Binding Employees}"
        ChipPadding="8,8,0,0"
        DisplayMemberPath="Name">
    <buttons:SfChipGroup.ChipLayout>
    <FlexLayout
        HorizontalOptions="Start"
        VerticalOptions="Center"
        Direction="Row"
        Wrap="Wrap"
        JustifyContent="Start"
        AlignContent="Start"
        AlignItems="Start"/>
    </buttons:SfChipGroup.ChipLayout>
    </Grid>
    </ContentPage.Content>
</ContentPage>

```

```

</buttons:SfChipGroup>
</Grid>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.XForms.Buttons;
using System.Collections.ObjectModel;
using System.ComponentModel;
namespace Chips
{
    public partial class GettingStarted: ContentPage
    {
        public GettingStarted()
        {
            InitializeComponent();
            Grid grid = new Grid();
            SfChipGroup chipGroup = new SfChipGroup();
            grid.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
            chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
            chipGroup.DisplayMemberPath = "Name";
            chipGroup.ChipPadding = new Thickness(8, 8, 0, 0);
            this.Content = grid;
        }
    }
}

```



### Set types of chip group

The functionality of chips control differ based on its [Type](#) property.

By default type of chips control have Input type. Input chip types have close button, using it chip can be removed dynamically from children and the layout.

The following code example uses the **Action** type. In Action type, **Command** property of **SfChipGroup** is executed when any chip in the group is tapped. Here the Employee name of corresponding chip is set as label text when the Command is executed.

### XML

```
<ContentPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Chips"
x:Class="Chips.GettingStarted">
  <ContentPage.BindingContext>
    <local:ViewModel/>
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <StackLayout>
      <buttons:SfChipGroup
        Command="{Binding ActionCommand}"
        ItemsSource="{Binding Employees}"
        Type="Action">
      </buttons:SfChipGroup>
      <StackLayout Orientation="Horizontal">
        <Label
          Text="Name:"
          FontAttributes="Bold"
          FontSize="14" />
        <Label
          Text="{Binding Result}"
          FontAttributes="Bold"
          FontSize="14" />
      </StackLayout>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

### C#

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Windows.Input;
using Xamarin.Forms;
namespace Chips
{
    public class ViewModel : INotifyPropertyChanged
    {
        private ICommand actionCommand;
        private ObservableCollection<Person> employees;
        private string result;
        public ICommand ActionCommand
        {
            get
            {
            }
```

```
return actionCommand;
}
set
{
    actionCommand = value;
}
}
public ObservableCollection<Person> Employees
{
    get
    {
        return employees;
    }
    set
    {
        Employees = value;
        OnPropertyChanged("Employees");
    }
}
public string Result
{
    get
    {
        return result;
    }
    set
    {
        result = value;
        OnPropertyChanged("Result");
    }
}
public ViewModel()
{
    ActionCommand = new Command(HandleAction);
    employees = new ObservableCollection<Person>();
    employees.Add(new Person() { Name = "John" });
    employees.Add(new Person() { Name = "James" });
    employees.Add(new Person() { Name = "Linda" });
    employees.Add(new Person() { Name = "Rose" });
    employees.Add(new Person() { Name = "Mark" });
}
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string property)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(property));
    }
}
private void HandleAction(object obj)
{
    Result = (obj as Person).Name.ToString();
}
}
```



Name: John

You can find the complete getting started sample from this [link](#).

### Populating chips

Chips can be populated with either business objects and SfChip. This section explain how to populate the chips control with business objects and SfChip.

#### Populating business objects as items

Business objects can be populated in Chips using the `ItemsSource` property.

Refer to this [documentation](#) to know more details about populating the chips control with list of employee details.

#### Populating SfChip as items

Chips control also provides support to create and set SfChip as item. It can be achieved using the `Items` property.

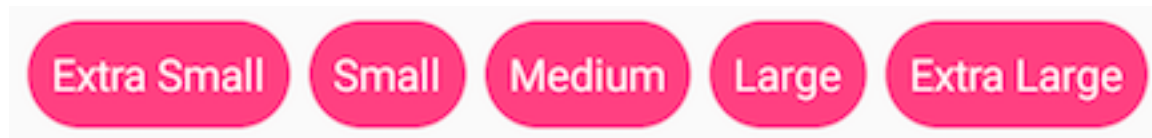
#### XML

```
<ContentPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Chips"
x:Class="Chips.GettingStarted">
<ContentPage.Content>
<buttons:SfChipGroup Type="Action">
<buttons:SfChipGroup.Items>
<buttons:SfChip Text="Extra Small"/>
<buttons:SfChip Text="Small"/>
<buttons:SfChip Text="Medium"/>
<buttons:SfChip Text="Large"/>
<buttons:SfChip Text="Extra Large"/>
</buttons:SfChipGroup.Items>
</buttons:SfChipGroup>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace Chips
{
    public partial class GettingStarted: ContentPage
    {
        public GettingStarted()
        {
            InitializeComponent();
            Grid grid = new Grid();
```

```
var chipGroup = new SfChipGroup() {Type = SfChipsType.Action};
grid.Children.Add(chipGroup);
chipGroup.Items.Add(new SfChip() {Text="Extra Small"});
chipGroup.Items.Add(new SfChip() {Text="Small"});
chipGroup.Items.Add(new SfChip() {Text="Medium"});
chipGroup.Items.Add(new SfChip() {Text="Large"});
chipGroup.Items.Add(new SfChip() {Text="Extra Large"});
this.Content = grid;
}
```



### Set the type for chip group

The functionality of chips control differ based on its **Type** property. No operation can be performed in a chip group unless the Type property is set. The chips control provides four different types, and each has its own functionality. The types are,

- Action
- Choice
- Filter
- Input

#### Input

Arranges the chips in a layout and enables the close button for each chip. Using the close button, a chip can be removed from children and layout as well. It additionally has support to add an optional **InputView** at the end of the layout, from which users can obtain the chip text for creating a chip at run time.

#### Choice

Allows users to select a single chip from a group of items. Selecting a chip will automatically deselect the previously selected chips. The selected chip color can be customized using the **SelectedChipBackgroundColor** and **SelectedChipTextColor** properties. The **SelectedItem** property holds the instance of recently selected chip.

#### Filter

Allows users to select more than one chip in a group of chips. The selected chips are indicated by selection indicator in this type. The selection indicator can be customized using the **SelectionIndicatorColor** property. Use the **SelectedItems** property to get the list of selected chips.

#### Action

Executes the **Command** of ChipGroup and raises the Clicked event when a chip is clicked.

In this example, the Input chip is used to add an employee at run time. To get the employee name as input, an entry is added as **InputView**.

#### XML

```

<ContentPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Chips"
x:Class="Chips.GettingStarted">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<buttons:SfChipGroup
x:Name="chipGroup"
ItemsSource="{Binding Employees}"
ChipPadding="8,8,0,0"
DisplayMemberPath="Name">
<buttons:SfChipGroup.ChipLayout>
<FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
<buttons:SfChipGroup.InputView>
<Entry
Margin="10,10,0,0"
WidthRequest="110"
Completed="Entry_Completed"/>
</buttons:SfChipGroup.InputView>
</buttons:SfChipGroup>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using Syncfusion.XForms.Buttons;
using System.Collections.ObjectModel;
using System.ComponentModel;
namespace Chips
{
public partial class GettingStarted: ContentPage
{
public GettingStarted()
{
InitializeComponent();
Grid grid = new Grid();
SfChipGroup chipGroup = new SfChipGroup();
grid.Children.Add(chipGroup);
FlexLayout layout = new FlexLayout()
{
Direction = FlexDirection.Row,
Wrap = FlexWrap.Wrap,
HorizontalOptions = LayoutOptions.Start,

```

```

VerticalOptions = LayoutOptions.Center,
AlignContent = FlexAlignContent.Start,
JustifyContent = FlexJustify.Start,
AlignItems = FlexAlignItems.Start,
};
var entry= new Entry { Margin = new Thickness(10, 10, 0, 0), WidthRequest = 110 };
entry.Completed += Entry_Completed;
chipGroup.InputView = entry;
chipGroup.ChipLayout = layout;
this.BindingContext = new ViewModel();
chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
chipGroup.DisplayMemberPath = "Name";
chipGroup.ChipPadding = new Thickness(8, 8, 0, 0);
this.Content = grid;
}
}
}

```

The following code example demonstrates the dynamic addition of chips into a chip group with the completed event in entry.

### C#

```

using Xamarin.Forms;
using Syncfusion.XForms.Buttons;
namespace Chips
{
    public partial class GettingStarted : ContentPage
    {
        public GettingStarted()
        {
            InitializeComponent();
        }
        private void Entry_Completed(object sender, System.EventArgs e)
        {
            Entry entry = sender as Entry;
            if (entry != null && !string.IsNullOrEmpty(entry.Text))
            {
                viewModel.Employees.Add(new Person() { Name = entry.Text });
            }
        }
    }
}

```

---

**Note:** The `InputView` is visible only in Input type.

---

### Customization of SfChip

The chip control supports to customize the background color, border color, close button color, and more. The chip control can be customized using the following properties:

#### ShowCloseButton

The [ShowCloseButton](#) property sets the visible state of close button in SfChip.

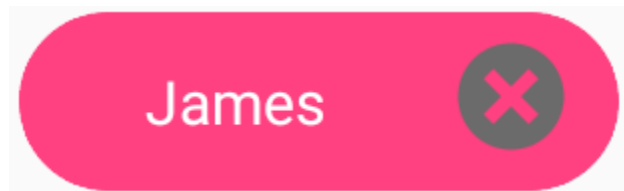
### XML



```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
ShowCloseButton="true" >
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

**C#**

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout();
SfChip chip = new SfChip();
chip.Text = "James";
chip.WidthRequest = 150;
chip.HorizontalOptions = LayoutOptions.Center;
chip.VerticalOptions = LayoutOptions.Center;
chip.ShowCloseButton = true;
stackLayout.Children.Add(chip);
this.Content = stackLayout;
}
}
}
```



**Note:** The default value of ShowCloseButton is [false].

**ShowSelectionIndicator**

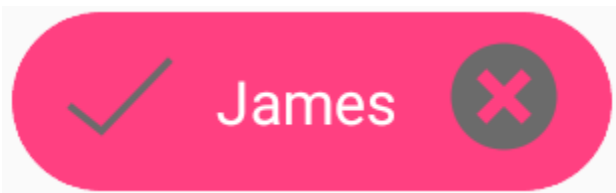
The [ShowSelectionIndicator](#) property sets the visible state of selection indicator in SfChip.

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
ShowCloseButton="true"
ShowSelectionIndicator="true"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

**C#**

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
            chip.ShowCloseButton = true;
            chip.ShowSelectionIndicator = true;
            stackLayout.Children.Add(chip);
            this.Content = stackLayout;
        }
    }
}
```



**Note:** The default value of ShowSelectionIndicator is [false].

## CloseButtonColor

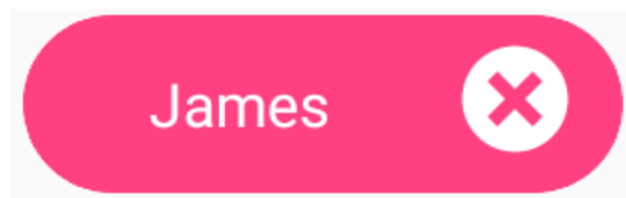
The [CloseButtonColor](#) property customizes the color of the close button in SfChip.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
ShowCloseButton="true"
CloseButtonColor="White"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
            chip.ShowCloseButton = true;
            chip.CloseButtonColor = Color.White;
            stackLayout.Children.Add(chip);
            this.Content = stackLayout;
        }
    }
}
```



**Note:** The default value of CloseButtonColor is [Color.FromHex("#6b6b6b")].

#### SelectionIndicatorColor

The [SelectionIndicatorColor](#) property customizes the selection indicator color in SfChip.

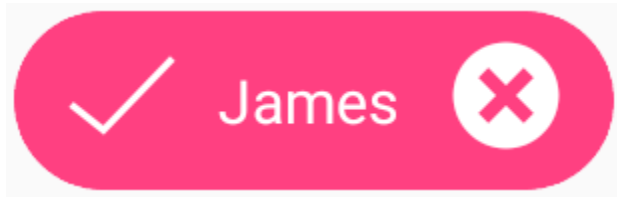
#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
  <ContentPage.Content>
    <StackLayout Margin="8,8,8,8" >
      <buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
ShowCloseButton="true"
ShowSelectionIndicator="true"
CloseButtonColor = "White"
SelectionIndicatorColor = "White"
      >
    </buttons:SfChip>
  </StackLayout>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
  public partial class MainPage : ContentPage
  {
    public MainPage()
    {
      InitializeComponent();
      StackLayout stackLayout = new StackLayout();
      SfChip chip = new SfChip();
      chip.Text = "James";
      chip.WidthRequest = 150;
      chip.HorizontalOptions = LayoutOptions.Center;
      chip.VerticalOptions = LayoutOptions.Center;
      chip.ShowCloseButton = true;
      chip.ShowSelectionIndicator = true;
      chip.CloseButtonColor = Color.White;
    }
  }
}
```

```
chip.SelectionIndicatorColor = Color.White;
stackLayout.Children.Add(chip);
this.Content = stackLayout;
}
}
```



**Note:** The default value of SelectionIndicatorColor is [Color.FromHex("#6b6b6b")].

### BackgroundColor

The [BackgroundColor](#) property customizes the background color of SfChip.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
  <ContentPage.Content>
    <StackLayout Margin="8,8,8,8" >
      <buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
ShowCloseButton="true"
ShowSelectionIndicator="true"
CloseButtonColor = "White"
SelectionIndicatorColor = "White"
BackgroundColor="Aqua"
>
    </buttons:SfChip>
  </StackLayout>
</ContentPage.Content>
</ContentPage>
```

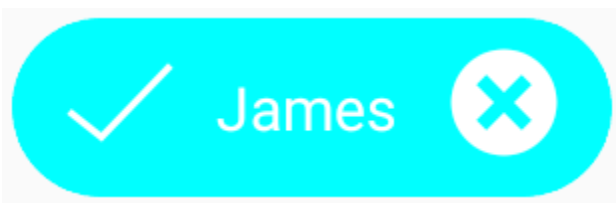
### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

```

StackLayout stackLayout = new StackLayout();
SfChip chip = new SfChip();
chip.Text = "James";
chip.WidthRequest = 150;
chip.HorizontalOptions = LayoutOptions.Center;
chip.VerticalOptions = LayoutOptions.Center;
chip.ShowSelectionIndicator = true;
chip.CloseButtonColor = Color.White;
chip.SelectionIndicatorColor = Color.White;
chip.BackgroundColor = Color.Aqua;
stackLayout.Children.Add(chip);
this.Content = stackLayout;
}
}
}

```



**Note:** The default value of BackgroundColor is [Color.Accent].

#### BorderColor

The [BorderColor](#) property customizes the color of border in SfChip.

#### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
BorderWidth="4"
BorderColor="Black"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

#### C#

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization

```

```

{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout();
SfChip chip = new SfChip();
chip.Text = "James";
chip.WidthRequest = 150;
chip.HorizontalOptions = LayoutOptions.Center;
chip.VerticalOptions = LayoutOptions.Center;
chip.BorderWidth = 4;
chip.BorderColor = Color.Black;
stackLayout.Children.Add(chip);
this.Content = stackLayout;
}
}
}

```



**Note:** The default value of BorderColor is [Color.Transparent].

#### BorderWidth

The [BorderWidth](#) property customizes the thickness of border in SfChip.

#### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
BorderWidth="8"
BorderColor="Black"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>

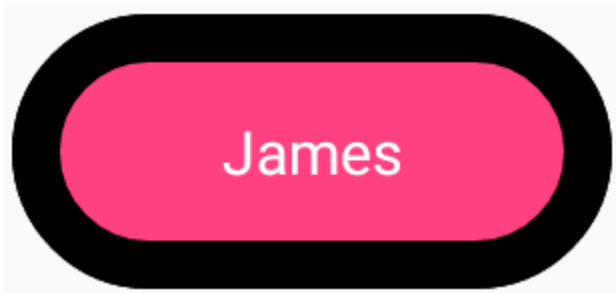
```

**C#**

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
            chip.BorderWidth = 8;
            chip.BorderColor = Color.Black;
            stackLayout.Children.Add(chip);
            this.Content = stackLayout;
        }
    }
}

```




---

**Note:** Default value of BorderWidth

---

Android : [0d]

iOS : [0d]

UWP : [2d]

**CornerRadius**

The **CornerRadius** property is used to customize the rounded edges in SfChip as demonstrated in the following code sample.

**XML**

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
    <ContentPage.Content>

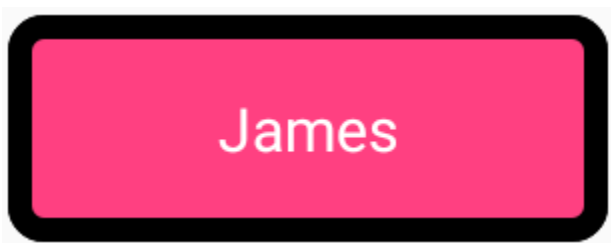
```



```
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
BorderWidth="4"
CornerRadius = "4"
BorderColor="Black"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout();
SfChip chip = new SfChip();
chip.Text = "James";
chip.WidthRequest = 150;
chip.HorizontalOptions = LayoutOptions.Center;
chip.VerticalOptions = LayoutOptions.Center;
chip.BorderWidth = 4;
chip.CornerRadius = 4;
chip.BorderColor = Color.Black;
stackLayout.Children.Add(chip);
this.Content = stackLayout;
}
}
}
```



**Note:** The default value of CornerRadius is [Thickness(0)].

## FontAttributes

The [FontAttributes](#) property customizes the font style of text in SfChip.

## XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
ShowCloseButton="true"
ShowSelectionIndicator="true"
CloseButtonColor = "White"
SelectionIndicatorColor = "White"
FontAttributes="Italic" >
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
            chip.ShowSelectionIndicator = true;
            chip.CloseButtonColor = Color.White;
            chip.SelectionIndicatorColor = Color.White;
            chip.FontAttributes = FontAttributes.Italic;
            stackLayout.Children.Add(chip);
            this.Content = stackLayout;
        }
    }
}
```



### FontFamily

The [FontFamily](#) property customizes the font family of text in SfChip.

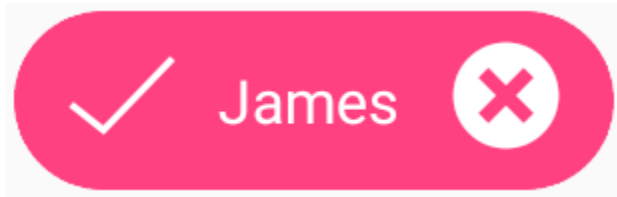
### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
ShowCloseButton="true"
ShowSelectionIndicator="true"
CloseButtonColor = "White"
SelectionIndicatorColor = "White"
FontFamily="Arial"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
            chip.ShowSelectionIndicator = true;
            chip.CloseButtonColor = Color.White;
            chip.SelectionIndicatorColor = Color.White;
        }
    }
}
```

```
chip.FontFamily = "Arial";
stackLayout.Children.Add(chip);
this.Content = stackLayout;
}
}
```



### FontSize

The [FontSize](#) property customizes the size of text in SfChip.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
ShowCloseButton="true"
ShowSelectionIndicator="true"
CloseButtonColor = "White"
SelectionIndicatorColor = "White"
FontSize = "10"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

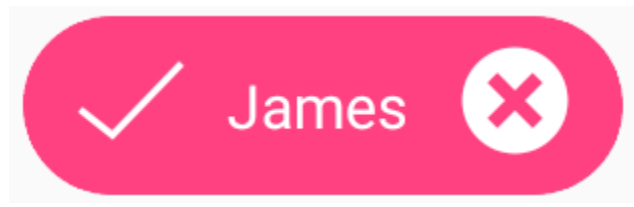
### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout();
SfChip chip = new SfChip();
```

```

chip.Text = "James";
chip.WidthRequest = 150;
chip.HorizontalOptions = LayoutOptions.Center;
chip.VerticalOptions = LayoutOptions.Center;
chip.ShowSelectionIndicator = true;
chip.CloseButtonColor = Color.White;
chip.SelectionIndicatorColor = Color.White;
chip.FontSize = 10;
stackLayout.Children.Add(chip);
this.Content = stackLayout;
}
}
}

```




---

**Note:** Default Value of FontSize

---

Android : [14d]

iOS : [15d]

UWP : [15d]

TextColor

The [TextColor](#) property customizes the color of text in SfChip.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
ShowCloseButton="true"
ShowSelectionIndicator="true"
CloseButtonColor = "White"
SelectionIndicatorColor = "White"
BackgroundColor="Aqua"
TextColor="Black"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
            chip.ShowSelectionIndicator = true;
            chip.CloseButtonColor = Color.White;
            chip.SelectionIndicatorColor = Color.White;
            chip.BackgroundColor = Color.Aqua;
            chip.TextColor = Color.Black;
            stackLayout.Children.Add(chip);
            this.Content = stackLayout;
        }
    }
}

```



**Note:** The default value of TextColor is [Color.White].

**TextAlignment**

The [HorizontalTextAlignment](#) and [VerticalTextAlignment](#) properties customize the alignment of text in SfChip.

**XML**

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
    <ContentPage.Content>
        <StackLayout Margin="8,8,8,8" >
            <buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"

```

```

VerticalOptions="Center"
ShowSelectionIndicator="true"
SelectionIndicatorColor = "White"
BackgroundColor="Aqua"
HorizontalTextAlignment="Start"
VerticalTextAlignment="Start"
TextColor="Black"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>

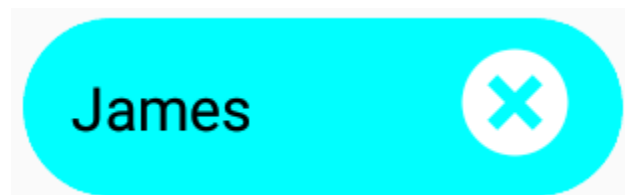
```

**C#**

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
            chip.ShowSelectionIndicator = true;
            chip.SelectionIndicatorColor = Color.White;
            chip.BackgroundColor = Color.Aqua;
            chip.HorizontalTextAlignment = TextAlignment.Start;
            chip.VerticalTextAlignment = TextAlignment.Start;
            chip.TextColor = Color.Black;
            stackLayout.Children.Add(chip);
            this.Content = stackLayout;
        }
    }
}

```



**Note:** The default values of HorizontalTextAlignment and VerticalTextAlignment are [TextAlignment.Center].

## ShowIcon

You can enable the icon image using the [ShowIcon](#) property to know whether any image appears to the SfChip.

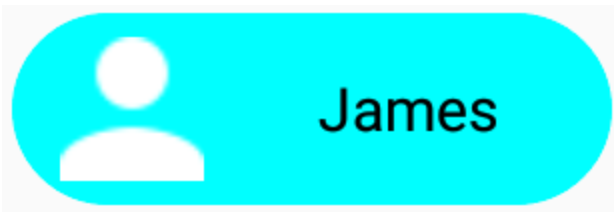
## XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
BackgroundColor="Aqua"
TextColor="Black"
ImageSource="ChipUserContact.png"
ShowIcon="true"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
            chip.BackgroundColor = Color.Aqua;
            chip.TextColor = Color.Black;
            chip.ImageSource = "ChipUserContact.png";
            chip.ShowIcon = true;
            stackLayout.Children.Add(chip);
            this.Content = stackLayout;
        }
    }
}
```






---

**Note:** The default value of `ShowIcon` is `[false]`.

---

#### ImageSource

The [ImageSource](#) property customizes the icon image of `SfChip` by adding a custom image.

---

**Note:** Enable the [ShowIcon](#) property to enable the [ImageSource](#) property.

---

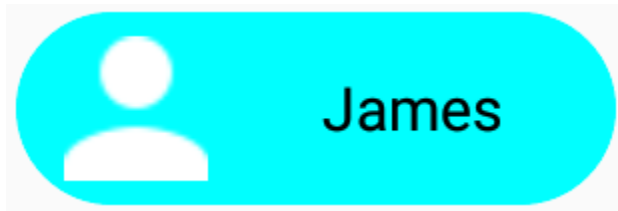
#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
BackgroundColor="Aqua"
TextColor="Black"
ImageSource="ChipUserContact.png"
ShowIcon="true"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
```

```
chip.BackgroundColor = Color.Aqua;
chip.TextColor = Color.Black;
chip.ImageSource = "ChipUserContact.png";
chip.ShowIcon = true;
stackLayout.Children.Add(chip);
this.Content = stackLayout;
}
}
}
```



### ImageWidth

The [ImageWidth](#) property customizes the width of icon image in SfChip.

**Note:** Enable the [ShowIcon](#) property to enable the [ImageSource](#) property.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
BackgroundColor="Aqua"
TextColor="Black"
ImageSource="ChipUserContact.png"
ImageWidth="25"
ShowIcon="true"
>
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
public partial class MainPage : ContentPage
```

```

{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout();
SfChip chip = new SfChip();
chip.Text = "James";
chip.WidthRequest = 150;
chip.HorizontalOptions = LayoutOptions.Center;
chip.VerticalOptions = LayoutOptions.Center;
chip.BackgroundColor = Color.Aqua;
chip.TextColor = Color.Black;
chip.ImageSource = "ChipUserContact.png";
chip.ImageWidth = 25;
chip.ShowIcon = true;
stackLayout.Children.Add(chip);
this.Content = stackLayout;
}
}
}

```



**Note:** The default value of ImageWidth is [32].

#### ImageAlignment

The [ImageAlignment](#) property customizes the alignment of icon image in SfChip.

**Note:** Enable the [ShowIcon](#) property to enable the [ImageSource](#) property.

#### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.Content>
<StackLayout Margin="8,8,8,8" >
<buttons:SfChip Text="James"
WidthRequest="150"
HorizontalOptions="Center"
VerticalOptions="Center"
BackgroundColor="Aqua"
TextColor="Black"
ImageSource="ChipUserContact.png"
ImageAlignment="End"
ImageWidth="25"
ShowIcon="true"
>

```

```

</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>

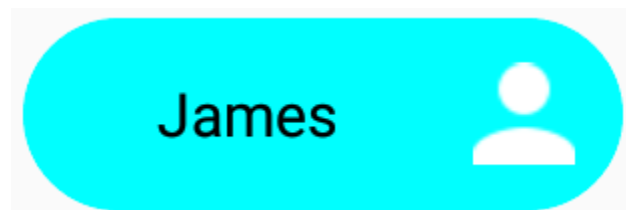
```

## C#

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();
            SfChip chip = new SfChip();
            chip.Text = "James";
            chip.WidthRequest = 150;
            chip.HorizontalOptions = LayoutOptions.Center;
            chip.VerticalOptions = LayoutOptions.Center;
            chip.BackgroundColor = Color.Aqua;
            chip.TextColor = Color.Black;
            chip.ImageSource = "ChipUserContact.png";
            chip.ImageAlignment = Alignment.End;
            chip.ImageWidth = 25;
            chip.ShowIcon = true;
            stackLayout.Children.Add(chip);
            this.Content = stackLayout;
        }
    }
}

```



**Note:** The default value of ImageAlignment is [Alignment.Start].

## Command

The [Command](#) property associates a command with an instance of SfChip. This property is most often set with MVVM pattern to bind callbacks back into the ViewModel.

## XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"

```

```

x:Class="ChipCustomization.MainPage">
<ContentPage.BindingContext>
<local:CommandDemoViewModel />
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout Margin="8,8,0,0">
<buttons:SfChip x:Name="Chip"
Text="Chip"
HorizontalOptions="Center"
VerticalOptions="Center"
WidthRequest="150"
ShowCloseButton="true"
BackgroundColor="{Binding Background}"
Command="{Binding ButtonCommand}">
</buttons:SfChip>
</StackLayout>
</ContentPage.Content>
</ContentPage>

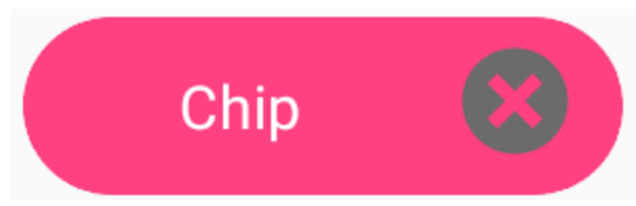
```

**C#**

```

// ViewModel class for Command Demo.
public class CommandDemoViewModel : INotifyPropertyChanged
{
    private Color _background = Color.Accent;
    public Color Background
    {
        get { return _background; }
        set
        {
            _background = value;
            NotifyPropertyChanged();
        }
    }
    private void NotifyPropertyChanged([CallerMemberName] String propertyName =
        "")
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
    public event PropertyChangedEventHandler PropertyChanged;
    public CommandDemoViewModel()
    {
        BackgroundColor();
        this.Background=Color.Accent;
    }
    private void BackgroundColor()
    {
        this.Background = this.Background == Color.DeepSkyBlue ? Color.Accent :
        Color.DeepSkyBlue;
    }
    public ICommand ButtonCommand => new Command(BackgroundColor);
}

```



**Note:** The default value of Command is [null].

### Customization of SfChipGroup

The chip group supports to customize the chip's background color, border color, text color, and more. The chip group can be customized using the following properties:

### InputView

The [InputView](#) property allows to provide a view to the input chip. In this example, the input chip is used to add an employee at run time. To get the employee name as input, an entry is added as InputView.

### XML

```
<ContentPage
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Chips"
x:Class="Chips.GettingStarted">
  <ContentPage.BindingContext>
  <local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
  <buttons:SfChipGroup
x:Name="chipGroup"
ItemsSource="{Binding Employees}"
ChipPadding="8,8,0,0"
DisplayMemberPath="Name">
  <buttons:SfChipGroup.ChipLayout>
  <FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
  </buttons:SfChipGroup.ChipLayout>
  <buttons:SfChipGroup.InputView>
  <Entry
Margin="10,10,0,0"
WidthRequest="110"
Completed="Entry_Completed"/>
  </buttons:SfChipGroup.InputView>
  </buttons:SfChipGroup>
</ContentPage.Content>
</ContentPage>
```

### C#

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace Chips
{
    public partial class GettingStarted: ContentPage
    {
        public GettingStarted()
        {
            InitializeComponent();
            Grid grid = new Grid();
            SfChipGroup chipGroup = new SfChipGroup();
            grid.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            var entry= new Entry { Margin = new Thickness(10, 10, 0, 0), WidthRequest = 110 };
            entry.Completed += Entry_Completed;
            chipGroup.InputView = entry;
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
            chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
            chipGroup.DisplayMemberPath = "Name";
            chipGroup.ChipPadding = new Thickness(8, 8, 0, 0);
            this.Content = grid;
        }
    }
}

```



**Note:** The InputView is visible only in the Input type. The default value of InputView is [null].

[SelectedChipBackgroundColor](#)

The [SelectedChipBackgroundColor](#) property customizes the background color of the selected chip.

#### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.BindingContext>

```

```

<local:ViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout Margin="10,10,10,10">
<buttons:SfChipGroup
ItemsSource="{Binding Employees}"
DisplayMemberPath="Name"
SelectedChipBackgroundColor="Fuchsia"
Type="Choice">
</buttons:SfChipGroup>
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stack = new StackLayout();
SfChipGroup chipGroup = new SfChipGroup();
stack.Children.Add(chipGroup);
FlexLayout layout = new FlexLayout()
{
Direction = FlexDirection.Row,
Wrap = FlexWrap.Wrap,
HorizontalOptions = LayoutOptions.Start,
VerticalOptions = LayoutOptions.Center,
AlignContent = FlexAlignContent.Start,
JustifyContent = FlexJustify.Start,
AlignItems = FlexAlignItems.Start,
};
chipGroup.ChipLayout = layout;
this.BindingContext = new ViewModel();
chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
chipGroup.DisplayMemberPath = "Name";
chipGroup.SelectedChipBackgroundColor = Color.Fuchsia;
chipGroup.Type = SfChipsType.Choice;
this.Content = stack;
}
}
}

```

John

James

Jacob



---

**Note:** The default value of `SelectedChipBackgroundColor` is `[Color.Accent]`.

---

### SelectedChipTextColor

The [SelectedChipTextColor](#) property customizes the text color of the selected chip.

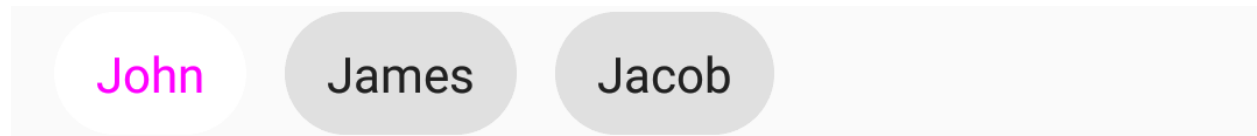
### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
  <ContentPage.BindingContext>
    <local:ViewModel/>
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <StackLayout Margin="10,10,10,10">
      <buttons:SfChipGroup
        ItemsSource="{Binding Employees}"
        DisplayMemberPath="Name"
        SelectedChipBackgroundColor="White"
        SelectedChipTextColor="Fuchsia"
        Type="Choice">
      </buttons:SfChipGroup>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stack = new StackLayout();
            SfChipGroup chipGroup = new SfChipGroup();
            stack.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
        }
    }
}
```

```
chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
chipGroup.DisplayMemberPath = "Name";
chipGroup.SelectedChipBackgroundColor = Color.White;
chipGroup.SelectedChipTextColor = Color.Fuchsia;
chipGroup.Type = SfChipsType.Choice;
this.Content = stack;
}
}
}
```



**Note:** The default value of SelectedChipTextColor is [Color.White].

### ChipBackgroundColor

The [ChipBackgroundColor](#) property customizes the background color of the SfChipGroup.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
  <ContentPage.BindingContext>
    <local:ViewModel/>
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <StackLayout Margin="10,10,10,10">
      <buttons:SfChipGroup
ItemsSource="{Binding Employees}"
DisplayMemberPath="Name"
ChipBackgroundColor="Aqua"
Type="Choice">
      </buttons:SfChipGroup>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

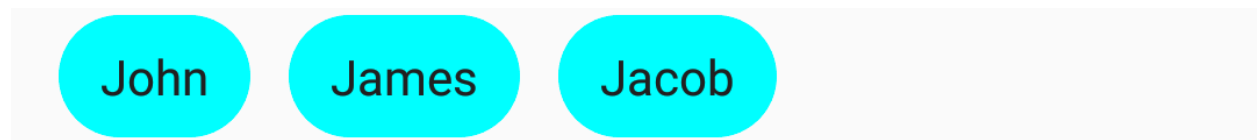
### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stack = new StackLayout();
```

```

SfChipGroup chipGroup = new SfChipGroup();
stack.Children.Add(chipGroup);
FlexLayout layout = new FlexLayout()
{
    Direction = FlexDirection.Row,
    Wrap = FlexWrap.Wrap,
    HorizontalOptions = LayoutOptions.Start,
    VerticalOptions = LayoutOptions.Center,
    AlignContent = FlexAlignContent.Start,
    JustifyContent = FlexJustify.Start,
    AlignItems = FlexAlignItems.Start,
};
chipGroup.ChipLayout = layout;
this.BindingContext = new ViewModel();
chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
chipGroup.DisplayMemberPath = "Name";
chipGroup.ChipBackgroundColor = Color.Aqua;
chipGroup.Type = SfChipsType.Choice;
this.Content = stack;
}
}
}

```



**Note:** The default value of ChipBackgroundColor is [Color.FromHex("#E0E0E0")].

ChipBorderColor

The [ChipBorderColor](#) property customizes the border color of the SfChipGroup.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
    <ContentPage.BindingContext>
        <local:ViewModel/>
    </ContentPage.BindingContext>
    <ContentPage.Content>
        <StackLayout Margin="10,10,10,10">
            <buttons:SfChipGroup
ItemsSource="{Binding Employees}"
DisplayMemberPath="Name"
ChipBorderWidth="5"
ChipBorderColor="Black"
ChipBackgroundColor="Aqua">
            </buttons:SfChipGroup>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>

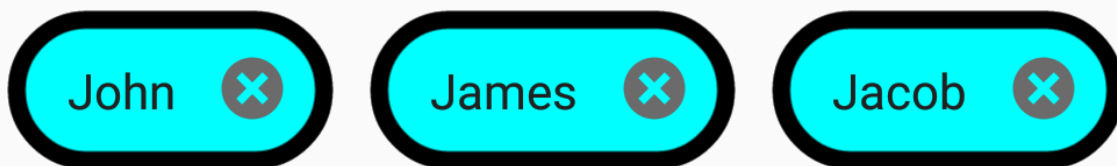
```

**C#**

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stack = new StackLayout();
            SfChipGroup chipGroup = new SfChipGroup();
            stack.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
            chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
            chipGroup.DisplayMemberPath = "Name";
            chipGroup.ChipBorderWidth = 5;
            chipGroup.ChipBorderColor = Color.Black;
            chipGroup.ChipBackgroundColor = Color.Aqua;
            this.Content = stack;
        }
    }
}

```



**Note:** The default value of ChipBorderColor is [Color.Transparent].

**ChipTextColor**

The [ChipTextColor](#) property customizes the text color of the SfChipGroup.

**XML**

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"

```

```

xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.BindingContext>
<local:ViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout Margin="10,10,10,10">
<buttons:SfChipGroup
ItemsSource="{Binding Employees}"
DisplayMemberPath="Name"
ChipTextColor="Blue">
</buttons:SfChipGroup>
<buttons:SfChipGroup.ChipLayout>
<FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stack = new StackLayout();
            SfChipGroup chipGroup = new SfChipGroup();
            stack.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
            chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
            chipGroup.DisplayMemberPath = "Name";
        }
    }
}

```

```
chipGroup.ChipTextColor = Color.Blue;
this.Content = stack;
}
}
}
```



**Note:** The default value of ChipTextColor is [Color.FromHex("#212121")].

### ChipTextSize

The [ChipTextSize](#) property customizes the text size of the SfChipGroup.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.BindingContext>
<local:ViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout Margin="8,8,0,0">
<buttons:SfChipGroup
ItemsSource="{Binding Employees}"
DisplayMemberPath="Name"
ChipTextSize="10">
</buttons:SfChipGroup>
<buttons:SfChipGroup.ChipLayout>
<FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
public partial class MainPage : ContentPage
```

```

{
public MainPage()
{
InitializeComponent();
StackLayout stack = new StackLayout();
SfChipGroup chipGroup = new SfChipGroup();
stack.Children.Add(chipGroup);
FlexLayout layout = new FlexLayout()
{
Direction = FlexDirection.Row,
Wrap = FlexWrap.Wrap,
HorizontalOptions = LayoutOptions.Start,
VerticalOptions = LayoutOptions.Center,
AlignContent = FlexAlignContent.Start,
JustifyContent = FlexJustify.Start,
AlignItems = FlexAlignItems.Start,
};
chipGroup.ChipLayout = layout;
this.BindingContext = new ViewModel();
chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
chipGroup.DisplayMemberPath = "Name";
chipGroup.ChipTextSize = 10;
this.Content = stack;
}
}
}

```



**Note:** The default value of `ChipTextSize` is [14d].

### ChipPadding

The [ChipPadding](#) property sets spacing between each chip.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.BindingContext>
<local:ViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout Margin="8,8,0,0">
<buttons:SfChipGroup
ItemsSource="{Binding Employees}"
DisplayMemberPath="Name"
ChipPadding="8,0,0,0">
</buttons:SfChipGroup>
<buttons:SfChipGroup.ChipLayout>
<FlexLayout

```

```

HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
</StackLayout>
</ContentPage.Content>
</ContentPage>

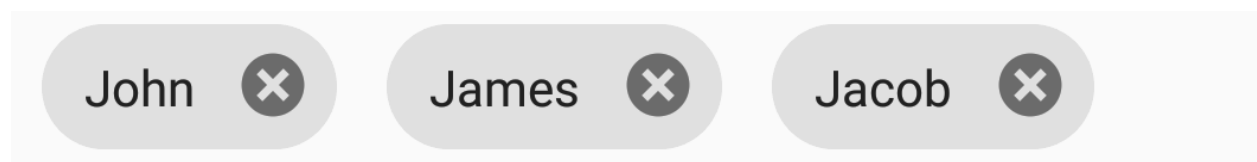
```

**C#**

```

using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stack = new StackLayout();
            SfChipGroup chipGroup = new SfChipGroup();
            stack.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
            chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
            chipGroup.ChipPadding = new Thickness(8, 8, 0, 0);
            chipGroup.DisplayMemberPath = "Name";
            this.Content = stack;
        }
    }
}

```



**Note:** The default value of ChipPadding is [Thickness(5d, 0, 0, 0)].



## ChipBorderWidth

The [ChipBorderWidth](#) property customizes the border width of the SfChipGroup.

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.BindingContext>
<local:ViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout Margin="8,8,0,0">
<buttons:SfChipGroup
ItemsSource="{Binding Employees}"
ChipBorderWidth="5"
DisplayMemberPath="Name"
ChipBorderColor="Black">
</buttons:SfChipGroup>
<buttons:SfChipGroup.ChipLayout>
<FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

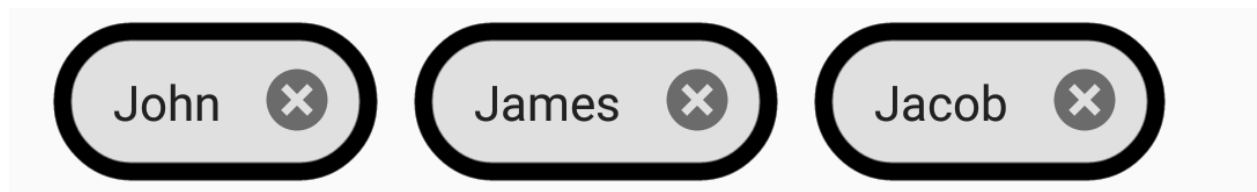
**C#**

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stack = new StackLayout();
SfChipGroup chipGroup = new SfChipGroup();
stack.Children.Add(chipGroup);
FlexLayout layout = new FlexLayout()
{
Direction = FlexDirection.Row,
Wrap = FlexWrap.Wrap,
HorizontalOptions = LayoutOptions.Start,
```

```

VerticalOptions = LayoutOptions.Center,
AlignContent = FlexAlignContent.Start,
JustifyContent = FlexJustify.Start,
AlignItems = FlexAlignItems.Start,
};
chipGroup.ChipLayout = layout;
this.BindingContext = new ViewModel();
chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
chipGroup.ChipBorderWidth = 5;
chipGroup.ChipBorderColor = Color.Black;
chipGroup.DisplayMemberPath = "Name";
this.Content = stack;
}
}
}

```



**Note:** The default value of ChipBorderWidth is [0d].

#### ItemHeight

The [ItemHeight](#) property customizes the height of the items in the SfChipGroup.

#### XML

```

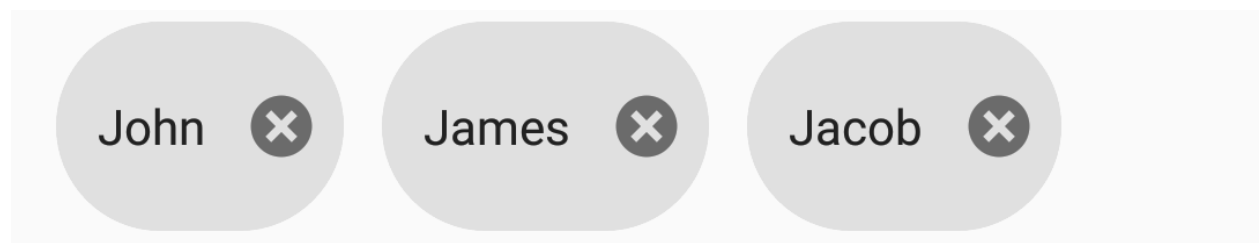
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
  <ContentPage.BindingContext>
    <local:ViewModel/>
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <StackLayout Margin="8,8,0,0">
      <buttons:SfChipGroup
ItemsSource="{Binding Employees}"
ItemHeight="60"
DisplayMemberPath="Name">
      </buttons:SfChipGroup>
    <buttons:SfChipGroup.ChipLayout>
      <FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
    </buttons:SfChipGroup.ChipLayout>
  </StackLayout>

```

```
</ContentPage.Content>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stack = new StackLayout();
            SfChipGroup chipGroup = new SfChipGroup();
            stack.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
            chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
            chipGroup.ItemHeight = 60;
            chipGroup.DisplayMemberPath = "Name";
            this.Content = stack;
        }
    }
}
```



**Note:** The default value of ItemHeight is [double.NaN].

## ShowIcon

You can enable the icon image using the [ShowIcon](#) property to know whether any image appears on the SfChipGroup.

## XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
```

```

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<Grid>
<buttons:SfChipGroup
ItemsSource="{Binding Employees}"
ChipPadding="8,8,0,0"
ImageMemberPath="Image"
ShowIcon="true"
ChipBackgroundColor="Aqua"
DisplayMemberPath="Name">
<buttons:SfChipGroup.ChipLayout>
<FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
</buttons:SfChipGroup>
</Grid>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

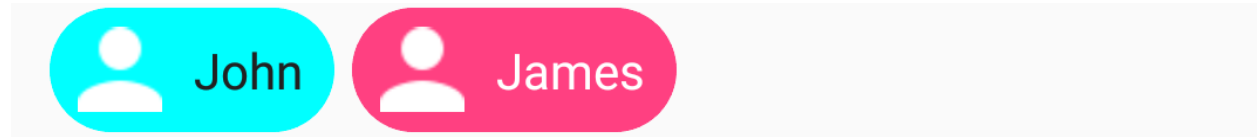
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stack = new StackLayout();
SfChipGroup chipGroup = new SfChipGroup();
stack.Children.Add(chipGroup);
FlexLayout layout = new FlexLayout()
{
Direction = FlexDirection.Row,
Wrap = FlexWrap.Wrap,
HorizontalOptions = LayoutOptions.Start,
VerticalOptions = LayoutOptions.Center,
AlignContent = FlexAlignContent.Start,
JustifyContent = FlexJustify.Start,
AlignItems = FlexAlignItems.Start,

```

```
};
chipGroup.ChipLayout = layout;
this.BindingContext = new ViewModel();
chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
chipGroup.DisplayMemberPath = "Name";
chipGroup.ImageMemberPath = "Image";
chipGroup.ChipImageWidth = 30;
chipGroup.SelectionIndicatorColor = Color.Black;
chipGroup.CloseButtonColor = Color.White;
chipGroup.ChipBackgroundColor = Color.Aqua;
chipGroup.Type = SfChipsType.Input;
chipGroup.ShowIcon = true;
chipGroup.ChipPadding = new Thickness(8, 8, 0, 0);
this.Content = stack;
}
}
public class Person
{
    public string Name
    {
        get;
        set;
    }
    public string Image
    {
        get;
        set;
    }
}
public class ViewModel : INotifyPropertyChanged
{
    private ObservableCollection<Person> employees;
    public ObservableCollection<Person> Employees
    {
        get
        {
            return employees;
        }
        set
        {
            Employees = value;
            OnPropertyChanged("Employees");
        }
    }
    public ViewModel()
    {
        employees = new ObservableCollection<Person>();
        employees.Add(new Person() { Image = "ChipUserContact.png", Name = "John" });
        employees.Add(new Person() { Image = "ChipUserContact.png", Name = "James" });
    }
    public event PropertyChangedEventHandler PropertyChanged;
    public void OnPropertyChanged(string property)
    {
        if (PropertyChanged != null)
        {

```

```
PropertyChanged(this, new PropertyChangedEventArgs(property));
}
}
}
```



### CloseButtonColor

The [CloseButtonColor](#) property customizes the color of close button in the SfChipGroup.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout Margin="8,8,0,0">
<buttons:SfChipGroup
ItemsSource="{Binding Employees}"
ImageMemberPath="Image"
Type="Input"
ChipImageWidth="30"
ShowIcon="true"
CloseButtonColor="White"
ChipBackgroundColor="Aqua"
DisplayMemberPath="Name">
<buttons:SfChipGroup.ChipLayout>
<FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
</buttons:SfChipGroup>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
```

```

namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stack = new StackLayout();
            SfChipGroup chipGroup = new SfChipGroup();
            stack.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
            chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
            chipGroup.DisplayMemberPath = "Name";
            chipGroup.ImageMemberPath = "Image";
            chipGroup.ChipImageWidth = 30;
            chipGroup.SelectionIndicatorColor = Color.Black;
            chipGroup.CloseButtonColor = Color.White;
            chipGroup.ChipBackgroundColor = Color.Aqua;
            chipGroup.Type = SfChipsType.Input;
            chipGroup.ShowIcon = true;
            this.Content = stack;
        }
    }
}

```

---

**Note:** The default value of CloseButtonColor is [Color.Black].

---

### SelectionIndicatorColor

The [SelectionIndicatorColor](#) property customizes the selection indicator color of the SfChipGroup.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
    <ContentPage.BindingContext>
        <local:ViewModel x:Name="viewModel"/>
    </ContentPage.BindingContext>
    <ContentPage.Content>
        <StackLayout Margin="8,8,0,0">
            <buttons:SfChipGroup
ItemsSource="{Binding Employees}"

```

```
Type="Filter"
SelectionIndicatorColor="Black"
CloseButtonColor="White"
ChipBackgroundColor="Aqua"
DisplayMemberPath="Name">
<buttons:SfChipGroup.ChipLayout>
<FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
</buttons:SfChipGroup>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stack = new StackLayout();
            SfChipGroup chipGroup = new SfChipGroup();
            stack.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
            chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
            chipGroup.DisplayMemberPath = "Name";
            chipGroup.SelectionIndicatorColor = Color.Black;
            chipGroup.CloseButtonColor = Color.White;
            chipGroup.ChipBackgroundColor = Color.Aqua;
            chipGroup.Type = SfChipsType.Filter;
            this.Content = stack;
        }
    }
}
```



}



**Note:** The default value of `SelectionIndicatorColor` is `[Color.White]`.

### ChipImageWidth

The `[ChipImageWidth]` property customizes the width of icon image in the `SfChipGroup`.

### XML

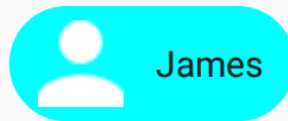
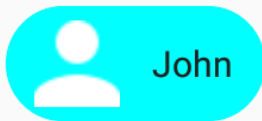
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ChipCustomization"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="ChipCustomization.MainPage">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout Margin="8,8,0,0">
<buttons:SfChipGroup
ItemsSource="{Binding Employees}"
ImageMemberPath="Image"
Type="Choice"
ChipImageWidth="30"
ShowIcon="true"
ChipBackgroundColor="Aqua"
DisplayMemberPath="Name">
<buttons:SfChipGroup.ChipLayout>
<FlexLayout
HorizontalOptions="Start"
VerticalOptions="Center"
Direction="Row"
Wrap="Wrap"
JustifyContent="Start"
AlignContent="Start"
AlignItems="Start"/>
</buttons:SfChipGroup.ChipLayout>
</buttons:SfChipGroup>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Syncfusion.XForms.Buttons;
```

```
using Xamarin.Forms;
namespace ChipCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stack = new StackLayout();
            SfChipGroup chipGroup = new SfChipGroup();
            stack.Children.Add(chipGroup);
            FlexLayout layout = new FlexLayout()
            {
                Direction = FlexDirection.Row,
                Wrap = FlexWrap.Wrap,
                HorizontalOptions = LayoutOptions.Start,
                VerticalOptions = LayoutOptions.Center,
                AlignContent = FlexAlignContent.Start,
                JustifyContent = FlexJustify.Start,
                AlignItems = FlexAlignItems.Start,
            };
            chipGroup.ChipLayout = layout;
            this.BindingContext = new ViewModel();
            chipGroup.SetBinding(SfChipGroup.ItemsSourceProperty, "Employees");
            chipGroup.DisplayMemberPath = "Name";
            chipGroup.ImageMemberPath = "Image";
            chipGroup.ChipImageWidth = 30;
            chipGroup.ChipBackgroundColor = Color.Aqua;
            chipGroup.Type = SfChipsType.Choice;
            chipGroup.ShowIcon = true;
            this.Content = stack;
        }
    }
    // Model Class
    public class Person
    {
        public string Name
        {
            get;
            set;
        }
        public string Image
        {
            get;
            set;
        }
    }
    //ViewModel Class
    public class ViewModel : INotifyPropertyChanged
    {
        private ObservableCollection<Person> employees;
        public ObservableCollection<Person> Employees
        {
            get
            {
                return employees;
            }
        }
    }
}
```

```
set
{
    Employees = value;
    OnPropertyChanged("Employees");
}
}
public ViewModel()
{
    employees = new ObservableCollection<Person>();
    employees.Add(new Person() { Image = "ChipUserContact.png", Name = "John"
    });
    employees.Add(new Person() { Image = "ChipUserContact.png", Name = "James"
    });
}
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string property)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(property));
    }
}
}
```



**Note:** The default value of `ChipImageWidth` is [26].

## SfCircularGauge

### Overview

Essential Gauge for Xamarin.Forms helps you to visualize the numeric values over a circular scale. The appearance of the gauge is fully customized to integrate your applications without fault.

### Key features

[SfCircularGauge](#) is a composed control of several scales. Scales will be an integrated UI part of the circular gauge.

[SfCircularGauge](#) is a composite UI element with the following subparts:

- Scales
- Ranges
- Pointers
- Headers
- Annotations

The circular gauge control is highly customizable control with variety of simple APIs to modify its basic look and feel. You can position ranges, ticks, labels, and range pointers as needed.

### Scales

Circular gauge scale contains labels, tick marks, and a rim to customize its basic look and feel. It defines the start angle, sweep direction, sweep angle, overall minimum and maximum values, the frequency of labels, and tick marks.

**Ranges**

Range is a visual element that depicts the start and end values of inner divisions within the scale's range. Each scale is capable of displaying one or more ranges, and each range can depict different zones or regions of same metrics, such as high, low, and average temperatures.

**Pointers**

Pointer is an element that points out the values of the bound property on a scale. A circular scale will have one or more pointers that can be used to measure different values. Each pointer has the **Value** property, which informs the current value to the user visually.

**Headers**

Header can be used to set a unique header for the circular gauge. The user can add text as the header in a circular gauge.

**Annotations**

Annotations allows you to mark the specific area of interest in a circular gauge. You can place custom views, text, and images as annotations by using the annotations feature.



## Getting Started

This section explains the steps required to configure the [SfCircularGauge](#), and also explains the steps required to add basic elements to [SfCircularGauge](#) through various APIs available within it.

To get start quickly with Xamarin Circular Gauge control, you can check on this video:

```
<style>#XamarinGaugeVideoTutorial{width : 90% !important; height: 300px !important }</style>
```

```
<iframe id='XamarinGaugeVideoTutorial'  
src='https://www.youtube.com/embed/Rjd9NxDFoWo'></iframe>
```

## Adding SfCircularGauge reference

You can add SfCircularGauge reference using one of the following methods:

### Method 1: Adding SfCircularGauge reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfCircularGauge to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfGauge](#), and then install it.

![Adding SfCircularGauge reference from NuGet](getting-started\_images/Adding SfCircularGauge reference.png)

---

**Note:** Install the same version of SfCircularGauge NuGet in all the projects.

---

### Method 2: Adding SfCircularGauge reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfCircularGauge control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfCircularGauge assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfGauge.Android.dll Syncfusion.SfGauge.XForms.Android.dll Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfGauge.iOS.dll Syncfusion.SfGauge.XForms.iOS.dll Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfGauge.UWP.dll Syncfusion.SfGauge.XForms.UWP.dll Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Launching an application on each platform with SfCircularGauge.

To use the SfCircularGauge control inside an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

### iOS

To launch the SfCircularGauge in iOS, call the `SfGaugeRenderer.Init()` in the `FinishedLaunching` overridden method of the AppDelegate class after the Xamarin.Forms Framework has been initialized and before the `LoadApplication` is called, as demonstrated in the following code example.

### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.SfGauge.XForms.iOS.SfGaugeRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

### Universal Windows Platform (UWP)

You need to initialize the circular gauge view assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with circular gauge in Release mode in UWP platform.

### C#

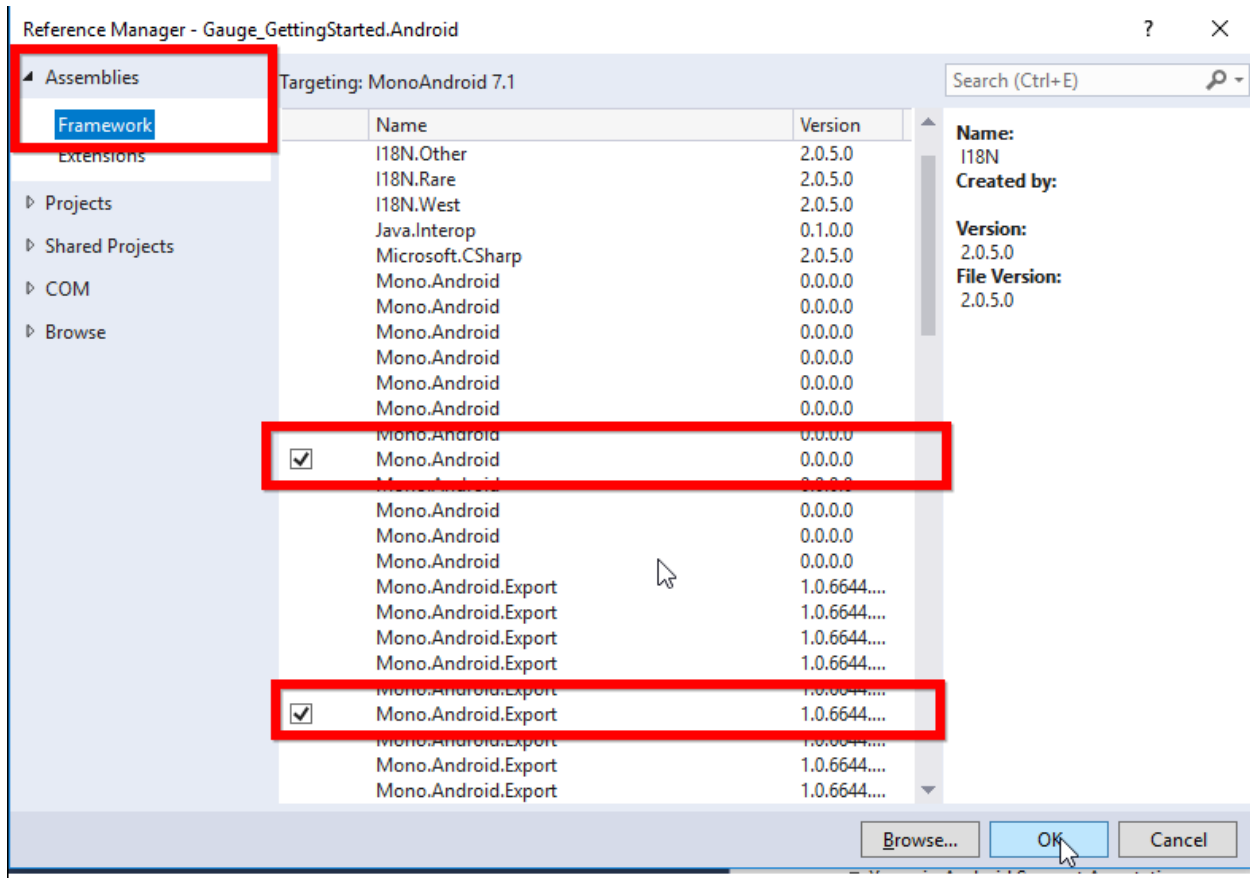
```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    // Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.SfGauge.XForms.UWP.SfGaugeRenderer)
        .GetTypeInfo().Assembly);
    // Replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Android

The Android platform does not require any additional configuration to render the circular gauge.

### Reference Mono.Android.Export

1. In the Solution Explorer in the Android project, right-click on References and choose Add Reference.
2. In the Add Reference window, select the Assemblies tab and choose the Framework.
3. In the Framework tab, ensure Mono.Android and Mono.Android.Export is checked and click ok.



*Adding namespace for the assemblies*

#### **XML**

```
xmlns:gauge="clr-namespace:Syncfusion.SfGauge.XForms;assembly=Syncfusion.SfGauge.XForms"
```

#### **C#**

```
using Syncfusion.SfGauge.XForms;
```

#### Initialize gauge

You can initialize the [SfCircularGauge](#) control with a required optimal name by using the included namespace.

#### **XML**

```
<gauge:SfCircularGauge/>
```

#### **C#**

```
SfCircularGauge circularGauge = new SfCircularGauge ();  
this.Content = circularGauge;
```



### Adding header

You can assign a unique header to [SfCircularGauge](#) by using the [Header](#) property and position it by using the [Position](#) property as you want.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Headers>
    <gauge:Header Text="Speedometer" ForegroundColor="Black" TextSize="20" />
  </gauge:SfCircularGauge.Headers>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
Header header = new Header();
header.Text = "Speedometer";
header.ForegroundColor = Color.Black;
header.TextSize = 20;
circularGauge.Headers.Add(header);
```

### Configuring scales

You can configure the [Scale](#) elements by using following APIs, which are available in [SfCircularGauge](#):

- StartAngle
- SweepAngle
- StartValue
- EndValue
- Interval
- RimThickness
- RimColor

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale />
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
ObservableCollection<Scale> scales = new ObservableCollection<Scale>>();
Scale scale = new Scale();
scales.Add(scale);
circularGauge.Scales = scales;
```

### Adding ranges

You can add ranges to [SfCircularGauge](#) by creating ranges collection using the [Range](#) property.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Ranges>
        <gauge:Range StartValue="0" EndValue="40"/>
      </gauge:Scale.Ranges>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
Range range = new Range();
range.StartValue = 0;
range.EndValue = 40;
scale.Ranges.Add(range);
scales.Add(scale);
circularGauge.Scales = scales;
```

### Adding a needle pointer

Create a [Needle Pointer](#), and associate it with a scale that is to be displayed the current value.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:NeedlePointer Value="60" />
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 60;
scale.Pointers.Add(needlePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```

### Adding a range pointer

[Range Pointer](#) provides an alternative way to indicate the current value.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
```

```
<gauge:RangePointer Value="60" />
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
ObservableCollection<Scale> scales = new ObservableCollection<Scale>>();
Scale scale = new Scale();
RangePointer rangePointer = new RangePointer();
rangePointer.Value = 60;
scale.Pointers.Add(rangePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```

Adding a marker pointer

[Marker Pointer](#) points the current value in scale.

**XML**

```
<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Scales>
<gauge:Scale>
<gauge:Scale.Pointers>
<gauge:MarkerPointer Value="70" />
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale=new Scale();
MarkerPointer markerPointer = new MarkerPointer();
markerPointer.Value = 70;
scale.Pointers.Add(markerPointer);
scales.Add(scale);
circularGauge.Scales = scales;
```

The following code example gives you the complete code of above configurations.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:gauge="clr-
namespace:Syncfusion.SfGauge.XForms;assembly=Syncfusion.SfGauge.XForms"
xmlns:local="clr-namespace:CircularGauge;assembly=CircularGauge"
x:Class="CircularGauge.UGSample">
<gauge:SfCircularGauge VerticalOptions="FillAndExpand"
HorizontalOptions="FillAndExpand" Margin="10">
<gauge:SfCircularGauge.Headers>
```

```

<gauge:Header Text="Speedometer" ForegroundColor="Black" TextSize="20" />
</gauge:SfCircularGauge.Headers>
<gauge:SfCircularGauge.Scales>
<gauge:Scale>
<gauge:Scale.Ranges>
<gauge:Range StartValue="0" EndValue="40"/>
</gauge:Scale.Ranges>
<gauge:Scale.Pointers>
<gauge:NeedlePointer Value="60" />
<gauge:RangePointer Value="60" />
<gauge:MarkerPointer Value="70" />
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
</ContentPage>

```

**C#**

```

using Syncfusion.SfGauge.XForms;
namespace CircularGauge
{
public partial class UGSample : ContentPage
{
public UGSample()
{
InitializeComponent();
//Initializing circular gauge
SfCircularGauge circularGauge = new SfCircularGauge();
circularGauge.Margin = 10;
//Adding header
Header header = new Header();
header.Text = "Speedometer";
header.ForegroundColor = Color.Black;
header.TextSize = 20;
circularGauge.Headers.Add(header);
//Initializing scales for circular gauge
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scales.Add(scale);
//Adding range
Range range = new Range();
range.StartValue = 0;
range.EndValue = 40;
scale.Ranges.Add(range);
//Adding needle pointer
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 60;
scale.Pointers.Add(needlePointer);
//Adding range pointer
RangePointer rangePointer = new RangePointer();
rangePointer.Value = 60;
scale.Pointers.Add(rangePointer);
//Adding marker pointer
MarkerPointer markerPointer = new MarkerPointer();
markerPointer.Value = 70;

```

```
scale.Pointers.Add(markerPointer);  
scales.Add(scale);  
circularGauge.Scales = scales;  
this.Content = circularGauge;  
}  
}  
}
```

The following circular gauge is created as a result of the above codes.



You can find the complete getting started sample from this [link](#).

## Scales

Scales contain a collection of [Scale](#) elements, which integrates labels, tick marks, and a rim to customize the basic look and feel of the circular gauge.

## Scale

[Scale](#) contains the sub elements such as rim, ticks, labels, ranges, and pointers. It defines the radius, start angle, sweep direction, sweep angle, overall minimum and maximum values, frequency of labels, and tick marks. It will have multiple ranges.

A range is a visual element, which begins and ends at specified values within a [Scale](#). It will have one or more pointers to point out the values in scale.

#### XML

```
<gauge:SfCircularGauge>  
  <gauge:SfCircularGauge.Scales>  
    <gauge:Scale/>  
  </gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();  
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();  
Scale scale = new Scale();  
scales.Add(scale);  
circularGauge.Scales = scales;
```



Setting start and end values for scale

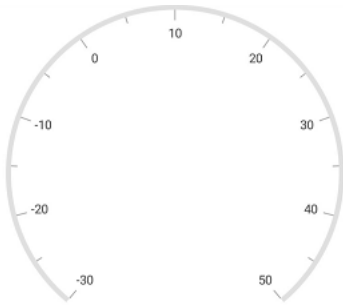
The [StartValue](#) and [EndValue](#) properties allow you to set the start and end values for scale.

#### XML

```
<gauge:SfCircularGauge>  
  <gauge:SfCircularGauge.Scales>  
    <gauge:Scale StartValue="-30" EndValue="50"/>  
  </gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();  
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();  
Scale scale = new Scale();  
scale.StartValue = -30;  
scale.EndValue = 50;  
scales.Add(scale);  
circularGauge.Scales = scales;
```



Setting start and sweep angles for scale

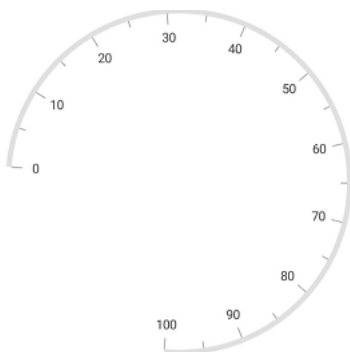
The [StartAngle](#) and [SweepAngle](#) properties allow you to set the start and end angles for scale.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale StartAngle="185" SweepAngle="270"/>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.StartAngle = 185;
scale.SweepAngle = 270;
scales.Add(scale);
circularGauge.Scales = scales;
```



Setting interval for scale

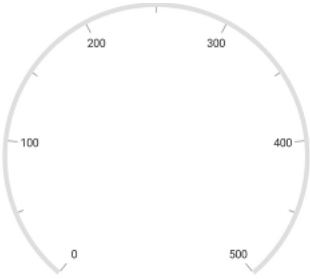
The [Interval](#) property allows you to set the interval for scale.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale StartValue = "0" EndValue = "500" Interval = "100" />
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();  
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();  
Scale scale = new Scale();  
scale.StartValue = 0;  
scale.EndValue = 500;  
scale.Interval = 100;  
scales.Add(scale);  
circularGauge.Scales = scales;
```

**Setting auto interval for scale**

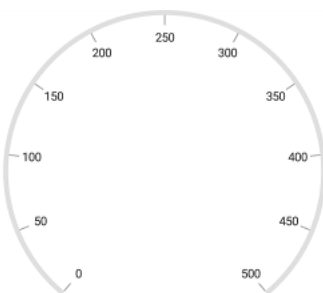
By default, the interval value is calculated by 10. By using the [EnableAutoInterval](#) property, you can set auto interval based on the start and end values.

**XML**

```
<gauge:SfCircularGauge>  
  <gauge:SfCircularGauge.Scales>  
    <gauge:Scale StartValue="0" EndValue="500" EnableAutoInterval = "True"/>  
  </gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();  
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();  
Scale scale = new Scale();  
scale.StartValue = 0;  
scale.EndValue = 500;  
scale.EnableAutoInterval = true;  
scales.Add(scale);  
circularGauge.Scales = scales;
```





### Setting scale direction for scale

The [Direction](#) property allows you to render the gauge scale in either clockwise or counterclockwise direction.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale Direction="AntiClockwise"/>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.Direction = ScaleDirection.AntiClockwise;
scales.Add(scale);
circularGauge.Scales = scales;
```



### Setting maximum labels

The [MaximumLabels](#) property defines the count of the scale labels in 100 pixels. By default, the count of maximum labels for 100 pixels is 3.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale EndValue="200" MaximumLabels="4" MinorTicksPerInterval="1" />
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.EndValue = 200;
scale.MaximumLabels = 4;
scale.MinorTicksPerInterval = 1;
scales.Add(scale);
sfCircularGauge.Scales = scales;
Content = sfCircularGauge;
```



### Setting multiple scales for scale

It helps you to add multiple scales to the same circular gauge. You can customize all the scales in a [Scales](#) collection.

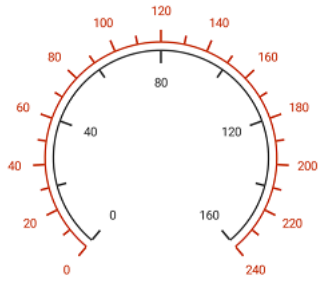
#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale x:Name="scale1"
      StartValue="0" EndValue="240" Interval="20" MinorTicksPerInterval="1"
      RimColor="#C62E0A"
      LabelOffset="0.88" LabelColor="#C62E0A" ScaleStartOffset="0.7"
      ScaleEndOffset="0.69">
      <gauge:Scale.MajorTickSettings>
        <gauge:TickSettings StartOffset="0.7" EndOffset="0.77" Thickness="2"
          Color="#C62E0A"/>
      </gauge:Scale.MajorTickSettings>
      <gauge:Scale.MinorTickSettings>
        <gauge:TickSettings StartOffset="0.7" EndOffset="0.75" Thickness="2"
          Color="#C62E0A"/>
      </gauge:Scale.MinorTickSettings>
    </gauge:Scale>
    <gauge:Scale x:Name="scale2"
      StartValue="0" EndValue="160" Interval="40" MinorTicksPerInterval="1"
      RimColor="#333333"
      LabelOffset="0.45" LabelColor="#333333" ScaleStartOffset="0.65"
      ScaleEndOffset="0.64">
      <gauge:Scale.MajorTickSettings>
        <gauge:TickSettings StartOffset="0.64" EndOffset="0.57" Thickness="2"
          Color="#333333"/>
      </gauge:Scale.MajorTickSettings>
      <gauge:Scale.MinorTickSettings>
        <gauge:TickSettings StartOffset="0.64" EndOffset="0.59" Thickness="2"
          Color="#333333"/>
      </gauge:Scale.MinorTickSettings>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

```
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.StartValue = 0;
scale.EndValue = 240;
scale.Interval = 20;
scale.MinorTicksPerInterval = 1;
scale.RimColor = Color.FromHex("#C62E0A");
scale.LabelOffset = 0.88;
scale.LabelColor = Color.FromHex("#C62E0A");
scale.ScaleStartOffset = 0.7;
scale.ScaleEndOffset = 0.69;
TickSettings majorTicks = new TickSettings();
majorTicks.StartOffset = 0.7;
majorTicks.EndOffset = 0.77;
majorTicks.Thickness = 2;
majorTicks.Color = Color.FromHex("#C62E0A");
scale.MajorTickSettings = majorTicks;
TickSettings minorTicks = new TickSettings();
minorTicks.StartOffset = 0.7;
minorTicks.EndOffset = 0.75;
minorTicks.Thickness = 2;
minorTicks.Color = Color.FromHex("#C62E0A");
scale.MinorTickSettings = minorTicks;
scales.Add(scale);
Scale circularScale = new Scale();
circularScale.StartValue = 0;
circularScale.EndValue = 160;
circularScale.Interval = 40;
circularScale.MinorTicksPerInterval = 1;
circularScale.RimColor = Color.FromHex("#333333");
circularScale.LabelOffset = 0.45;
circularScale.LabelColor = Color.FromHex("#333333");
circularScale.ScaleStartOffset = 0.65;
circularScale.ScaleEndOffset = 0.64;
TickSettings majorTick = new TickSettings();
majorTick.StartOffset = 0.64;
majorTick.EndOffset = 0.57;
majorTick.Thickness = 2;
majorTick.Color = Color.FromHex("#333333");
circularScale.MajorTickSettings = majorTick;
TickSettings minorTick = new TickSettings();
minorTick.StartOffset = 0.64;
minorTick.EndOffset = 0.59;
minorTick.Thickness = 2;
minorTick.Color = Color.FromHex("#333333");
circularScale.MinorTickSettings = minorTick;
scales.Add(circularScale);
circularGauge.Scales = scales;
```



## Events

You can change the default label by hooking the [LabelCreated](#) event. Based on your requirements, the labels can be changed by using the `LabelContent` property of `LabelCreatedEventArgs`.

## XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale x:Name="scale" StartAngle="270" StartValue="0" EndValue="16"
      Interval="2"
      SweepAngle="360" MinorTicksPerInterval="1" ShowLastLabel="False"
      LabelCreated="scale_LabelCreated" />
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

## C#

```
private void scale_LabelCreated(object sender, LabelCreatedEventArgs args)
{
    switch ((string)args.LabelContent)
    {
        case "0":
            args.LabelContent = "N";
            break;
        case "2":
            args.LabelContent = "NE";
            break;
        case "4":
            args.LabelContent = "E";
            break;
        case "6":
            args.LabelContent = "SE";
            break;
        case "8":
            args.LabelContent = "S";
            break;
        case "10":
            args.LabelContent = "SW";
            break;
        case "12":
            args.LabelContent = "W";
            break;
        case "14":
            args.LabelContent = "NW";
            break;
    }
}
```

```
}
}
```



## Rim

Scale determines the structure of a circular gauge by using the circular rim. By setting the [StartAngle](#) and [SweepAngle](#) properties, you can change the shape of the circular gauge into a full circular gauge, half circular gauge, or quarter circular gauge.

The [StartValue](#) and [EndValue](#) properties will determine the overall range of the circular rim.

## XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale StartAngle="270" SweepAngle="360" StartValue="0" EndValue="360"
      Interval="20" MinorTicksPerInterval="0" ShowFirstLabel="False"/>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.StartAngle = 270;
scale.SweepAngle = 360;
scale.StartValue = 0;
scale.EndValue = 360;
scale.Interval = 20;
scale.MinorTicksPerInterval = 0;
scale.ShowFirstLabel = false;
scales.Add(scale);
circularGauge.Scales = scales;
```



### Rim customization

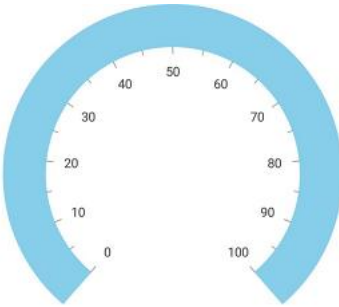
The color and thickness of rim can be set by using the [RimColor](#) and [RimThickness](#) properties. To increase the [RimThickness](#), set the [RadiusFactor](#).

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale RadiusFactor="1" RimThickness="40" RimColor="SkyBlue"
      LabelOffset = "0.6">
      <gauge:Scale.MajorTickSettings>
        <gauge:TickSettings Offset = "0.75"/>
      </gauge:Scale.MajorTickSettings>
      <gauge:Scale.MinorTickSettings>
        <gauge:TickSettings Offset = "0.75"/>
      </gauge:Scale.MinorTickSettings>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.RadiusFactor = 1;
scale.RimThickness = 40;
scale.LabelOffset = 0.6;
scale.MajorTickSettings.Offset = 0.75;
scale.MinorTickSettings.Offset = 0.75;
scale.RimColor = Color.SkyBlue;
scales.Add(scale);
circularGauge.Scales = scales;
```



Setting position for rim

You can customize the position of [Scales](#) in the following two ways:

1. [RadiusFactor](#) with the [RimThickness](#) property.
2. The [ScaleStartOffset](#) and [ScaleEndOffset](#) properties.

*Setting radius factor for rim*

#### XML

```
<gauge:SfCircularGauge>  
  <gauge:SfCircularGauge.Scales>  
    <gauge:Scale RadiusFactor = "0.7" RimThickness = "30">  
    </gauge:Scale>  
  </gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();  
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();  
Scale scale = new Scale();  
scale.RadiusFactor = 0.7;  
scale.RimThickness = 30;  
scales.Add(scale);  
circularGauge.Scales = scales;
```



*Setting scale start and end offsets for rim*

#### XML

```
<gauge:SfCircularGauge>  
  <gauge:SfCircularGauge.Scales>  
    <gauge:Scale ScaleStartOffset="0.6" ScaleEndOffset = "0.7">  
    </gauge:Scale>  
  </gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

```
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge circularGauge = new SfCircularGauge();  
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();  
Scale scale = new Scale();  
scale.ScaleStartOffset = 0.6;  
scale.ScaleEndOffset = 0.7;  
scales.Add(scale);  
circularGauge.Scales = scales;
```



## Show rim

The [ShowRim](#) property is a Boolean property, which is used to enable or disable the rim in circular gauge.

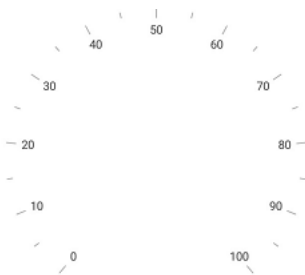
## XML

```
<gauge:SfCircularGauge>  
  <gauge:SfCircularGauge.Scales>  
    <gauge:Scale ShowRim = "False">  
    </gauge:Scale>  
  </gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge circularGauge = new SfCircularGauge();  
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();  
Scale scale = new Scale();  
scale.ShowRim = false;  
scales.Add(scale);  
circularGauge.Scales = scales;
```





## Tick Setting

The **TickSettings** property helps you to identify the gauge's data value by marking the gauge scale in regular increments.

### Show ticks for scale

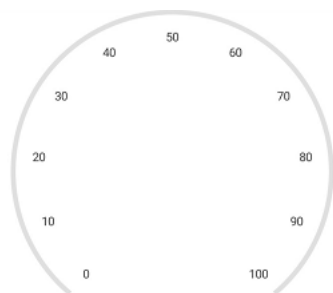
The [ShowTicks](#) property allows you to enable or disable the ticks of circular gauge.

## XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale ShowTicks="False"/>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.ShowTicks = false;
scales.Add(scale);
circularGauge.Scales = scales;
```



## Ticks customization

The [Interval](#) property is used to calculate the tick counts for a scale. Similar to ticks, minor ticks are calculated by using the [MinorTicksPerInterval](#) property.

Color and thickness of the tick are set by using the [Color](#) and [Thickness](#) UI properties. You can also customize the length of the ticks by using the [Length](#) property. First, you should set the **Offset** property for ticks, then increase the length of the ticks.

### Customize major ticks for scale

## XML

```

<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Scales>
<gauge:Scale>
<gauge:Scale.MajorTickSettings>
<gauge:TickSettings Color ="Brown" Thickness="4" Length="15" Offset="0.97"/>
</gauge:Scale.MajorTickSettings>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

**C#**

```

SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
TickSettings majorTicks = new TickSettings();
majorTicks.Length = 15;
majorTicks.Color = Color.Brown;
majorTicks.Thickness = 4;
majorTicks.Offset = 0.97;
scale.MajorTickSettings = majorTicks;
scales.Add(scale);
circularGauge.Scales = scales;

```



Customize minor ticks for scale

**XML**

```

<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Scales>
<gauge:Scale>
<gauge:Scale.MinorTickSettings>
<gauge:TickSettings Color ="SkyBlue" Thickness="4" Length="4" Offset =
"0.97" />
</gauge:Scale.MinorTickSettings>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

**C#**

```

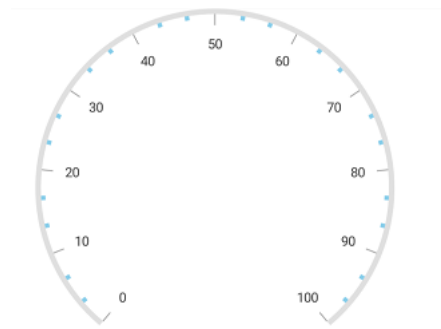
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
TickSettings minorTicks = new TickSettings();
minorTicks.Length = 4;

```

```

minorTicks.Color = Color.SkyBlue;
minorTicks.Thickness = 4;
minorTicks.Offset = 0.97;
scale.MinorTickSettings = minorTicks;
scales.Add(scale);
circularGauge.Scales = scales;

```



### Setting position for ticks

The major and minor ticks can be positioned far away from the rim by using the following two ways:

1. **Offset** property.
2. **StartOffset** and **EndOffset** properties.

### Setting offset for scale

#### XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.MajorTickSettings>
        <gauge:TickSettings Offset = "0.5"/>
      </gauge:Scale.MajorTickSettings>
      <gauge:Scale.MinorTickSettings>
        <gauge:TickSettings Offset = "0.5"/>
      </gauge:Scale.MinorTickSettings>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

#### C#

```

SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
TickSettings majorTicks = new TickSettings();
majorTicks.Offset = 0.5;
scale.MajorTickSettings = majorTicks;
TickSettings minorTicks = new TickSettings();
minorTicks.Offset = 0.5;
scale.MinorTickSettings = minorTicks;
scales.Add(scale);
circularGauge.Scales = scales;

```



*Setting scale start and end offset for scale*

#### **XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.MajorTickSettings>
        <gauge:TickSettings StartOffset = "0.3" EndOffset = "0.4"/>
      </gauge:Scale.MajorTickSettings>
      <gauge:Scale.MinorTickSettings>
        <gauge:TickSettings StartOffset = "0.3" EndOffset = "0.35"/>
      </gauge:Scale.MinorTickSettings>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### **C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
TickSettings majorTicks = new TickSettings();
majorTicks.StartOffset = 0.3;
majorTicks.EndOffset = 0.4;
scale.MajorTickSettings = majorTicks;
TickSettings minorTicks = new TickSettings();
minorTicks.StartOffset = 0.3;
minorTicks.EndOffset = 0.35;
scale.MinorTickSettings = minorTicks;
scales.Add(scale);
circularGauge.Scales = scales;
```



## Labels

The [Scale](#) labels associate a numeric value with major scale tick marks.

### Label color customization

The label color can be changed using the [LabelColor](#) property.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale LabelColor= "Blue"/ >
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.LabelColor = Color.Blue;
scales.Add(scale);
circularGauge.Scales = scales;
```



### Label font customization

The label font can be customized by using the [LabelFontSize](#), [FontAttribute](#), and [FontFamily](#) properties.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale FontAttributes = "Bold" LabelFontSize = "20" >
    <gauge:Scale.FontFamily>
      <OnPlatform x:String iOS = "Chalkduster" Android =
        "algerian.ttf" WinPhone="Chiller" />
    </gauge:Scale.FontFamily>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
```

```
scale.FontAttributes = FontAttributes.Bold;
scale.FontFamily = Device.RuntimePlatform == Device.iOS ? "Chalkduster" :
Device.RuntimePlatform == Device.Android ? "algerian.ttf" : "Chiller";
scale.LabelFontSize = 20;
scales.Add(scale);
circularGauge.Scales = scales;
```



### Setting position for labels

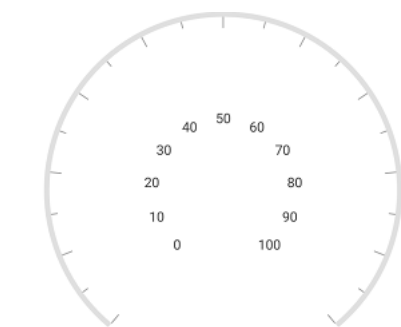
The labels can be positioned far away from the ticks by using the [LabelOffset](#) property.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale LabelOffset = "0.4"/>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.LabelOffset = 0.4;
scales.Add(scale);
circularGauge.Scales = scales;
```



### Setting number of decimal digits for labels

The [NumberOfDecimalDigits](#) property is used to set the number of decimal digits to be displayed in the scale labels.

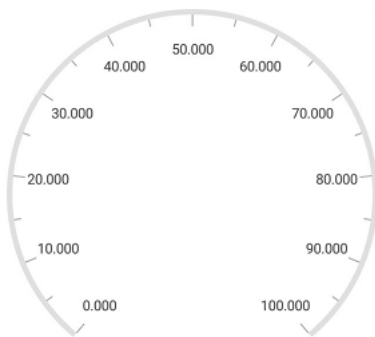
#### XML

```
<gauge:SfCircularGauge>
```

```
<gauge:SfCircularGauge.Scales>
<gauge:Scale NumberOfDecimalDigits = "3"/>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.NumberOfDecimalDigits = 3;
scales.Add(scale);
circularGauge.Scales = scales;
```



## Setting postfix and prefix for labels

You can postfix/prefix values to the scale labels by using the [LabelPostfix](#) and [LabelPrefix](#) properties, respectively.

*Label postfix*

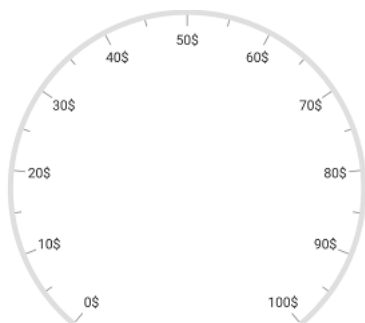
**LabelPostfix** property allows you to postfix the values to the scale labels.

**XML**

```
<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Scales>
<gauge:Scale LabelPostfix = "$" />
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.LabelPostfix = "$";
scales.Add(scale);
circularGauge.Scales = scales;
```



### Label prefix

**LabelPrefix** property allows you to prefix the values to the scale labels.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale LabelPrefix = "$" />
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.LabelPrefix = "$";
scales.Add(scale);
circularGauge.Scales = scales;
```



### Edge label customization

You can customize the edge label by using the [ShowFirstLabel](#) and [ShowLastLabel](#) properties, which are Boolean properties. The [ShowFirstLabel](#) property is used to enable or disable first label, and the [ShowLastLabel](#) property is used to enable or disable the last label in circular gauge.

### XML

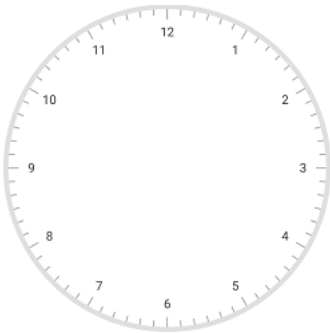
```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale ShowFirstLabel = "False" StartValue = "0" EndValue = "12"
      Interval = "1" MinorTicksPerInterval = "5" StartAngle = "270" SweepAngle =
        "360" />
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```



```
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.StartValue = 0;
scale.Interval = 1;
scale.MinorTicksPerInterval = 5;
scale.EndValue = 12;
scale.StartAngle = 270;
scale.SweepAngle = 360;
scale.ShowFirstLabel = false;
scales.Add(scale);
circularGauge.Scales = scales;
```

**Show labels**

The [ShowLabels](#) property is a Boolean property, which is used to enable or disable the labels in circular gauge.

**XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale ShowLabels = "False"/>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.ShowLabels = false;
scales.Add(scale);
circularGauge.Scales = scales;
```



### Setting auto angle for label

Scale labels can be rotated automatically based on the current angle. To enable or disable the auto angle, use the [EnableAutoAngle](#) property.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale EnableAutoAngle = "True"/>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.EnableAutoAngle = true;
scales.Add(scale);
circularGauge.Scales = scales;
```



### Custom labels for circular gauge

Circular scale also supports custom label format using the [CustomLabels](#) property.

You can give labels in an array that you want to place in scale.

#### XML

```
Namespace:
xmlns:sys="clr-namespace:System;assembly=mcorlib"
...
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.CustomLabels>
```

```
<x:Array Type="{x:Type x:Double}">
  <sys:Double>0</sys:Double>
  <sys:Double>23</sys:Double>
  <sys:Double>45</sys:Double>
  <sys:Double>67</sys:Double>
  <sys:Double>85</sys:Double>
  <sys:Double>100</sys:Double>
</x:Array>
</gauge:Scale.CustomLabels>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.CustomLabels = new double[] { 0, 23, 45, 67, 85, 100 };
scales.Add(scale);
circularGauge.Scales = scales;
Content = circularGauge;
```



## Ranges

Range is a visual element, which begins and ends at specified values within a scale.

Setting start and end values for range

Start and end values of ranges are set by using the [StartValue](#) and [EndValue](#) properties.

## XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Ranges>
        <gauge:Range StartValue="0" EndValue="50" />
      </gauge:Scale.Ranges>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

```

</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

## C#

```

SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
Range range = new Range();
range.StartValue = 0;
range.EndValue = 50;
scale.Ranges.Add(range);
scales.Add(scale);
circularGauge.Scales = scales;

```



## Range customization

An UI of a range is customized by using the [Color](#) and [Thickness](#) properties. First, you should set the [Offset](#) property for range, then increase the thickness of the range.

## XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale RimThickness="10" RimColor="#E0E0E0" LabelColor="#424242"
      LabelOffset="0.9" MinorTicksPerInterval="3">
      <gauge:Scale.MajorTickSettings>
        <gauge:TickSettings Thickness="3" Length="10" Offset="0.8"/>
      </gauge:Scale.MajorTickSettings>
      <gauge:Scale.MinorTickSettings>
        <gauge:TickSettings Length="5" Offset="0.8"/>
      </gauge:Scale.MinorTickSettings>
      <gauge:Scale.Ranges>
        <gauge:Range StartValue="0" EndValue="50" Thickness="70" Offset="0.8"
          Color = "Pink"/>
      </gauge:Scale.Ranges>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

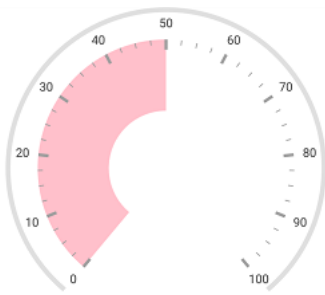
## C#

```

SfCircularGauge circularGauge = new SfCircularGauge();

```

```
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.RimThickness = 10;
scale.RimColor = Color.FromHex("#E0E0E0");
scale.LabelColor = Color.FromHex("#424242");
scale.MajorTickSettings.Thickness = 3;
scale.LabelOffset = 0.9;
scale.MinorTickSettings.Length = 5;
scale.MajorTickSettings.Length = 10;
scale.MajorTickSettings.Offset = 0.8;
scale.MinorTickSettings.Offset = 0.8;
scale.MinorTicksPerInterval = 3;
Range range = new Range();
range.StartValue = 0;
range.EndValue = 50;
range.Thickness = 70;
range.Offset = 0.8;
range.Color = Color.Pink;
scale.Ranges.Add(range);
scales.Add(scale);
circularGauge.Scales = scales;
```



### Setting position for range

The range can be placed inside the scale, outside the scale, or on the scale by using the following two ways:

1. The [Offset](#) property with the [Thickness](#) property.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Ranges>
        <gauge:Range StartValue = "0" EndValue = "100" Offset = "0.3" Thickness =
"20" />
      </gauge:Scale.Ranges>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
```

```
Scale scale = new Scale();
Range range = new Range();
range.StartValue = 0;
range.EndValue = 100;
range.Offset = 0.3;
range.Thickness = 20;
scale.Ranges.Add(range);
scales.Add(scale);
circularGauge.Scales = scales;
```



2.The [InnerStartOffset](#), [InnerEndOffset](#), [OuterStartOffset](#), and [OuterEndOffset](#) properties.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Ranges>
        <gauge:Range StartValue="10" EndValue="80" InnerStartOffset = "0.83"
          InnerEndOffset = "0.6" OuterStartOffset = "0.85" OuterEndOffset =" 0.8"/>
      </gauge:Scale.Ranges>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
Range range = new Range();
range.StartValue = 10;
range.EndValue = 80;
range.InnerStartOffset = 0.83;
range.InnerEndOffset = 0.6;
range.OuterStartOffset = 0.85;
range.OuterEndOffset = 0.8;
scale.Ranges.Add(range);
scales.Add(scale);
circularGauge.Scales = scales;
```



### Setting range color for labels

You can set range colors to labels using the [UseRangeColorForLabels](#) property.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale UseRangeColorForLabels="True">
      <gauge:Scale.Ranges>
        <gauge:Range StartValue = "0" EndValue = "100" Thickness = "20" />
      </gauge:Scale.Ranges>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.UseRangeColorForLabels = true;
Range range = new Range();
range.StartValue = 0;
range.EndValue = 100;
range.Thickness = 20;
scale.Ranges.Add(range);
scales.Add(scale);
circularGauge.Scales = scales;
this.Content = circularGauge;
```



### Setting multiple ranges

In addition to the default range, you can add n number of ranges to a scale by using the [Ranges](#) property.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Headers>
    <gauge:Header Text="Poor" Position="0.1,0.55" TextSize="20"
      ForegroundColor="#F03E3E" FontAttributes = "Bold"/>
    <gauge:Header Text="Good" Position="0.87,0.55" TextSize="20"
      ForegroundColor="#27beb7" FontAttributes = "Bold"/>
  </gauge:SfCircularGauge.Headers>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale StartValue = "0" EndValue = "100" StartAngle = "180"
      SweepAngle = "180" Interval = "10"
      ShowLabels = "False" ShowTicks = "False" ShowRim = "False" RimThickness =
        "40"
      RadiusFactor = "0.9" RimColor = "#e0e0e0">
      <gauge:Scale.Pointers>
        <gauge:NeedlePointer Value="70" KnobRadius="0" KnobStrokeColor = "#0682F6"
          KnobStrokeWidth = "6"
          Color="OrangeRed" KnobColor="White" LengthFactor="0.66" Type="Triangle"
          Thickness = "10"/>
      </gauge:Scale.Pointers>
      <gauge:Scale.Ranges>
        <gauge:Range StartValue = "0" EndValue = "35" Color = "#F03E3E" Thickness =
          "40" Offset = "0.9" />
        <gauge:Range StartValue = "35" EndValue = "75" Color = "#FFDD00" Thickness =
          "40" Offset = "0.9" />
        <gauge:Range StartValue = "75" EndValue = "100" Color = "#27beb7" Thickness
          = "40" Offset = "0.9" />
      </gauge:Scale.Ranges>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#



```

SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new
ObservableCollection<Syncfusion.SfGauge.XForms.Scale>();
Scale scale = new Syncfusion.SfGauge.XForms.Scale();
scale.StartValue = 0;
scale.EndValue = 100;
scale.StartAngle = 180;
scale.SweepAngle = 180;
scale.Interval = 10;
scale.ShowLabels = false;
scale.ShowTicks = false;
scale.ShowRim = false;
scale.RimThickness = 40;
scale.RadiusFactor = 0.9;
scale.RimColor = Color.FromHex("#e0e0e0");
Header header1 = new Header();
header1.Text = "Poor";
header1.TextSize = 20;
header1.Position = new Point(0.1, 0.55);
header1.ForegroundColor = Color.FromHex("#F03E3E");
header1.FontAttributes = FontAttributes.Bold;
circularGauge.Headers.Add(header1);
Header header2 = new Header();
header2.Text = "Good";
header2.TextSize = 20;
header2.Position = new Point(0.87, 0.55);
header2.ForegroundColor = Color.FromHex("#27beb7");
header2.FontAttributes = FontAttributes.Bold;
circularGauge.Headers.Add(header2);
//Poor
Range range = new Range();
range.StartValue = 0;
range.EndValue = 35;
range.Offset = 0.9;
range.Thickness = 40;
range.Color = Color.FromHex("#F03E3E");
scale.Ranges.Add(range);
//Average
Range range1 = new Range();
range1.StartValue = 35;
range1.EndValue = 75;
range1.Offset = 0.9;
range1.Thickness = 40;
range1.Color = Color.FromHex("#FFDD00");
scale.Ranges.Add(range1);
//Good
Range range2 = new Range();
range2.StartValue = 75;
range2.EndValue = 100;
range2.Offset = 0.9;
range2.Thickness = 40;
range2.Color = Color.FromHex("#27beb7");
scale.Ranges.Add(range2);
NeedlePointer pointer = new NeedlePointer();
pointer.Thickness = 10;
pointer.LengthFactor = 0.66;
pointer.Color = Color.OrangeRed;

```

```

pointer.KnobColor = Color.White;
pointer.Type = PointerType.Triangle;
pointer.KnobRadius = 0;
pointer.Value = 70;
pointer.KnobStrokeColor = Color.FromHex("#0682F6");
pointer.KnobStrokeWidth = 6;
scale.Pointers.Add(pointer);
scales.Add(scale);
circularGauge.Scales = scales;

```



### Setting gradient color for range

You can give smooth color transition to range by specifying the different colors based on range value.

#### XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale LabelFontSize = "16" ShowRim = "False"
      LabelOffset = "0.95" StartValue = "0" EndValue = "100" Interval = "10"
      RimThickness="28" MinorTicksPerInterval = "4" >
      <gauge:Scale.MajorTickSettings>
        <gauge:TickSettings Thickness="1" EndOffset="0.83" StartOffset="0.75"
          Length="8" />
      </gauge:Scale.MajorTickSettings>
      <gauge:Scale.MinorTickSettings>
        <gauge:TickSettings Thickness="0.7" EndOffset="0.79" StartOffset="0.75"/>
      </gauge:Scale.MinorTickSettings>
      <gauge:Scale.Ranges>
        <gauge:Range StartValue = "0" EndValue = "100" Thickness = "30" Offset =
          "0.6">
          <gauge:Range.GradientStops>
            <gauge:GaugeGradientStop Value="0" Color="#30B32D"/>
            <gauge:GaugeGradientStop Value="50" Color="#FFDD00"/>
            <gauge:GaugeGradientStop Value="80" Color="#F03E3E"/>
          </gauge:Range.GradientStops>
        </gauge:Range>
      </gauge:Scale.Ranges>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

#### C#

```

SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new
ObservableCollection<Syncfusion.SfGauge.XForms.Scale>();
Scale scale = new Syncfusion.SfGauge.XForms.Scale();

```

```

scale.StartValue = 0;
scale.EndValue = 100;
scale.Interval = 10;
scale.LabelFontSize = 16;
scale.LabelOffset = 0.95;
scale.RimThickness = 28;
scale.ShowRim = false;
scale.MinorTicksPerInterval = 4;
TickSettings major = new TickSettings();
major.Thickness = 1;
major.EndOffset = 0.83;
major.StartOffset = 0.75;
major.Length = 8;
scale.MajorTickSettings = major;
TickSettings minor = new TickSettings();
minor.Thickness = 0.7;
minor.EndOffset = 0.79;
minor.StartOffset = 0.75;
scale.MinorTickSettings = minor;
scales.Add(scale);
circularGauge.Scales = scales;
Range range = new Range();
range.Offset = 0.6;
range.Thickness = 30;
range.StartValue = 0;
range.EndValue = 100;
scale.Ranges.Add(range);
ObservableCollection<GaugeGradientStop> gradientColor1 = new
ObservableCollection<GaugeGradientStop>();
GaugeGradientStop gaugeGradientStop = new GaugeGradientStop();
gaugeGradientStop.Value = 0;
gaugeGradientStop.Color = Color.FromHex("#30B32D");
gradientColor1.Add(gaugeGradientStop);
GaugeGradientStop gaugeGradientStop1 = new GaugeGradientStop();
gaugeGradientStop1.Value = 50;
gaugeGradientStop1.Color = Color.FromHex("#FFDD00");
gradientColor1.Add(gaugeGradientStop1);
GaugeGradientStop gaugeGradientStop2 = new GaugeGradientStop();
gaugeGradientStop2.Value = 80;
gaugeGradientStop2.Color = Color.FromHex("#F03E3E");
gradientColor1.Add(gaugeGradientStop2);
range.GradientStops = gradientColor1;
circularGauge.Scales = scales;

```



## Pointers

You can add multiple pointers to the gauge to point multiple values on the same scale. It is used to show low and high values at the same time. The value of the pointer is set by using the [Value](#) property.

### Needle pointer

[Needle Pointer](#) contains three parts, namely needle, knob, and tail and that can be placed on a gauge to mark the values.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:NeedlePointer Value="70" />
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 70;
scale.Pointers.Add(needlePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```



### Setting needle pointer type

The appearance of the needle pointer can be customized by using the [Type](#) property. The default value of this property is [BarPointer](#). This is an enum property, and it has the following options:

1. Bar
2. Triangle

### Setting bar pointer type

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
```

```
<gauge:Scale.Pointers>
<gauge:NeedlePointer Value="60" Type="Bar"/>
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 60;
needlePointer.Type = PointerType.Bar;
scale.Pointers.Add(needlePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```



*Setting needle pointer type*

**XML**

```
<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Scales>
<gauge:Scale>
<gauge:Scale.Pointers>
<gauge:NeedlePointer Value="60" Type="Triangle"/>
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 60;
needlePointer.Type = PointerType.Triangle;
scale.Pointers.Add(needlePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```



### Needle pointer customization

The length of the needle is controlled by using the [LengthFactor](#) property. The [LengthFactor](#) property's minimum and maximum bounds are 0 and 1. The needle's UI is customized by using the [Color](#) and [Thickness](#) properties.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:NeedlePointer Value="60" Color="DeepSkyBlue" LengthFactor="0.7"
          Thickness="7"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 60;
needlePointer.Color = Color.DeepSkyBlue;
needlePointer.Thickness = 7;
needlePointer.LengthFactor = 0.7;
scale.Pointers.Add(needlePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```



*Knob customization*

Knob of the needle pointer can be customized by using the [KnobColor](#), [KnobRadius](#), [KnobRadiusFactor](#), [KnobStrokeColor](#), and [KnobStrokeWidth](#) properties. You can set the radius of knob to pixel and percentage values by using the [KnobRadius](#) and [KnobRadiusFactor](#) properties.

**XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:NeedlePointer Value="10" KnobRadius="15" KnobStrokeColor="#007DD1"
          KnobStrokeWidth="8" KnobColor="White" KnobRadiusFactor="0.1"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 10;
needlePointer.KnobRadius = 15;
needlePointer.KnobStrokeColor = Color.FromHex("#007DD1");
needlePointer.KnobColor = Color.White;
needlePointer.KnobStrokeWidth = 8;
needlePointer.KnobRadiusFactor = 0.1;
scale.Pointers.Add(needlePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```

*Setting tail for needle pointer*

Tail of the needle pointer can be customized by using the [TailColor](#), [TailLengthFactor](#), [TailStrokeColor](#), and [TailStrokeWidth](#) properties.

**XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
```

```
<gauge:NeedlePointer Value="90" KnobRadius = "15" TailColor="#757575"
TailLengthFactor="0.2" TailStrokeWidth="1" TailStrokeColor="#757575" />
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 90;
needlePointer.KnobRadius = 15;
needlePointer.TailColor = Color.FromHex("#757575");
needlePointer.TailLengthFactor = 0.2;
needlePointer.TailStrokeWidth = 1;
needlePointer.TailStrokeColor = Color.FromHex("#757575");
scale.Pointers.Add(needlePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```

**Range pointer**

A range pointer is an accenting line or shaded background range that can be placed on a gauge to mark the values. The [RangeStart](#) property allows you to set the starting value of the range pointer.

**XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:RangePointer RangeStart="15" Value="85" />
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
RangePointer rangePointer = new RangePointer();
```



```
rangePointer.RangeStart = 15;
rangePointer.Value = 85;
scale.Pointers.Add(rangePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```



### *Range pointer customization*

The range pointer's UI is customized by using the [Color](#) and [Thickness](#) properties. First, you should set the [Offset](#) property for range pointer, and then increase the thickness of the range pointer.

### **XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:RangePointer Value="60" Color="DarkCyan" Thickness="30"
          Offset="0.7"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### **C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
RangePointer rangePointer = new RangePointer();
rangePointer.Value = 60;
rangePointer.Color = Color.DarkCyan;
rangePointer.Thickness = 30;
rangePointer.Offset = 0.7;
scale.Pointers.Add(rangePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```



### Setting position for range pointer

The [RangePointer](#) in the scale can be placed inside or outside of the scale by using the following two ways:

1. The [Offset](#) property.
2. The [StartOffset](#) and [EndOffset](#) properties.

### Setting offset for range pointer

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:RangePointer Value="100" Offset="0.3" Thickness = "30"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
RangePointer rangePointer = new RangePointer();
rangePointer.Value = 100;
rangePointer.Offset = 0.3;
rangePointer.Thickness = 30;
scale.Pointers.Add(rangePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```



Setting start and end offset for range pointer

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:RangePointer RangeStart="15" Value="85" StartOffset="0.5"
EndOffset="0.7"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
RangePointer rangePointer = new RangePointer();
rangePointer.RangeStart = 15;
rangePointer.Value = 85;
rangePointer.StartOffset = 0.5;
rangePointer.EndOffset = 0.7;
scale.Pointers.Add(rangePointer);
scales.Add(scale);
circularGauge.Scales = scales;
```



Setting range cap for range pointer

The [RangeCap](#) property provides options to position the range cap of the [RangePointer](#), which contains the start, end, both, and none options. The [RangeCap](#) property is an enum property.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale LabelOffset="0.75" ScaleStartOffset="0.9" ScaleEndOffset="1">
      <gauge:Scale.MajorTickSettings>
        <gauge:TickSetting Offset="0.9"/>
      </gauge:Scale.MajorTickSettings>
      <gauge:Scale.MinorTickSettings>
        <gauge:TickSetting Offset="0.9"/>
      </gauge:Scale.MinorTickSettings>
      <gauge:Scale.Pointers>
        <gauge:RangePointer RangeStart="20" Value="80" RangeCap="End"
StartOffset="0.9" EndOffset="1"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

```

</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

## C#

```

SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.MajorTickSettings.Offset = 0.9;
scale.MinorTickSettings.Offset = 0.9;
scale.LabelOffset = 0.75;
scale.ScaleStartOffset = 0.9;
scale.ScaleEndOffset = 1;
RangePointer rangePointer = new RangePointer();
rangePointer.RangeStart = 20;
rangePointer.StartOffset = 0.9;
rangePointer.EndOffset = 1;
rangePointer.Value = 80;
rangePointer.RangeCap = RangeCap.End;
scale.Pointers.Add(rangePointer);
scales.Add(scale);
circularGauge.Scales = scales;

```



## Marker pointer

The different types of marker shapes are used to mark the pointer values in a scale. You can change the marker shape by using the [MarkerShape](#) property. Gauge supports the following types of marker shapes:

- Circle
- Rectangle
- Triangle
- Inverted triangle
- Diamond
- Image

The image is used to denote the pointer value instead of rendering the marker shape. It can be achieved by setting the [MarkerShape](#) to Image, and assigning the image path to [ImageSource](#) in pointer.

## XML

```

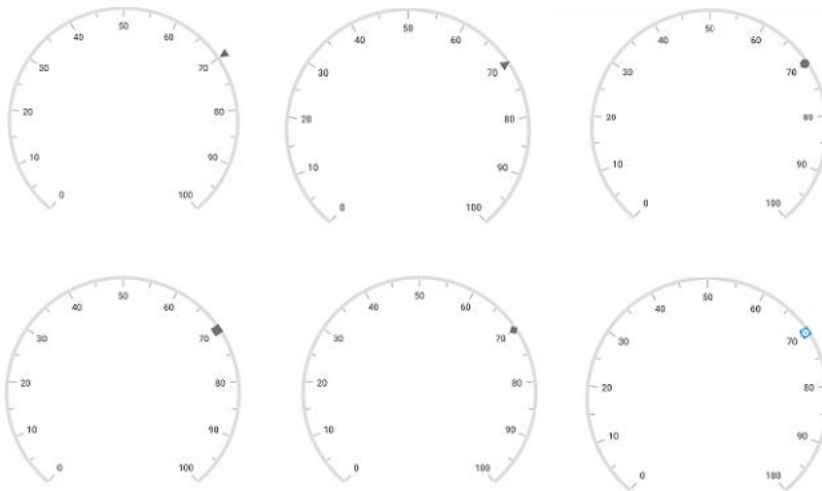
<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Scales>
<gauge:Scale>
<gauge:Scale.Pointers>

```

```
<gauge:MarkerPointer Value="70" MarkerShape = "Triangle"/>
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
MarkerPointer markerPointer = new MarkerPointer();
markerPointer.Value = 70;
markerPointer.MarkerShape = MarkerShape.Triangle;
scale.Pointers.Add(markerPointer);
scales.Add(scale);
circularGauge.Scales = scales;
```

*Setting image marker shape***XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:MarkerPointer Value="40" MarkerShape = "Image" ImageSource =
"icon.png"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
```

```

Scale scale = new Scale();
ObservableCollection<Pointer> pointers = new
ObservableCollection<Pointer>();
MarkerPointer markerPointer = new MarkerPointer();
markerPointer.Value = 40;
markerPointer.MarkerShape = MarkerShape.Image;
markerPointer.ImageSource = "icon.png";
pointers.Add(markerPointer);
scale.Pointers = pointers;
scales.Add(scale);
circularGauge.Scales = scales;

```



#### Marker pointer customization

The marker can be customized in terms of color, width, and height by using the [Color](#), [MarkerWidth](#), and [MarkerHeight](#) properties in pointer. First, you should set the [Offset](#) property for marker pointer, then increase the height and width of the marker pointer.

#### XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:MarkerPointer Value="70" Color="Pink" MarkerHeight="20"
MarkerWidth="20" Offset="1"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

#### C#

```

SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
MarkerPointer markerPointer = new MarkerPointer();
markerPointer.Value = 70;
markerPointer.Color = Color.Pink;
markerPointer.MarkerHeight = 20;
markerPointer.MarkerWidth = 20;
markerPointer.Offset = 1;
scale.Pointers.Add(markerPointer);
scales.Add(scale);
circularGauge.Scales = scales;

```



### Setting multiple pointers

In addition to the default pointer, you can add n number of pointers to a scale by using the [Pointers](#) property.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Pointers>
        <gauge:MarkerPointer Value="60" />
      </gauge:Scale.Pointers>
    </gauge:Scale>
    <gauge:Scale ShowTicks ="False" ShowLabels ="False" ScaleStartOffset ="0.5"
      ScaleEndOffset = "0.6">
      <gauge:Scale.Pointers>
        <gauge:NeedlePointer Value="40" LengthFactor = "0.3"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
ObservableCollection<Pointer> pointers = new
ObservableCollection<Pointer>();
MarkerPointer markerPointer = new MarkerPointer();
markerPointer.Value = 40;
pointers.Add(markerPointer);
Scale scale1 = new Scale();
scale1.ShowLabels = false;
scale1.ShowTicks = false;
scale1.ScaleStartOffset = 0.5;
scale1.ScaleEndOffset = 0.6;
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 60;
needlePointer.LengthFactor = 0.3;
pointers.Add(needlePointer);
scale1.Pointers = pointers;
scale.Pointers = pointers;
scales.Add(scale);
scales.Add(scale1);
circularGauge.Scales = scales;
```



### Setting animation for pointer

The [EnableAnimation](#) property is a Boolean property that enables or disables the animation of the pointers in circular gauge.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale RimColor="LightGray" RimThickness="30" RadiusFactor="1"
      ShowTicks="False"
      StartValue="0" EndValue="100" Interval="10" LabelOffset="0.75"
      LabelColor="#424242"
      LabelFontSize="15" >
      <gauge:Scale.Pointers>
        <gauge:RangePointer Color="Orange" Thickness="30" Offset="1"
          EnableAnimation="True"
          AnimationDuration="5" Value="80" />
        <gauge:NeedlePointer Thickness="7" LengthFactor="0.55" Color="LightGray"
          KnobColor="White" TailColor="LightGray" TailLengthFactor="0.2"
          Type="Triangle" KnobRadius="12" Value="80"
          AnimationDuration="5" TailStrokeWidth="2" TailStrokeColor="LightGray"
          KnobStrokeColor="LightGray" KnobStrokeWidth="8" />
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new
ObservableCollection<Syncfusion.SfGauge.XForms.Scale>();
Scale scale = new Syncfusion.SfGauge.XForms.Scale();
scale.RimColor = Color.LightGray;
scale.RimThickness = 30;
scale.RadiusFactor = 1;
scale.ShowTicks = false;
scale.StartValue = 0;
scale.EndValue = 100;
scale.Interval = 10;
scale.LabelOffset = 0.75;
scale.LabelColor = Color.FromHex("#424242");
scale.LabelFontSize = 15;
RangePointer pointer1 = new RangePointer();
pointer1.Color = Color.Orange;
```



```

pointer1.Thickness = 30;
pointer1.Offset = 1;
pointer1.EnableAnimation = true;
pointer1.AnimationDuration = 5;
pointer1.Value = 80;
scale.Pointers.Add(pointer1);
NeedlePointer pointer2 = new NeedlePointer();
pointer2.Thickness = 7;
pointer2.LengthFactor = 0.55;
pointer2.Color = Color.LightGray;
pointer2.KnobColor = Color.White;
pointer2.TailColor = Color.LightGray;
pointer2.TailLengthFactor = 0.2;
pointer2.Type = PointerType.Triangle;
pointer2.KnobRadius = 15;
pointer2.KnobRadius = 12;
pointer2.Value = 80;
pointer2.AnimationDuration = 5;
pointer2.TailStrokeWidth = 2;
pointer2.TailStrokeColor = Color.LightGray;
pointer2.KnobStrokeColor = Color.LightGray;
pointer2.KnobStrokeWidth = 8;
scale.Pointers.Add(pointer2);
circularGauge.Scales.Add(scale);

```



#### Setting pointer drag

Pointers can be dragged over the scale value. It can be achieved by clicking and dragging the pointer. To enable or disable the pointer drag, use the `EnableDragging` property.

#### XML

```

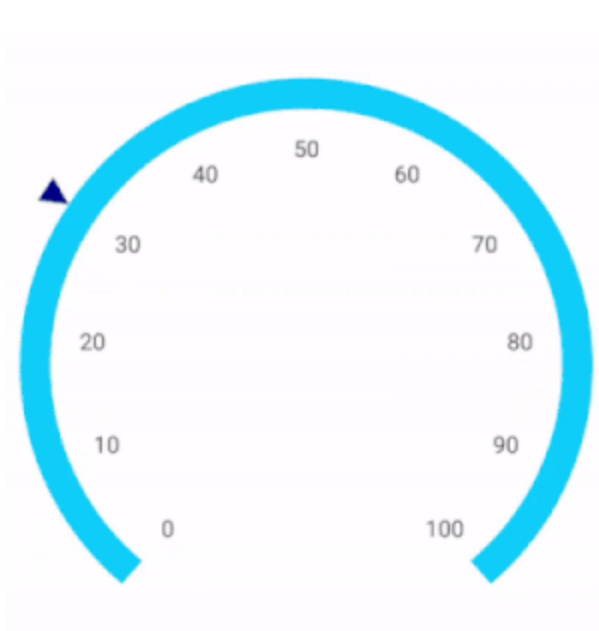
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale RimColor="DeepSkyBlue" RimThickness="20" RadiusFactor="1"
      ShowTicks="False"

```

```
StartValue="0" EndValue="100" Interval="10" LabelOffset="0.75"  
LabelColor="#424242"  
LabelFontSize="15">  
<gauge:Scale.Pointers>  
<gauge:MarkerPointer MarkerShape="InvertedTriangle" MarkerHeight="18"  
MarkerWidth="18"  
Value="30" EnableAnimation="False" EnableDragging="True"/>  
</gauge:Scale.Pointers>  
</gauge:Scale>  
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge circularGauge = new SfCircularGauge(this);  
CircularScale scale = new CircularScale();  
scale.RimColor = Color.DeepSkyBlue;  
scale.RimWidth = 20;  
scale.RadiusFactor = 1;  
scale.ShowTicks = false;  
scale.StartValue = 0;  
scale.EndValue = 100;  
scale.Interval = 10;  
scale.LabelOffset = 0.75;  
scale.LabelColor = Color.ParseColor("#424242");  
scale.LabelTextSize = 15;  
MarkerPointer pointer1 = new MarkerPointer();  
pointer1.MarkerShape =  
Com.Syncfusion.Gauges.SfCircularGauge.Enums.MarkerShape.InvertedTriangle;  
pointer1.Color = Color.DarkBlue;  
pointer1.MarkerHeight = 18;  
pointer1.MarkerWidth = 18;  
pointer1.Value = 30;  
pointer1.EnableAnimation = false;  
pointer1.EnableDragging = true;  
scale.CircularPointers.Add(pointer1);  
circularGauge.CircularScales.Add(scale);
```



## Header

The [Header](#) support allows you to show text inside the gauge control. A circular gauge can be made self-descriptive about the data. It can be measured with use of the header.

Adding header in circular gauge

### Header

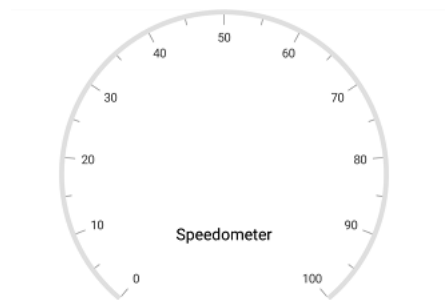
The [Header](#) can be used to set a unique header for the circular gauge. You can add text as headers in a circular gauge. Multiple headers also can be added in a circular gauge.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Headers>
    <gauge:Header Text="Speedometer" ForegroundColor="Black" />
  </gauge:SfCircularGauge.Headers>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
Header header = new Header();
header.Text = "Speedometer";
header.ForegroundColor = Color.Black;
circularGauge.Headers.Add(header);
```



### Setting position for header

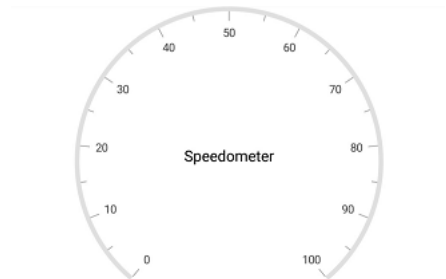
The [Position](#) property is used to place the header in a circular gauge. The value for [Position](#) should be specified in offset value. In the Point value, which has been given for the [Position](#), first value represent x-coordinate and second value represents y-coordinate. By default, it is placed at (0.5, 0.7).

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Headers>
    <gauge:Header Text="Speedometer" Position="0.5,0.5" ForegroundColor="Black"
  />
  </gauge:SfCircularGauge.Headers>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
Header header = new Header();
header.Text = "Speedometer";
header.ForegroundColor = Color.Black;
header.Position = new Point(0.5, 0.5);
circularGauge.Headers.Add(header);
```



### Customization of header

You can customize the header's text by using the [FontFamily](#), [FontAttribute](#) and [TextSize](#) properties.

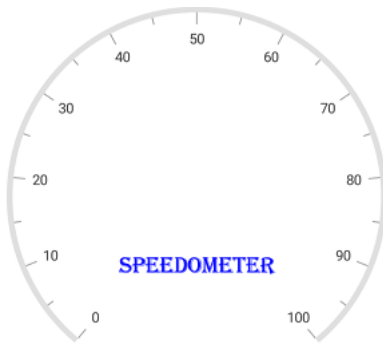
### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Headers>
    <gauge:Header Text="Speedometer" TextSize="20" ForegroundColor="Blue"
    FontAttributes="Bold">
    <gauge:Header.FontFamily>
```

```
<OnPlatform x:TypeArguments="x:String" iOS="Chalkduster"
Android="algerian.ttf" WinPhone="Chiller" />
</gauge:Header.FontFamily>
</gauge:Header>
</gauge:SfCircularGauge.Headers>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
Header header = new Header();
header.Text = "Speedometer";
header.TextSize = 20;
header.FontAttributes = FontAttributes.Bold;
header.FontFamily = Device.RuntimePlatform == Device.iOS ? "Chalkduster" :
Device.RuntimePlatform == Device.Android ? "algerian.ttf" : "Chiller";
header.ForegroundColor = Color.Blue;
circularGauge.Headers.Add(header);
this.content = circularGauge;
```

**Alignment of header**

You can align header to the **Start**, **Center** and **End** using the [HorizontalHeaderPosition](#) and [VerticalHeaderPosition](#) properties.

*Setting horizontal header position*

**XML**

```
<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Headers>
<gauge:Header Text="Speedometer" ForegroundColor="Black"
HorizontalHeaderPosition="Start"/>
</gauge:SfCircularGauge.Headers>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
Header header = new Header();
header.Text = "Speedometer";
header.ForegroundColor = Color.Black;
header.HorizontalHeaderPosition = ViewAlignment.Start;
circularGauge.Headers.Add(header);
```

```
Content = circularGauge;
```



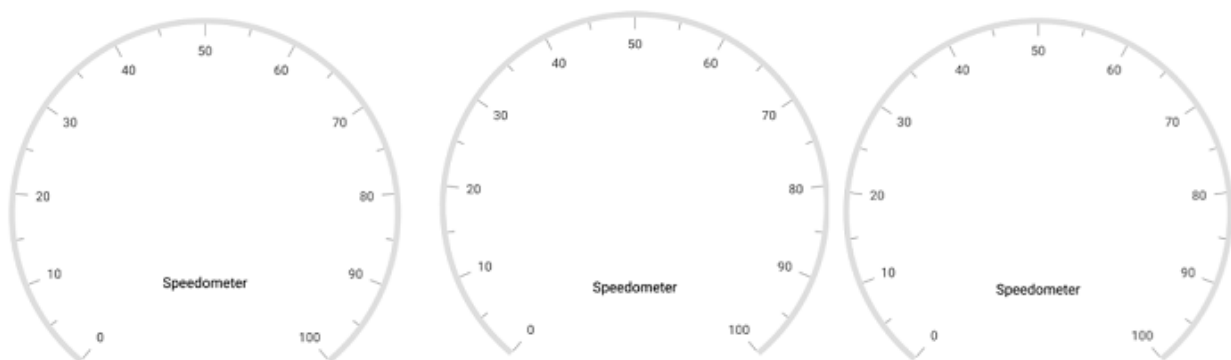
*Setting vertical header position*

#### **XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Headers>
    <gauge:Header Text="Speedometer" ForegroundColor="Black"
      VerticalHeaderPosition="Start"/>
  </gauge:SfCircularGauge.Headers>
</gauge:SfCircularGauge>
```

#### **C#**

```
SfCircularGauge circularGauge = new SfCircularGauge();
Header header = new Header();
header.Text = "Speedometer";
header.ForegroundColor = Color.Black;
header.VerticalHeaderPosition = ViewAlignment.Start;
circularGauge.Headers.Add(header);
Content = circularGauge;
```



#### **Annotations**

[SfCircularGauge](#) supports [Annotations](#), which allows you to mark the specific area of interest in circular gauge. You can place custom views as annotations. The text and images also can be added by using [Annotations](#) property.

### Setting view annotation

When the annotation allows you to place custom elements, a gauge can be initialized to the element, and this can be used to place the annotation in another gauge. The Following properties are used to customize the Annotations:

- [Angle](#): Used to place the View at the given Angle.
- [Offset](#): Used to move the View from the center to edge of the circular gauge. The value should be range from 0 to 1.

The following code is used to create the Annotations.

### XML

```
<gauge:SfCircularGauge HeightRequest="80" WidthRequest="80">
  <gauge:SfCircularGauge.Annotations>
    <gauge:GaugeAnnotation Angle="90" Offset="0.5">
      <gauge:GaugeAnnotation.View>
        <gauge:SfCircularGauge HeightRequest="80" WidthRequest="80">
          <gauge:SfCircularGauge.Annotations>
            <gauge:GaugeAnnotation Angle="270" Offset="0.5">
              <gauge:GaugeAnnotation.View>
                <Label Text="10s" FontSize="12" HeightRequest="20" WidthRequest="35"
                  TextColor="Black"
                  HorizontalTextAlignment="Center" VerticalTextAlignment="Center"/>
              </gauge:GaugeAnnotation.View>
            </gauge:GaugeAnnotation>
          </gauge:SfCircularGauge.Annotations>
        </gauge:SfCircularGauge.Scales>
        <gauge:Scale StartAngle="270" SweepAngle="360" ShowLabels="False"
          StartValue="0" EndValue="60" Interval="5"
          RimColor="#EDEEEF">
          <gauge:Scale.MajorTickSettings>
            <gauge:TickSettings Color="Black" StartOffset="1" EndOffset="0.85"
              Thickness="2"/>
          </gauge:Scale.MajorTickSettings>
          <gauge:Scale.MinorTickSettings>
            <gauge:TickSettings Color="Black" StartOffset="1" EndOffset="0.90"
              Thickness="0.5"/>
          </gauge:Scale.MinorTickSettings>
          <gauge:Scale.Ranges>
            <gauge:Range StartValue="0" EndValue="30" Color="Gray"
              InnerStartOffset="0.925"
              OuterStartOffset="1" InnerEndOffset="0.925" OuterEndOffset="1"/>
          </gauge:Scale.Ranges>
          <gauge:Scale.Pointers>
            <gauge:NeedlePointer Type="Triangle" KnobRadius="4" Thickness="3"
              EnableAnimation="False"
              Color="Black" KnobColor="Black"/>
          </gauge:Scale.Pointers>
        </gauge:Scale>
      </gauge:SfCircularGauge.Scales>
    </gauge:SfCircularGauge>
  </gauge:GaugeAnnotation.View>
</gauge:GaugeAnnotation>
<gauge:GaugeAnnotation Angle="0" Offset="0.5">
```

```

<gauge:GaugeAnnotation.View>
<Label x:Name="LabelAnnotation1" Text="4:55PM" FontSize="14"
HeightRequest="25" WidthRequest="75" TextColor="Black"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"/>
</gauge:GaugeAnnotation.View>
</gauge:GaugeAnnotation>
<gauge:GaugeAnnotation Angle="180" Offset="0.5">
<gauge:GaugeAnnotation.View>
<gauge:SfCircularGauge HeightRequest="80" WidthRequest="80">
<gauge:SfCircularGauge.Annotations>
<gauge:GaugeAnnotation Angle="270" Offset="0.5">
<gauge:GaugeAnnotation.View>
<Label x:Name="LabelAnnotation3" Text="55M" FontSize="12" HeightRequest="20"
WidthRequest="35" TextColor="Black"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"/>
</gauge:GaugeAnnotation.View>
</gauge:GaugeAnnotation>
</gauge:SfCircularGauge.Annotations>
<gauge:SfCircularGauge.Scales>
<gauge:Scale StartAngle="270" SweepAngle="360" ShowLabels="False"
StartValue="0" EndValue="60" Interval="5"
RimColor="#EDEEEF">
<gauge:Scale.MajorTickSettings>
<gauge:TickSettings Color="Black" StartOffset="1" EndOffset="0.85"
Thickness="2"/>
</gauge:Scale.MajorTickSettings>
<gauge:Scale.MinorTickSettings>
<gauge:TickSettings Color="Black" StartOffset="1" EndOffset="0.90"
Thickness="0.5"/>
</gauge:Scale.MinorTickSettings>
<gauge:Scale.Ranges>
<gauge:Range StartValue="0" EndValue="30" Color="Gray"
InnerStartOffset="0.925"
OuterStartOffset="1" InnerEndOffset="0.925" OuterEndOffset="1"/>
</gauge:Scale.Ranges>
<gauge:Scale.Pointers>
<gauge:NeedlePointer Type="Triangle" KnobRadius="4" Thickness="3"
EnableAnimation="False"
Color="Black" KnobColor="Black"/>
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
</gauge:GaugeAnnotation.View>
</gauge:GaugeAnnotation>
</gauge:SfCircularGauge.Annotations>
<gauge:SfCircularGauge.Scales>
<gauge:Scale StartAngle="270" SweepAngle="360" ShowLabels="False"
StartValue="0" EndValue="12" Interval="1"
RimColor="#EDEEEF" MinorTicksPerInterval="4" LabelColor="Gray"
LabelOffset="0.8"
ScaleEndOffset="0.925" LabelFontSize="14" ShowFirstLabel="False">
<gauge:Scale.MajorTickSettings>
<gauge:TickSettings Color="Black" StartOffset="1" EndOffset="0.9"
Thickness="3"/>
</gauge:Scale.MajorTickSettings>
<gauge:Scale.MinorTickSettings>

```



```

<gauge:TickSettings Color="Black" StartOffset="1" EndOffset="0.95"
Thickness="1"/>
</gauge:Scale.MinorTickSettings>
<gauge:Scale.Ranges>
<gauge:Range StartValue="0" EndValue="3" Color="Gray"
InnerStartOffset="0.925"
OuterStartOffset="1" InnerEndOffset="0.925" OuterEndOffset="1"/>
</gauge:Scale.Ranges>
<gauge:Scale.Pointers>
<gauge:NeedlePointer KnobRadius="6" Thickness="3.5" EnableAnimation="False"
KnobStrokeWidth="5" TailLengthFactor="0.20" TailColor="Black"
KnobStrokeColor="Black" Color="Black" KnobColor="White"
LengthFactor="0.75"/>
<gauge:NeedlePointer KnobRadius="6" Type="Triangle" Thickness="5"
EnableAnimation="False"
Color="Black" KnobColor="White" LengthFactor="0.4"/>
<gauge:NeedlePointer KnobRadius="6" Type="Triangle" Thickness="5"
EnableAnimation="False"
Color="Black" KnobColor="White" LengthFactor="0.65"/>
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

**C#**

```

Label LabelAnnotation1 = new Label();
LabelAnnotation1.Text = "4:55PM";
LabelAnnotation1.FontSize = 14;
LabelAnnotation1.HeightRequest = 25;
LabelAnnotation1.WidthRequest = 75;
LabelAnnotation1.TextColor = Color.Black;
LabelAnnotation1.HorizontalTextAlignment = TextAlignment.Center;
LabelAnnotation1.VerticalTextAlignment = TextAlignment.Center;
Label LabelAnnotation2 = new Label();
LabelAnnotation2.Text = "10s";
LabelAnnotation2.FontSize = 12;
LabelAnnotation2.HeightRequest = 20;
LabelAnnotation2.WidthRequest = 35;
LabelAnnotation2.TextColor = Color.Black;
LabelAnnotation2.HorizontalTextAlignment = TextAlignment.Center;
LabelAnnotation2.VerticalTextAlignment = TextAlignment.Center;
Label LabelAnnotation3 = new Label();
LabelAnnotation3.Text = "55M";
LabelAnnotation3.FontSize = 12;
LabelAnnotation3.HeightRequest = 20;
LabelAnnotation3.WidthRequest = 35;
LabelAnnotation3.TextColor = Color.Black;
LabelAnnotation3.HorizontalTextAlignment = TextAlignment.Center;
LabelAnnotation3.VerticalTextAlignment = TextAlignment.Center;
SfCircularGauge Annotation1 = new SfCircularGauge();
Annotation1.HeightRequest = Device.OnPlatform(70, 80, 100);
Annotation1.WidthRequest = Device.OnPlatform(70, 80, 100);
CircularGaugeAnnotationCollection annotations = new
CircularGaugeAnnotationCollection();
GaugeAnnotation gaugeAnnotation = new GaugeAnnotation();

```

```
gaugeAnnotation.View = LabelAnnotation2;
gaugeAnnotation.Angle = 270;
gaugeAnnotation.Offset = .5;
annotations.Add(gaugeAnnotation);
Annotation1.Annotations = annotations;
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.StartAngle = 270;
scale.SweepAngle = 360;
scale.ShowLabels = false;
scale.StartValue = 0;
scale.EndValue = 60;
scale.Interval = 5;
scale.RimColor = Color.FromRgb(237, 238, 239);
scale.MajorTickSettings.Color = Color.Black;
scale.MajorTickSettings.StartOffset = 1;
scale.MajorTickSettings.EndOffset = .85;
scale.MajorTickSettings.Thickness = 2;
scale.MinorTickSettings.Color = Color.Black;
scale.MinorTickSettings.StartOffset = 1;
scale.MinorTickSettings.EndOffset = .90;
scale.MinorTickSettings.Thickness = 0.5;
ObservableCollection<Range> ranges = new ObservableCollection<Range>();
Range range = new Range();
range.StartValue = 0;
range.EndValue = 30;
range.Color = Color.Gray;
range.InnerStartOffset = 0.925;
range.OuterStartOffset = 1;
range.InnerEndOffset = 0.925;
range.OuterEndOffset = 1;
ranges.Add(range);
scale.Ranges = ranges;
ObservableCollection<Pointer> pointers = new
ObservableCollection<Pointer>();
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Type = PointerType.Triangle;
needlePointer.KnobRadius = 4;
needlePointer.Thickness = 3;
needlePointer.EnableAnimation = false;
needlePointer.KnobColor = Color.Black;
needlePointer.Color = Color.Black;
pointers.Add(needlePointer);
scale.Pointers = pointers;
scales.Add(scale);
Annotation1.Scales = scales;
Annotation1.Parent = this;
SfCircularGauge Annotation2 = new SfCircularGauge();
Annotation2.HeightRequest = Device.OnPlatform(70, 80, 100);
Annotation2.WidthRequest = Device.OnPlatform(70, 80, 100);
CircularGaugeAnnotationCollection annotations1 = new
CircularGaugeAnnotationCollection();
GaugeAnnotation gaugeAnnotation1 = new GaugeAnnotation();
gaugeAnnotation1.View = LabelAnnotation3;
gaugeAnnotation1.Angle = 270;
gaugeAnnotation1.Offset = .5;
annotations1.Add(gaugeAnnotation1);
```

```
Annotation2.Annotations = annotations1;
ObservableCollection<Scale> scales1 = new ObservableCollection<Scale>();
Scale scale1 = new Scale();
scale1.StartAngle = 270;
scale1.SweepAngle = 360;
scale1.StartValue = 0;
scale1.EndValue = 60;
scale1.Interval = 5;
scale1.ShowLabels = false;
scale1.RimColor = Color.FromRgb(237, 238, 239);
scale1.MajorTickSettings.Color = Color.Black;
scale1.MajorTickSettings.StartOffset = 1;
scale1.MajorTickSettings.EndOffset = 0.85;
scale1.MajorTickSettings.Thickness = 2;
scale1.MinorTickSettings.Color = Color.Black;
scale1.MinorTickSettings.StartOffset = 1;
scale1.MinorTickSettings.EndOffset = 0.90;
scale1.MinorTickSettings.Thickness = 0.5;
ObservableCollection<Range> ranges1 = new ObservableCollection<Range>();
Range range1 = new Range();
range1.StartValue = 0;
range1.EndValue = 30;
range1.Color = Color.Gray;
range1.InnerStartOffset = 0.925;
range1.OuterStartOffset = 1;
range1.InnerEndOffset = 0.925;
range1.OuterEndOffset = 1;
ranges1.Add(range1);
scale1.Ranges = ranges1;
ObservableCollection<Pointer> pointers1 = new
ObservableCollection<Pointer>();
NeedlePointer needlePointer1 = new NeedlePointer();
needlePointer1.Type = PointerType.Triangle;
needlePointer1.KnobRadius = 4;
needlePointer1.Thickness = 3;
needlePointer1.EnableAnimation = false;
needlePointer1.KnobColor = Color.Black;
needlePointer1.Color = Color.Black;
pointers1.Add(needlePointer1);
scale1.Pointers = pointers1;
scales1.Add(scale1);
Annotation2.Scales = scales1;
Annotation2.Parent = this;
SfCircularGauge gauge = new SfCircularGauge();
CircularGaugeAnnotationCollection annotations3 = new
CircularGaugeAnnotationCollection();
GaugeAnnotation gaugeAnnotation2 = new GaugeAnnotation();
gaugeAnnotation2.View = Annotation1;
gaugeAnnotation2.Angle = 90;
gaugeAnnotation2.Offset = Device.OnPlatform(.5, .5, .6);
annotations3.Add(gaugeAnnotation2);
GaugeAnnotation gaugeAnnotation3 = new GaugeAnnotation();
gaugeAnnotation3.View = LabelAnnotation1;
gaugeAnnotation3.Angle = 00;
gaugeAnnotation3.Offset = .5;
annotations3.Add(gaugeAnnotation3);
GaugeAnnotation gaugeAnnotation4 = new GaugeAnnotation();
```

```
gaugeAnnotation4.View = Annotation2;
gaugeAnnotation4.Angle = 180;
gaugeAnnotation4.Offset = Device.OnPlatform(.5, .5, .6);
annotations3.Add(gaugeAnnotation4);
gauge.Annotations = annotations3;
ObservableCollection<Scale> scales2 = new ObservableCollection<Scale>();
Scale scale2 = new Scale();
scale2.StartValue = 0;
scale2.EndValue = 12;
scale2.Interval = 1;
scale2.MinorTicksPerInterval = 4;
scale2.RimColor = Color.FromRgb(237, 238, 239);
scale2.LabelColor = Color.Gray;
scale2.LabelOffset = Device.OnPlatform(.8, .8, .875);
scale2.ScaleEndOffset = .925;
scale2.StartAngle = 270;
scale2.SweepAngle = 360;
scale2.LabelFontSize = 14;
scale2.ShowFirstLabel = false;
scale2.MinorTickSettings.Color = Color.Black;
scale2.MinorTickSettings.StartOffset = 1;
scale2.MinorTickSettings.EndOffset = 0.95;
scale2.MinorTickSettings.Thickness = 1;
scale2.MajorTickSettings.Color = Color.Black;
scale2.MajorTickSettings.StartOffset = 1;
scale2.MajorTickSettings.EndOffset = 0.9;
scale2.MajorTickSettings.Thickness = 3;
ObservableCollection<Range> ranges2 = new ObservableCollection<Range>();
Range range2 = new Range();
range2.StartValue = 0;
range2.EndValue = 3;
range2.Color = Color.Gray;
range2.InnerStartOffset = 0.925;
range2.OuterStartOffset = 1;
range2.InnerEndOffset = 0.925;
range2.OuterEndOffset = 1;
ranges2.Add(range2);
scale2.Ranges = ranges2;
ObservableCollection<Pointer> pointers2 = new
ObservableCollection<Pointer>();
NeedlePointer needlePointer2 = new NeedlePointer();
needlePointer2.EnableAnimation = false;
needlePointer2.KnobRadius = 6;
needlePointer2.LengthFactor = .75;
needlePointer2.KnobColor = Color.White;
needlePointer2.Color = Color.Black;
needlePointer2.Thickness = 3.5;
needlePointer2.KnobStrokeColor = Color.Black;
needlePointer2.KnobStrokeWidth = 5;
needlePointer2.TailLengthFactor = 0.20;
needlePointer2.TailColor = Color.Black;
pointers2.Add(needlePointer2);
NeedlePointer needlePointer3 = new NeedlePointer();
needlePointer3.EnableAnimation = false;
needlePointer3.KnobRadius = 6;
needlePointer3.LengthFactor = .4;
needlePointer3.KnobColor = Color.White;
```

```

needlePointer3.Color = Color.Black;
needlePointer3.Thickness = 5;
needlePointer3.Type = PointerType.Triangle;
pointers2.Add(needlePointer3);
NeedlePointer needlePointer4 = new NeedlePointer();
needlePointer4.EnableAnimation = false;
needlePointer4.KnobRadius = 6;
needlePointer4.LengthFactor = .65;
needlePointer4.KnobColor = Color.White;
needlePointer4.Color = Color.Black;
needlePointer4.Thickness = 5;
needlePointer4.Type = PointerType.Triangle;
pointers2.Add(needlePointer4);
scale2.Pointers = pointers2;
scales2.Add(scale2);
gauge.Scales = scales2;

```



### Setting image annotation

Annotations provide options to add any image over the gauge control with respect to its offset position. You can add multiple images in single control.

### XML

```

<gauge:SfCircularGauge HeightRequest="500" WidthRequest="500"
BackgroundColor="White" Margin="20">
  <gauge:SfCircularGauge.Annotations>
    <gauge:GaugeAnnotation Angle="270" Offset="0.2">
      <gauge:GaugeAnnotation.View>
        <Image Source="weather.jpg" HeightRequest="100" WidthRequest="100"/>
      </gauge:GaugeAnnotation.View>
    </gauge:GaugeAnnotation>
  </gauge:SfCircularGauge.Annotations>
  <gauge:SfCircularGauge.Headers>
    <gauge:Header Text="73.2" Position="0.48, 0.6" ForegroundColor="#424242"
FontAttributes="Bold" TextSize="20"/>
    <gauge:Header Text="o" Position="0.55,0.58" ForegroundColor="#424242"
FontAttributes="Bold" TextSize="12"/>
    <gauge:Header Text="F" Position="0.58,0.6" ForegroundColor="#424242"
FontAttributes="Bold" TextSize="20"/>
  </gauge:SfCircularGauge.Headers>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale ShowLabels="False" ShowTicks="False" RimThickness="20"
RadiusFactor="1" RimColor="#e0e0e0"
StartAngle="90" SweepAngle="360" StartValue="0" EndValue="100"
Interval="10">
      <gauge:Scale.Pointers>

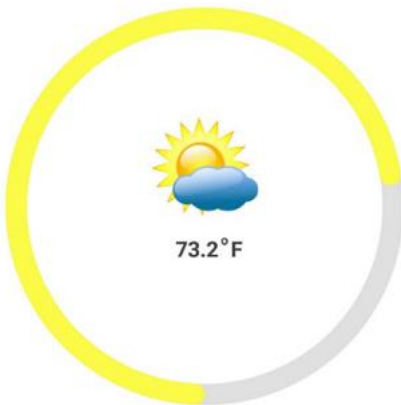
```

```
<gauge:RangePointer Value="73.2" Offset="1" Thickness="20" RangeCap="Both"
Color="#FCFB48"/>
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge gauge = new SfCircularGauge();
gauge.HeightRequest = 500;
gauge.WidthRequest = 500;
gauge.Margin = new Thickness(20);
gauge.BackgroundColor = Color.White;
ObservableCollection<Scale> scales = new
ObservableCollection<Syncfusion.SfGauge.XForms.Scale>();
Scale scale = new Syncfusion.SfGauge.XForms.Scale();
scale.ShowLabels = false;
scale.ShowTicks = false;
scale.RimThickness = 20;
scale.RadiusFactor = 1;
scale.RimColor = Color.FromHex("#e0e0e0");
scale.StartAngle = 90;
scale.SweepAngle = 360;
scale.StartValue = 0;
scale.EndValue = 100;
scale.Interval = 10;
RangePointer pointer = new RangePointer();
pointer.Value = 73.2;
pointer.Offset = 1;
pointer.Thickness = 20;
pointer.RangeCap = RangeCap.Both;
pointer.Color = Color.FromRgb(252, 251, 72);
scale.Pointers.Add(pointer);
scales.Add(scale);
Header header = new Header();
header.Text = "73.2";
header.Position = new Point(0.48, 0.6);
header.ForegroundColor = Color.FromHex("#424242");
header.FontAttributes = FontAttributes.Bold;
header.TextSize = 20;
gauge.Headers.Add(header);
Header header1 = new Header();
header1.Text = "o";
header1.Position = new Point(0.55, 0.58);
header1.ForegroundColor = Color.FromHex("#424242");
header1.FontAttributes = FontAttributes.Bold;
header1.TextSize = 12;
gauge.Headers.Add(header1);
Header header2 = new Header();
header2.Text = "F";
header2.Position = new Point(0.58, 0.6);
header2.ForegroundColor = Color.FromHex("#424242");
header2.FontAttributes = FontAttributes.Bold;
header2.TextSize = 20;
gauge.Headers.Add(header2);
```

```
GaugeAnnotation annotation = new GaugeAnnotation();
Image image = new Image();
image.Source = "weather.jpg";
image.HeightRequest = 100;
image.WidthRequest = 100;
annotation.View = image;
annotation.Angle = 270;
annotation.Offset = 0.2;
gauge.Annotations.Add(annotation);
gauge.Scales = scales;
```



#### Setting text annotation

You can add any text over the gauge control to enhance the readability. You can add multiple text instances in single control.

#### XML

```
<gauge:SfCircularGauge HeightRequest="500" WidthRequest="500">
  <gauge:SfCircularGauge.Headers>
    <gauge:Header Text="13M" Position="0.5, 0.5" ForegroundColor="#0682F6"
      FontAttributes="Bold" TextSize="20"/>
  </gauge:SfCircularGauge.Headers>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale ShowLabels="False" ShowTicks="False" ShowRim="False"
      StartAngle="160" SweepAngle="270" StartValue="0" EndValue="15" >
      <gauge:Scale.Pointers>
        <gauge:RangePointer Value="13" Offset="0.8" Thickness="30" RangeCap="Start"
          Color="#0682F6"/>
        <gauge:MarkerPointer MarkerShape="Image" ImageSource="shot.jpg"
          MarkerWidth="60" MarkerHeight="60"
          Value="0" Offset="0.8"/>
        <gauge:MarkerPointer MarkerShape="Circle" MarkerWidth="40" MarkerHeight="40"
          Color="#9e9e9e"
          Value="13" Offset="0.8"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
    <gauge:Scale ShowLabels="False" ShowTicks="False" ShowRim="False"
      StartAngle="0" SweepAngle="360" >
      </gauge:Scale>
    </gauge:SfCircularGauge.Scales>
```

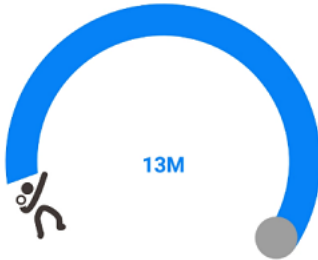
```
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge gauge = new SfCircularGauge();
gauge.HeightRequest = 500;
gauge.WidthRequest = 500;
ObservableCollection<Scale> scales = new
ObservableCollection<Syncfusion.SfGauge.XForms.Scale>();
Scale scale = new Syncfusion.SfGauge.XForms.Scale();
scale.StartAngle = 160;
scale.SweepAngle = 270;
scale.StartValue = 0;
scale.EndValue = 15;
scale.ShowLabels = false;
scale.ShowRim = false;
scale.ShowTicks = false;
RangePointer range = new RangePointer();
range.Color = Color.FromHex("#0682F6");
range.Offset = 0.8;
range.Value = 13;
range.RangeCap = RangeCap.Start;
range.Thickness = 30;
scale.Pointers.Add(range);
Scale scale1 = new Syncfusion.SfGauge.XForms.Scale();
scale1.StartAngle = 0;
scale1.SweepAngle = 360;
scale1.RimColor = Color.FromHex("#e0e0e0");
scale1.RadiusFactor = 0.8;
scale1.RimThickness = 30;
scale1.ShowTicks = false;
scale1.ShowRim = false;
scale1.ShowLabels = false;
scales.Add(scale1);
Header header = new Header();
header.FontAttributes = FontAttributes.Bold;
header.TextSize = 20;
header.Text = "13M";
header.Position = new Point(0.5, 0.5);
header.ForegroundColor = Color.FromHex("#0682F6");
gauge.Headers.Add(header);
MarkerPointer pointer1 = new MarkerPointer();
pointer1.MarkerShape = MarkerShape.Image;
pointer1.ImageSource = "shot.jpg";
pointer1.MarkerWidth = 60;
pointer1.MarkerHeight = 60;
pointer1.Value = 0;
pointer1.Offset = 0.8;
scale.Pointers.Add(pointer1);
MarkerPointer pointer2 = new MarkerPointer();
pointer2.MarkerShape = MarkerShape.Circle;
pointer2.MarkerWidth = 40;
pointer2.MarkerHeight = 40;
pointer2.Color = Color.FromHex("#9e9e9e");
pointer2.Value = 13;
pointer2.Offset = 0.8;
```



```
scale.Pointers.Add(pointer2);
scales.Add(scale);
gauge.Scales = scales;
```



### Set margin to annotation

You can adjust annotation by specifying the [ViewMargin](#) property in pixel, which adjusts annotation elements from their current position.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Annotations>
    <gauge:GaugeAnnotation ViewMargin="10,50">
      <gauge:GaugeAnnotation.View>
        <Label Text="800 GB" TextColor="Black" FontSize="25"/>
      </gauge:GaugeAnnotation.View>
    </gauge:GaugeAnnotation>
  </gauge:SfCircularGauge.Annotations>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale ShowLabels="False" ShowTicks="False" RimThickness="25"
      RadiusFactor="1" RimColor="#e0e0e0"
      StartAngle="90" SweepAngle="360" StartValue="0" EndValue="100"
      Interval="10">
      <gauge:Scale.Pointers>
        <gauge:RangePointer Value="80" Offset="1" Thickness="25" RangeCap="Both"
          Color="#01bdae"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

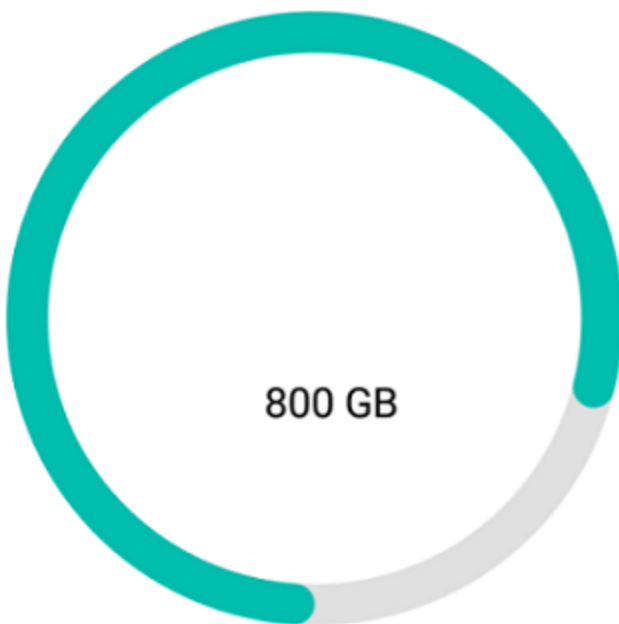
### C#

```
SfCircularGauge gauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.ShowLabels = false;
scale.ShowTicks = false;
scale.RimThickness = 25;
scale.RadiusFactor = 1;
scale.RimColor = Color.FromHex("#e0e0e0");
scale.StartAngle = 90;
```

```

scale.SweepAngle = 360;
scale.StartValue = 0;
scale.EndValue = 100;
scale.Interval = 10;
RangePointer pointer = new RangePointer();
pointer.Value = 80;
pointer.Offset = 1;
pointer.Thickness = 25;
pointer.RangeCap = RangeCap.Both;
pointer.Color = Color.FromHex("#01bdae");
scale.Pointers.Add(pointer);
scales.Add(scale);
GaugeAnnotation annotation = new GaugeAnnotation();
annotation.ViewMargin = new Point(10, 50);
Label label = new Label();
label.Text = "800 GB";
label.FontSize = 25;
label.TextColor = Color.Black;
annotation.View = label;
gauge.Annotations.Add(annotation);
gauge.Scales = scales;
Content = gauge;

```



#### Alignment of annotation

You can align annotations to the **Start**, **Center** and **End** using the [HorizontalAlignment](#) and [VerticalAlignment](#) properties.

#### Setting horizontal alignment

##### XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Annotations>
    <gauge:GaugeAnnotation HorizontalAlignment="Start">

```

```

<gauge:GaugeAnnotation.View>
<Label Text="800 GB" TextColor="Black" FontSize="25"/>
</gauge:GaugeAnnotation.View>
</gauge:GaugeAnnotation>
</gauge:SfCircularGauge.Annotations>
<gauge:SfCircularGauge.Scales>
<gauge:Scale ShowLabels="False" ShowTicks="False" RimThickness="25"
RadiusFactor="1" RimColor="#e0e0e0" StartAngle="90"
SweepAngle="360" StartValue="0" EndValue="100"
Interval="10">
<gauge:Scale.Pointers>
<gauge:RangePointer Value="80" Offset="1" Thickness="25" RangeCap="Both"
Color="#01bdae"/>
</gauge:Scale.Pointers>
</gauge:Scale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

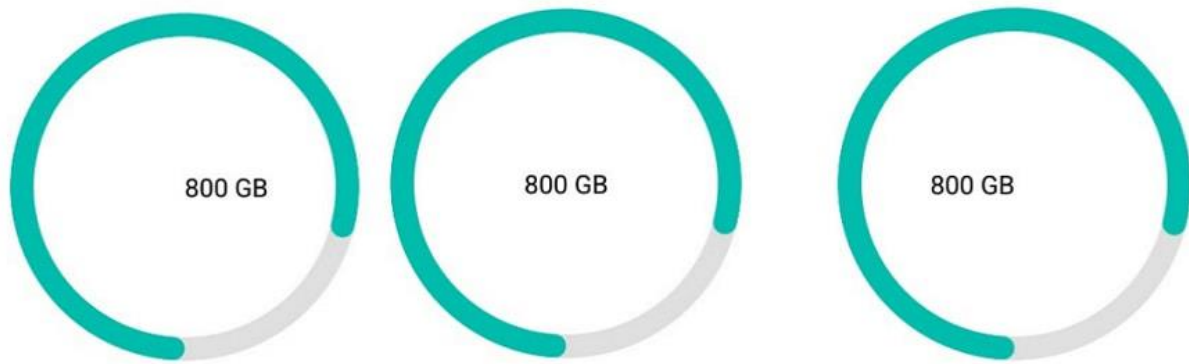
```

**C#**

```

SfCircularGauge gauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.ShowLabels = false;
scale.ShowTicks = false;
scale.RimThickness = 25;
scale.RadiusFactor = 1;
scale.RimColor = Color.FromHex("#e0e0e0");
scale.StartAngle = 90;
scale.SweepAngle = 360;
scale.StartValue = 0;
scale.EndValue = 100;
scale.Interval = 10;
RangePointer pointer = new RangePointer();
pointer.Value = 80;
pointer.Offset = 1;
pointer.Thickness = 25;
pointer.RangeCap = RangeCap.Both;
pointer.Color = Color.FromHex("#01bdae");
scale.Pointers.Add(pointer);
scales.Add(scale);
GaugeAnnotation annotation = new GaugeAnnotation();
annotation.HorizontalAlignment = ViewAlignment.Start;
Label label = new Label();
label.Text = "800 GB";
label.FontSize = 25;
label.TextColor = Color.Black;
annotation.View = label;
gauge.Annotations.Add(annotation);
gauge.Scales = scales;
Content = gauge;

```



*Setting vertical alignment*

#### **XML**

```
<gauge:SfCircularGauge BackgroundColor="White" Margin="20">
  <gauge:SfCircularGauge.Annotations>
    <gauge:GaugeAnnotation VerticalAlignment="Start">
      <gauge:GaugeAnnotation.View>
        <Label Text="800 GB" TextColor="Black" FontSize="25"/>
      </gauge:GaugeAnnotation.View>
    </gauge:GaugeAnnotation>
  </gauge:SfCircularGauge.Annotations>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale ShowLabels="False" ShowTicks="False" RimThickness="25"
      RadiusFactor="1" RimColor="#e0e0e0"
      StartAngle="90" SweepAngle="360" StartValue="0" EndValue="100"
      Interval="10">
      <gauge:Scale.Pointers>
        <gauge:RangePointer Value="80" Offset="1" Thickness="25" RangeCap="Both"
          Color="#01bdae"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### **C#**

```
SfCircularGauge gauge = new SfCircularGauge();
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
scale.ShowLabels = false;
scale.ShowTicks = false;
scale.RimThickness = 25;
scale.RadiusFactor = 1;
scale.RimColor = Color.FromHex("#e0e0e0");
scale.StartAngle = 90;
scale.SweepAngle = 360;
scale.StartValue = 0;
scale.EndValue = 100;
scale.Interval = 10;
RangePointer pointer = new RangePointer();
pointer.Value = 80;
pointer.Offset = 1;
```

```

pointer.Thickness = 25;
pointer.RangeCap = RangeCap.Both;
pointer.Color = Color.FromHex("#01bdae");
scale.Pointers.Add(pointer);
scales.Add(scale);
GaugeAnnotation annotation = new GaugeAnnotation();
annotation.VerticalAlignment = ViewAlignment.Start;
Label label = new Label();
label.Text = "800 GB";
label.FontSize = 25;
label.TextColor = Color.Black;
annotation.View = label;
gauge.Annotations.Add(annotation);
gauge.Scales = scales;
Content = gauge;

```



## How to

### Changing the gauge size

The [CircularCoefficient](#) property is used to change the diameter of the gauge.

It ranges from 0 to 1, and the default value is 1.

### XML

```

<gauge:SfCircularGauge CircularCoefficient="0.7">
  <gauge:SfCircularGauge.Headers>
    <gauge:Header Text="Speedometer" ForegroundColor="Black" />
  </gauge:SfCircularGauge.Headers>
  <gauge:SfCircularGauge.Scales>
    <gauge:Scale>
      <gauge:Scale.Ranges>
        <gauge:Range StartValue="0" EndValue="50" />
      </gauge:Scale.Ranges>
      <gauge:Scale.Pointers>
        <gauge:NeedlePointer Value="70" />
        <gauge:RangePointer RangeStart="15" Value="85" />
        <gauge:MarkerPointer Value="70" MarkerShape = "Triangle"/>
      </gauge:Scale.Pointers>
    </gauge:Scale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

**C#**

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
sfCircularGauge.CircularCoefficient = 0.7f;
Header header = new Header();
header.Text = "Speedometer";
header.ForegroundColor = Color.Black;
sfCircularGauge.Headers.Add(header);
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();
Scale scale = new Scale();
NeedlePointer needlePointer = new NeedlePointer();
needlePointer.Value = 70;
scale.Pointers.Add(needlePointer);
scales.Add(scale);
RangePointer rangePointer = new RangePointer();
rangePointer.RangeStart = 15;
rangePointer.Value = 85;
scale.Pointers.Add(rangePointer);
MarkerPointer markerPointer = new MarkerPointer();
markerPointer.Value = 70;
markerPointer.MarkerShape = MarkerShape.Triangle;
scale.Pointers.Add(markerPointer);
Range range = new Range();
range.StartValue = 0;
range.EndValue = 50;
scale.Ranges.Add(range);
sfCircularGauge.Scales = scales;
Content = sfCircularGauge;

```

**Gauge alignment**

The [IsCenterAligned](#) property is used to align the gauge to the center. In semi-circular gauge, bottom space will be removed using the [IsCenterAligned](#) property.

**XML**

```

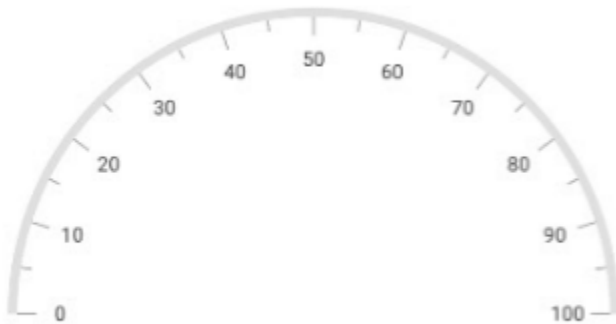
<gauge:SfCircularGauge IsCenterAligned="True">

```

```
<gauge:SfCircularGauge.Scales>  
<gauge:Scale StartAngle="180" SweepAngle="180"/>  
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

## C#

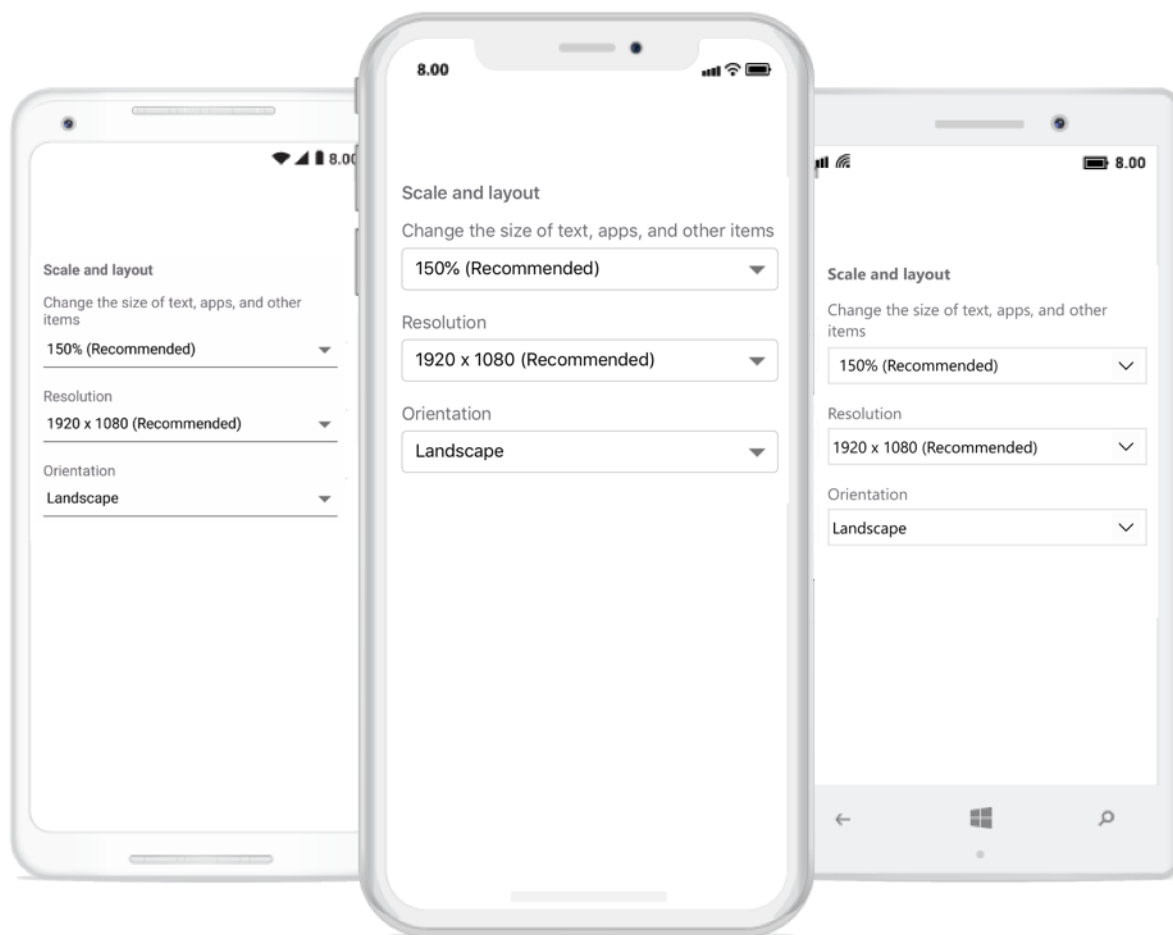
```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
sfCircularGauge.IsCenterAligned = true;  
ObservableCollection<Scale> scales = new ObservableCollection<Scale>();  
Scale scale = new Scale();  
scale.StartAngle = 180;  
scale.SweepAngle = 180;  
scales.Add(scale);  
sfCircularGauge.Scales = scales;  
Content = sfCircularGauge;
```



## SfComboBox

### Overview

The combo box is a textbox component that allows users to type a value or choose an option from the list of predefined options. This control has several out of box features such as data binding, filtering, UI customization, and more.



## Key features

- Editable mode – Supports both editable and non-editable text box to choose selected item from given data source.
- Filtering mode – Provides options to support both filtering and non-filtering suggestion lists. Provides three ways to display filtered suggestions. These include displaying suggestions using the drop-down list, which appends the first suggestion to text, and a combination of both.
- Suggestion modes – Suggestions can be filtered in different modes, such as StartsWith, EndWith, Contains, Equals, Custom and more. ComboBox provides both case-sensitive and case-insensitive modes. The items only filter the Allow Filtering property has enabled.
- MultiSelection – Provides two different ways to select multiple items from the suggestion list. They are using Token representation and Delimiter. In Token mode, the text can be wrapped in two ways. They are Wrap and None. In Wrap mode text will be wrapped to next line. When using None, the text will be wrapped horizontally.
- Customization Support – Provide options to customize both the Entry and Suggestion drop down.
- Header and Footer – Header and Footer content can be given in the top and bottom of the Suggestion list in SfComboBox
- Highlighting Text – Highlights the matching text in the suggestion list based on the input given in it.



- **Watermark** – Supports explanatory text inside the combo box control until the user inputs text. Watermark is restored again if user clears the text in control.

## Getting Started

This section explains the steps required to create the combo box control, populate it with data and filter the suggestions. This section covers only the minimal features that are needed to get started with the control.

### Adding SfComboBox reference

You can add SfComboBox reference using one of the following methods:

#### Method 1: Adding SfComboBox reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfComboBox). To add SfComboBox to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfComboBox](https://www.nuget.org/packages/Syncfusion.Xamarin.SfComboBox), and then install it.

![Adding SfComboBox reference from NuGet](images/Getting-Started/Adding SfComboBox reference.png)

---

**Note:** Install the same version of SfComboBox NuGet in all the projects.

---

#### Method 2: Adding SfComboBox reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfComboBox control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfComboBox assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfComboBox.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfComboBox.XForms.dll Syncfusion.SfComboBox.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfComboBox.XForms.dll Syncfusion.SfComboBox.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfComboBox.XForms.dll Syncfusion.SfComboBox.XForms.UWP.dll Syncfusion.Core.XForms.dll

	Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
--	--

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** If you are adding the references from toolbox, this step is not needed.

#### *Additional Step for iOS*

Create an instance of the `SfComboBoxRenderer` in `FinishedLaunching` overridden method of an `AppDelegate` class in iOS project as shown in the following codes:

#### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    new Syncfusion.XForms.iOS.ComboBox.SfComboBoxRenderer();
    global::Xamarin.Forms.Forms.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### *Additional step for UWP*

This step is required only if the application is deployed in release mode with .NET native tool chain enabled. This is for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the `SfComboBox` assembly at `OnLaunched` overridden method of the `App` class in UWP project is the suggested workaround. The following code example show to resolve this issue.

#### **C#**

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    #if DEBUG
    if (System.Diagnostics.Debugger.IsAttached)
    {
        this.DebugSettings.EnableFrameRateCounter = true;
    }
    #endif
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame == null)
    {
        rootFrame = new Frame();
        rootFrame.NavigationFailed += OnNavigationFailed;
        List<System.Reflection.Assembly> assembliesToInclude = new
        List<System.Reflection.Assembly>();
        // Add all the renderer assemblies your app uses
        assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.ComboBox.SfComboBoxRender
        er).GetTypeInfo().Assembly);
    }
```

```
// Replace the Xamarin.Forms.Forms.Init(e);
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
{
    //TODO: Load state from previously suspended application
}
// Place the frame in the current Window
Window.Current.Content = rootFrame;
}
if (rootFrame.Content == null)
{
    // When the navigation stack isn't restored navigate to the first page,
    // configuring the new page by passing required information as a navigation
    // parameter
    rootFrame.Navigate(typeof(MainPage), e.Arguments);
}
// Ensure the current window is active
Window.Current.Activate();
}
```

### Initializing ComboBox

Import the SfComboBox namespace in respective page as shown in the following code.

#### XML

```
xmlns:combobox="clr-
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
```

#### C#

```
using Syncfusion.XForms.ComboBox;
```

Then initialize an empty combobox as shown in the following code,

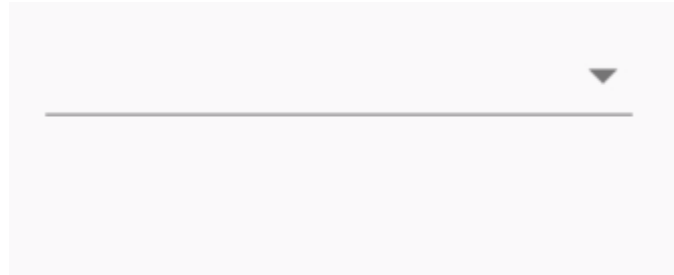
#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:combobox="clr-
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
xmlns:local="clr-namespace:NamespaceName"
x:Class="NamespaceName.ClassName">
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"/>
</StackLayout>
</ContentPage>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
```

```
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
layout.Children.Add(comboBox);
Content = layout;
```



### Populating ComboBox with data

Now, a list of string with resolution list is created and added to the SfComboBox data source property. This list is populated as suggestion list by setting the 'DataSource' property based on text entry. You can customize the drop-down height using the MaximumDropDownHeight property. Add the DataSource for the SfComboBox as shown in the following code.

### XML

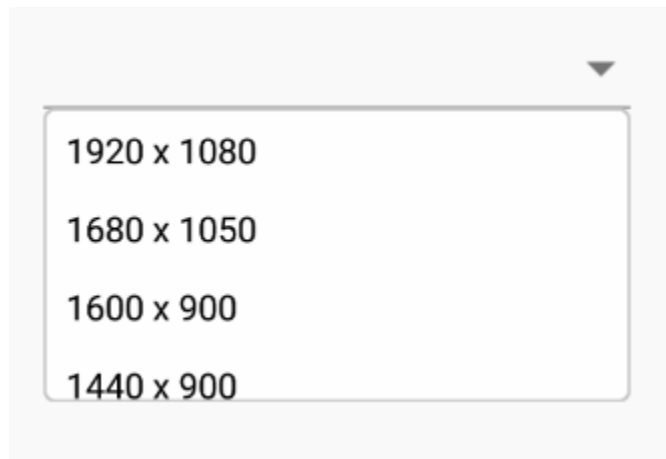
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:combobox="clr-
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
xmlns:local="clr-namespace:NamespaceName"
x:Class="NamespaceName.ClassName">
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox">
<combobox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String> 1920 x 1080 </x:String>
<x:String> 1680 x 1050 </x:String>
<x:String> 1600 x 900 </x:String>
<x:String> 1440 x 900 </x:String>
<x:String> 1400 x 1050 </x:String>
<x:String> 1366 x 768 </x:String>
<x:String> 1360 x 768 </x:String>
<x:String> 1280 x 1024 </x:String>
<x:String> 1280 x 960 </x:String>
<x:String> 1280 x 720 </x:String>
<x:String> 854 x 480 </x:String>
<x:String> 800 x 480 </x:String>
<x:String> 480 x 640 </x:String>
<x:String> 480 x 320 </x:String>
<x:String> 432 x 240 </x:String>
<x:String> 360 x 640 </x:String>
<x:String> 320 x 240 </x:String>
</ListCollection:List>
</combobox:SfComboBox.ComboBoxSource>
</combobox:SfComboBox>
```

```
</StackLayout>
</ContentPage>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> resolutionList = new List<String>();
resolutionList.Add("1920 x 1080");
resolutionList.Add("1680 x 1050");
resolutionList.Add("1600 x 900");
resolutionList.Add("1440 x 900");
resolutionList.Add("1400 x 1050");
resolutionList.Add("1366 x 768");
resolutionList.Add("1360 x 768");
resolutionList.Add("1280 x 1024");
resolutionList.Add("1280 x 960");
resolutionList.Add("1280 x 720");
resolutionList.Add("854 x 480");
resolutionList.Add("800 x 480");
resolutionList.Add("480 x 640");
resolutionList.Add("480 x 320");
resolutionList.Add("432 x 240");
resolutionList.Add("360 x 640");
resolutionList.Add("320 x 240");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.ComboBoxSource = resolutionList;
layout.Children.Add(comboBox);
Content = layout;
```

Refer [this](#) link to learn more about the options available in SfComboBox to populate data.



### ComboBox modes

The combo box control supports both editable and non-editable text boxes to choose selected items in given data source. You can select an item from the suggestion list.

N > The default value of the `IsEditableMode` property is false.

#### Non-editable combo box

Non-editable mode prevents users from typing and allows them to select items from the drop-down list. In non-editable mode, the suggestion box can be displayed by clicking the control or drop-down button.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    MaximumDropDownHeight="200" IsEditableMode="false">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> 1920 x 1080 </x:String>
        <x:String> 1680 x 1050 </x:String>
        <x:String> 1600 x 900 </x:String>
        <x:String> 1440 x 900 </x:String>
        <x:String> 1400 x 1050 </x:String>
        <x:String> 1366 x 768 </x:String>
        <x:String> 1360 x 768 </x:String>
        <x:String> 1280 x 1024 </x:String>
        <x:String> 1280 x 960 </x:String>
        <x:String> 1280 x 720 </x:String>
        <x:String> 854 x 480 </x:String>
        <x:String> 800 x 480 </x:String>
        <x:String> 480 X 640 </x:String>
        <x:String> 480 x 320 </x:String>
        <x:String> 432 x 240 </x:String>
        <x:String> 360 X 640 </x:String>
        <x:String> 320 x 240 </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

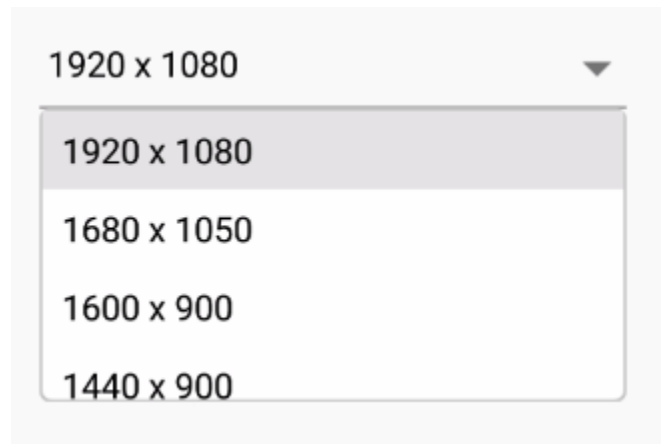
### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> resolutionList = new List<String>();
resolutionList.Add("1920 x 1080");
resolutionList.Add("1680 x 1050");
resolutionList.Add("1600 x 900");
resolutionList.Add("1440 x 900");
resolutionList.Add("1400 x 1050");
resolutionList.Add("1366 x 768");
resolutionList.Add("1360 x 768");
resolutionList.Add("1280 x 1024");
```

```

resolutionList.Add("1280 x 960");
resolutionList.Add("1280 x 720");
resolutionList.Add("854 x 480");
resolutionList.Add("800 x 480");
resolutionList.Add("480 x 640");
resolutionList.Add("480 x 320");
resolutionList.Add("432 x 240");
resolutionList.Add("360 x 640");
resolutionList.Add("320 x 240");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.MaximumDropDownHeight = 200;
comboBox.IsEditableMode = false;
comboBox.ComboBoxSource = resolutionList;
layout.Children.Add(comboBox);
Content = layout;

```



#### Editable combo box

In editable mode, the combo box allows users to edit in the text box that shows suggestions in drop-down list based on the input. With the previous codes, the 'IsEditableMode' property can be set to true. This helps users to edit the combo box control.

#### XML

```

<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    MaximumDropDownHeight="200" IsEditableMode="true">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> 1920 x 1080 </x:String>
        <x:String> 1680 x 1050 </x:String>
        <x:String> 1600 x 900 </x:String>
        <x:String> 1440 x 900 </x:String>
        <x:String> 1400 x 1050 </x:String>
        <x:String> 1366 x 768 </x:String>
        <x:String> 1360 x 768 </x:String>
        <x:String> 1280 x 1024 </x:String>
        <x:String> 1280 x 960 </x:String>
        <x:String> 1280 x 720 </x:String>
        <x:String> 854 x 480 </x:String>
        <x:String> 800 x 480 </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>

```

```
<x:String> 480 X 640 </x:String>
<x:String> 480 x 320 </x:String>
<x:String> 432 x 240 </x:String>
<x:String> 360 X 640 </x:String>
<x:String> 320 x 240 </x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> resolutionList = new List<String>();
resolutionList.Add("1920 x 1080");
resolutionList.Add("1680 x 1050");
resolutionList.Add("1600 x 900");
resolutionList.Add("1440 x 900");
resolutionList.Add("1400 x 1050");
resolutionList.Add("1366 x 768");
resolutionList.Add("1360 x 768");
resolutionList.Add("1280 x 1024");
resolutionList.Add("1280 x 960");
resolutionList.Add("1280 x 720");
resolutionList.Add("854 x 480");
resolutionList.Add("800 x 480");
resolutionList.Add("480 X 640");
resolutionList.Add("480 x 320");
resolutionList.Add("432 x 240");
resolutionList.Add("360 X 640");
resolutionList.Add("320 x 240");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.MaximumDropDownHeight = 200;
comboBox.IsEditableMode = true;
comboBox.ComboBoxSource = resolutionList;
layout.Children.Add(comboBox);
Content = layout;
```





### Retrieving selected values

When selecting an item from the drop-down list, the selection changed event will be called. Using the following code snippet, a dialogue box will be displayed when a new item is selected from the suggestion box.

### XML

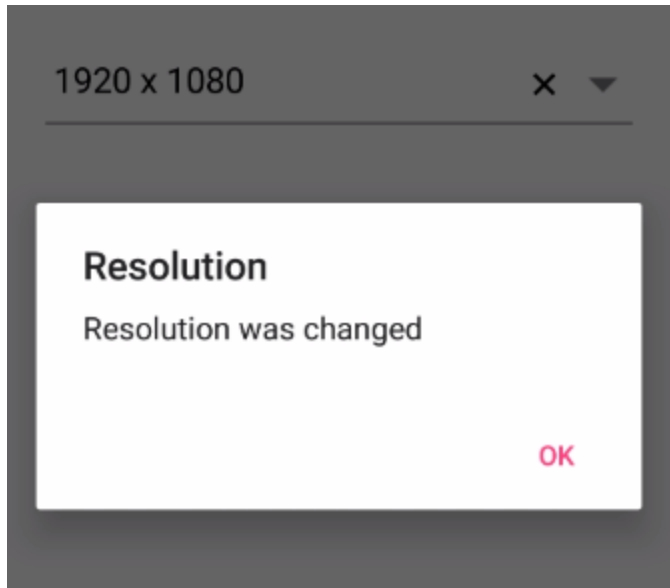
```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    MaximumDropDownHeight="200" IsEditableMode="true"
    SelectionChanged="ComboBox_SelectionChanged">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> 1920 x 1080 </x:String>
        <x:String> 1680 x 1050 </x:String>
        <x:String> 1600 x 900 </x:String>
        <x:String> 1440 x 900 </x:String>
        <x:String> 1400 x 1050 </x:String>
        <x:String> 1366 x 768 </x:String>
        <x:String> 1360 x 768 </x:String>
        <x:String> 1280 x 1024 </x:String>
        <x:String> 1280 x 960 </x:String>
        <x:String> 1280 x 720 </x:String>
        <x:String> 854 x 480 </x:String>
        <x:String> 800 x 480 </x:String>
        <x:String> 480 X 640 </x:String>
        <x:String> 480 x 320 </x:String>
        <x:String> 432 x 240 </x:String>
        <x:String> 360 X 640 </x:String>
        <x:String> 320 x 240 </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
```

```
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
List<String> resolutionList = new List<String>();
resolutionList.Add("1920 x 1080");
resolutionList.Add("1680 x 1050");
resolutionList.Add("1600 x 900");
resolutionList.Add("1440 x 900");
resolutionList.Add("1400 x 1050");
resolutionList.Add("1366 x 768");
resolutionList.Add("1360 x 768");
resolutionList.Add("1280 x 1024");
resolutionList.Add("1280 x 960");
resolutionList.Add("1280 x 720");
resolutionList.Add("854 x 480");
resolutionList.Add("800 x 480");
resolutionList.Add("480 x 640");
resolutionList.Add("480 x 320");
resolutionList.Add("432 x 240");
resolutionList.Add("360 x 640");
resolutionList.Add("320 x 240");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.MaximumDropDownHeight = 200;
comboBox.IsEditableMode = true;
comboBox.ComboBoxSource = resolutionList;
comboBox.SelectionChanged += ComboBox_SelectionChanged;
layout.Children.Add(comboBox);
Content = layout;
}

private void ComboBox_SelectionChanged(object sender,
Syncfusion.XForms.ComboBox.SelectionChangedEventArgs e)
{
    Xamarin.Forms.Application.Current.MainPage.DisplayAlert("Resolution",
"Resolution was changed", "OK");
}
```



The complete Getting Started sample is available in [this](#) link.

### Populating data

SfComboBox control can be populated with a list of string or business objects, which assists the users when typing. Users can choose one item from the filtered suggestion list.

The [DataSource] property is used to populate data in the combo box control. This section explains how to populate the combo box control with list of string and list of employee details separately.

### Populating string data

Create an instance of string list and populate items as shown in the following codes.

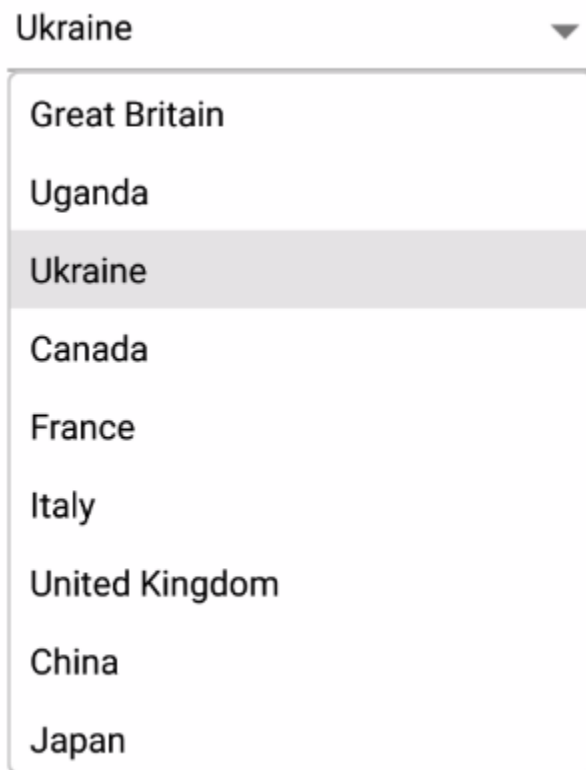
#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Great Britain </x:String>
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> Canada </x:String>
        <x:String> France </x:String>
        <x:String> Italy </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> China </x:String>
        <x:String> Japan </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
```

```
VerticalOptions = LayoutOptions.Start,  
HorizontalOptions = LayoutOptions.Start,  
Padding = new Thickness(30)  
};  
List<String> countryNames = new List<String>();  
countryNames.Add("Great Britain");  
countryNames.Add("Uganda");  
countryNames.Add("Ukraine");  
countryNames.Add("Canada");  
countryNames.Add("France");  
countryNames.Add("Italy");  
countryNames.Add("United Kingdom");  
countryNames.Add("China");  
countryNames.Add("Japan");  
SfComboBox comboBox = new SfComboBox();  
comboBox.HeightRequest = 40;  
comboBox.ComboBoxSource = countryNames;  
layout.Children.Add(comboBox);  
Content = layout;
```



### Populating business objects

Apart from string data, SfComboBox can deal business object data also. Now create Model and ViewModel classes to populate the combo box control with employee details.

#### Create and Initialize business models

Define a simple model class employee with fields ID and name, and then populate employee data in ViewModel.

**C#**

```

public class Employee
{
    private int id;
    public int ID
    {
        get { return id; }
        set { id = value; }
    }
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}

public class EmployeeViewModel
{
    private ObservableCollection<Employee> employeeCollection;
    public ObservableCollection<Employee> EmployeeCollection
    {
        get { return employeeCollection; }
        set { employeeCollection = value; }
    }
    public EmployeeViewModel()
    {
        employeeCollection = new ObservableCollection<Employee>();
        employeeCollection.Add(new Employee() { ID = 1, Name = "Frank" });
        employeeCollection.Add(new Employee() { ID = 2, Name = "James" });
        employeeCollection.Add(new Employee() { ID = 3, Name = "Steve" });
        employeeCollection.Add(new Employee() { ID = 4, Name = "Lucas" });
        employeeCollection.Add(new Employee() { ID = 5, Name = "Mark" });
        employeeCollection.Add(new Employee() { ID = 6, Name = "Michael" });
        employeeCollection.Add(new Employee() { ID = 7, Name = "Aldrin" });
        employeeCollection.Add(new Employee() { ID = 8, Name = "Jack" });
        employeeCollection.Add(new Employee() { ID = 9, Name = "Howard" });
    }
}

```

*Populate data in ComboBox*

Now populate this EmployeeViewModel data in SfComboBox control by binding with `[DataSource]` property.

**XML**

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:combobox="clr-namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
xmlns:local="clr-namespace:NamespaceName"
x:Class="NamespaceName.ClassName">
    <ContentPage.BindingContext>
        <local:EmployeeViewModel/>
    </ContentPage.BindingContext>
    <StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">

```

```
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"  
DataSource="{Binding EmployeeCollection}" DisplayMemberPath="Name" />  
</StackLayout>  
</ContentPage>
```

## C#

```
StackLayout layout = new StackLayout()  
{  
    VerticalOptions = LayoutOptions.Start,  
    HorizontalOptions = LayoutOptions.Start,  
    Padding = new Thickness(30)  
};  
EmployeeViewModel employee = new EmployeeViewModel();  
SfComboBox comboBox = new SfComboBox();  
comboBox.HeightRequest = 40;  
comboBox.BindingContext = employee;  
comboBox.DataSource = employee.EmployeeCollection;  
comboBox.DisplayMemberPath = "Name";  
layout.Children.Add(comboBox);  
Content = layout;
```

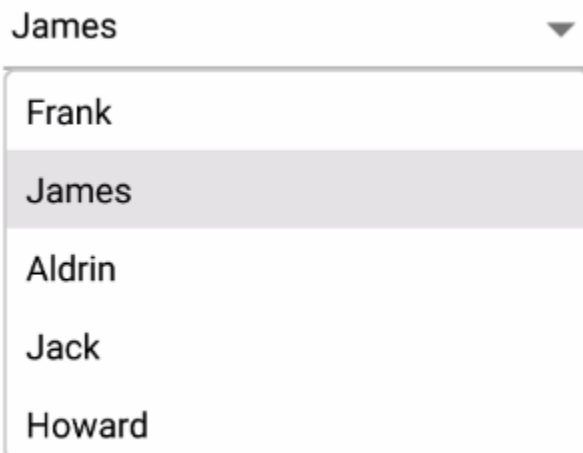
**Note:** Set the EmployeeViewModel instance as the BindingContext of your control; this is done to bind properties of EmployeeViewModel to SfComboBox.

### Setting DisplayMemberPath

The combo box control is populated with a list of employees. But the employee model contains two properties ID and Name, So it is necessary to intimate by which property it should filter suggestions. The [DisplayMemberPath] property specifies the property path with type of filtering is done on business objects.

## C#

```
comboBox.DisplayMemberPath = "Name";
```



### Setting ItemTemplate

The [ItemTemplate] property helps to decorate suggestion items with custom templates. The following code explains the steps to add an image to the suggestion list item.

#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
DisplayMemberPath="Name" DataSource="{Binding EmployeeCollection}">
<combobox:SfComboBox.ItemTemplate>
<DataTemplate>
<StackLayout Orientation="Horizontal">
<Image Source="User.png" WidthRequest="12"/>
<Label Text="{Binding Name}" />
</StackLayout>
</DataTemplate>
</combobox:SfComboBox.ItemTemplate>
</combobox:SfComboBox>
</StackLayout>
```

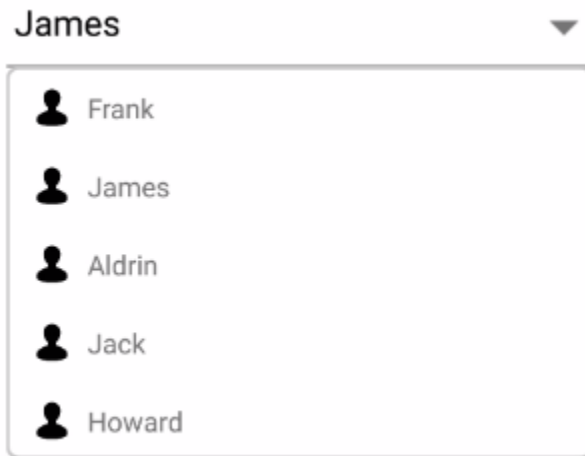
The ItemTemplate in the previous XAML code is translated to C# which is shown in the following code.

#### C#

```
DataTemplate itemTemplate = new DataTemplate(() =>
{
    StackLayout stack;
    Image image;
    Label label;
    stack = new StackLayout();
    stack.Orientation = StackOrientation.Horizontal;
    image = new Image();
    image.Source = (FileImageSource) ImageSource.FromFile("User.png");
    label = new Label();
    label.SetBinding(Label.TextProperty, "Name");
    stack.Children.Add(image);
    stack.Children.Add(label);
    return new ViewCell { View = stack };
});
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
EmployeeViewModel employee = new EmployeeViewModel();
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.BindingContext = employee;
comboBox.DataSource = employee.EmployeeCollection;
comboBox.ItemTemplate = itemTemplate;
layout.Children.Add(comboBox);
Content = layout;
```

Refer to [this](#) link to learn more about the customizing options available in the combo box control.

**Note:** Add the required image in drawable folder(Android), Resources folder(iOS) and at project location for UWP.



### DataTemplateSelector

SfComboBox supports DataTemplateSelector, which is used to choose a DataTemplate based on data object.

#### XML

```
<ContentPage.BindingContext>
<local:MobileDetailViewModel/>
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="default">
<ViewCell>
<Grid Padding="5">
<Label Text="{Binding Mobile}" TextColor="Green"/>
</Grid>
</ViewCell>
</DataTemplate>
<DataTemplate x:Key="specific">
<ViewCell>
<Grid Padding="5">
<Label Text="{Binding Mobile}" BackgroundColor="LightGray" TextColor="Red"/>
</Grid>
</ViewCell>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox:SfComboBox DataSource="{Binding MobileCollection}"
DisplayMemberPath="Mobile">
<comboBox:SfComboBox.ItemTemplate>
<local:DataTemplateSelectorViewModel DefaultTemplate="{StaticResource
default}" SpecificDataTemplate="{StaticResource specific}" />
</comboBox:SfComboBox.ItemTemplate>
</comboBox:SfComboBox>
```



```
</StackLayout>
```

**C#**

```
public partial class MainPage : ContentPage
{
    DataTemplate defaultTemplate;
    DataTemplate specifictempalte;
    public MainPage()
    {
        InitializeComponent();
        MobileDetailViewModel mobileDetailViewModel = new MobileDetailViewModel();
        defaultTemplate = new DataTemplate(() =>
        {
            Grid grid = new Grid();
            grid.Padding = new Thickness(5);
            Label label = new Label();
            label.SetBinding(Label.TextProperty, "Mobile");
            label.TextColor = Color.Green;
            grid.Children.Add(label);
            return new ViewCell { View = grid };
        });
        specifictempalte = new DataTemplate(() =>
        {
            Grid grid = new Grid();
            grid.Padding = new Thickness(5);
            Label label = new Label();
            label.SetBinding(Label.TextProperty, "Mobile");
            label.BackgroundColor = Color.LightGray;
            label.TextColor = Color.Red;
            grid.Children.Add(label);
            return new ViewCell { View = grid };
        });
        StackLayout layout = new StackLayout()
        {
            VerticalOptions = LayoutOptions.Start,
            HorizontalOptions = LayoutOptions.Start,
            Padding = new Thickness(30)
        };
        SfComboBox comboBox = new SfComboBox();
        comboBox.HeightRequest = 40;
        comboBox.DataSource = mobileDetailViewModel.MobileCollection;
        this.BindingContext = mobileDetailViewModel;
        comboBox.DisplayMemberPath = "Mobile";
        comboBox.ItemTemplate = new DataTemplateSelectorViewModel { DefaultTemplate = defaultTemplate, SpecificDataTemplate = specifictempalte };
        layout.Children.Add(comboBox);
        Content = layout;
    }
}
```

*Create and Initialize Business Models*

Define a simple model class MobileDetail with fields IsAvailableInStock, Mobile and populate mobile detail in ViewModel.

**C#**

```
public class MobileDetailViewModel
{
    public ObservableCollection<MobileDetail> MobileCollection { get; set; }
    public MobileDetailViewModel()
    {
        this.MobileCollection = new ObservableCollection<MobileDetail>()
        {
            new MobileDetail () { Mobile="Samasung S8", IsAvailableInStock=false },
            new MobileDetail () { Mobile="Samasung S9", IsAvailableInStock=true },
            new MobileDetail () { Mobile="Samsung S10", IsAvailableInStock=true },
            new MobileDetail () { Mobile="Samsung S10 plus", IsAvailableInStock=true },
            new MobileDetail () { Mobile="iPhone 7", IsAvailableInStock=true },
            new MobileDetail () { Mobile="iPhone 8", IsAvailableInStock=true },
            new MobileDetail () { Mobile="iPhone X", IsAvailableInStock=false },
            new MobileDetail () { Mobile="iPhone XR", IsAvailableInStock=false },
            new MobileDetail () { Mobile="iPhone XS", IsAvailableInStock=true },
        };
    }
}

public class MobileDetail
{
    public string Mobile { get; set; }
    public bool IsAvailableInStock { get; set; }
}
```

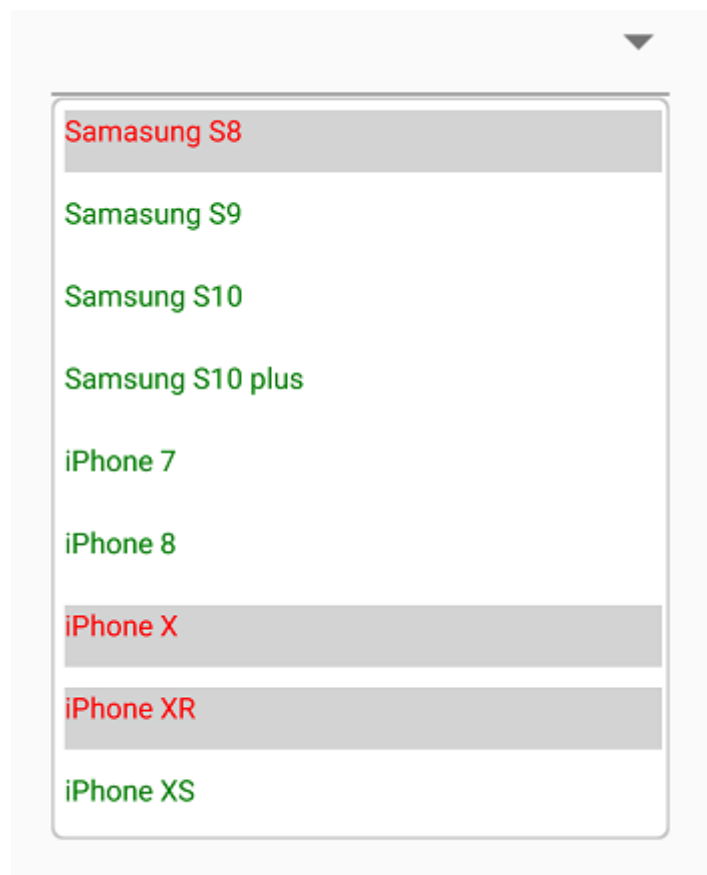
## OnSelectTemplate

The OnSelectTemplate is an overridden method to return a particular DataTemplate. The following code sample demonstrates how to use the OnSelectTemplate method.

**C#**

```
public class DataTemplateSelectorViewModel : DataTemplateSelector
{
    public DataTemplate DefaultTemplate { get; set; }
    public DataTemplate SpecificDataTemplate { get; set; }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject container)
    {
        var message = item as MobileDetail;
        if (message.IsAvailableInStock == null)
            return null;
        return message.IsAvailableInStock == false ? SpecificDataTemplate :
            DefaultTemplate;
    }
}
```

The following screenshot illustrates the output of above code.



We have attached sample for reference. You can download the sample from the following link.

Sample Link: [SfComboBox\\_DataTemplateSelector](#)

### Handling Selected Items

SfComboBox provides a way to handle the selected item using the following properties:

- SelectedIndex
- SelectedIndices
- SelectedItem

#### SelectedIndex

You can get or set the index of the selected item using the [SelectedIndex](#) property. It is applicable only when [MultiSelectMode](#) is set to None.

*Set the index of item to be selected*

The [SelectedIndex](#) property holds the index of selected item in suggestion list.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ComboBox"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
```

```

xmlns:comboBox="clr
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
x:Class="ComboBox.MainPage">
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox:SfComboBox x:Name="comboBox" HeightRequest="40"
MultiSelectMode="None" SelectedIndex="1">
<comboBox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
</ContentPage>

```

## C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace ComboBox
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfComboBox comboBox = new SfComboBox()
            {
                HeightRequest = 40,
                SelectedIndex = 1,
                MultiSelectMode=MultiSelectMode.None,
                DataSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",

```

```

"Argentina",
"Anguilla",
"Albania",
"Algeria",
"Andorra",
"Angola"
}
};
stackLayout.Children.Add(comboBox);
this.Content = stackLayout;
}
}
}

```

#### *Retrieve the index of selected item*

When an item is selected from suggestion list, its index can be retrieved using the [SelectedIndex](#) property.

The following code sample demonstrates how to retrieve [SelectedIndex](#) and display it in an alert.

#### **XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ComboBox"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:comboBox="clr
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
x:Class="ComboBox.MainPage">
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox:SfComboBox x:Name="comboBox" HeightRequest="40"
MultiSelectMode="None" SelectionChanged="ComboBox_SelectionChanged">
<comboBox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
</ContentPage>

```

#### **C#**

```

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace ComboBox
{
    public partial class MainPage : ContentPage
    {
        SfComboBox comboBox;
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            comboBox = new SfComboBox()
            {
                HeightRequest = 40, MultiSelectMode=MultiSelectMode.None,
                DataSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",
                    "Argentina",
                    "Anguilla",
                    "Albania",
                    "Algeria",
                    "Andorra",
                    "Angola"
                }
            };
            comboBox.SelectionChanged += comboBox_SelectionChanged;
            stackLayout.Children.Add(comboBox);
            this.Content = stackLayout;
        }
        private void comboBox_SelectionChanged(object sender,
            Syncfusion.XForms.ComboBox.SelectionChangedEventArgs e)
        {
            DisplayAlert("Selection Changed", "SelectedIndex: " +
                comboBox.SelectedIndex, "OK");
        }
    }
}

```

### SelectedIndices

You can get or set the indices of the selected items using the [SelectedIndices](#) property. It is applicable when [MultiSelectMode](#) is set to either Token or Delimiter.

#### *Set the indices of items*

The [SelectedIndices](#) property holds the indices of the selected items in suggestion list.

### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ComboBox"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:comboBox="clr
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
x:Class="ComboBox.MainPage">
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox:SfComboBox x:Name="comboBox" HeightRequest="40"
MultiSelectMode="Token" SelectedIndices="{Binding SelectedIndices}">
<comboBox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
</ContentPage>
```

## C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace ComboBox
{
    public partial class MainPage : ContentPage
    {
        private object selectedIndices;
        public object SelectedIndices
        {
            get { return selectedIndices; }
            set { selectedIndices = value; }
        }
        public MainPage()
        {
            InitializeComponent();
            SelectedIndices = new List<int>() { 2, 4, 7 };
            this.BindingContext = this;
        }
    }
}
```

*Retrieve the indices of selected item*

When an item is selected from suggestion list, its index can be retrieved using the [SelectedIndices](#) property.

The following code sample demonstrates how to retrieve [SelectedIndices](#) and display them in ListView.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ComboBox"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:comboBox="clr
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
x:Class="ComboBox.MainPage">
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox:SfComboBox x:Name="comboBox"HeightRequest="40"
MultiSelectMode="Token">
<comboBox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
<ListView
x:Name="MainListView"
RowHeight="30"
ItemsSource="{Binding Source={x:Reference comboBox},Path=SelectedIndices}">
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout Orientation="Horizontal">
<Label Text="SelectedIndex:"/>
<Label Text="{Binding}" />
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</StackLayout>
</ContentPage>
```

**C#**



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace ComboBox
{
    public partial class MainPage : ContentPage, INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void NotifyPropertyChanged([CallerMemberName] string
        propertyName = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
        private object selectedIndices;
        public object SelectedIndices
        {
            get { return selectedIndices; }
            set
            {
                selectedIndices = value;
                NotifyPropertyChanged("SelectedIndices");
            }
        }
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = this;
        }
    }
}

```

### SelectedItem

The **SelectedItem** property is used to select a particular item from the suggestion list. You can either get or set the **SelectedItem**.

#### How to set the SelectedItem

The following code sample demonstrates how to set **SelectedItem**.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ComboBox"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:comboBox="clr
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
x:Class="ComboBox.MainPage">
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox:SfComboBox x:Name="comboBox" HeightRequest="40"
SelectedItem="Angola">
<comboBox:SfComboBox.ComboBoxSource>

```

```

<ListCollection:List x:TypeArguments="x:String">
  <x:String>Antigua and Barbuda</x:String>
  <x:String>American Samoa</x:String>
  <x:String>Afghanistan</x:String>
  <x:String>Antarctica</x:String>
  <x:String>Argentina</x:String>
  <x:String>Anguilla</x:String>
  <x:String>Albania</x:String>
  <x:String>Algeria</x:String>
  <x:String>Andorra</x:String>
  <x:String>Angola</x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
</ContentPage>

```

## C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace ComboBox
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            SfComboBox comboBox = new SfComboBox()
            {
                HeightRequest = 40,
                SelectedItem = "Angola",
                MultiSelectMode = MultiSelectMode.None,
                DataSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",
                    "Argentina",
                    "Anguilla",
                    "Albania",
                    "Algeria",
                    "Andorra",
                    "Angola"
                }
            };
        }
    }
}

```

```

};
stackLayout.Children.Add(comboBox);
this.Content = stackLayout;
}
}
}

```

### Retrieve the selected item

When an item is selected from suggestion list, it can be retrieved using the [SelectedItem](#) property.

The `SelectedValuePath` API is used to retrieve the value of the selected item in drop-down when the item is selected. The `SelectedValue` property is updated based on the selection. For loading the default values in the control, use only the `SelectedItem`, and the `SelectedValue` is used for retrieving the value of our desired property[ID] based on the selection.

There are scenarios in which `SelectedValue` can have duplicate items when `DisplayMemberPath` is loaded with Countries and `SelectedValue` is bound to Continent; it will be the same for many countries. In such scenarios, you cannot populate based on `SelectedValue`.

The following code sample demonstrates how to retrieve [SelectedItem](#) and display it in an alert.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ComboBox"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:comboBox="clr
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
x:Class="ComboBox.MainPage">
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox:SfComboBox x:Name="comboBox" HeightRequest="40"
MultiSelectMode="None" SelectionChanged="ComboBox_SelectionChanged">
<comboBox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Antigua and Barbuda</x:String>
<x:String>American Samoa</x:String>
<x:String>Afghanistan</x:String>
<x:String>Antarctica</x:String>
<x:String>Argentina</x:String>
<x:String>Anguilla</x:String>
<x:String>Albania</x:String>
<x:String>Algeria</x:String>
<x:String>Andorra</x:String>
<x:String>Angola</x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
</ContentPage>

```

### C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace ComboBox
{
    public partial class MainPage : ContentPage
    {
        SfComboBox comboBox;
        public MainPage()
        {
            InitializeComponent();
            StackLayout stackLayout = new StackLayout()
            {
                VerticalOptions = LayoutOptions.Start,
                HorizontalOptions = LayoutOptions.Start,
                Padding = 30
            };
            comboBox = new SfComboBox()
            {
                HeightRequest = 40,
                MultiSelectMode = MultiSelectMode.None,
                DataSource = new List<string>()
                {
                    "Antigua and Barbuda",
                    "American Samoa",
                    "Afghanistan",
                    "Antarctica",
                    "Argentina",
                    "Anguilla",
                    "Albania",
                    "Algeria",
                    "Andorra",
                    "Angola"
                }
            };
            comboBox.SelectionChanged += ComboBox_SelectionChanged;
            stackLayout.Children.Add(comboBox);
            this.Content = stackLayout;
        }
        private void ComboBox_SelectionChanged(object sender,
            Syncfusion.XForms.ComboBox.SelectionChangedEventArgs e)
        {
            DisplayAlert("Selection Changed", "SelectedItem: " + comboBox.SelectedItem,
                "OK");
        }
    }
}

```

### Diacritic Sensitivity

The control does not stick with one type of keyboard, so you can populate items from a language with letters containing diacritics, and search for them with English characters from an en-US keyboard. Users can enable or disable the diacritic sensitivity with the [IgnoreDiacritic](#) property. In the below code

example we have illustrate how to enables the diacritic sensitivity so that items in the suggestion list get populated by entering any diacritic character of that alphabet.

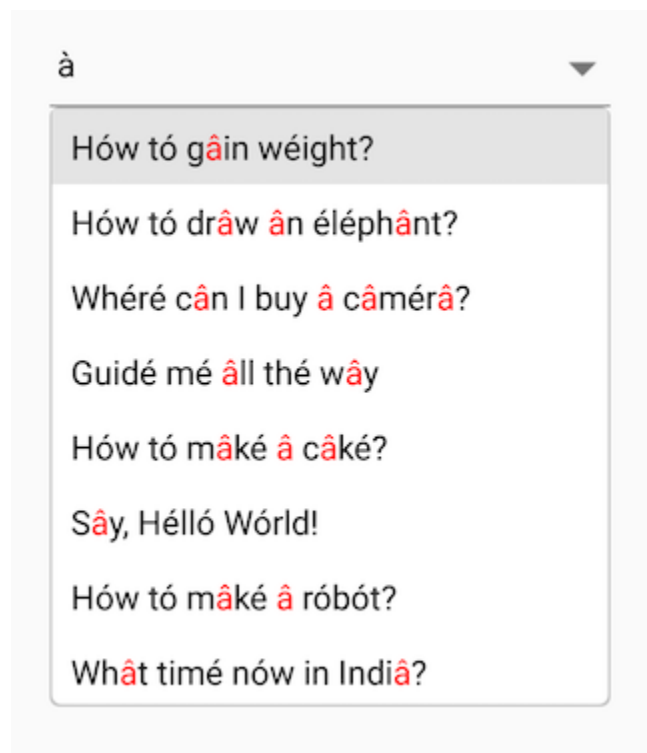
**Note:** Diacritic Sensitivity support has enhanced only on iOS and Android platform.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="True" AllowFiltering="True"
    TextHighlightMode="MultipleOccurrence" SuggestionMode="Contains"
    HighlightedTextColor="Red" IgnoreDiacritic="false">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String>Hów tó gâin wéight?</x:String>
        <x:String>Hów tó drâw ân éléphânt?</x:String>
        <x:String>Whéré cân I buy â câmérâ?</x:String>
        <x:String>Guidé mé âll thé wây</x:String>
        <x:String>Hów tó mâké â câké?</x:String>
        <x:String>Sây, Hélló Wórlđ!</x:String>
        <x:String>Hów tó mâké â róbót?</x:String>
        <x:String>Whât tímé nów in Indiâ?</x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout mainLayout = new StackLayout()
{
    Padding = new Thickness(30)
};
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.TextHighlightMode = OccurrenceMode.MultipleOccurrence;
comboBox.SuggestionMode = SuggestionMode.Contains;
comboBox.HighlightedTextColor = Color.Red;
comboBox.IgnoreDiacritic = false;
List<String> diacritic = new List<String>();
diacritic.Add("Hów tó gâin wéight?");
diacritic.Add("Hów tó drâw ân éléphânt?");
diacritic.Add("Whéré cân I buy â câmérâ?");
diacritic.Add("Guidé mé âll thé wây");
diacritic.Add("Hów tó mâké â câké?");
diacritic.Add("Sây, Hélló Wórlđ!");
diacritic.Add("Hów tó mâké â róbót?");
diacritic.Add("Whât tímé nów in Indiâ?");
comboBox.ComboBoxSource = diacritic;
mainLayout.Children.Add(comboBox);
Content = mainLayout;
```



### Multiple selection

Select multiple items from a suggestion list. There are two ways to perform multi selection in the combo box control.

- Token Representation
- Delimiter

### Token representation

Selected items will be displayed with a customizable token representation. Users can remove each tokenized item with the close button. And `IsSelectedItemsVisibleInDropDown` property is used to restrict the selected items as visible or not in dropDown.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <comboBox:SfComboBox HeightRequest="40" x:Name="comboBox"
    MultiSelectMode="Token" TokensWrapMode="Wrap"
    IsSelectedItemsVisibleInDropDown="false" />
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
SfComboBox comboBox = new SfComboBox();
```

```
comboBox.DropDownItemHeight = 50;
comboBox.MultiSelectMode = MultiSelectMode.Token;
comboBox.TokensWrapMode = TokensWrapMode.Wrap;
comboBox.IsSelectedItemsVisibleInDropDown = false;
layout.Children.Add(comboBox);
Content = layout;
```

### Wrap mode of token

The selected item can be displayed as token inside the SfComboBox in two ways. They are

- **Wrap** - When the **TokensWrapMode** is set to **Wrap** the selected items will be wrapped to the next line of the SfComboBox.
- **None** - When the **TokensWrapMode** is set to **None** the selected item will be wrapped in horizontal orientation.

Define a simple model class Employee with fields ID, Name and populate employee data in ViewModel.

### C#

```
// Create a Employee Class which holds the Name and image.
public class Employee
{
    private string image;
    public string Image
    {
        get { return image; }
        set { image = value; }
    }
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}

// Create EmployeeViewModel class holds the collection of employee data.
public class EmployeeViewModel : INotifyPropertyChanged
{
    private ObservableCollection<Employee> employeeCollection;
    public ObservableCollection<Employee> EmployeeCollection
    {
        get { return employeeCollection; }
        set { employeeCollection = value; }
    }
    public EmployeeViewModel()
    {
        employeeCollection = new ObservableCollection<Employee>();
        employeeCollection.Add(new Employee() { Image = "John.png", Name = "John" });
        employeeCollection.Add(new Employee() { Image = "Justin.png", Name = "Justin" });
        employeeCollection.Add(new Employee() { Image = "Jerome.png", Name = "Jerome" });
        employeeCollection.Add(new Employee() { Image = "Jessica.png", Name = "Jessica" });
    }
}
```

```

employeeCollection.Add(new Employee() { Image = "Victoria.png", Name =
"Victoria" });
}
public event PropertyChangedEventHandler PropertyChanged;
private void RaisePropertyChanged(String name)
{
if (PropertyChanged != null)
this.PropertyChanged(this, new PropertyChangedEventArgs(name));
}
}

```

Now populate this EmployeeViewModel data in SfComboBox control by binding with [DataSource] property.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:combobox="clr-
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
xmlns:local="clr-namespace:NamespaceName"
x:Class="NamespaceName.ClassName">
<ContentPage.BindingContext>
<local:EmployeeViewModel/>
</ContentPage.BindingContext>
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
DropDownItemHeight="50" DisplayMemberPath="Name" ImageMemberPath="Image"
MultiSelectMode="Token" DataSource="{Binding EmployeeCollection}"/>
</StackLayout>
</ContentPage>

```

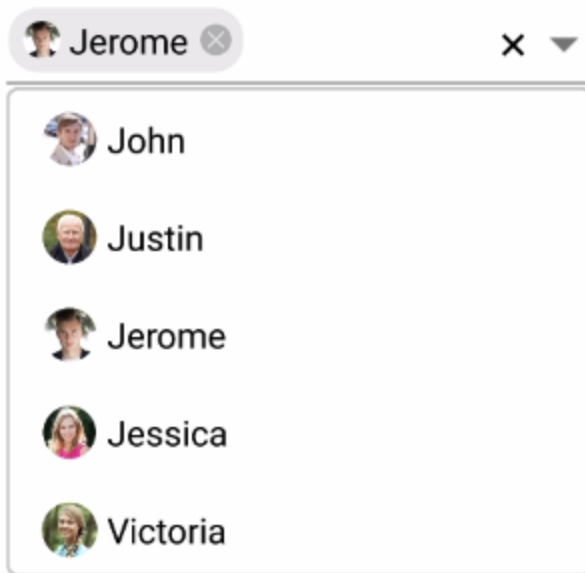
### C#

```

StackLayout layout = new StackLayout()
{
VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
SfComboBox comboBox = new SfComboBox();
comboBox.DropDownItemHeight = 50;
comboBox.DisplayMemberPath = "Name";
comboBox.ImageMemberPath = "Image";
comboBox.MultiSelectMode = MultiSelectMode.Token;
this.BindingContext = new EmployeeViewModel();
// Set TokensWrapMode to Wrap
comboBox.TokensWrapMode = TokensWrapMode.Wrap;
comboBox.SetBinding(SfComboBox.DataSourceProperty, "EmployeeCollection",
BindingMode.TwoWay);
layout.Children.Add(comboBox);
Content = layout;

```





#### Token customization

Token can be customized in the following ways:

- **TextColor** - Sets the color of the text inside the token.
- **FontSize** - Sets the size of the font inside the token.
- **FontFamily** - Sets the font family for the text inside the token.
- **BackgroundColor** - Sets the background color for token.
- **SelectedBackgroundColor** - Sets the background color of the token when it is selected.
- **IsCloseButtonVisible** - Enables and disables the close button inside the SfComboBox.
- **DeleteButtonColor** - Sets the color of the close button inside the SfComboBox.
- **CornerRadius** - Sets the corner radius for the token.
- **DeleteButtonPlacement** - sets the placement of delete button. **Left** and **Right** are the placement options. By default, it is set placed at right side of the token.

---

**Note:** CornerRadius support has enhanced only on iOS and Android platform.

---

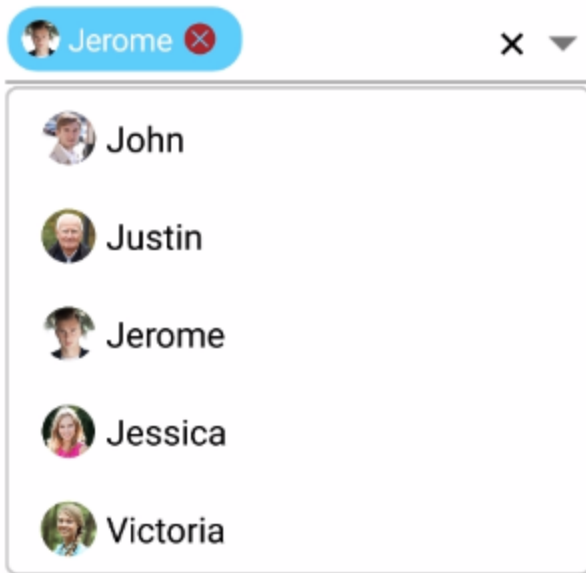
#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:combobox="clr-
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
xmlns:local="clr-namespace:NamespaceName"
x:Class="NamespaceName.ClassName">
<ContentPage.BindingContext>
<local:EmployeeViewModel/>
</ContentPage.BindingContext>
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
DropDownItemHeight="50" DisplayMemberPath="Name" ImageMemberPath="Image"
```

```
MultiSelectMode="Token" TokensWrapMode="Wrap" DataSource="{Binding
EmployeeCollection}">
<combobox:SfComboBox.TokenSettings>
<combobox:TokenSettings FontSize="16" BackgroundColor="#66ccff"
TextColor="White" SelectedBackgroundColor="#ffffe0"
DeleteButtonColor="Color.Brown" IsCloseButtonVisible="true"
CornerRadius="15" DeleteButtonPlacement="Right">
</combobox:TokenSettings>
</combobox:SfComboBox.TokenSettings>
</combobox:SfComboBox>
</StackLayout>
</ContentPage>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
SfComboBox comboBox = new SfComboBox();
comboBox.DropDownItemHeight = 50;
comboBox.DisplayMemberPath = "Name";
comboBox.ImageMemberPath = "Image";
comboBox.MultiSelectMode = MultiSelectMode.Token;
this.BindingContext = new EmployeeViewModel();
// Token Customization
TokenSettings token = new TokenSettings();
token.FontSize = 16;
token.BackgroundColor = Color.FromHex("#66ccff");
token.TextColor = Color.White;
token.SelectedBackgroundColor = Color.FromHex("#ffffe0");
token.DeleteButtonColor = Color.Brown;
token.IsCloseButtonVisible = true;
token.CornerRadius = 15;
token.DeleteButtonPlacement = DeleteButtonPlacement.Right;
comboBox.TokenSettings = token;
comboBox.TokensWrapMode = TokensWrapMode.Wrap;
comboBox.SetBinding(SfComboBox.DataSourceProperty, "EmployeeCollection",
BindingMode.TwoWay);
layout.Children.Add(comboBox);
Content = layout;
```



### Delimiter

When selecting multiple items, the selected items can be divided with a desired character given for a delimiter. You can set delimiter character using the `Delimiter` property.

**Note:** Delimiter support has enhanced only on iOS and Android platform.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    MultiSelectMode="Delimiter" Delimiter=",">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Afghanistan </x:String>
        <x:String> Albania </x:String>
        <x:String> Mexico </x:String>
        <x:String> Norway </x:String>
        <x:String> Singapore </x:String>
        <x:String> Thailand </x:String>
        <x:String> China </x:String>
        <x:String> United States </x:String>
        <x:String> Zimbabwe </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
```

```
Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Afghanistan");
countryNames.Add("Albania");
countryNames.Add("Mexico");
countryNames.Add("Norway");
countryNames.Add("Singapore");
countryNames.Add("Thailand");
countryNames.Add("China");
countryNames.Add("United States");
countryNames.Add("Zimbabwe");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.MultiSelectMode = MultiSelectMode.Delimiter;
comboBox.Delimiter = ",";
layout.Children.Add(comboBox);
Content = layout;
```

Singapore,United States, ▼

Afghanistan  
Akrotiri  
Mexico  
Norway  
Singapore  
Thailand  
China  
United States  
Zimbabwe

#### *Selection indicator*

The combobox enables the user to indicate the selected item from the datasource when selecting multiple items from the dropdown. It can be performed by enabling `EnableSelectionIndicator` property.

#### **XML**

```
<combobox:SfComboBox HeightRequest="40" ShowSuggestionsOnFocus="true"
IsSelectedItemsVisibleInDropDown="true" IndicatorText="A"
IndicatorFontFamily="sample.ttf" IndicatorTextSize="15"
IndicatorTextColor="Red" EnableSelectionIndicator="true"
MultiSelectMode="Token" x:Name="comboBox" DataSource="{Binding
EmployeeCollection}"/>
```

**C#**

```
comboBox.MultiSelectMode=MultiSelectMode.Token;
comboBox.ShowSuggestionsOnFocus=true;
comboBox.IsSelectedItemsVisibleInDropDown=true;
comboBox.IndicatorText= "A";
comboBox.IndicatorFontFamily= "sample.ttf";
comboBox.IndicatorTextSize= "15";
comboBox.IndicatorTextColor = Color.Red;
comboBox.EnableSelectionIndicator= true;
```

*Item padding*

The autocomplete enables the user to provide padding for the items inside dropdown using `ItemPadding` property.

**XML**

```
<combobox:SfComboBox ShowSuggestionsOnFocus="true" ItemPadding="20,10,0,0"
MultiSelectMode="Token" x:Name="comboBox" DataSource="{Binding
EmployeeCollection}"/>
```

**C#**

```
comboBox.MultiSelectMode=MultiSelectMode.Token;
comboBox.ShowSuggestionsOnFocus=true;
comboBox.ItemPadding= new Thickness(20,10,0,0);;
```

*No Results Found*

When the entered item is not in the suggestion list, SfComboBox displays a text that indicates no search results found. You can set the desire text to be displayed for indicating no results found using the [NoResultsFoundText](#) property.

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:ListCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
xmlns:combobox="clr-
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
xmlns:local="clr-namespace:ComboBox_Sample"
x:Class="ComboBox_Sample.MainPage">
<StackLayout Padding="0, 30, 0, 0">
<combobox:SfComboBox AllowFiltering="true"
IsEditableMode="true"
x:Name="comboBox"
HeightRequest="40"
```

```

NoResultsFoundText="Country not in the list"
MultiSelectMode="None">
<combobox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String> Great Britain </x:String>
<x:String> Uganda </x:String>
<x:String> Ukraine </x:String>
<x:String> Canada </x:String>
<x:String> France </x:String>
<x:String> Italy </x:String>
<x:String> United Kingdom </x:String>
<x:String> China </x:String>
<x:String> Japan </x:String>
</ListCollection:List>
</combobox:SfComboBox.ComboBoxSource>
</combobox:SfComboBox>
</StackLayout>
</ContentPage>

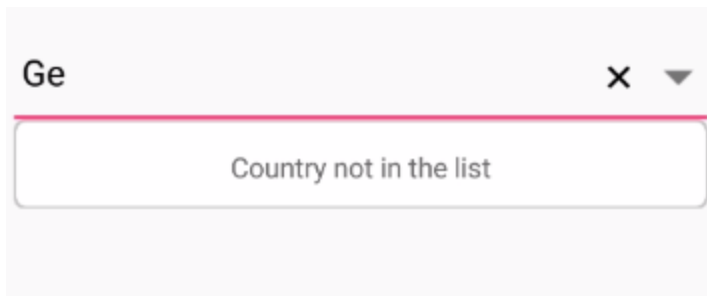
```

**C#**

```

using System;
using System.Collections.Generic;
using Xamarin.Forms;
using Syncfusion.XForms.ComboBox;
namespace ComboBox_Sample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            StackLayout mainLayout = new StackLayout() { Padding=new
            Thickness(0,30,0,0)};
            SfComboBox comboBox = new SfComboBox();
            comboBox.ComboBoxSource = new List<string>() { "Uganda", "Great Britain",
            "Ukraine", "Canada", "France", "Italy", "United Kingdom", "China", "Japan"
            };
            comboBox.IsEditableMode = true;
            comboBox.AllowFiltering = true;
            comboBox.HeightRequest = 40;
            comboBox.NoResultsFoundText = "Country not in the list";
            mainLayout.Children.Add(comboBox);
            this.Content = mainLayout;
        }
    }
}

```



### Customizing NoResultsFoundText

The `NoResultsFoundTextColor`, `NoResultsFoundFontSize`, `NoResultsFoundFontAttributes`, and `NoResultsFoundFontFamily` properties are used to customize the foreground color, font size, font attribute, and font family of `NoResultsFoundText`.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" NoResultsFoundText="Country not in the list"
    NoResultsFoundTextColor="DarkGreen" NoResultsFoundFontSize="20"
    NoResultsFoundFontAttributes="Bold" NoResultsFoundFontFamily="Pacifico.ttf"
  />
</StackLayout>
```

### C#

```
comboBox.NoResultsFoundText="Country not in the list";
comboBox.NoResultsFoundTextColor = Color.Blue;
comboBox.NoResultsFoundFontSize = 20;
comboBox.NoResultsFoundFontAttributes = FontAttributes.Bold;
comboBox.NoResultsFoundFontFamily = "Pacifico.ttf"
comboBox.ComboBoxSource = new List<string>() { "Uganda", "Great Britain",
"Ukraine", "Canada", "France", "Italy", "United Kingdom", "China", "Japan"
};
```



## Highlighting matched text

You can highlight matching characters in a suggestion list to pick an item with more clarity by following two ways:

- First Occurrence
- Multiple Occurrence

Highlighting can be indicated with various customizing styles by enabling the following properties.

- HighlightedTextColor - Sets the color of the highlighted text for differentiating the highlighted characters.
- HighlightTextFontAttributes - Sets the FontAttributes of the highlighted text.

### First occurrence

Highlights the first position of the matching characters in the suggestion list.

#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true"
    TextHighlightMode="FirstOccurrence" HighlightedTextColor="Red"
    HighlightedTextFontAttributes="Bold" SuggestionMode="StartsWith">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Albania </x:String>
        <x:String> Algeria </x:String>
        <x:String> American Samoa </x:String>
        <x:String> Andorra </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Albania");
countryNames.Add("Algeria");
countryNames.Add("American Samoa");
countryNames.Add("Andorra");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.StartsWith;
comboBox.TextHighlightMode = OccurrenceMode.FirstOccurrence;
```



```
comboBox.HighlightedTextColor = Color.Red;
comboBox.HighlightedTextFontAttributes = FontAttributes.Bold;
layout.Children.Add (comboBox);
Content = layout;
```



### Multiple occurrence

Highlights the matching characters that present everywhere in the suggestion list for Contains case in SuggestionMode.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <comboBox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true"
    TextHighlightMode="MultipleOccurrence" HighlightedTextColor="Red"
    HighlightedTextFontAttributes="Bold" SuggestionMode="Contains">
    <comboBox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Albania </x:String>
        <x:String> Algeria </x:String>
        <x:String> American Samoa </x:String>
        <x:String> Andorra </x:String>
      </ListCollection:List>
    </comboBox:SfComboBox.DataSource>
  </comboBox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Albania");
countryNames.Add("Algeria");
countryNames.Add("American Samoa");
```

```
countryNames.Add("Andorra");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.Contains;
comboBox.TextHighlightMode = OccurrenceMode.MultipleOccurrence;
comboBox.HighlightedTextColor = Color.Red;
comboBox.HighlightedTextFontAttributes = FontAttributes.Bold;
layout.Children.Add(comboBox);
Content = layout;
```



## Header and Footer

You can provide header and footer views in the suggestion list in SfComboBox by enabling the `ShowDropDownHeaderView` and the `ShowDropDownFooterView` properties.

### Header content

You can provide Header content for header at the top of the ComboBox's Suggestion box. The `DropDownHeaderView` property is used to set the content of the header. The height of the header in the SfComboBox can be adjusted using the property `DropDownHeaderViewHeight` property.

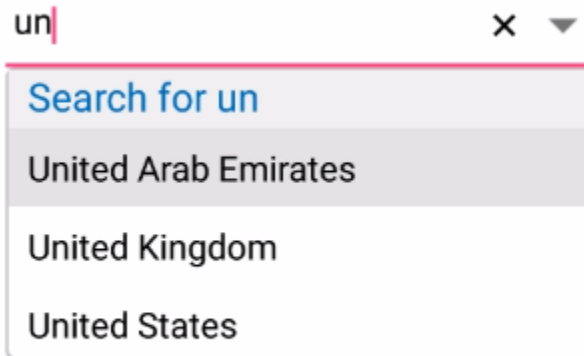
### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
    <combobox:SfComboBox.DropDownHeaderView>
      <StackLayout BackgroundColor="#f0f0f0" >
        <Label x:Name="label2" FontSize="20" VerticalTextAlignment="Center"
          HorizontalOptions="Center" VerticalOptions="Center" TextColor="#006bcd" />
      </StackLayout>
    </combobox:SfComboBox.DropDownHeaderView>
  </combobox:SfComboBox>
</StackLayout>
```

```
</StackLayout>
</combobox:SfComboBox.DropDownHeaderView>
</combobox:SfComboBox>
</StackLayout>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.ShowDropDownHeaderView = true;
//Set the height of the Header View
comboBox.DropDownHeaderViewHeight = 50;
comboBox.ValueChanged += (object sender,
    Syncfusion.XForms.ComboBox.ValueChangedEventArgs e) =>
{
    label2.Text = "Search for " + e.Value;
}
layout.Children.Add(comboBox);
Content = layout;
```



### Footer Content

You can provide content for footer at the bottom of the ComboBox's Suggestion box. The `DropDownFooterView` property is used to set the content for footer. The height of the Header in the SfComboBox can be adjusted using the `DropDownFooterViewHeight` property.

The following code example shows how to set Footer content in SfComboBox.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
  IsEditableMode="true" AllowFiltering="true">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
    <combobox:SfComboBox.DropDownFooterView>
      <StackLayout BackgroundColor="#f0f0f0" >
        <Label Text="Add New" BackgroundColor="#f0f0f0" TextColor="#006bcd"
        VerticalTextAlignment="Center" VerticalOptions="Center"
        HorizontalTextAlignment="Center" FontSize="20"/>
      </StackLayout>
    </combobox:SfComboBox.DropDownFooterView>
  </combobox:SfComboBox>
</StackLayout>
```

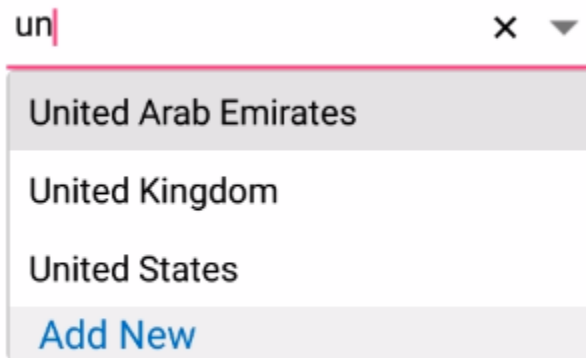
### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.ShowDropDownFooterView = true;
StackLayout customFooterView = new StackLayout();
Label label = new Label()
{
    Text = "Add New",
    BackgroundColor = Color.FromHex("#f0f0f0"),
```

```

TextColor = Color.FromHex("#006bcd"),
VerticalOptions = LayoutOptions.Center,
VerticalTextAlignment = TextAlignment.Center,
HorizontalTextAlignment = TextAlignment.Center,
FontSize = 20
};
customFooterView.Children.Add(label);
comboBox.DropDownFooterView = customFooterView;
layout.Children.Add(comboBox);
Content = layout;

```



### ComboBox modes

The combo box control supports both editable and non-editable text box to choose selected items in given data source. Users can select one item from the suggestion list.

**IsEditableMode** property is used to enable the user input in SfComboBox control. The default value is false.

#### Editable combo box

In editable mode, the combo box allows users to edit in the text box that shows the suggestions in drop-down list based on the input.

#### XML

```

<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>

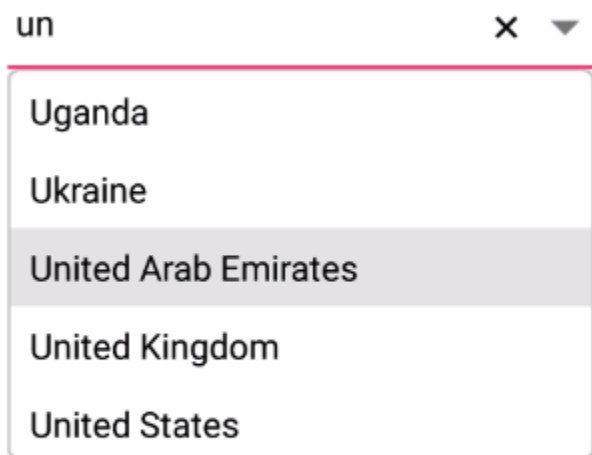
```

**C#**

```

StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
layout.Children.Add(comboBox);
Content = layout;

```

**Non-Editable combo box**

Non-editable mode prevents users from typing and allows them to select from drop-down list.

**XML**

```

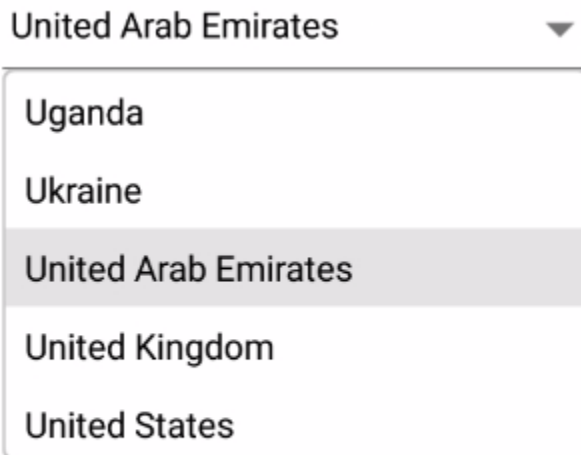
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="false">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>

```

```
<x:String> United States </x:String>
</ListCollection:List>
</comboBox:SfComboBox.DataSource>
</comboBox:SfComboBox>
</StackLayout>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = false;
layout.Children.Add(comboBox);
Content = layout;
```



## Dealing with suggestion box

Suggestion box is a drop-down list box which displays the filtered suggestions inside a popup. This section explains the properties that deals with the drop-down list in SfComboBox control.

### Suggestion box placement mode

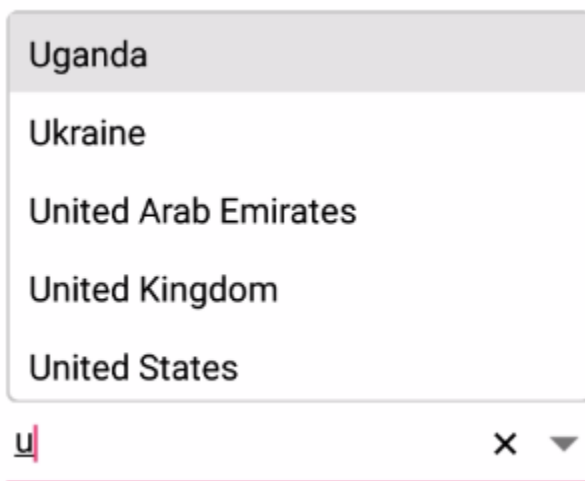
Suggestion Box can be placed either at the top or bottom using the **SuggestionBoxPlacement** property. By default, it is placed at the bottom.

**XML**

```
<StackLayout VerticalOptions="Center" HorizontalOptions="Center"
Padding="30">
<comboBox:SfComboBox HeightRequest="40" SuggestionBoxPlacement="Top"
x:Name="comboBox">
<comboBox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String> Uganda </x:String>
<x:String> Ukraine </x:String>
<x:String> United Arab Emirates </x:String>
<x:String> United Kingdom </x:String>
<x:String> United States </x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
```

**C#**

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.ComboBoxSource = countryNames;
comboBox.SuggestionBoxPlacement = SuggestionBoxPlacement.Top;
layout.Children.Add(comboBox);
Content = layout;
```





### Maximum suggestion box height

The maximum height of the suggestion box in the SfComboBox control can be varied using the `MaximumDropDownHeight` property.

#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    MaximumDropDownHeight="100">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Great Britain </x:String>
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> Canada </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> France </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> China </x:String>
        <x:String> United States </x:String>
        <x:String> Japan </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Great Britain");
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("Canada");
countryNames.Add("United Arab Emirates");
countryNames.Add("France");
countryNames.Add("United Kingdom");
countryNames.Add("China");
countryNames.Add("United States");
countryNames.Add("Japan");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.ComboBoxSource = countryNames;
comboBox.MaximumDropDownHeight = 100;
layout.Children.Add(comboBox);
Content = layout;
```



### Opening suggestion box on focus

Suggestion Box can be shown whenever control receives focus using the `ShowSuggestionsOnFocus` property. Suggestion list is the complete list of data source.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    ShowSuggestionsOnFocus="true">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Great Britain </x:String>
        <x:String> Canada </x:String>
        <x:String> France </x:String>
        <x:String> China </x:String>
        <x:String> Japan </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Great Britain");
countryNames.Add("Canada");
countryNames.Add("France");
countryNames.Add("China");
countryNames.Add("Japan");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.ComboBoxSource = countryNames;
comboBox.ShowSuggestionsOnFocus = true;
comboBox.IsEditableMode = true;
comboBox.SelectionChanged += (sender, e) =>
{
    DisplayAlert("Selection Changed", "Selected Value: " +
        comboBox.SelectedValue.ToString(), "OK");
};
```

```
layout.Children.Add(comboBox);
Content = layout;
```



#### Delay opening suggestion box

The **PopupDelay** property is used to delay the suggestion box opening process. It gets milliseconds as input in integer data type.

In this example, a time duration of 3 seconds is set to popup delay.

#### XML

```
<StackLayout VerticalOptions="StartAndExpand"
HorizontalOptions="StartAndExpand" Padding="30">
<comboBox:SfComboBox HeightRequest="40" x:Name="comboBox" PopupDelay="3000">
<comboBox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String> Uganda </x:String>
<x:String> Ukraine </x:String>
<x:String> United Arab Emirates </x:String>
<x:String> United Kingdom </x:String>
<x:String> United States </x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
```

```
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.ComboBoxSource = countryNames;
comboBox.PopupDelay = 3000;
layout.Children.Add(comboBox);
Content = layout;
```

### Delay before searching algorithm starts

The **SearchDelay** property is used to delay the searching algorithm process. It gets milliseconds as input in integer data type.

In this example, a time duration of 3 seconds is set to search delay.

### XML

```
<StackLayout VerticalOptions="StartAndExpand"
HorizontalOptions="StartAndExpand" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
SearchDelay="3000">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.ComboBoxSource = countryNames;
comboBox.SearchDelay = 3000;
layout.Children.Add(comboBox);
Content = layout;
```

### Avoid opening suggestion box

APIs are available to avoid pop-ups and retrieve filtered suggestion items that help you arrange lists or items control.

#### XML

```
<StackLayout VerticalOptions="StartAndExpand"
HorizontalOptions="StartAndExpand" Padding="30">
  <comboBox:SfComboBox HeightRequest="40" x:Name="comboBox"
  SuggestionBoxPlacement="None">
    <comboBox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </comboBox:SfComboBox.ComboBoxSource>
  </comboBox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.SuggestionBoxPlacement = SuggestionBoxPlacement.None;
comboBox.ComboBoxSource = countryNames;
layout.Children.Add(comboBox);
Content = layout;
```

### Customizing ComboBox

The combo box control provides user friendly customizing options for both entry part and drop-down part. In this section, customizing the entire ComboBox control is explained.

#### Customizing the entry

The [TextColor](#), [TextSize](#), [FontAttributes](#), [FontFamily](#) and [BorderColor](#) properties are used to customize the foreground color, font size, font attribute, font family and border color of the entry part.

##### Text color

The combo box control provides the user to customize the foreground color of the text inside the entry part.

*Text size*

The combo box control provides the user to customize the text size of the text inside the entry part using [TextSize](#) property.

*Font attributes*

The combo box control provides the user to customize the font attribute of the text inside the entry part using [FontAttributes](#) property.

*Font family*

The combo box control provides the user to customize the font family of the text inside the entry part using [FontFamily](#) property.

*Border color*

The combo box control provides the user to customize the border color of the entry box using [BorderColor](#) property.

**XML**

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <comboBox:SfComboBox HeightRequest="45" x:Name="comboBox" Text="Sample text"
    FontAttributes="Bold" TextColor="#1976d2" TextSize="20"
    BorderColor="#1976d2"/>
</StackLayout>
```

**C#**

```
StackLayout stackLayout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
SfComboBox comboBox = new SfComboBox()
{
    HeightRequest = 45,
    Text = "Sample text",
    TextColor = Color.FromHex("1976d2"),
    TextSize = 20,
    BorderColor = Color.FromHex("1976d2")
    FontAttributes = FontAttributes.Bold
};
stackLayout.Children.Add(comboBox);
this.Content = stackLayout;
```

Sample text

*Changing delete button color*

The [ClearButtonColor](#) property is used to modify the delete button color. The following code example shows changing delete button color.

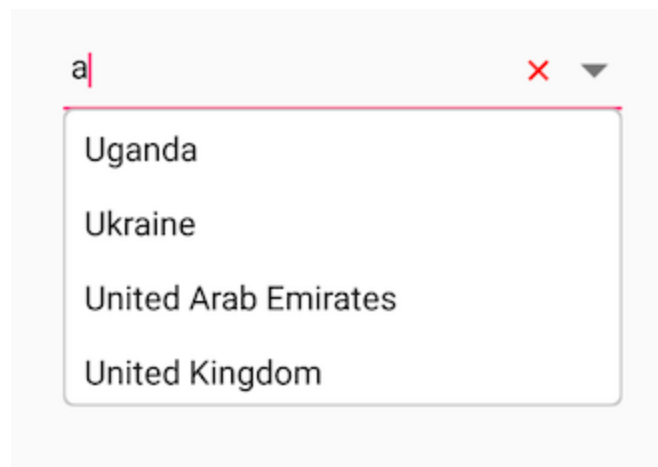
**XML**

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
```

```
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
IsEditableMode="True" ClearButtonColor="Red">
<combobox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>United Kingdom</x:String>
</ListCollection:List>
</combobox:SfComboBox.ComboBoxSource>
</combobox:SfComboBox>
</StackLayout>
```

**C#**

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
comboBox.ClearButtonColor = Color.Yellow;
comboBox.IsEditableMode = true;
layout.Children.Add(comboBox);
Content = layout;
```

*Changing delete button visibility*

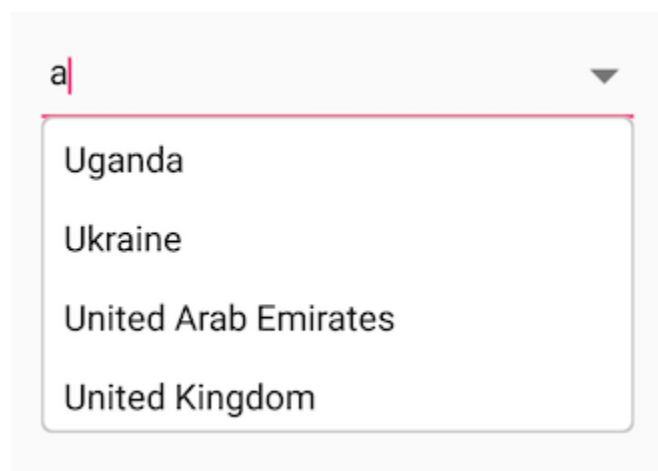
The [ShowClearButton](#) property is used to modify the visibility of delete button. The following code example shows changing delete button visibility.

**XML**

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
IsEditableMode="True" IsEditableMode="True" ShowClearButton="False">
<combobox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>United Kingdom</x:String>
</ListCollection:List>
</combobox:SfComboBox.ComboBoxSource>
</combobox:SfComboBox>
</StackLayout>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
comboBox.ShowClearButton = false;
comboBox.IsEditableMode = true;
layout.Children.Add(comboBox);
Content = layout;
```



### Changing border visibility

The [ShowBorder](#) property is used to modify the visibility of border. The following code example shows changing border visibility.

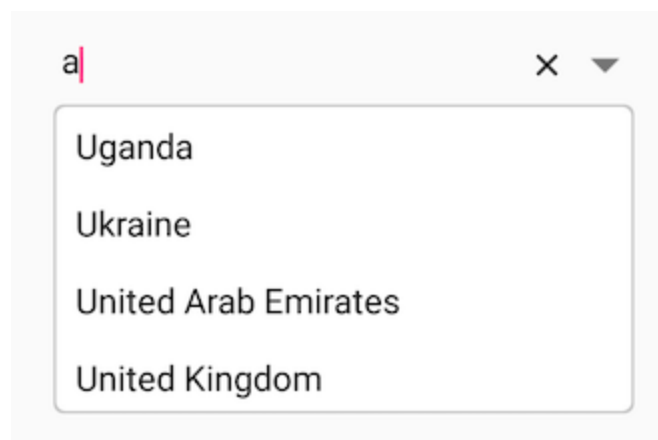


**XML**

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox:SfComboBox HeightRequest="40" x:Name="comboBox"
IsEditableMode="True" IsEditableMode="True" ShowBorder="False">
<comboBox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>United Kingdom</x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
```

**C#**

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
comboBox.ShowBorder = false;
comboBox.IsEditableMode = true;
layout.Children.Add(comboBox);
Content = layout;
```

**CustomView for ComboBox**

CustomView property has used to provide the custom view instead of entry in ComboBox. It's default height and width has control height and width.

**XML**

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox">
    <combobox:SfComboBox.CustomView>
      <Label x:Name="customLabel" HeightRequest="40" Text="ComboBox"
        VerticalTextAlignment="Center"/>
    </combobox:SfComboBox.CustomView>
  </combobox:SfComboBox>
</StackLayout>
```

**C#**

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
Label customLabel = new Label();
customLabel.HeightRequest = 40;
customLabel.VerticalTextAlignment = TextAlignment.Center;
customLabel.Text = "ComboBox";
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.SuggestionBoxPlacement = SuggestionBoxPlacement.Top;
comboBox.CustomView = customLabel;
layout.Children.Add(comboBox);
Content = layout;
```

ComboBox

**Custom template for suggestion items**

The **ItemTemplate** property helps to decorate suggestion items with custom templates. The following code explains the steps to add an image to the suggestion list item.

**C#**

```
public class Person
{
    private int age;
    public int Age
    {
        get { return age; }
        set { age = value; }
    }
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
```

```

}
public class PersonViewModel
{
    private ObservableCollection<Person> personCollection;
    public ObservableCollection<Person> PersonCollection
    {
        get { return personCollection; }
        set { personCollection = value; }
    }
    public PersonViewModel()
    {
        personCollection = new ObservableCollection<Person>();
        personCollection.Add(new Person() { Age = 21, Name = "Aldan" });
        personCollection.Add(new Person() { Age = 25, Name = "Clara" });
        personCollection.Add(new Person() { Age = 23, Name = "Aldrin" });
        personCollection.Add(new Person() { Age = 25, Name = "Mark" });
        personCollection.Add(new Person() { Age = 25, Name = "Lucas" });
        personCollection.Add(new Person() { Age = 24, Name = "Alan" });
        personCollection.Add(new Person() { Age = 25, Name = "James" });
        personCollection.Add(new Person() { Age = 22, Name = "Aaron" });
    }
}

```

Now populate this PersonViewModel data in SfComboBox control by binding with `[DataSource]` property.

#### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:combobox="clr-
namespace:Syncfusion.XForms.ComboBox;assembly=Syncfusion.SfComboBox.XForms"
xmlns:local="clr-namespace:NamespaceName"
x:Class="NamespaceName.ClassName">
<ContentPage.BindingContext>
<local:PersonViewModel/>
</ContentPage.BindingContext>
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
DisplayMemberPath="Name" DataSource="{Binding PersonCollection}">
<combobox:SfComboBox.ItemTemplate>
<DataTemplate>
<StackLayout Orientation="Horizontal">
<Image Source="User.png" WidthRequest="12"/>
<Label Text="{Binding Name}" />
</StackLayout>
</DataTemplate>
</combobox:SfComboBox.ItemTemplate>
</combobox:SfComboBox>
</StackLayout>
</ContentPage>

```

#### C#

```

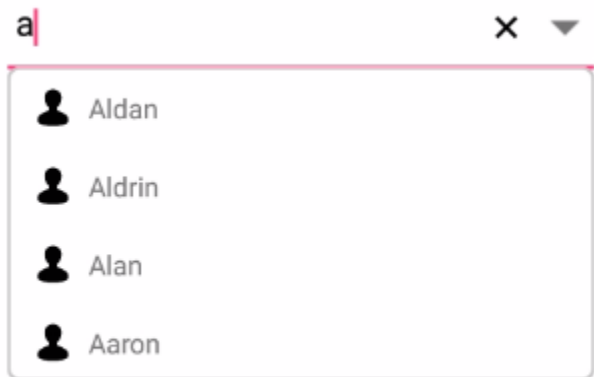
DataTemplate itemTemplate = new DataTemplate(() =>
{

```

```

StackLayout stack;
Image image;
Label label;
stack = new StackLayout();
stack.Orientation = StackOrientation.Horizontal;
image = new Image();
image.Source = (FileImageSource)ImageSource.FromFile("User.png");
label = new Label();
label.SetBinding(Label.TextProperty, "Name");
stack.Children.Add(image);
stack.Children.Add(label);
return new ViewCell { View = stack };
});
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
SfComboBox comboBox = new ComboBox();
comboBox.HeightRequest = 40;
comboBox.BindingContext = new PersonViewModel();
Binding binding = new Binding("PersonCollection");
binding.Source = this;
binding.Mode = BindingMode.TwoWay;
comboBox.SetBinding(Label.DataSourceProperty, binding);
comboBox.ItemTemplate = itemTemplate;
layout.Children.Add(comboBox);
Content = layout;

```



### Customizing the suggestion box

#### Changing suggestion item height

The [DropDownItemHeight](#) property is used to modify the height of suggestion items in drop-down list. The following code example shows changing suggestion item height.

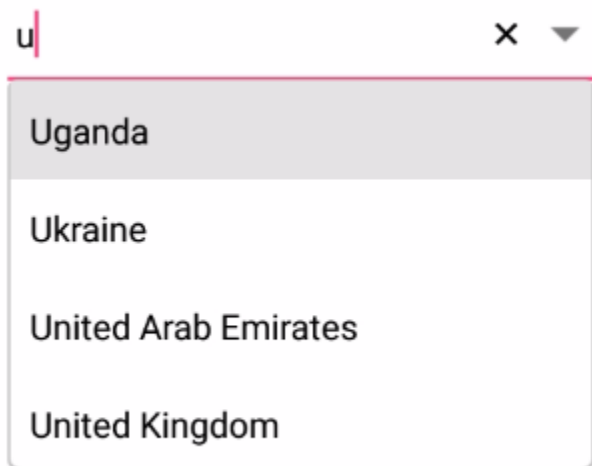
#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
```

```
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
DropDownItemHeight="50">
<combobox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String> Uganda </x:String>
<x:String> Ukraine </x:String>
<x:String> United Arab Emirates </x:String>
<x:String> United Kingdom </x:String>
</ListCollection:List>
</combobox:SfComboBox.ComboBoxSource>
</combobox:SfComboBox>
</StackLayout>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
comboBox.DropDownItemHeight = 50;
layout.Children.Add(comboBox);
Content = layout;
```



### *Changing suggestion box width*

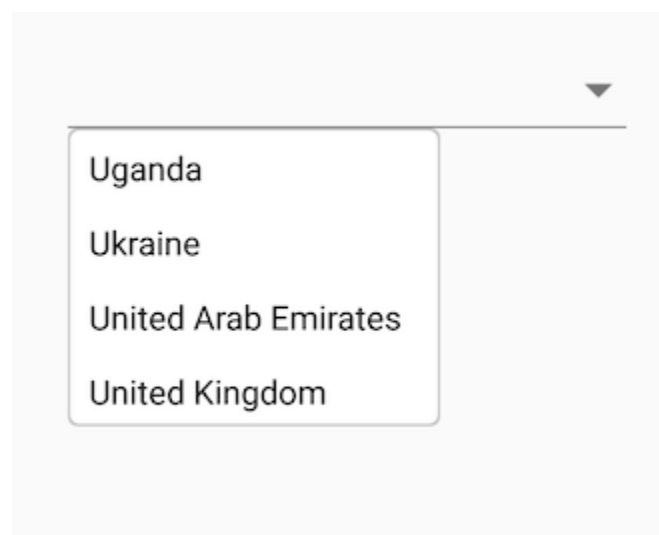
The [DropDownWidth](#) property is used to modify the width of suggestion box. The following code example shows changing suggestion box width.

## XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox:SfComboBox HeightRequest="40" x:Name="comboBox"
DropDownWidth="300">
<comboBox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String>Uganda</x:String>
<x:String>Ukraine</x:String>
<x:String>United Arab Emirates</x:String>
<x:String>United Kingdom</x:String>
</ListCollection:List>
</comboBox:SfComboBox.ComboBoxSource>
</comboBox:SfComboBox>
</StackLayout>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
comboBox.DropDownWidth = 300;
layout.Children.Add(comboBox);
Content = layout;
```



### Changing suggestion box corner radius

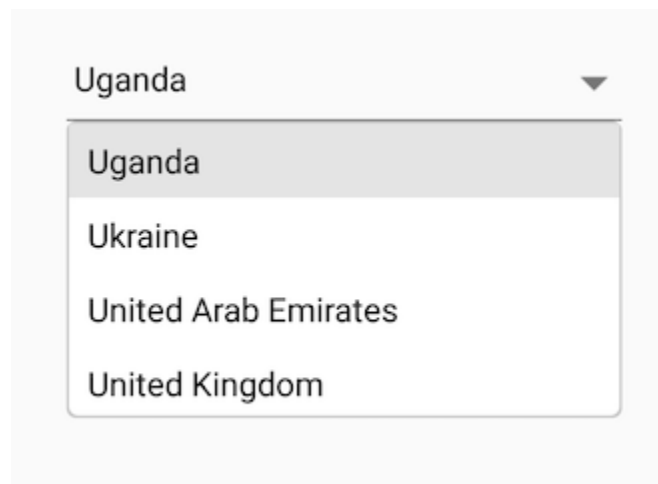
The [DropDownCornerRadius](#) property is used to modify the corner radius of suggestion box. The following code example shows changing suggestion box corner radius.

#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    DropDownCornerRadius="3">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String>Uganda</x:String>
        <x:String>Ukraine</x:String>
        <x:String>United Arab Emirates</x:String>
        <x:String>United Kingdom</x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
comboBox.DropDownCornerRadius = 3;
layout.Children.Add(comboBox);
Content = layout;
```



*Changing suggestion box background color*

The [DropDownBackgroundColor](#) property is used to modify the background color of suggestion box. The following code example shows changing suggestion box background color.

**XML**

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    DropDownBackgroundColor="Yellow">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String>Uganda</x:String>
        <x:String>Ukraine</x:String>
        <x:String>United Arab Emirates</x:String>
        <x:String>United Kingdom</x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

**C#**

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
comboBox.DropDownBackgroundColor = Color.Yellow;
layout.Children.Add(comboBox);
Content = layout;
```





#### *Changing the border color of suggestion box*

The `DropDownBorderColor` property is used to change the border color of suggestion box. The following code example demonstrates how to change the border color of suggestion box.

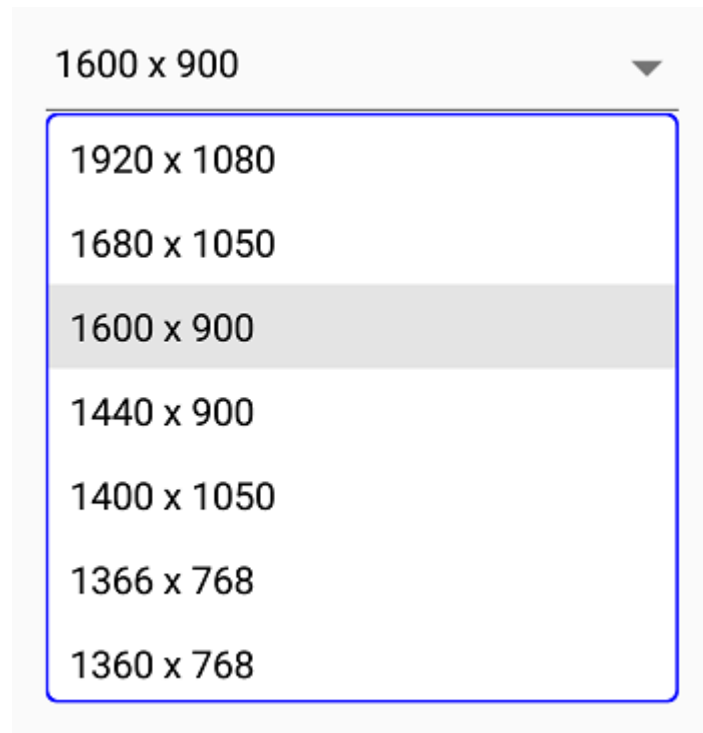
#### **XML**

```
<StackLayout>
  <comboBox:SfComboBox HeightRequest="40" DropDownBorderColor="Blue">
    <comboBox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String>1920 x 1080</x:String>
        <x:String>1680 x 1050</x:String>
        <x:String>1600 x 900</x:String>
        <x:String>1440 x 900</x:String>
        <x:String>1400 x 1050</x:String>
        <x:String>1366 x 768</x:String>
        <x:String>1360 x 768</x:String>
        <x:String>1280 x 1024</x:String>
        <x:String>1280 x 960</x:String>
        <x:String>1280 x 720</x:String>
        <x:String>854 x 480</x:String>
        <x:String>800 x 480</x:String>
        <x:String>480 x 640</x:String>
        <x:String>480 x 320</x:String>
        <x:String>432 x 240</x:String>
        <x:String>360 x 640</x:String>
        <x:String>320 x 240</x:String>
      </ListCollection:List>
    </comboBox:SfComboBox.ComboBoxSource>
  </comboBox:SfComboBox>
</StackLayout>
```

#### **C#**

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
}
```

```
};  
List<String> resolutionList = new List<String>();  
resolutionList.Add("1920 x 1080");  
resolutionList.Add("1680 x 1050");  
resolutionList.Add("1600 x 900");  
resolutionList.Add("1440 x 900");  
resolutionList.Add("1400 x 1050");  
resolutionList.Add("1366 x 768");  
resolutionList.Add("1360 x 768");  
resolutionList.Add("1280 x 1024");  
resolutionList.Add("1280 x 960");  
resolutionList.Add("1280 x 720");  
resolutionList.Add("854 x 480");  
resolutionList.Add("800 x 480");  
resolutionList.Add("480 x 640");  
resolutionList.Add("480 x 320");  
resolutionList.Add("432 x 240");  
resolutionList.Add("360 x 640");  
resolutionList.Add("320 x 240");  
SfComboBox comboBox = new SfComboBox();  
comboBox.HeightRequest = 40;  
comboBox.ComboBoxSource = resolutionList;  
comboBox.DropDownBorderColor = Color.Blue;  
layout.Children.Add(comboBox);  
Content = layout;
```



#### *Customizing suggestion items*

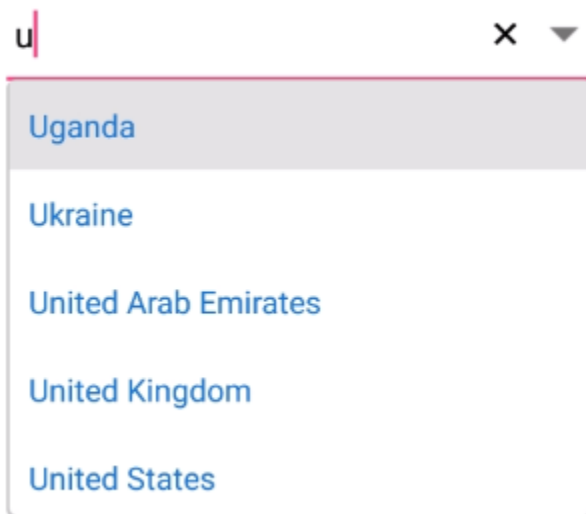
Suggestion box items can be customized using the [DropDownItemFontAttributes](#), [DropDownItemFontFamily](#), [DropDownTextSize](#) and [DropDownTextColor](#) properties.

#### **XML**

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
DropDownTextSize="16" DropDownTextColor="#1976d2">
<combobox:SfComboBox.ComboBoxSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String> Uganda </x:String>
<x:String> Ukraine </x:String>
<x:String> United Arab Emirates </x:String>
<x:String> United Kingdom </x:String>
<x:String> United States </x:String>
</ListCollection:List>
</combobox:SfComboBox.ComboBoxSource>
</combobox:SfComboBox>
</StackLayout>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
comboBox.DropDownTextColor = Color.FromHex("#1976d2");
comboBox.DropDownTextSize = 16;
layout.Children.Add(comboBox);
Content = layout;
```



### Changing selected item color in suggestion box

The [SelectedDropDownItemColor](#) property is used to modify text color of selected item in drop down. The following code example shows changing the selected item text color in drop down.

#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    SelectedDropDownItemColor="Blue">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String>Uganda</x:String>
        <x:String>Ukraine</x:String>
        <x:String>United Arab Emirates</x:String>
        <x:String>United Kingdom</x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
comboBox.SelectedDropDownItemColor = Color.Blue;
layout.Children.Add(comboBox);
Content = layout;
```



## DropDown button customization

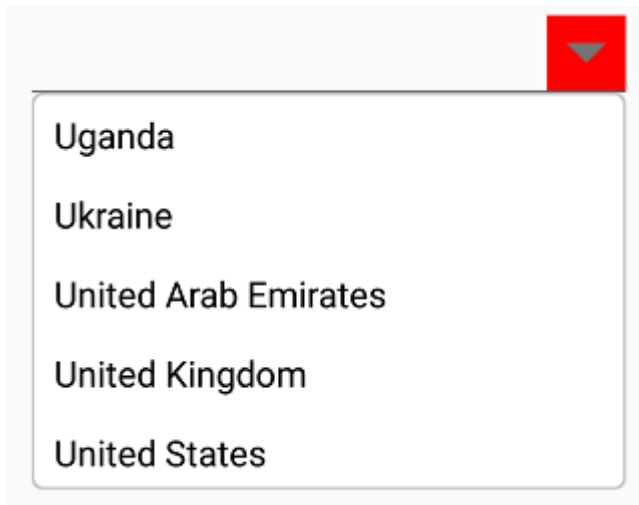
This section explains various DropDown button settings available in SfComboBox control.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox">
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
    <combobox:SfComboBox.DropDownButtonSettings>
      <combobox:DropDownButtonSettings Width="40" Height="40"
        HighlightedBackgroundColor="Green" BackgroundColor="Red"
        HighlightFontColor="Red"/>
    </combobox:SfComboBox.DropDownButtonSettings>
  </combobox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
DropDownButtonSettings dropDownButtonSettings = new
DropDownButtonSettings();
dropDownButtonSettings.Height = 40;
dropDownButtonSettings.Width = 40;
dropDownButtonSettings.HighlightedBackgroundColor = Color.Green;
dropDownButtonSettings.BackgroundColor = Color.Red;
dropDownButtonSettings.HighlightFontColor = Color.Red;
comboBox.DropDownButtonSettings = dropDownButtonSettings;
layout.Children.Add(comboBox);
Content = layout;
```



[View for drop down button](#)

This section explains how to provide view to the drop down button.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <comboBox:SfComboBox HeightRequest="40" x:Name="comboBox">
    <comboBox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String>Uganda</x:String>
        <x:String>Ukraine</x:String>
        <x:String>United Arab Emirates</x:String>
        <x:String>United Kingdom</x:String>
        <x:String>United States</x:String>
      </ListCollection:List>
    </comboBox:SfComboBox.ComboBoxSource>
    <comboBox:SfComboBox.DropDownButtonSettings>
      <comboBox:DropDownButtonSettings Width="40" Height="40">
        <comboBox:DropDownButtonSettings.View>
          <Label WidthRequest="30" Text="Click" HorizontalTextAlignment="Center"
            VerticalTextAlignment="Center"/>
        </comboBox:DropDownButtonSettings.View>
      </comboBox:DropDownButtonSettings>
    </comboBox:SfComboBox.DropDownButtonSettings>
  </comboBox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
```

```
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.DataSource = countryNames;
DropDownButtonSettings dropDownButtonSettings = new
DropDownButtonSettings();
dropDownButtonSettings.Height = 40;
dropDownButtonSettings.Width = 40;
Label label = new Label();
label.Text = "Click";
label.VerticalTextAlignment = TextAlignment.Center;
label.HorizontalTextAlignment = TextAlignment.Center;
label.WidthRequest = 30;
dropDownButtonSettings.View = label;
comboBox.DropDownButtonSettings = dropDownButtonSettings;
layout.Children.Add(comboBox);
Content = layout;
```



### Watermark

Watermark provides a short note about the type of input to enter in the editor control. Watermarks are visible only if the text is empty. It will reappear if the text is cleared. The following example, explains the usability of watermark that hints users to start with the character “U”.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <comboBox:SfComboBox HeightRequest="40" Watermark="Enter 'U' to filter
  suggestions" x:Name="comboBox" />
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
```

```
SfComboBox comboBox = new SfComboBox();
comboBox.Watermark = "Enter 'U' to filter suggestions";
layout.Children.Add(comboBox);
Content = layout;
```

Enter 'U' to filter suggestions ▼

### Changing Watermark Text Color

Text color of watermark can be customized using [WatermarkColor](#) property.

#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" Watermark="Enter some text"
WatermarkColor="#1976d2" x:Name="comboBox" />
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
SfComboBox comboBox = new SfComboBox();
comboBox.Watermark = "Enter some text";
comboBox.WatermarkColor = Color.FromHex("1976d2");
layout.Children.Add(comboBox);
Content = layout;
```

Enter some text ▼

### Filtering

The combo box enables the filter option for filtering the suggestions in the drop-down.

#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
IsEditableMode="true" AllowFiltering="true" MaximumDropDownHeight="150">
<combobox:SfComboBox.DataSource>
<ListCollection:List x:TypeArguments="x:String">
<x:String> Afghanistan </x:String>
<x:String> Albania </x:String>
<x:String> Algeria</x:String>
<x:String> American Samoa </x:String>
<x:String> Andorra </x:String>
<x:String> Angola </x:String>
```



```
<x:String> Anguilla</x:String>
<x:String> Antarctica </x:String>
<x:String> Antigua and Barbuda </x:String>
<x:String> Argentina </x:String>
</ListCollection:List>
</comboBox:SfComboBox.DataSource>
</comboBox:SfComboBox>
</StackLayout>
```

## C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Afghanistan");
countryNames.Add("Albania");
countryNames.Add("Algeria");
countryNames.Add("American Samoa");
countryNames.Add("Andorra");
countryNames.Add("Angola");
countryNames.Add("Anguilla");
countryNames.Add("Antarctica");
countryNames.Add("Antigua and Barbuda");
countryNames.Add("Argentina");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
layout.Children.Add(comboBox);
Content = layout;
```



### Filtering types

The string comparison for filtering suggestions can be changed using the `SuggestionMode` property. The default filtering type is “StartsWith”, and it is case insensitive. The available filtering modes are,

- StartsWith
- StartsWithCaseSensitive
- Contains
- ContainsWithCaseSensitive
- Equals
- EqualsWithCaseSensitive
- EndsWith
- EndsWithCaseSensitive
- Custom

### Filtering words that starts with input text

Displays all the matches that start with the typed characters in control. This strategy is case in-sensitive.

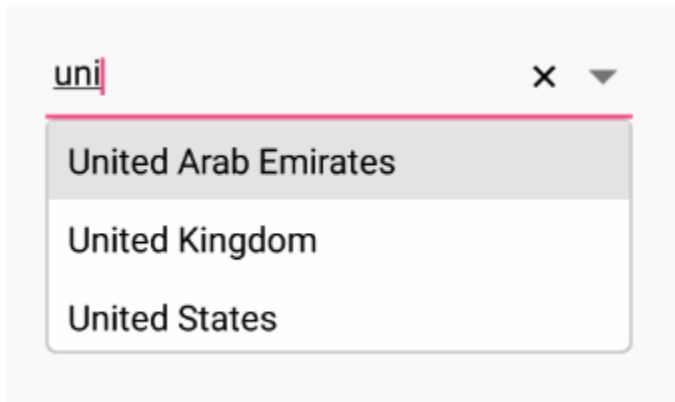
#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <comboBox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true" SuggestionMode="StartsWith">
    <comboBox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> UUnited Arab Emirates</x:String>
        <x:String> United Kingdom</x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </comboBox:SfComboBox.DataSource>
  </comboBox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.StartsWith;
```

```
layout.Children.Add(comboBox);
Content = layout;
```



Filtering words that starts with input text - case sensitive

Displays all the matches that start with the typed characters in control. This strategy is case sensitive.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <comboBox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true"
    SuggestionMode="StartsWithCaseSensitive">
    <comboBox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates</x:String>
        <x:String> United Kingdom</x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </comboBox:SfComboBox.DataSource>
  </comboBox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
```

```
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.StartsWithCaseSensitive;
layout.Children.Add(comboBox);
Content = layout;
```



#### *Filtering words that contain input text*

Displays all the matches that contain the typed characters in control. This strategy is case in-sensitive.

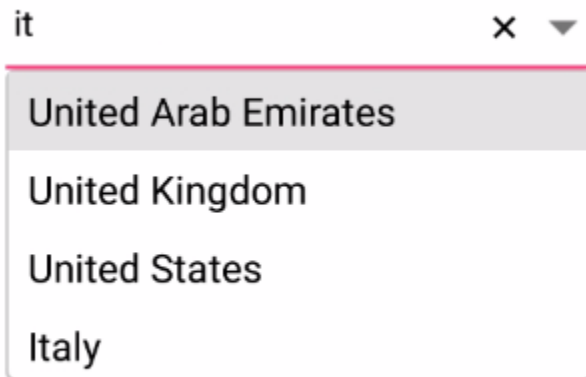
#### **XML**

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true" SuggestionMode="Contains">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates</x:String>
        <x:String> United Kingdom</x:String>
        <x:String> United States </x:String>
        <x:String> Italy </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>
```

#### **C#**

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
countryNames.Add("Italy");
SfComboBox comboBox = new SfComboBox();
```

```
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.Contains;
layout.Children.Add(comboBox);
Content = layout;
```



Filtering words that contain input text - case sensitive

Displays all the matches that contains the typed characters in control. This strategy is case sensitive.

#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true"
    SuggestionMode="ContainsWithCaseSensitive">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
        <x:String> Italy </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
```

```

List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
countryNames.Add("Italy");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.ContainsWithCaseSensitive;
layout.Children.Add(comboBox);
Content = layout;

```



#### *Filtering words that equals the input text*

Displays all the words that completely match with the typed characters in control. This strategy is case in-sensitive.

#### **XML**

```

<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfCpomboBox HeightRequest="40" x:Name="comboBox"
  IsEditableMode="true" AllowFiltering="true" SuggestionMode="Equals">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>

```

#### **C#**

```

StackLayout layout = new StackLayout()
{

```

```

VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
countryNames.Add("Italy");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.Equals;
layout.Children.Add(comboBox);
Content = layout;

```

#### Filtering words that equals input text - case sensitive

Displays all the words that completely match with the typed characters in control. This strategy is case sensitive.

#### XML

```

<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true"
    SuggestionMode="EqualsWithCaseSensitive">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>

```

#### C#

```

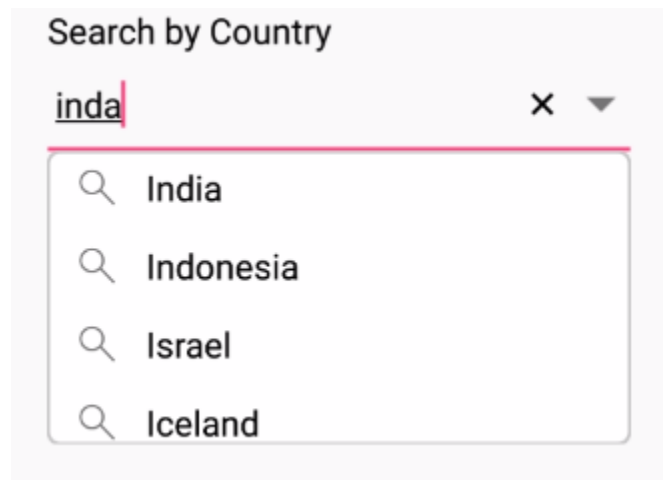
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");

```

```
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.EqualsWithCaseSensitive;
layout.Children.Add(comboBox);
Content = layout;
```

### Custom

Filters items in the suggestion list based on a custom search by user. This helps to apply our typo toleration functionality to the control.



### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" DropDownTextSize="20"
  x:Name="comboBox" IsEditableMode="true" AllowFiltering="true"
  ComboBoxMode="Suggest" MaximumDropDownHeight="200" SuggestionMode="Custom">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Albania </x:String>
        <x:String> Algeria </x:String>
        <x:String> American Samoa </x:String>
        <x:String> Andorra </x:String>
        <x:String> Anguilla </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>
```

### C#

```
public ComboBoxPage()
{
    InitializeComponent();
    StackLayout layout = new StackLayout()
    {
```



```

VerticalOptions = LayoutOptions.Start,
HorizontalOptions = LayoutOptions.Start,
Padding = new Thickness(30)
};
List<string> list = new List<string>();
list.Add("Albania");
list.Add("Algeria");
list.Add("American Samoa");
list.Add("Andorra");
list.Add("Angola");
list.Add("Anguilla");
comboBox.ComboBoxSource = list;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.Filter = ContainingSpaceFilter;
layout.Children.Add(comboBox);
Content = layout;
}
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;
                }
            }
            return false;
        }
        return true;
    }
    catch (Exception)
    {
        return (text.Contains(search));
    }
}
else
    return false;
}

```

#### *Filtering words that end with input text*

Displays all the matches that end with the typed characters in control. This strategy is case in-sensitive.

#### **XML**

```

<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
<comboBox: SfComboBox HeightRequest="40" WidthRequest="180" x:Name="comboBox"
IsEditableMode="true" AllowFiltering="true" SuggestionMode="EndsWith">
<comboBox: SfComboBox.DataSource>
<ListCollection:List x:TypeArguments="x:String">

```

```

<x:String> Uganda </x:String>
<x:String> Ukraine </x:String>
<x:String> United Arab Emirates </x:String>
<x:String> United Kingdom </x:String>
<x:String> United States </x:String>
</ListCollection:List>
</combobox:SfComboBox.DataSource>
</combobox:SfComboBox>
</StackLayout>

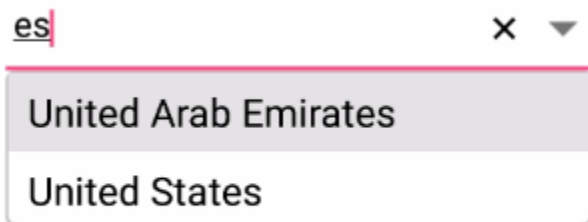
```

## C#

```

StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.EndsWith;
layout.Children.Add(comboBox);
Content = layout;

```



Filtering words that end with input text - case sensitive

Displays all the matches that ends with the typed characters in control. This strategy is case sensitive.

## XML

```

<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true"
    SuggestionMode="EndsWithCaseSensitive">
    <combobox:SfComboBox.DataSource>

```

```

<ListCollection:List x:TypeArguments="x:String">
  <x:String> Uganda </x:String>
  <x:String> Ukraine </x:String>
  <x:String> United Arab Emirates </x:String>
  <x:String> United Kingdom </x:String>
  <x:String> United States </x:String>
</ListCollection:List>
</comboBox:SfComboBox.DataSource>
</comboBox:SfComboBox>
</StackLayout>

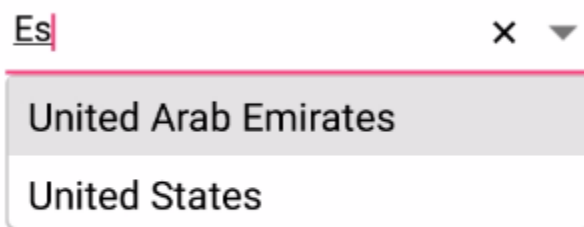
```

## C#

```

StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.SuggestionMode = SuggestionMode.EndsWithCaseSensitive;
layout.Children.Add(comboBox);
Content = layout;

```



## ComboBox modes

The combo box provides three different ways to display the filtered suggestions. They are

- Suggest - Displays suggestions in drop-down list
- Append - Appends the first suggestion to text
- SuggestAppend - Both suggests and appends.

The `ComboBoxMode` property is used to choose the suggestion display mode in SfComboBox control. The default value is Suggest.

#### Suggesting choices in List

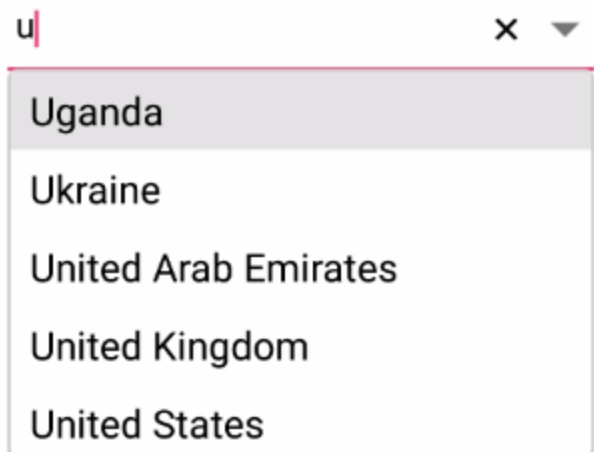
The filtered suggestions are displayed in a drop-down list. User can pick an item from the list.

#### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true" ComboBoxMode="Suggest">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>
```

#### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.ComboBoxMode = ComboBoxMode.Suggest;
layout.Children.Add(comboBox);
Content = layout;
```



### Appending suggestions to text

The first item in filtered suggestions is appended to SfComboBox text. In this mode, drop down remains closed.


### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true" ComboBoxMode="Append">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
```

```
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.ComboBoxMode = ComboBoxMode.Append;
layout.Children.Add(comboBox);
Content = layout;
```



### Suggesting choices and appending suggestions to text

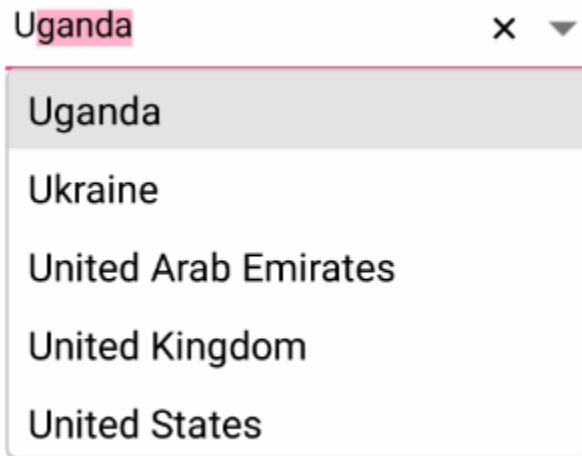
The text is appended to the first matched item in the suggestions collection, and filtered suggestions are displayed in a drop-down list. Users can pick an item from a list directly or use up and down keys for browsing the list.

### XML

```
<StackLayout VerticalOptions="Start" HorizontalOptions="Start" Padding="30">
  <combobox:SfComboBox HeightRequest="40" x:Name="comboBox"
    IsEditableMode="true" AllowFiltering="true" ComboBoxMode="SuggestAppend">
    <combobox:SfComboBox.DataSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Uganda </x:String>
        <x:String> Ukraine </x:String>
        <x:String> United Arab Emirates </x:String>
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.DataSource>
  </combobox:SfComboBox>
</StackLayout>
```

### C#

```
StackLayout layout = new StackLayout()
{
    VerticalOptions = LayoutOptions.Start,
    HorizontalOptions = LayoutOptions.Start,
    Padding = new Thickness(30)
};
List<String> countryNames = new List<String>();
countryNames.Add("Uganda");
countryNames.Add("Ukraine");
countryNames.Add("United Arab Emirates");
countryNames.Add("United Kingdom");
countryNames.Add("United States");
SfComboBox comboBox = new SfComboBox();
comboBox.HeightRequest = 40;
comboBox.DataSource = countryNames;
comboBox.IsEditableMode = true;
comboBox.AllowFiltering = true;
comboBox.ComboBoxMode = ComboBoxMode.SuggestAppend;
layout.Children.Add(comboBox);
Content = layout;
```

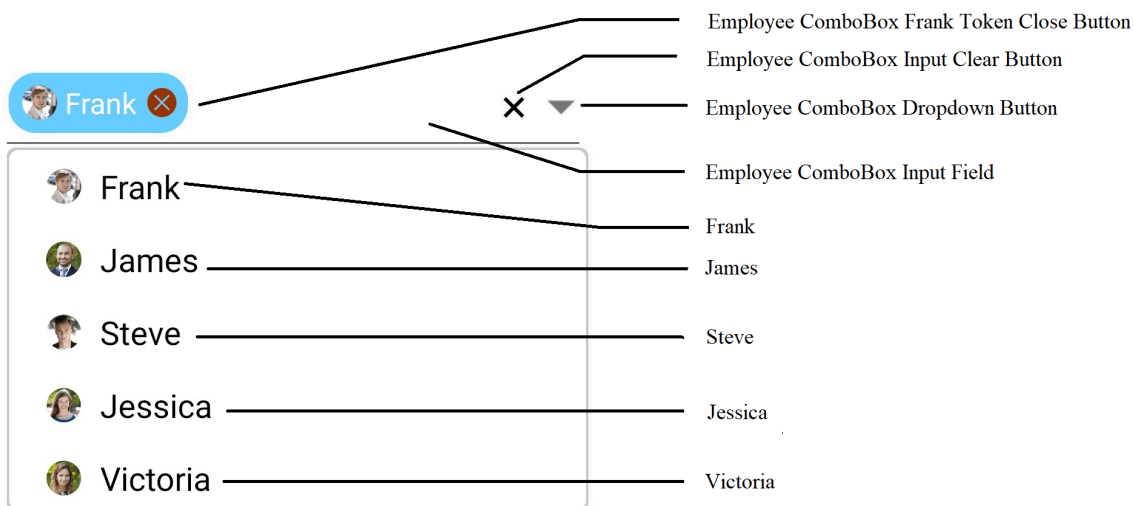


### AutomationId

The SfComboBox control has built-in **AutomationId** for inner elements. The **AutomationId** API allows the automation framework to find and interact with the inner elements of the SfComboBox control. To keep unique AutomationId, these inner elements' AutomationIds are updated based on the control's **AutomationId**.

For example, If you set SfComboBox's **AutomationId** as "Employee ComboBox", then the automation framework will interact with the drop-down button as "Employee ComboBox Dropdown Button".

The following screenshot illustrates the AutomationIds of inner elements. The automation framework will interact with the dropdown for scrolling the items as "Employee ComboBox Dropdown". You can also interact with the elements inside the HeaderView and FooterView with the element's AutomationId. The Automation framework will not interact with the Input Clear Button when the **MultiSelectMode** is None and Delimiter mode.



## SfDataForm

### SfDataForm

The SfDataForm control helps editing the data fields of any data object. It can be used to develop various forms such as login, reservation, data entry, etc. Key features includes the following:

- Layout and grouping: Supports to linear, grid layout and floating label layout with grouping support. Supports customizing the layout with different heights for each item.
- Caption customization: Supports loading the image as caption for the editor.
- Editors: Built-in support for text, numeric, numeric up-down, picker, date picker, time picker, switch, drop-down, autoComplete and checkbox editors.
- Custom editor: Supports loading the custom editors.
- Validation: Built-in support to validate the data based on the [INotifyDataErrorInfo](#) and data annotations. It also programmatically supports validation handling.

### Getting started

This section explains the quick overview to use the [SfDataForm](#) for Xamarin.Forms in your application.

#### Assembly deployment

After installing Essential Studio for Xamarin, you can find all the required assemblies in the installation folders, {Syncfusion Essential Studio Installed location} \Essential Studio\16.x.x.x\Xamarin\lib

Eg: C:\Program Files (x86) \Syncfusion\Essential Studio\16.1.0.24\Xamarin\lib

---

**Note:** Assemblies can be found in unzipped package location in Mac.

---

#### Adding SfDataForm reference

You can add SfDataForm reference using one of the following methods:

##### Method 1: Adding SfDataForm reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](#). To add SfDataForm to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfDataForm](#), and then install it.

![Adding SfDataForm reference from NuGet](SfDataForm\_images/Adding SfDataForm reference.png)

---

**Note:** Install the same version of SfDataForm NuGet in all the projects.

---

##### Method 2: Adding SfDataForm reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfDataForm control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

##### Method 3: Adding SfDataForm assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfDataForm.XForms.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.SfNumericUpDown.XForms.dll Syncfusion.SfComboBox.XForms.dll
-----	---



	Syncfusion.SfAutoComplete.XForms.dll Syncfusion.Buttons.XForms.dll Syncfusion.SfMaskedEdit.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll System.ComponentModel.Annotations
Android	Syncfusion.SfDataForm.XForms.dll Syncfusion.SfDataForm.XForms.Android.dll Syncfusion.SfNumericTextBox.Android.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.SfNumericTextBox.XForms.Android.dll Syncfusion.SfNumericUpDown.Android.dll Syncfusion.SfNumericUpDown.XForms.dll Syncfusion.SfNumericUpDown.XForms.Android.dll Syncfusion.SfComboBox.XForms.dll Syncfusion.SfComboBox.XForms.Android.dll Syncfusion.SfAutoComplete.XForms.dll Syncfusion.SfAutoComplete.XForms.Android.dll Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.Android.dll Syncfusion.SfMaskedEdit.XForms.dll Syncfusion.SfMaskedEdit.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfDataForm.XForms.dll Syncfusion.SfDataForm.XForms.iOS.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.SfNumericTextBox.XForms.iOS.dll Syncfusion.SfNumericUpDown.iOS.dll Syncfusion.SfNumericUpDown.XForms.dll Syncfusion.SfNumericUpDown.XForms.iOS.dll Syncfusion.SfComboBox.XForms.dll Syncfusion.SfComboBox.XForms.iOS.dll Syncfusion.SfAutoComplete.XForms.dll Syncfusion.SfAutoComplete.XForms.iOS.dll Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.iOS.dll Syncfusion.SfMaskedEdit.XForms.dll Syncfusion.SfMaskedEdit.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfDataForm.XForms.dll Syncfusion.SfDataForm.XForms.UWP.dll Syncfusion.SfInput.UWP.dll

	Syncfusion.SfShared.UWP.dll Syncfusion.SfNumericTextBox.XForms.UWP.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.SfNumericUpDown.XForms.UWP.dll Syncfusion.SfNumericUpDown.XForms.dll Syncfusion.SfComboBox.XForms.dll Syncfusion.SfComboBox.XForms.UWP.dll Syncfusion.SfAutoComplete.XForms.dll Syncfusion.SfAutoComplete.XForms.UWP.dll Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.UWP.dll Syncfusion.SfMaskedEdit.XForms.dll Syncfusion.SfMaskedEdit.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
--	---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

#### Launching the data form on each platform

To use the data form inside an application, each platform application must initialize the data form renderer. This initialization step varies from platform to platform and is discussed in the following sections:

##### Android

The Android launches the data form without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

**Note:** If you are adding the references from toolbox, this step is not needed.

##### iOS

To launch the data form in iOS, call the `SfDataFormRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework initialization and before the `LoadApplication` is called, as demonstrated in the following code example:

##### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    SfDataFormRenderer.Init ();
    LoadApplication (new App ());
    ...
}
```

### Universal Windows Platform (UWP)

The UWP launches the data form without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

### ReleaseMode issue in UWP platform

The known Framework issue in UWP platform is that the custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the data form assemblies in **App.xaml.cs** file in UWP project as in the following code snippet:

### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add 'using System.Reflection;'
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfDataFormRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfNumericTextBoxRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfNumericUpDownRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfSegmentedControlRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfComboBoxRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfCheckBoxRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfRadioButtonRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfMaskedEditRenderer).GetTypeInfo().Assembly);
    ;
    assembliesToInclude.Add(typeof(SfTextInputLayoutRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfAutoCompleteRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Creating the data form

In this section, you will create Xamarin.Forms application with **SfDataForm**. The control should be configured entirely in C# code.

- Creating the project.
- Adding data form in Xamarin.Forms.
- Creating data object.
- Setting data object.

### Creating the project

Create a new BlankApp (.Net Standard) application in Xamarin Studio or Visual Studio for Xamarin.Forms.

### Adding data form in Xamarin.Forms

To add the data form to your application, follow the steps:

1. Add required assemblies as discussed in assembly deployment section.
2. Import the control namespace as `xmlns:dataForm="clr-namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"` in XAML Page.
3. Create an instance of data form control and add as a view to the linear layout.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<dataForm:SfDataForm x:Name="dataForm"/>
</ContentPage>
```

### C#

```
using Syncfusion.XForms.DataForm;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        SfDataForm dataForm;
        public MainPage()
        {
            InitializeComponent();
            dataForm = new SfDataForm();
        }
    }
}
```

### Creating data object

The `SfDataForm` is a data edit control so, create a data object to edit the data object.

Here, the data object named **ContactsInfo** created with some properties.

### C#

```
public class ContactsInfo
{
    private string firstName;
    private string middleName;
    private string lastName;
```

```
private string contactNo;
private string email;
private string address;
private DateTime? birthDate;
private string groupName;
public ContactsInfo()
{
}
public string FirstName
{
get { return this.firstName; }
set
{
this.firstName = value;
}
}
public string MiddleName
{
get { return this.middleName; }
set
{
this.middleName = value;
}
}
public string LastName
{
get { return this.lastName; }
set
{
this.lastName = value;
}
}
public string ContactNumber
{
get { return contactNo; }
set
{
this.contactNo = value;
}
}
public string Email
{
get { return email; }
set
{
email = value;
}
}
public string Address
{
get { return address; }
set
{
address = value;
}
}
public DateTime? BirthDate
```

```

{
    get { return birthDate; }
    set
    {
        birthDate = value;
    }
}
public string GroupName
{
    get { return groupName; }
    set
    {
        groupName = value;
    }
}
}

```

**Note:** If you want your data model to respond to property changes, then implement `INotifyPropertyChanged` interface in your model class.

Create a model repository class with `ContactsInfo` property initialized with required data in a new class file as shown in the following code example and save it `ViewModel.cs` file:

### C#

```

public class ViewModel
{
    private ContactsInfo contactsInfo;
    public ContactsInfo ContactsInfo
    {
        get { return this.contactsInfo; }
        set { this.contactsInfo = value; }
    }
    public ViewModel()
    {
        this.contactsInfo = new ContactsInfo();
    }
}

```

### Setting data object

To populate the labels and editors in the data form, set the [DataObject](#) property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:GettingStarted"
    xmlns:dataForm="clr-namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
    x:Class="GettingStarted.MainPage">
    <ContentPage.BindingContext>
    <local:ViewModel/>
    </ContentPage.BindingContext>
    <dataForm:SfDataForm x:Name="dataForm"
    DataObject="{Binding ContactsInfo}"/>

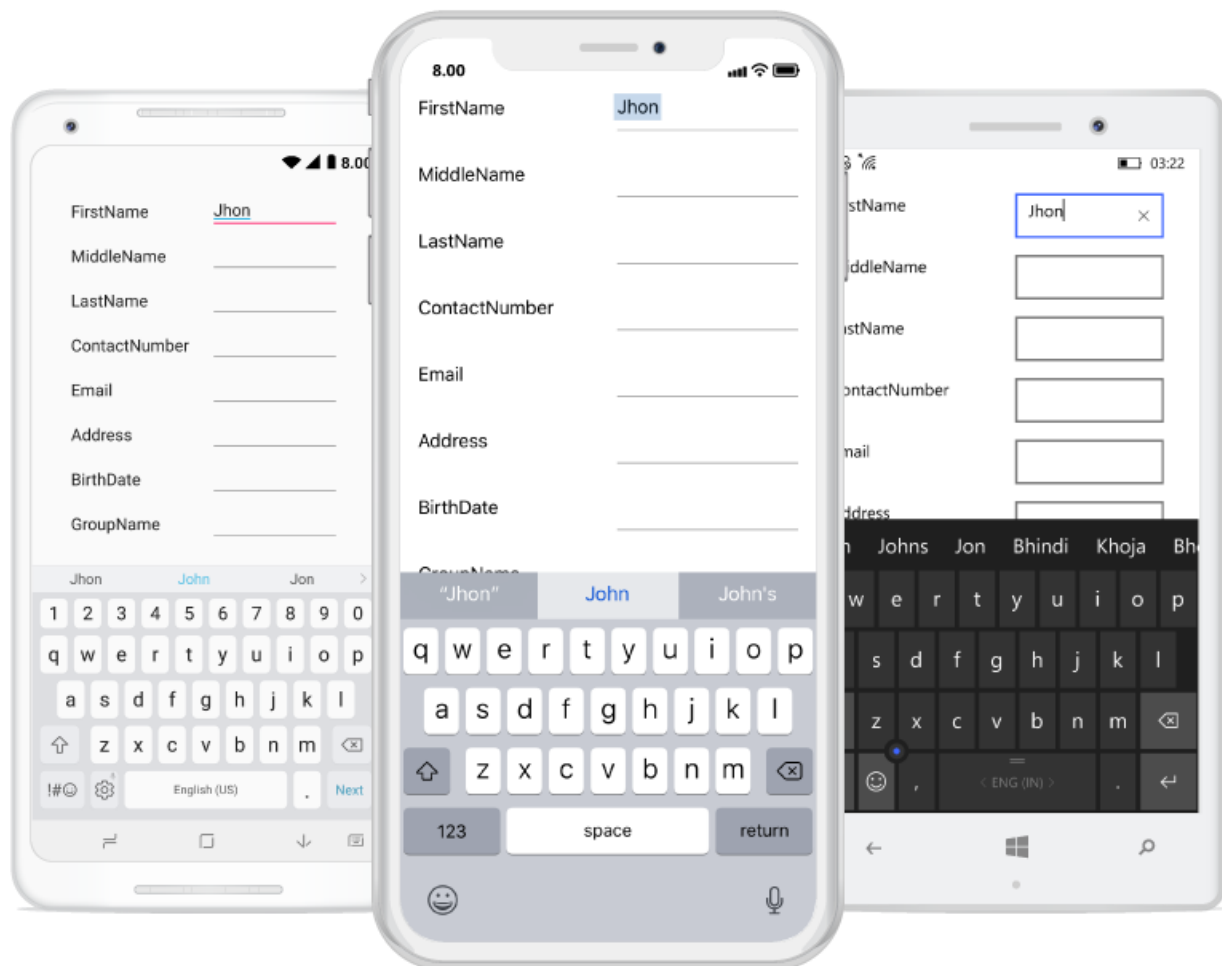
```

```
</ContentPage>
```

### C#

```
dataForm.DataObject = new ContactsInfo();
```

Now, run the application to render the data form to edit the data object as in the following screenshot:



You can download the entire source code of this demo for Xamarin.Forms from here [DataFormGettingStarted](#).

### Defining editors

The data form control automatically generates [DataFormItems](#) (which has UI settings of data field) when the data object set to the `SfDataForm.DataObject` property. The [DataFormItem](#) encapsulates the layout and editor setting for the data field appearing in the data form. When the `DataFormItems` are generated, you can handle the `SfDataForm.AutoGeneratingDataFormItem` event to customize or cancel the `DataFormItem`.

The type of input editor generated for the data field depends on the type and attribute settings of the property. The following table lists the `DataFormItem` and its constraints for generation:

Generated DataFormItem Type	Data Type / Attribute
<a href="#">DataFormTextItem</a>	Default DataFormItem generated for the String type and the properties with [DataType(DataType.Text)], [DataType(DataType.MultilineText)] and [DataType(DataType.Password)] attributes.
<a href="#">DataFormNumericItem</a>	Generated for the Int or Double type property. [DataType(DataType.Currency)]. [DataType("Percent")]
<a href="#">DataFormDateItem</a>	Generated for the DateTime type property. [DataType(DataType.Date)]. [DataType(DataType.DateTime)].
<a href="#">DataFormTimeItem</a>	Generated for the DateTime type property. [DataType(DataType.Time)].
<a href="#">DataFormPickerItem</a>	Generated for the Enum type property. [EnumDataTypeAttribute]
<a href="#">DataFormSegmentItem</a>	Generated for the Enum type property. [EnumDataTypeAttribute]
<a href="#">DataFormCheckBoxItem</a>	Generated for the Bool type property. [BoolDataTypeAttribute]
<a href="#">DataFormMaskedEditTextItem</a>	Generated for the PhoneNumber type property. [DataType(DataType.PhoneNumber)]
<a href="#">DataFormAutoCompleteItem</a>	Generated for the Enum type property. [EnumDataTypeAttribute]
<a href="#">DataFormDropDownItem</a>	Generated for the Enum type property. [EnumDataTypeAttribute]

The following list of editors are supported:

Editor	Data Type/Attribute	Input control loaded
Text	The String type property and any other type apart from the following specified cases.	[Entry](https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.entry?view=xamarin-forms)
MultilineText	The String type property with multi line text. [DataType(DataType.Multiline)]	<a href="#">Editor</a>
Numeric	Int or Double type property.	<a href="#">SfNumericTextBox</a>
Percent	The Int or Double type Property with percent value. [DataType("Percent")].	{{'SfNumericTextBox'  markdownify }}
Currency	The Int or Double type property with currency value. [DataType(DataType.Currency)].	<a href="#">SfNumericTextBox</a>
Date	The DateTime type property with date value. [DataType(DataType.Date)][DataType(DataType.DateTime)]	<a href="#">DatePicker</a>



Time	Property with [DataType(DataType.Time)] attribute.	<a href="#">TimePicker</a>
NumericUpDown	Int or Double type property.	<a href="#">SfNumericUpDown</a>
Segment	Enum type property.	<a href="#">SfSegmentedControl</a>
Bool	Bool type property.	<a href="#">SfCheckBox</a>
Switch	Bool type property.	<a href="#">Switch</a>
Picker	Enum and List type property. [EnumDataTypeAttribute]	<a href="#">Picker</a>
DropDown	Enum and List type property. [EnumDataTypeAttribute]	<a href="#">SfComboBox</a>
AutoComplete	Enum and List type property. [EnumDataTypeAttribute]	<a href="#">SfAutoComplete</a>
Password	The String type property with [DataType(DataType.Password)] attribute.	{{'Entry'  markdownify }}
RadioGroup	Enum and List type property. [EnumDataTypeAttribute]	<a href="#">SfRadioGroup</a>
MaskedEditText	Property with [DataType(DataType.PhoneNumber)] attribute.	<a href="#">SfMaskedEdit</a>

## Layout options

### Label position

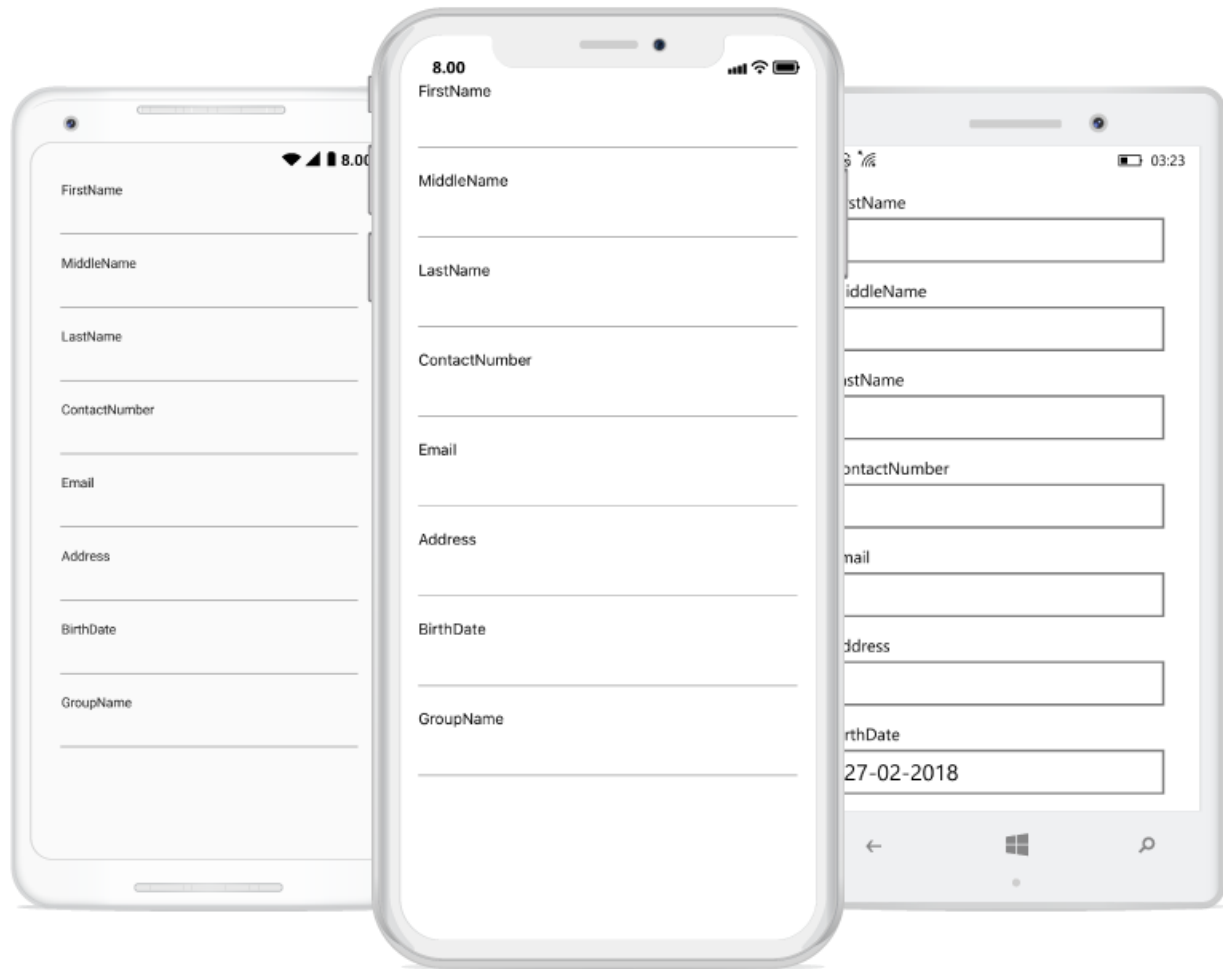
By default, the data form arranges the label at left side and input control at the right side. You can change the label position by setting the [SfDataForm.LabelPosition](#) property. You can position the label from left to top of the input control by setting the `LabelPosition` as `Top`.

### XML

```
<dataForm:SfDataForm LabelPosition="Top"/>
```

### C#

```
dataForm.LabelPosition = LabelPosition.Top;
```



### *Grid layout*

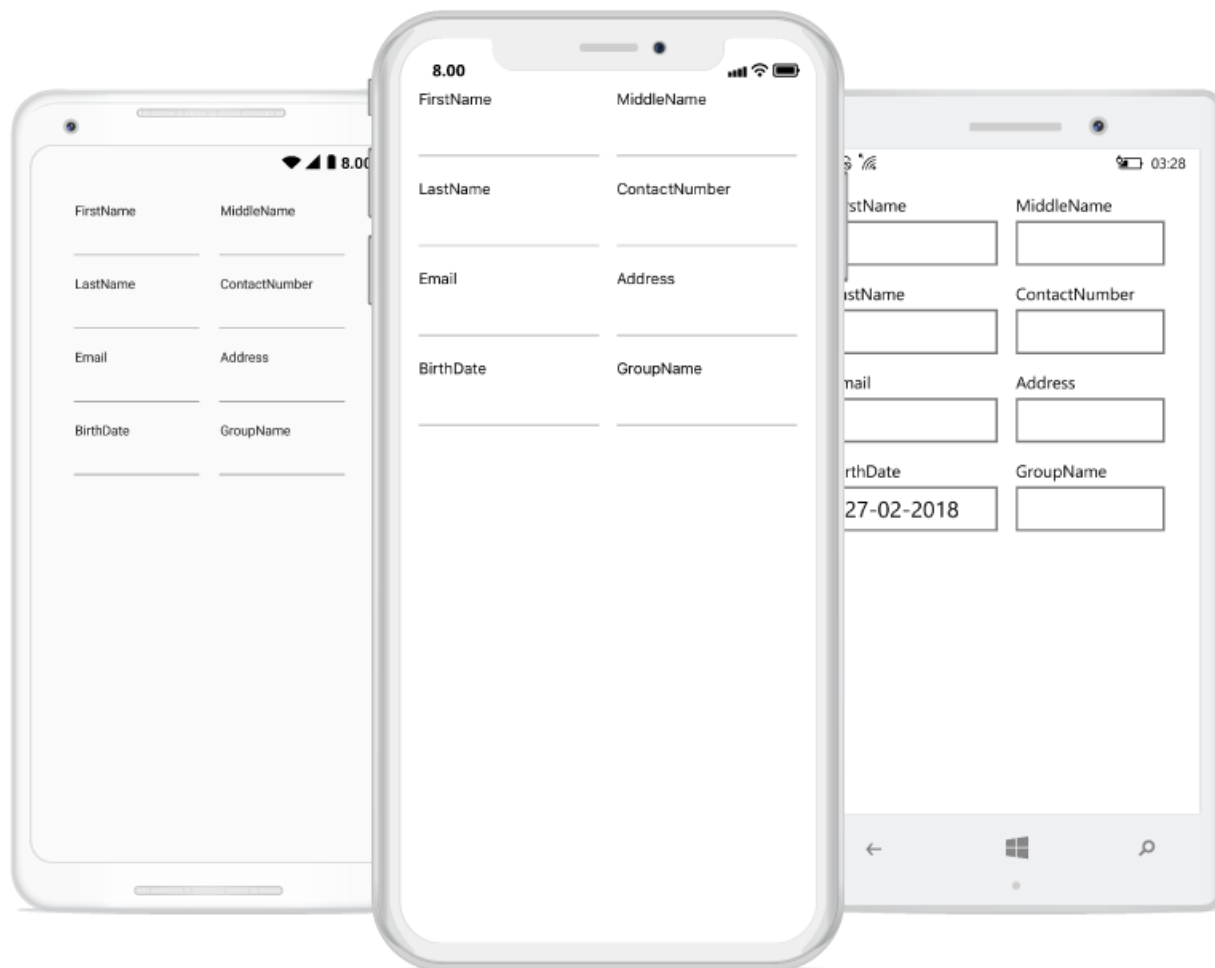
By default, the data form arranges one data field per row. It is possible to have more than one data field per row by setting the [ColumnCount](#) property which provides grid like layout for the data form.

### **XML**

```
<dataForm:SfDataForm ColumnCount="2"/>
```

### **C#**

```
dataForm.ColumnCount = 2;
```



### Loading DataForm inside StackLayout

StackLayout positions the child elements one after another either horizontally or vertically. Space of the [StackLayout](#) depends on the [HorizontalOptions](#) and [VerticalOptions](#) properties. Views in a stack layout can be sized based on space in the layout using layout options.

The DataForm control can be loaded inside any layout such as [Grid](#), [StackLayout](#), etc. When loading DataForm inside a [StackLayout](#), set the [HorizontalOptions](#) and [VerticalOptions](#) properties of DataForm, and set parent(StackLayout) of DataForm to [LayoutOptions.FillAndExpand](#).


Refer to the following code example to load the DataForm control inside a [StackLayout](#). Set the [VerticalOptions](#) and [HorizontalOptions](#) of the [StackLayout](#) and DataForm to [FillAndExpand](#).

#### XML

```
<StackLayout x:Name="stackLayout" VerticalOptions="FillAndExpand"
HorizontalOptions="FillAndExpand">
  <dataForm:SfDataForm x:Name="dataForm" VerticalOptions="FillAndExpand"
HorizontalOptions="FillAndExpand"/>
</StackLayout>
```

#### C#

```
public partial class MainPage : ContentPage
{
    StackLayout StackLayout;
    SfDataForm dataForm;
    public MainPage()
    {
        InitializeComponent();
        stackLayout = new StackLayout();
        stackLayout.VerticalOptions = LayoutOptions.FillAndExpand;
        stackLayout.HorizontalOptions = LayoutOptions.FillAndExpand;
        dataForm = new SfDataForm();
        dataForm.DataObject = new ContactInfo();
        dataForm.VerticalOptions = LayoutOptions.FillAndExpand;
        dataForm.HorizontalOptions = LayoutOptions.FillAndExpand;
        stackLayout.Children.Add(dataform);
        this.Content = stackLayout;
    }
}
```

 100% 12:54 PM

Name	John
Email	
Age	20
Salary	15,000

Loading DataForm with customized height and width

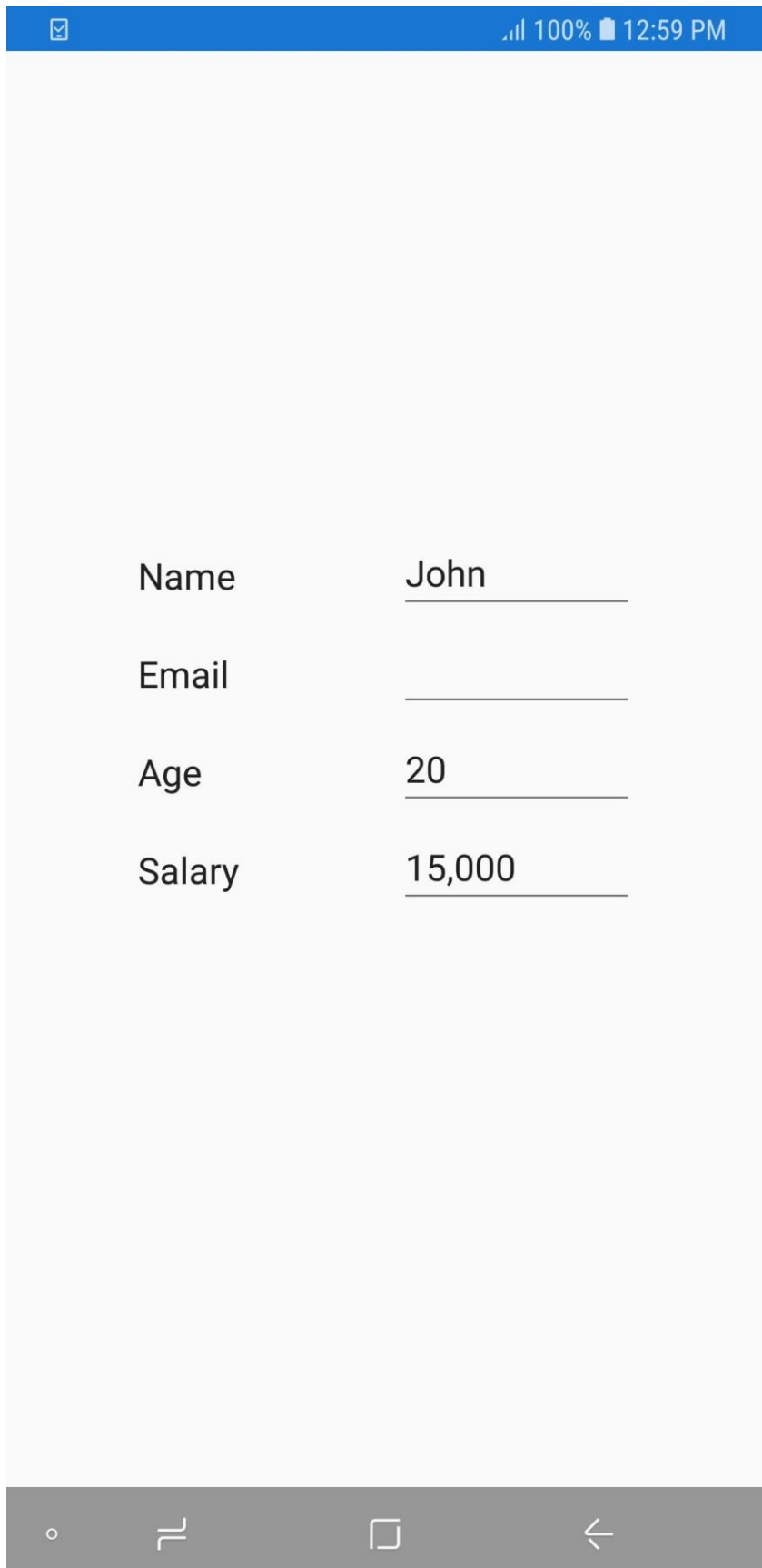
The DataForm can be loaded with specific height and width inside different layouts using the [SfDataForm.HeightRequest](#) and [SfDataForm.WidthRequest](#) properties.

#### **XML**

```
<dataForm:SfDataForm x:Name="dataForm" WidthRequest="300"
HeightRequest="300"
VerticalOptions="CenterAndExpand"
HorizontalOptions="Center"/>
```

#### **C#**

```
dataForm.HeightRequest = 300;
dataForm.WidthRequest = 300;
dataForm.VerticalOptions = LayoutOptions.CenterAndExpand;
dataForm.HorizontalOptions = LayoutOptions.Center;
```



The image shows a mobile application interface with a blue header bar. The header bar contains a checkmark icon on the left and signal strength, 100% battery, and the time 12:59 PM on the right. The main content area is a light gray background with a form. The form has four rows, each with a label on the left and a text input field on the right. The labels are 'Name', 'Email', 'Age', and 'Salary'. The input fields contain the values 'John', an empty field, '20', and '15,000' respectively. At the bottom of the screen is a gray navigation bar with four icons: a circle, a square, a square with a circle inside, and a left arrow.

Name	John
Email	
Age	20
Salary	15,000

## Editing

By default, the data form enables editing of the data field. You can disable editing by setting the [IsReadOnly](#) property of the data form. You can enable or disable editing for a particular data field by setting the [IsReadOnly](#) property of [DataFormItem](#) in the [AutoGeneratingDataFormItem](#) event. The data field editing behavior can also be defined by using [EditableAttribute](#).

## Working with the data form

### Auto-generating DataFormItemS for the data field

By default, the [DataFormItemS](#) will be generated based on the property type. For example, the [DataFormNumericItem](#) will be created for the `int` type property.

The [DataFormItem](#) generation depends on the type and attribute defined for the property.

The following tables lists the several types of [DataFormItem](#) and its constraints for auto generation:

Generated DataFormItem Type	Editor	Data Type / Attribute
<a href="#">DataFormTextItem</a>	Text	Default DataFormItem generated for the String type and the properties with [DataType(DataType.Text)], [DataType(DataType.MultilineText)] and [DataType(DataType.Password)] attributes.
<a href="#">DataFormNumericItem</a>	Numeric	Generated for Int, Double, Float, Decimal, Long types and also its nullable property with [DataType(DataType.Currency)] and [DataType("Percent")] attributes.
<a href="#">DataFormDateItem</a>	Date	Generated for DateTime type and properties with [DataType(DataType.Date)] and [DataType(DataType.DateTime)] attributes.
<a href="#">DataFormTimeItem</a>	Time	Generated for the property with [DataType(DataType.Time)] attribute.
<a href="#">DataFormPickerItem</a>	Picker	Generated for Enum type property and the property with [EnumDataTypeAttribute] attribute.
<a href="#">DataFormDropDownItem</a>	DropDown	Generated for Enum type property and the property with [EnumDataTypeAttribute] attribute.
<a href="#">DataFormAutoCompleteItem</a>	AutoComplete	Generated for Enum type property and the property with [EnumDataTypeAttribute] attribute.
<a href="#">DataFormMaskedEditTextItem</a>	MaskedEditText	Generated for the PhoneNumber type property.[DataType(DataType.PhoneNumber)]
<a href="#">DataFormItem</a>	CheckBox	Bool type



You can customize the property settings or cancel the generation of `DataFormItem` by handling the [AutoGeneratingDataFormItem](#) event.

#### Customize auto generated fields

You can customize or cancel the generated `DataFormItem` by handling the [AutoGeneratingDataFormItem](#) event. This event occurs when the field is auto-generated for public and non-static property of the data object.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-
namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem"/>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
}
```

[AutoGeneratingDataFormItemEventArgs](#) provides the information about the auto-generated. [AutoGeneratingDataFormItemEventArgs.DataFormItem](#) property returns the newly created `DataFormItem`.

#### Cancel DataFormItem generation of the data field

You can cancel the specific `DataFormItem` adding to the data form by handling the `AutoGeneratingDataFormItem` event or by defining display attribute to avoid the particular data field being displayed.

#### Using attributes

You can set [AutoGenerateField](#) to `false` for canceling the `DataFormItem` generation.

#### C#

```
private int id;
[Display(AutoGenerateField = false)]
public int ID
{
    get
    {
        return id;
    }
    set
    {
}
```

```
id = value;  
RaisePropertyChanged("ID");  
}  
}
```

### Using event

In the following code, the `DataFormItem` generation for the `MiddleName` property is canceled by setting the `Cancel` property to true.

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem.Name == "MiddleName")  
        e.Cancel = true;  
}
```

### Changing editor type

You can change the editor of the `DataFormItem` in the `AutoGeneratingDataFormItem` event.

In the following code, the editor is changed for `IsAvailable` field from `Bool` to `Switch`.

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem.Name == "IsAvailable")  
        e.DataFormItem.Editor = "Switch";  
}
```

### Changing property settings

You can change the property of `DataFormItem` in the `AutoGeneratingDataFormItem` event.

Here, `Salary` data field is restricted from being edited in the data form.

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem != null)  
    {  
        if (e.DataFormItem.Name == "Salary")  
            e.DataFormItem.IsReadOnly = true;  
    }  
}
```

### Changing DataFormItem visibility

You can change the `DataFormItem` visibility by using the `IsVisible` property in the `DataFormItem`.

Here, **Salary** data field will be hidden.

### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "Salary")
            e.DataFormItem.IsVisible = false;
    }
}
```

### Changing DataFormItem FontSize

Using the [EditorFontSize](#), [LabelFontSize](#), and [ValidationLabelFontSize](#) properties from **DataFormItem**, you can define the font size of the Editor, Label, and ValidationLabel. Changing the font size will be handled in the **AutoGeneratingDataFormItem** event.

You can define the font size as described as follows.

- Set the value to font size directly or use the named font sizes such as **Small**, **Medium**, and **Large**.

### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender, AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        // Setting Fontsize using NamedSize.
        e.DataFormItem.EditorFontSize = Device.GetNamedSize(NamedSize.Small,
            typeof(Label));
        e.DataFormItem.LabelFontSize = Device.GetNamedSize(NamedSize.Large,
            typeof(Label));
        e.DataFormItem.ValidationLabelFontSize =
            Device.GetNamedSize(NamedSize.Micro, typeof(Label));
        // Setting value to FontSize directly.
        e.DataFormItem.EditorFontSize = 8;
        e.DataFormItem.LabelFontSize = 8;
        e.DataFormItem.ValidationLabelFontSize = 8;
    }
}
```

### Setting watermark

You can display the watermark in the editor by defining the display attribute or using the **AutoGeneratingDataFormItem** event.

#### Using attribute

You can show the watermark in the editor by setting the [Prompt](#) in display attribute.

**C#**

```
private string middleName;
[Display(Prompt = "Enter middle name")]
public string MiddleName
{
    get { return this.middleName; }
    set
    {
        this.middleName = value;
    }
}
```

*Using event*

You can show the watermark in the editor by using the [PlaceholderText](#) property in [DataFormItem](#).

**C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "Description")
            e.DataFormItem.PlaceholderText = "Enter description";
    }
}
```

*Changing DataFormItem*

You can change the created [DataFormItem](#) and assign new [DataFormItem](#) in the [AutoGeneratingDataFormItem](#) event.

Here, [DataFormTextItem](#) with number keyboard is loaded for numeric value instead of [DataFormNumericItem](#).

**C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "ID")
            e.DataFormItem = new DataFormTextItem() { Name = "ID", Editor = "Text",
            KeyBoard = Keyboard.Numeric };
    }
}
```

*Adding or removing the data field displayed in the dataForm at runtime*

If you want to remove or add data fields item at runtime, you can use the [RefreshLayout](#) method which auto-generates the [DataFormItem](#) where you can skip certain item from display. By default, it will

generate the canceled items initially. If you want to regenerate all the items, you should pass argument as `true`.

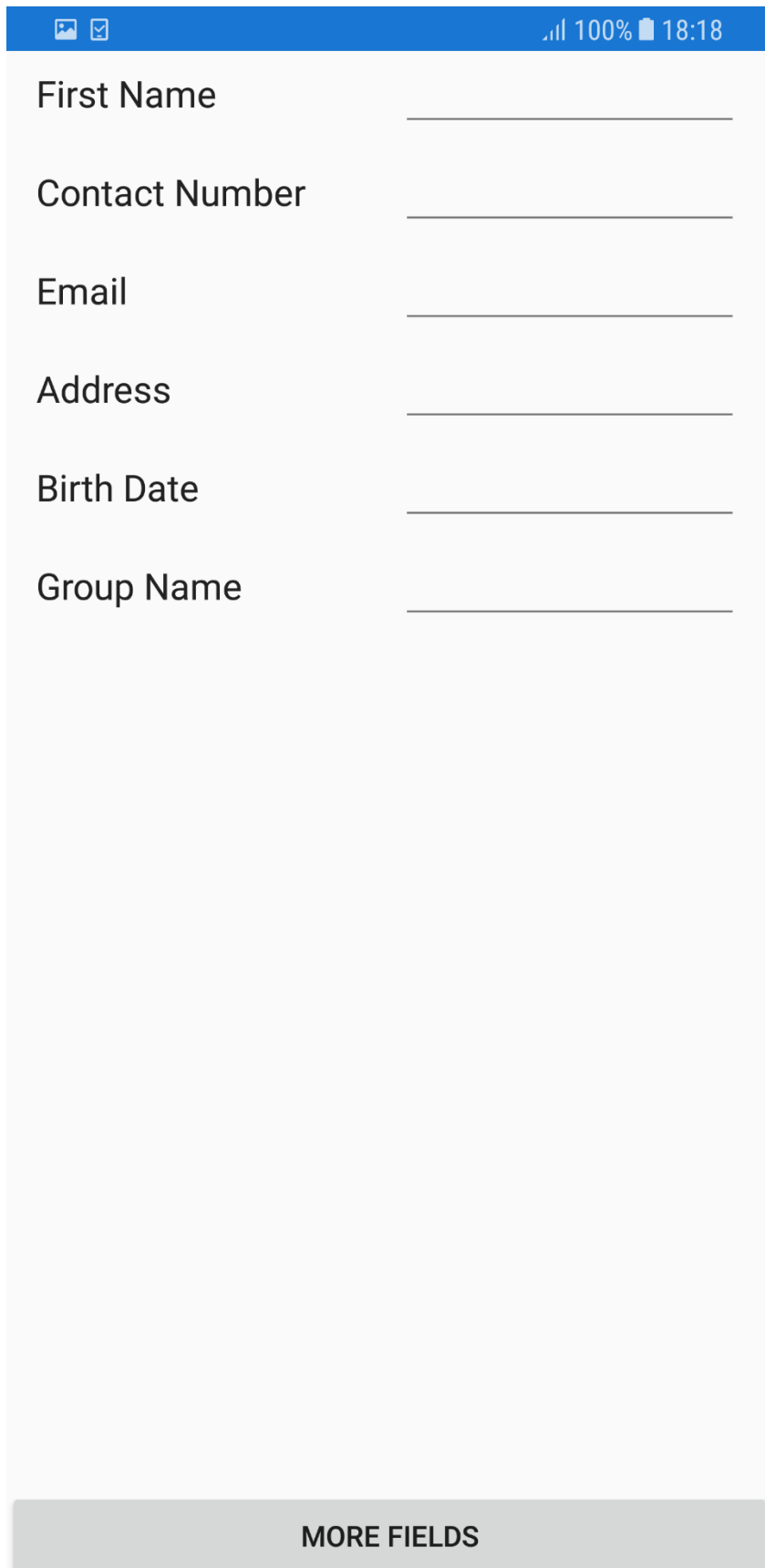
In the following code snippet, items are auto generated based on `refreshLayout` flag where you can change flag at runtime and call `RefreshLayout` method to add or remove items being displayed in the data form at runtime.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-
namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<dataForm:SfDataForm Grid.Row="0" x:Name="dataForm"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem"/>
<Button x:Name="Commit" Grid.Row="1" WidthRequest="100" HeightRequest="50"
Text="MORE FIELDS" Clicked="Button_Clicked"/>
</Grid>
</ContentPage.Content>
</ContentPage>
```

### C#

```
dataForm.DataObject = new ContactsInfo();
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (!refreshLayout)
        {
            if (e.DataFormItem.Name.Equals("MiddleName") ||
e.DataFormItem.Name.Equals("LastName"))
            e.Cancel = true;
        }
        else
        {
            if (e.DataFormItem.Name == "GroupName")
            e.Cancel = true;
        }
    }
}
```



The image shows a mobile application interface with a blue header bar. On the left of the header are two icons: a camera and a checkmark. On the right is the status bar showing signal strength, 100% battery, and the time 18:18. Below the header is a light gray form area containing six text input fields, each with a label to its left: 'First Name', 'Contact Number', 'Email', 'Address', 'Birth Date', and 'Group Name'. At the bottom of the form area is a gray button with the text 'MORE FIELDS'.

First Name

Contact Number

Email

Address

Birth Date

Group Name

MORE FIELDS

If you want to generate the MiddleName and LastName fields at runtime, you should set `refreshLayout` flag to `true` and call the [RefreshLayout](#) method which triggers `AutoGeneratingDataFormItem` event again and generates the items based on `refreshLayout` flag.

#### **C#**

```
private void Button_Click(object sender, System.EventArgs e)
{
    refreshLayout = true;
    dataForm.RefreshLayout();
}
```

Here, the MiddleName and LastName fields are generated at runtime after clicking the more field button.

The image shows a mobile application interface with a blue status bar at the top. The status bar contains icons for signal strength, 100% battery, and the time 18:20. Below the status bar is a light gray form with eight text input fields, each with a label to its left: 'First Name', 'Middle Name', 'Last Name', 'Contact Number', 'Email', 'Address', 'Birth Date', and 'Group Name'. Each label is followed by a horizontal line representing the input field. At the bottom of the form is a gray button with the text 'MORE FIELDS' in bold, uppercase letters.



The `GroupName` field is displayed initially in the data form. If you want to remove it at runtime, you should set `refreshLayout` flag to true and pass the argument as true in [RefreshLayout](#) method. It triggers `AutoGeneratingDataFormItem` event for all the fields where you can cancel 'GroupName' field item generation.

**C#**

```
private void Button_Click(object sender, System.EventArgs e)
{
    refreshLayout = true;
    dataForm.RefreshLayout(true);
}
```

Here, the `GroupName` field is removed at runtime.

A mobile application mockup featuring a data form. The top status bar is blue with icons for camera, gallery, signal strength, 100% battery, and the time 18:19. The form consists of seven text input fields, each with a label to its left: 'First Name', 'Middle Name', 'Last Name', 'Contact Number', 'Email', 'Address', and 'Birth Date'. At the bottom of the form is a grey button labeled 'MORE FIELDS'.

You can download the sample from [here](#).

### DataFormItemManager

The [DataFormItemManager](#) creates [DataFormItems](#) collection and handles value reflection and validation. It also overrides to handle the get and set property values from and to the data object.

### *Manually generate DataFormItems for DataObject*

By default, [DataFormItems](#) will be generated based on DataObject. If you need to generate DataFormItems manually, override the [DataFormItemManager](#) class and set it to [SfDataForm.ItemManager](#).

To create [DataFormItems](#), override the [GenerateDataFormItems](#) method.

### C#

```
// dataform item creating by setting DataObject.
dataForm.DataObject = new ContactsInfo();
dataForm.ItemManager = new DataFormItemManagerExt(dataForm);
public class DataFormItemManagerExt : DataFormItemManager
{
    SfDataForm sfDataForm;
    public DataFormItemManagerExt(SfDataForm dataForm) : base(dataForm)
    {
        sfDataForm = dataForm;
    }
    protected override List<DataFormItemBase>
    GenerateDataFormItems(PropertyInfoCollection itemProperties,
    List<DataFormItemBase> dataFormItems)
    {
        var items = new List<DataFormItemBase>();
        foreach (var propertyInfo in itemProperties)
        {
            DataFormItem dataFormItem;
            if (propertyInfo.Key == "ID")
                dataFormItem = new DataFormNumericItem() { Name = propertyInfo.Key, Editor = "Numeric", MaximumNumberDecimalDigits = 0 };
            else if (propertyInfo.Key == "Name")
                dataFormItem = new DataFormTextItem() { Name = propertyInfo.Key, Editor = "Text" };
            else
                dataFormItem = new DataFormTextItem() { Name = propertyInfo.Key, Editor = "Text" };
            items.Add(dataFormItem);
        }
        return items;
    }
}
```

You can download the source code of this demo from [GenerateDataFormItemsForDataObject](#)

### *Manually generate DataFormItems for data dictionary*

You can load the dataform with custom dictionary by generating DataFormItems manually. To create DataFormItems from dictionary, override the [GenerateDataFormItems](#) method.

### C#

```
// dataform item creating using dictionary.
dataForm.DataObject = new object();
var dictionary = new Dictionary<string, object>();
dictionary.Add("ID", 1);
dictionary.Add("Name", "John");
dataForm.ItemManager = new DataFormItemManagerExt(dataForm, dictionary);
public class DataFormItemManagerExt : DataFormItemManager
{
    Dictionary<string, object> dataFormDictionary;
    public DataFormItemManagerExt(SfDataForm dataForm, Dictionary<string,
    object> dictionary) : base(dataForm)
    {
        dataFormDictionary = dictionary;
    }
    protected override List<DataFormItemBase>
    GenerateDataFormItems(PropertyInfoCollection itemProperties,
    List<DataFormItemBase> dataFormItems)
    {
        var items = new List<DataFormItemBase>();
        foreach (var key in dataFormDictionary.Keys)
        {
            DataFormItem dataFormItem;
            if (key == "ID")
                dataFormItem = new DataFormNumericItem() { Name = key, Editor = "Numeric",
                MaximumNumberDecimalDigits = 0 };
            else if (key == "Name")
                dataFormItem = new DataFormTextItem() { Name = key, Editor = "Text" };
            else
                dataFormItem = new DataFormTextItem() { Name = key, Editor = "Text" };
            items.Add(dataFormItem);
        }
        return items;
    }
}
```

#### Handling reading and writing values to and from the dictionary

By default, the dictionary value will be shown in corresponding editor and value changes in editor will be committed again in dictionary value by using the [GetValue](#) and [SetValue](#) override methods in [DataFormItemManager](#).

Here, the value is read and written from/to dictionary instead of the data object.

#### C#

```
public class DataFormItemManagerExt : DataFormItemManager
{
    Dictionary<string, object> dataFormDictionary;
    public DataFormItemManagerExt(SfDataForm dataForm, Dictionary<string,
    object> dictionary) : base(dataForm)
    {
        dataFormDictionary = dictionary;
    }
    public override object GetValue(DataFormItem dataFormItem)
    {
        var value = dataFormDictionary[dataFormItem.Name];
        return value;
    }
}
```

```

}
public override void SetValue(DataFormItem dataFormItem, object value)
{
    dataFormDictionary[dataFormItem.Name] = value;
}
}

```

Here, the dataform is loaded with field from dictionary.

You can download the source code of this demo from [GenerateDataFormItemsForDictionary](#)

### Binding with dynamic data object

You can also load the dynamic object in SfDataForm **DataObject** and by default text editor will be generated for each dynamic object property. You can change the editor of DataFormItem for dynamic object property data type (default string) by using the AutoGeneratingDataFormItem event. You can find details about this [here](#)

### Limitations

DynamicObject supports only in Xamarin.Forms Android and data form item will be created for each dynamic object property. Currently Dynamic object is not supported in UWP and iOS platform and you can find more details about this for iOS platform under the following links.

<https://forums.xamarin.com/discussion/53941/expandoobject-crashing-ios>

[https://developer.xamarin.com/guides/ios/advanced\\_topics/limitations/](https://developer.xamarin.com/guides/ios/advanced_topics/limitations/)

### C#

```

dataForm.DataObject = new DynamicDataModel().ExpandoObject;
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
    AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "DateOfBirth")
        e.DataFormItem.Editor = "Date";
}
public class DynamicDataModel
{
    public dynamic ExpandoObject;
    public dynamic DynamicObject;
    public DynamicDataModel()
    {
        ExpandoObject = new ExpandoObject();
        ExpandoObject.FirstName = "John";
        ExpandoObject.LastName = "";
        ExpandoObject.DateOfBirth = DateTime.Now.Date;
        ExpandoObject.Email = "";
        DynamicObject = new Data();
        DynamicObject.FirstName = "John";
        DynamicObject.LastName = "";
        DynamicObject.DateOfBirth = DateTime.Now.Date;
        DynamicObject.Email = "";
    }
}
public class Data : DynamicObject, IDictionary<string, object>
{

```

```
Dictionary<string, object> list = new Dictionary<string, object>();
public override bool TrySetMember(SetMemberBinder binder, object value)
{
    list[binder.Name] = value;
    return true;
}
public override bool TryGetMember(GetMemberBinder binder, out object result)
{
    return list.TryGetValue(binder.Name, out result);
}
public override IEnumerable<string> GetDynamicMemberNames()
{
    return list.Keys;
}
public void Add(string key, object value)
{
    this.list.Add(key, value);
}
public bool ContainsKey(string key)
{
    return this.list.ContainsKey(key);
}
public ICollection<string> Keys
{
    get { return this.list.Keys; }
}
public bool Remove(string key)
{
    return this.list.Remove(key);
}
public bool TryGetValue(string key, out object value)
{
    return this.list.TryGetValue(key, out value);
}
public ICollection<object> Values
{
    get { return this.list.Values; }
}
object IDictionary<string, object>.this[string key]
{
    get { return this.list[key]; }
    set { this.list[key] = value; }
}
public void Add(KeyValuePair<string, object> item)
{
    this.list.Add(item.Key, item.Value);
}
public void Clear()
{
    this.list.Clear();
}
public bool Contains(KeyValuePair<string, object> item)
{
    return this.list.Contains(item);
}
public void CopyTo(KeyValuePair<string, object>[] array, int arrayIndex)
{

```

```
}  
public int Count  
{  
    get { return this.list.Count; }  
}  
public bool IsReadOnly  
{  
    get { return false; }  
}  
public bool Remove(KeyValuePair<string, object> item)  
{  
    return this.list.Remove(item.Key);  
}  
public IEnumerator<KeyValuePair<string, object>> GetEnumerator()  
{  
    return this.list.GetEnumerator();  
}  
System.Collections.IEnumerator  
System.Collections.IEnumerable.GetEnumerator()  
{  
    return this.list.GetEnumerator();  
}  
}
```



You can download the sample from [here](#)

#### Adding custom DataFormItems

Support has been provided to generate custom DataFormItems for the defined business model using the [Items](#) property of the `SfDataForm` class. You need to set the [AutoGenerateItems](#) property to false to restrict the auto generation of DataFormItems.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms">
```



```

x:Class="GettingStarted.MainPage">
<ContentPage.BindingContext>
<local:ViewModel x:Name="bindingContext"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding Details}"
AutoGenerateItems="false">
<dataForm:SfDataForm.Items>
<dataForm:DataFormTextItem Name="Name" Editor="Text"/>
<dataForm:DataFormTextItem Name="Password" Editor="Password"/>
<dataForm:DataFormMaskedEditTextItem Name="Phone" Editor="MaskedEditText"/>
<dataForm:DataFormTextItem Name="Address" Editor="MultilineText"/>
<dataForm:DataFormAutoCompleteItem Name="Countries" Editor="AutoComplete"
ItemsSource = "{Binding CountryNames}"/>
<dataForm:DataFormDateItem Name="BirthTime" Editor="Date"/>
</dataForm:SfDataForm.Items>
</dataForm:SfDataForm>
</ContentPage.Content>
</ContentPage>

```

## C#

```

ObservableCollection<DataFormItemBase> items = new
ObservableCollection<DataFormItemBase>();
items.Add(new DataFormTextItem() { Name = "Name", Editor = "Text" });
items.Add(new DataFormTextItem() { Name = "Password", Editor = "Password"
});
items.Add(new DataFormMaskedEditTextItem() { Name = "Phone", Editor =
"MaskedEditText" });
items.Add(new DataFormTextItem() { Name = "Address", Editor =
"MultilineText" });
items.Add(new DataFormAutoCompleteItem() { Name = "Countries", Editor =
"AutoComplete", ItemsSource = this.GetItemSource("Countries")});
items.Add(new DataFormTimeItem() { Name = "BirthTime", Editor = "Time" });
dataForm.AutoGenerateItems = false;
dataForm.Items = items;
public class ViewModel
{
private DataFormModel details;
private IList<string> countryNames;
public ViewModel()
{
details = new DataFormModel();
countryNames = new List<string>
{
"United states",
"United Kingdom",
"France",
"Belgium",
"Germany"
};
}
public DataFormModel Details
{
get { return this.details; }
set { this.details = value; }
}
}

```

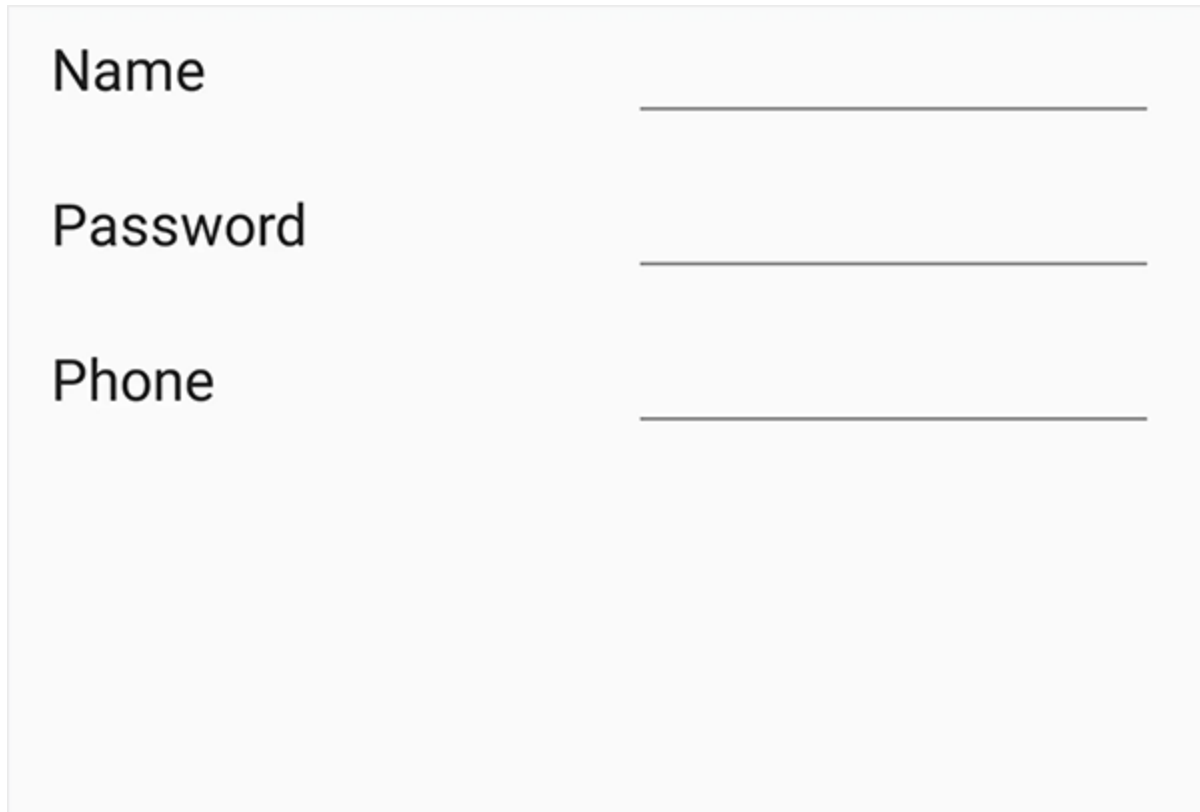
```
}  
public IList<string> CountryNames  
{  
    get { return this.countryNames; }  
    set { this.countryNames = value; }  
}  
private IList GetItemSource(string sourceName)  
{  
    var list = new List<string>();  
    if (sourceName == "Countries")  
    {  
        list.Add("Afghanistan");  
        list.Add("Akrotiri");  
        list.Add("Albania");  
        list.Add("Algeria");  
        list.Add("American Samoa");  
        list.Add("Andorra");  
        list.Add("Angola");  
        list.Add("Anguilla");  
        list.Add("Antarctica");  
        list.Add("Antigua and Barbuda");  
    }  
    return list;  
}
```

#### *Dynamically add custom dataform items*

Support has been provided to dynamically add the dataform items to collections using the [Items](#) property of **SfDataForm**.

#### **C#**

```
dataForm.Items.Add(new DataFormDropDownItem()  
{ Name = "StateName",  
  Editor = "DropDown",  
  ItemsSource = this.GetItemSource("StateName"),  
  PlaceholderText = "Select a State"  
});  
dataForm.Items.Add(new DataFormItem()  
{ Name = "Save",  
  Editor = "Switch"  
});
```

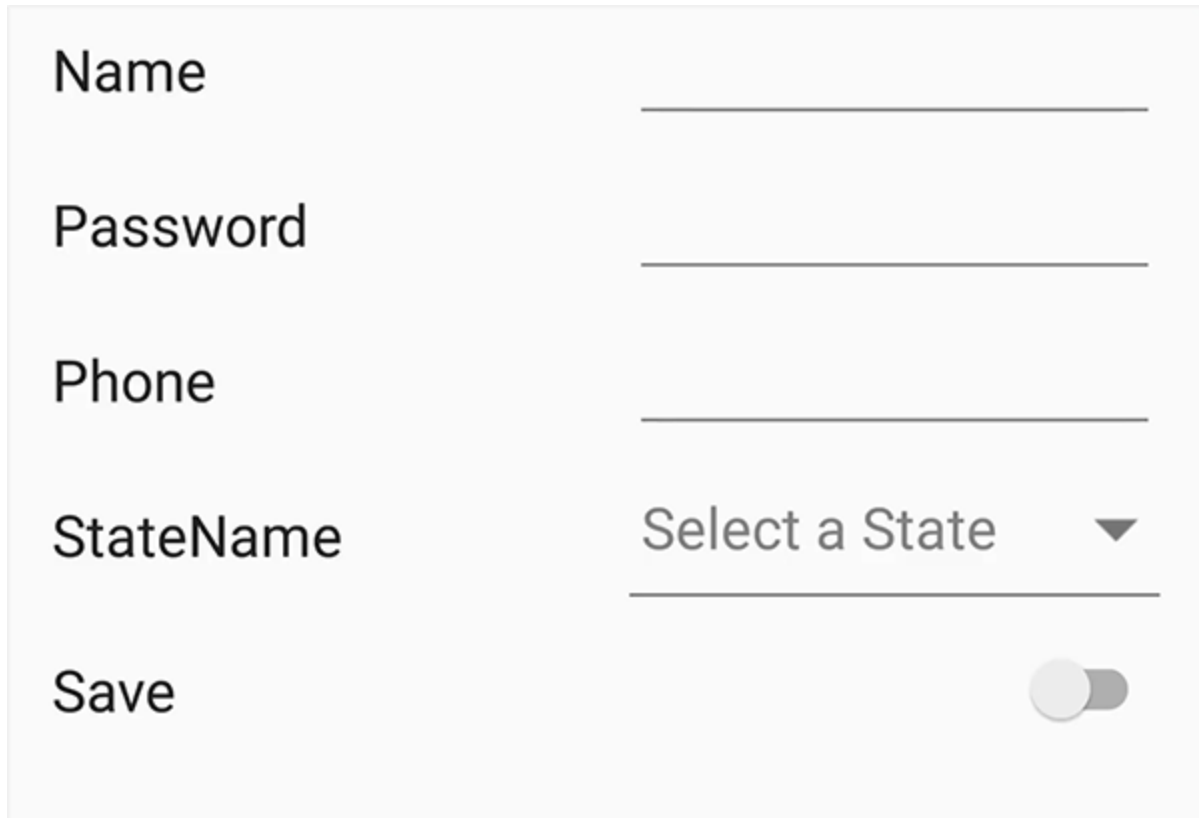
A screenshot of a data form with three input fields. The first field is labeled 'Name' and has a single horizontal line for input. The second field is labeled 'Password' and has a single horizontal line for input. The third field is labeled 'Phone' and has a single horizontal line for input. The form is displayed on a light gray background.

#### *Dynamically remove custom dataform items*

Support has been provided to dynamically remove the dataform items from collections using the [Items](#) property of `SfDataForm`.

#### **C#**

```
dataForm.Items.RemoveAt(2);
```



Name

Password

Phone

StateName

Select a State ▼

Save

#### *Dynamically clear custom dataform items*

Support has been provided to dynamically clear the dataform items using the [Items](#) property of `SfDataForm`.

#### **C#**

```
dataForm.Items.Clear();
```



Name

Password

Phone

*Dynamically reset custom dataform items*

Support has been provided to reset the dataform items using the [Items](#) property of **SfDataForm**.

**C#**

```
var item = dataForm.Items[2];
item = new DataFormNumericUpDownItem() { Name = "Age", Editor =
"NumericUpDown" };
dataForm.Items[2] = item;
```

*Dynamically add custom dataform group items*

Support has been provided to dynamically add custom group items using [Items](#) property of **SfDataForm**.

**C#**

```
DataFormGroupItem dataFormGroupItem = new DataFormGroupItem();
dataFormGroupItem.GroupName = "GroupItem";
dataFormGroupItem.IsExpanded = true;
dataFormGroupItem.DataFormItems = new DataFormItems();
dataFormGroupItem.DataFormItems.Add(new DataFormTextItem() { Name = "First
Name", Editor = "Text", GroupName = "GroupItem" });
dataFormGroupItem.DataFormItems.Add(new DataFormTextItem() { Name = "Middle
Name", Editor = "Text", GroupName = "GroupItem" });
dataFormGroupItem.DataFormItems.Add(new DataFormTextItem() { Name = "Last
Name", Editor = "Text", GroupName = "GroupItem" });
dataForm.Items.Add(dataFormGroupItem);
```

The screenshot shows a SfDataForm control with a single group item named "Name". The group is expanded, showing three text input fields: "First Name", "Middle Name", and "Last Name". Each field has a corresponding text input control.

**Editing**

The data form supports several built-in editors.

*Supported editors and associated DataFormItem*

Editor name	Editor class	Data Type/Attribute	Input control loaded
-------------	--------------	---------------------	----------------------

Text	<a href="#">DataFormTextEditor</a>	The String type property and any other type apart from the following specified cases.	[Entry](https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.entry?view=xamarin-forms)
Multiline Text	<a href="#">DataFormMultiLineTextEditor</a>	The String type property with multiline text. [DataType(DataType.MultilineText)]	<a href="#">Editor</a>
Numeric	<a href="#">DataFormNumericEditor</a>	The property of Int, Double, Float, Decimal, Long types and also its nullable property.	<a href="#">SfNumericTextBox</a>
Percent	<a href="#">DataFormNumericEditor</a>	The property of Int, Double, Float, Decimal, Long types and also its nullable property with [DataType(DataType.Percent)] attribute.	{{' <a href="#">SfNumericTextBox</a> '   markdownify }}
Currency	{{' <a href="#">DataFormNumericEditor</a> '   markdownify }}	The property of Int, Double, Float, Decimal, Long types and also its nullable property with [DataType(DataType.Currency)] attribute.	[SfNumericTextBox](https://help.syncfusion.com/xamarin/sfnumerictextbox/overview)
Date	<a href="#">DataFormDateTimeEditor</a>	The DateTime type property and the property with [DataType(DataType.Date)] and [DataType(DataType.DateTime)] attributes.	<a href="#">DatePicker</a>
Time	<a href="#">DataFormTimeEditor</a>	The property with [DataType(DataType.Time)] attribute.	<a href="#">TimePicker</a>
Numeric UpDown	<a href="#">DataFormNumericUpDownEditor</a>	Int or Double type property.	<a href="#">SfNumericUpDown</a>
Segment	<a href="#">DataFormSegmentedEditor</a>	Enum type property.	<a href="#">SfSegmentedControl</a>
Bool	<a href="#">DataFormCheckBoxEditor</a>	Bool type property.	<a href="#">SfCheckBox</a>

Switch	<a href="#">DataFormSwitchEditor</a>	Bool type property.	<a href="#">Switch</a>
Picker	<a href="#">DataFormPickerEditor</a>	Enum and List type property. [EnumDataTypeAttribute]	<a href="#">Picker</a>
DropDown	<a href="#">DataFormDropDownEditor</a>	Enum and List type property.[EnumDataTypeAttribute]	<a href="#">SfComboBox</a>
AutoComplete	<a href="#">DataFormAutoCompleteEditor</a>	Enum and List type property.[EnumDataTypeAttribute]	<a href="#">SfAutoComplete</a>
Password	<a href="#">DataFormPasswordEditor</a>	The String type property and property with [DataType(DataType.Password)] attribute.	{{'Entry'   markdownify }}
RadioGroup	<a href="#">DataFormRadioGroupEditor</a>	Enum and List type property. [EnumDataTypeAttribute]	<a href="#">SfRadioGroup</a>
MaskedEditText	<a href="#">DataFormMaskedEditTextEditor</a>	The property with [DataType(DataType.PhoneNumber)] attribute.	<a href="#">SfMaskedEdit</a>

### Changing editor for type

By default, the editors will be loaded based on the previous table. To change the editor for any type, use the [RegisterEditor](#) method and specify the type and editor.

#### C#

```
dataForm.RegisterEditor(typeof(int), "NumericUpDown");
```

Here, the **NumericUpDown** editor will be loaded for the integer type instead of numeric editor.

### Changing editor for property

To change the editor for any property, use the [RegisterEditor](#) method and specify the property name and editor.

#### C#

```
dataForm.RegisterEditor("IsAvailable", "Switch");
```

Here, the Switch editor will be loaded for the `IsAvailable` property (bool type) instead of `CheckBox` editor.

#### Customizing existing editor

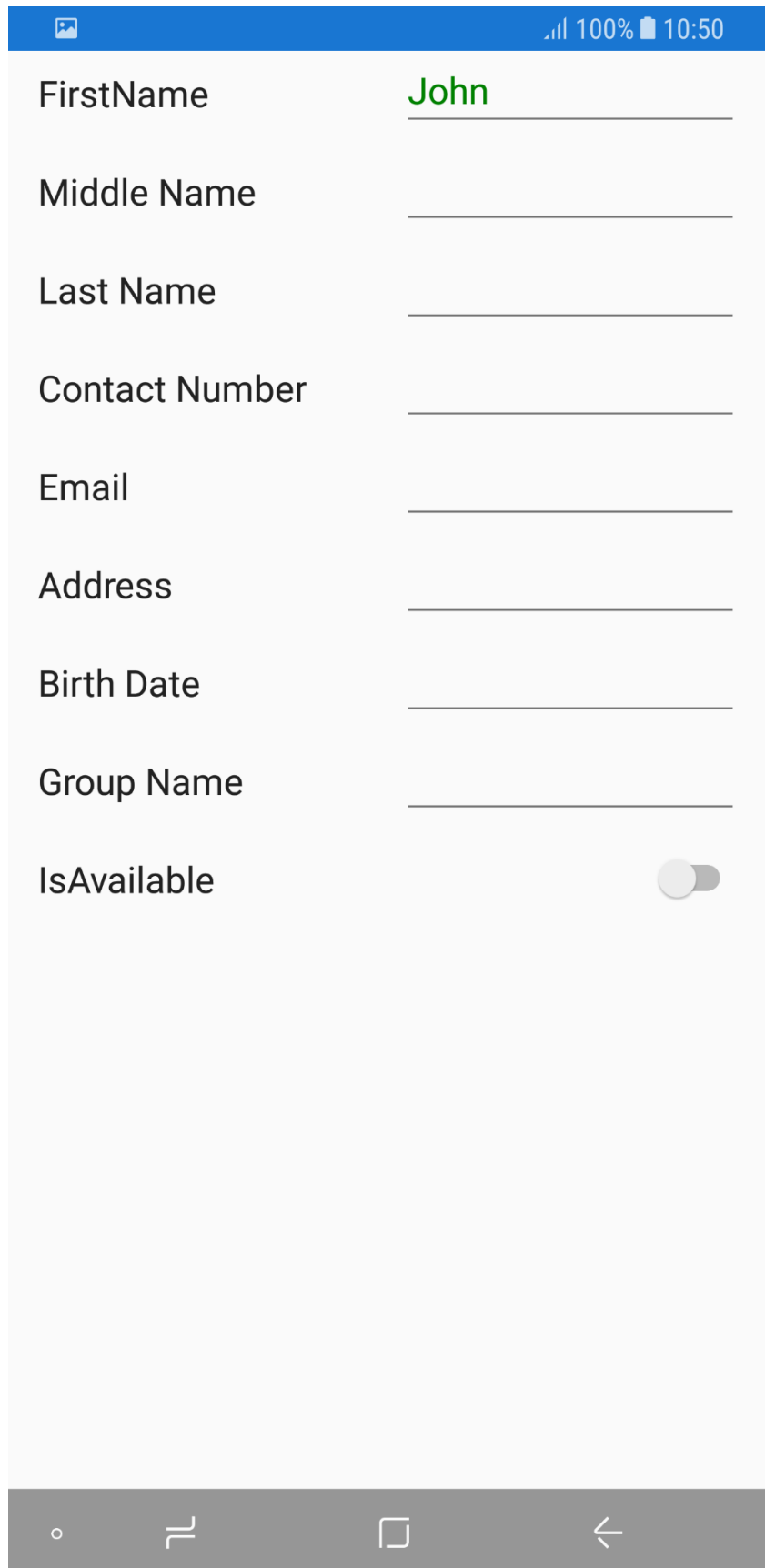
The existing editors defined in the previous table can be customized by overriding the default editors.

Here, the [DataFormTextEditor](#) is customized to set different foreground for the `FirstName` property text editor.

#### C#

```
public class CustomTextEditor : DataFormTextEditor
{
    public CustomTextEditor(SfDataForm dataForm) : base(dataForm)
    {
    }
    protected override void OnInitializeView(DataFormItem dataFormItem, Entry view)
    {
        if (dataFormItem.Name == "FirstName")
            view.TextColor = Color.Green;
        base.OnInitializeView(dataFormItem, view);
    }
}
dataForm.RegisterEditor("Text", new CustomTextEditor(dataForm));
```





First Name **John**

Middle Name

Last Name

Contact Number

Email

Address

Birth Date

Group Name

IsAvailable ☐

### Creating new custom editor

Create the custom editor by overriding the [DataFormEditor](#) class.

Property settings, commit, data validation can be handled by overriding the required methods. Here, the `Entry` is loaded for `Age` editor.

### C#

```
public class CustomTextEditor : DataFormEditor<Entry>
{
    public CustomTextEditor(SfDataForm dataForm) : base(dataForm)
    {
    }
    protected override Entry OnCreateEditorView(DataFormItem dataFormItem)
    {
        return new Entry();
    }
    protected override void OnInitializeView(DataFormItem dataFormItem, Entry view)
    {
        base.OnInitializeView(dataFormItem, view);
        view.Keyboard = Keyboard.Numeric;
        this.OnUpdateValue(dataFormItem, view);
        this.OnUpdateReadOnly(dataFormItem, view);
    }
    protected override void OnWireEvents(Entry view)
    {
        view.TextChanged += OnViewTextChanged;
        view.Focused += OnViewFocused;
        view.Unfocused += OnViewUnfocused;
    }
    private void OnViewPropertyChanged(object sender, PropertyChangedEventArgs e)
    {
        OnValidateValue(sender as Entry);
    }
    private void OnViewFocused(object sender, FocusEventArgs e)
    {
        var view = (sender as Entry);
        view.TextColor = Color.Green;
    }
    protected override bool OnValidateValue(Entry view)
    {
        return this.DataForm.Validate("Age");
    }
    private void OnViewUnfocused(object sender, FocusEventArgs e)
    {
        var view = sender as Entry;
        view.TextColor = Color.Red;
        if (this.DataForm.CommitMode == Syncfusion.XForms.DataForm.CommitMode.LostFocus ||
            this.DataForm.ValidationMode == ValidationMode.LostFocus)
            this.OnValidateValue(view);
        if (this.DataForm.CommitMode != Syncfusion.XForms.DataForm.CommitMode.LostFocus) return;
        this.OnCommitValue(view);
        OnValidateValue(sender as Entry);
    }
}
```

```


}
private void OnViewTextChanged(object sender, TextChangedEventArgs e)
{
    var view = sender as Entry;
    if (DataForm.CommitMode ==
        Syncfusion.XForms.DataForm.CommitMode.PropertyChanged ||
        DataForm.ValidationMode == ValidationMode.PropertyChanged)
    {
        this.OnValidateValue(view);
        if (this.DataForm.CommitMode !=
            Syncfusion.XForms.DataForm.CommitMode.PropertyChanged) return;
        this.OnCommitValue(view);
    }
    protected override void OnCommitValue(Entry view)
    {
        var dataFormItemView = view.Parent as DataFormItemView;
        this.DataForm.ItemManager.SetValue(dataFormItemView.DataFormItem,
            view.Text);
    }
    protected override void OnUpdateValue(DataFormItem dataFormItem, Entry view)
    {
        var cellValue = this.DataForm.ItemManager.GetValue(dataFormItem);
        if (cellValue != null && view.Text == cellValue.ToString())
            return;
        view.Text = cellValue == null ? string.Empty : cellValue.ToString();
    }
    protected override void OnUpdateReadOnly(DataFormItem dataFormItem, Entry
        view)
    {
        base.OnUpdateReadOnly(dataFormItem, view);
    }
    protected override void OnUnWireEvents(Entry view)
    {
        view.TextChanged -= OnViewTextChanged;
        view.Focused -= OnViewFocused;
        view.Unfocused -= OnViewUnfocused;
    }
}
dataForm.RegisterEditor("numeric", new CustomTextEditor(dataForm));
dataForm.RegisterEditor("Age", "numeric");
dataForm.ValidationMode = ValidationMode.LostFocus;

```

You should manually commit the custom DataFormItem editor value by using [OnCommitValue](#) override method of [DataFormEditor](#) class on custom editor [Value](#) or [Focus changed](#) event which is used to update the custom editor value in respective property in [DataObject](#) based on dataform [commit mode](#) set.

Also, you should manually validate the custom editor value in by using [OnValidateValue](#) override method of [DataFormEditor](#) class on custom editor [Value](#) or [Focus changed](#) event which is used to validate the custom editor value based on data form [validation mode](#) set. In the override method for [OnValidateValue](#), you need to return [DataForm.Validate\(string\)](#) method in order to validate the particular data item.

Also, you should manually update the value to the custom editor by using [OnUpdateValue](#) override method of [DataFormEditor](#) class on custom editor while bound the value from the Model class.

 97% 9:21 AM

FirstName	<u>John</u>
LastName	<u>Peter</u>
Email	<u></u>
Age	<u>20</u>

### Support for Email editor

You can load the Email editor by changing **Keyboard** type in the [AutoGeneratingDataFormItem](#) event.

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem != null && e.DataFormItem.Name == "Email")  
        (e.DataFormItem as DataFormTextItem).Keyboard = Keyboard.Email;  
}
```

The screenshot displays a mobile application interface with a blue status bar at the top showing a camera icon, signal strength, 100% battery, and the time 10:59. Below the status bar is a form with the following fields:

- FirstName: John
- Middle Name: (empty)
- Last Name: (empty)
- Contact Number: (empty)
- Email: john@ (with a pink underline)
- Address: (empty)
- Birth Date: (empty)
- Group Name: (empty)
- Salary: 0

At the bottom of the screen is a virtual keyboard with the following layout:

- Top row: gmail, yahoo, hotmail, >
- Row 1: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
- Row 2: q, w, e, r, t, y, u, i, o, p
- Row 3: a, s, d, f, g, h, j, k, l
- Row 4: ↑, z, x, c, v, b, n, m, ↵
- Row 5: !@#, ⚙️, @, EN(US), ., .com, Next
- Bottom row: ⇐, ⇐, ⇐, ⇐

### Commit mode

The [CommitMode](#) determines when the value should be committed to the data object.

The supported commit modes are as follows:

- `LostFocus`
- `PropertyChanged`
- `Explicit`

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-
namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
  <ContentPage.Content>
    <dataForm:SfDataForm x:Name="dataForm" CommitMode="LostFocus"/>
  </ContentPage.Content>
</ContentPage>
```

### C#

```
dataForm.CommitMode = CommitMode.LostFocus;
```

#### *LostFocus*

If the commit mode is `LostFocus`, the value is committed when the editor lost its focus.

#### *PropertyChanged*

The value will be committed immediately when it is changed.

#### *Explicit*

The value should be committed manually by calling the [SfDataForm.Commit](#) or [SfDataForm.Commit\(propertyName\)](#) method.

The following code commits the value of all the properties in the data object:

### C#

```
dataForm.Commit();
```

To commit the specific property value, pass the property name as argument.

### C#

```
dataForm.Commit("Name");
```

### Update editor value based on another editor

You can the update the editor value by using the [SfDataForm.UpdateEditor](#) method at runtime.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-
namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm"/>
</ContentPage.Content>
</ContentPage>
```

## C#

```
var expenseInfo = new ExpenseInfo();
expenseInfo.PropertyChanged += ExpenseInfo_PropertyChanged;
dataForm.DataObject = expenseInfo;
private void ExpenseInfo_PropertyChanged(object sender,
PropertyChangedEventArgs e)
{
    if (e.PropertyName == "Budget" || e.PropertyName == "Expense")
    {
        var item = sender as ExpenseInfo;
        item.Balance = item.Budget - item.Expense;
        dataForm.UpdateEditor("Balance");
    }
}
```

Here, the Balance property value is updated based on Budget and Expense properties. For updating value in editor, the `UpdateEditor` method is called.

You can download the sample from [here](#).

## Converter

To show the original value in different format or as different value, use the [Converter](#) attribute.

### *Changing original value of the DataForm property value using converter*

Here, the original value is multiplied by 10 and shown in editor. While committing, it is divided by 10 and stored in the data object.

## C#

```
public class ValueConverterExt : IPropertyValueConverter
{
    public object Convert(object value)
    {
        var amount = double.Parse(value.ToString());
        return amount * 10;
    }
    public object ConvertBack(object value)
    {
        var amount = double.Parse(value.ToString());
        return amount / 10;
    }
}
```



```
private double? amount = 1000;
[Converter(typeof(ValueConverterExt))]
public double? Amount
{
    get
    {
        return amount;
    }
    set
    {
        amount = value;
        RaisePropertyChanged("Amount");
    }
}
```

#### *Using date editor for DateTimeOffset DataForm property data type*

In SfDataForm, you cannot use date editor for DateTimeOffset property data type. To overcome this, you need to use Converter attribute to convert DateTimeOffset to DateTime value and vice-versa.

#### **C#**

```
private DateTimeOffset displayDate;
[Converter(typeof(ValueConverterExt))]
public DateTimeOffset DisplayDate
{
    get
    {
        return displayDate;
    }
    set
    {
        displayDate = value;
    }
}

public class ValueConverterExt : IPropertyValueConverter
{
    public object Convert(object value)
    {
        DateTime baseTime = new DateTime(2008, 6, 19, 7, 0, 0);
        DateTime targetTime;
        var dateTimeOffset = (DateTimeOffset)value;
        dateTimeOffset = new DateTimeOffset(baseTime,
            TimeZoneInfo.Local.GetUtcOffset(baseTime));
        targetTime = dateTimeOffset.DateTime;
        return targetTime;
    }
    public object ConvertBack(object value)
    {
        var dateTime = (DateTime)value;
        dateTime = new DateTime(2008, 6, 19, 7, 0, 0);
        dateTime = DateTime.SpecifyKind(dateTime, DateTimeKind.Local);
        DateTimeOffset dateTimeOffset = dateTime;
        return dateTimeOffset;
    }
}
```

You can download the source code of this demo from here [DateTimeOffsetConverter](#)

### Disable editing

You can disable editing by setting the [IsReadOnly](#) property of the data form.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm" IsReadOnly="True"/>
</ContentPage.Content>
</ContentPage>
```

### C#

```
dataForm.IsReadOnly = true;
```

You can also change the editing behavior by setting the [IsReadOnly](#) property of the [DataFormItem](#).

### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "Salary")
            e.DataFormItem.IsReadOnly = true;
    }
}
```

You can also change the editing behavior at runtime.

### C#

```
private void Button_Click(object sender, System.EventArgs e)
{
    var dataFormItem = dataForm.ItemManager.DataFormItems["FirstName"];
    dataFormItem.IsReadOnly = true;
}
```

**Note:** [DataFormItem.IsReadOnly](#) takes higher priority than [SfDataForm.IsReadOnly](#).

### Two-way data binding

When the DataForm business object properties are updated with two-way data binding support, the value will sync with underlying DataForm editors.

To enable two-way data binding support, set the value of [NotifyPropertyChanges](#) property to true in DataForm.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm ="clr-
namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm" NotifyPropertyChanges="True"/>
</ContentPage.Content>
</ContentPage>
```

### C#

```
dataForm.NotifyPropertyChanges = true;
```

You can download the entire source code of this demo from here [Two-wayDataBinding](#)

### Editors

The data form supports several built-in editors as follows:

Editor name	Editor class	Supported Data Type/Attribute	Input control loaded
Text	<a href="#">DataFormTextEditor</a>	The String type property and any other type apart from the below specified cases.	[Entry](https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.entry?view=xamarin-forms)
Multi lineText	<a href="#">DataFormMultiLineTextEditor</a>	The String type property with multi line text.[DataType(DataType.MultilineText)]	<a href="#">Editor</a>

Num eric	[DataFormNumericEditor](https://help.syncfusion.com/cr/cref_files/xamarin/Syncfusion.SfDataForm.XForms~Syncfusion.XForms.DataForm.Editors.DataFormNumericEditor.html)	The property of Int, Double, Float, Decimal, Long types and also its nullable property.	[SfNumericTextBox](https://help.syncfusion.com/xamarin/sfnumerictextbox/overview)
Perc ent	{{' <a href="#">DataFormNumericEditor</a> '   markdownify }}	The property of Int, Double, Float, Decimal, Long types and also its nullable property with [DataType(â€œPercentâ€•)] attribute.	{{' <a href="#">SfNumericTextBox</a> '   markdownify }}
Curr ency	[DataFormNumericEditor](https://help.syncfusion.com/cr/cref_files/xamarin/Syncfusion.SfDataForm.XForms~Syncfusion.XForms.DataForm.Editors.DataFormNumericEditor.html)	The property of Int, Double, Float, Decimal, Long types and also its nullable property with [DataType(DataType.Currency)] attribute.	[SfNumericTextBox](https://help.syncfusion.com/xamarin/sfnumerictextbox/overview)
Date	<a href="#">DataFormDateEditor</a>	DateTime type property and the property with [DataType(DataType.Date)] and	<a href="#">DatePicker</a>

		[DataType(DataType.DateTime)] attributes.	
Time	<a href="#">DataFormTimeEditor</a>	The property with [DataType(DataType.Time)] attribute.	<a href="#">TimePicker</a>
NumericUpDown	<a href="#">DataFormNumericUpDownEditor</a>	Int or Double type property.	<a href="#">SfNumericUpDown</a>
Segment	<a href="#">DataFormSegmentedEditor</a>	Enum type property.	<a href="#">SfSegmentedControl</a>
Bool	<a href="#">DataFormCheckBoxEditor</a>	Bool type property.	<a href="#">SfCheckBox</a>
Switch	<a href="#">DataFormSwitchEditor</a>	Bool type property.	<a href="#">Switch</a>
Picker	<a href="#">DataFormPickerEditor</a>	Enum and List type property. [EnumDataTypeAttribute]	<a href="#">Picker</a>
DropDown	<a href="#">DataFormDropDownEditor</a>	Enum and List type property. [EnumDataTypeAttribute]	<a href="#">SfComboBox</a>
Password	<a href="#">DataFormPasswordEditor</a>	The String type property and property with [DataType(DataType.Password)] attribute.	{{'Entry'  markdownify }}

Radi oGro up	<a href="#">DataFormRadioGroupEditor</a>	Enum and List type property.[En umDataType Attribute]	<a href="#">SfRadioGroup</a>
Mask edEd itTex t	<a href="#">DataFormMaskedEditTextEditor</a>	The property with [DataType(D ataType.Pho neNumber)] attribute.	<a href="#">SfMaskedEdit</a>
Auto Com plete	<a href="#">DataFormAutoCompleteEditor</a>	Enum and List type property.[En umDataType Attribute]	<a href="#">SfAutoComplete</a>

#### Text editor

In the text editor, the [Entry](#) is loaded.

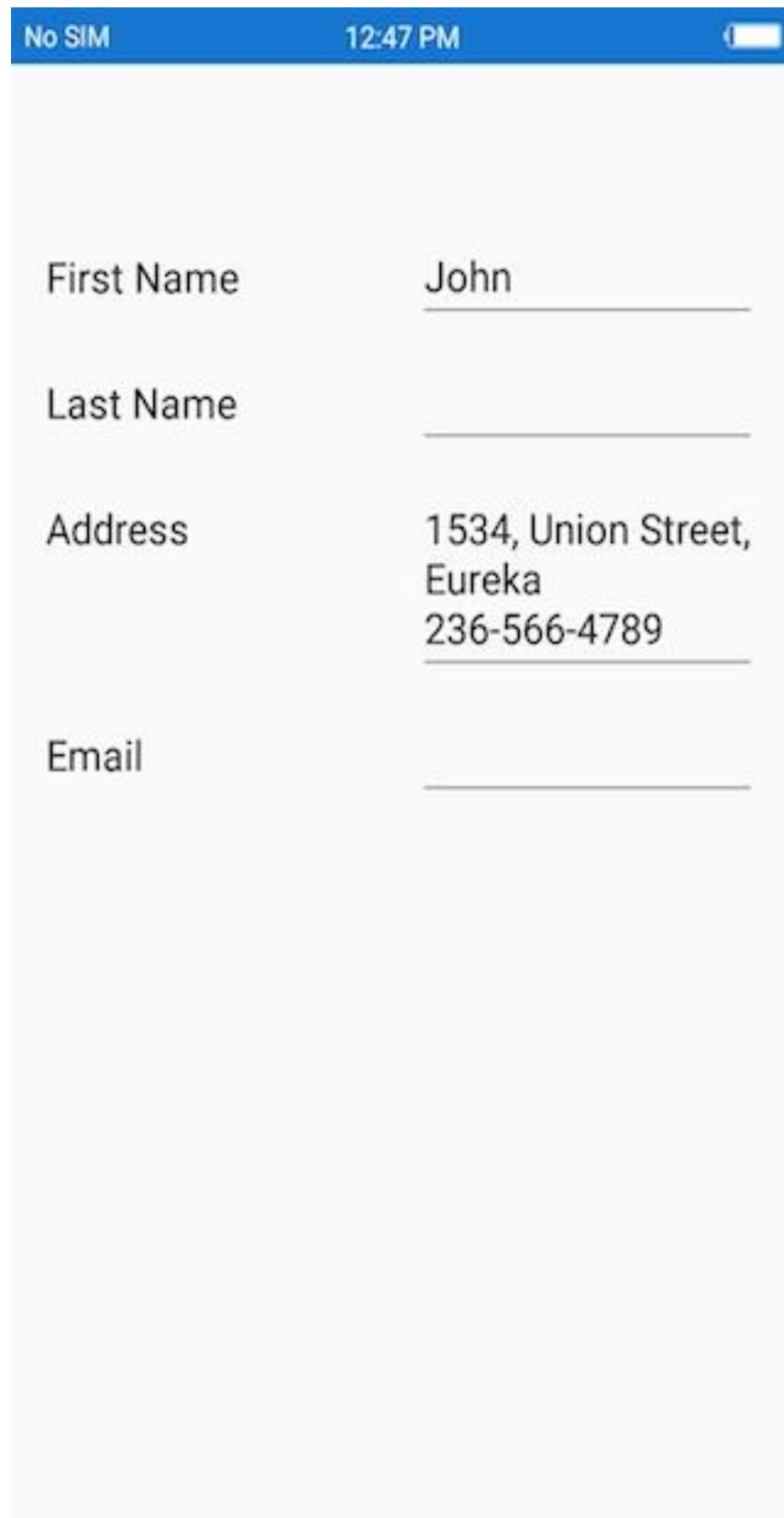
#### Multiline Text editor

In the `MultilineText` editor, the [Editor](#) is loaded.

And `MultilineText` editor height will auto expand/reduce based on the line wraps in editor , which allowing text to be readable without scrolling the editor.

#### C#

```
[DataType(DataType.MultilineText)]
public String Address { get; set; }
```



The image shows a mobile application interface with a blue status bar at the top. The status bar contains the text 'No SIM' on the left, '12:47 PM' in the center, and a battery icon on the right. Below the status bar is a light gray form area. The form contains four labeled text input fields. The first field is labeled 'First Name' and contains the text 'John'. The second field is labeled 'Last Name' and is empty. The third field is labeled 'Address' and contains the text '1534, Union Street, Eureka' followed by '236-566-4789' on the next line. The fourth field is labeled 'Email' and is empty. Each field has a thin gray underline.

First Name	John
Last Name	
Address	1534, Union Street, Eureka 236-566-4789
Email	

## Numeric editor

In the numeric editor, the [SfNumericTextBox](#) is loaded.

### *Change maximum NumberDecimalDigits in the numeric editor*

In the [SfNumericTextBox](#), two decimal digits will be displayed by default. You can change the number of decimal digits being displayed by setting the [MaximumNumberDecimalDigits](#) property in the [DataFormNumericItem](#).

## C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Amount")
    {
        (e.DataFormItem as DataFormNumericItem).MaximumNumberDecimalDigits = 3;
    }
}
```

## Date editor

In the date editor, the [DatePicker](#) will be loaded.

### *Setting null value in date editor*

In [DatePicker](#), the default date value (1/01/0001) is displayed by default. You can also set the null value by adding nullable [DateTime](#) data type for the date picker property in data form, which allows you to set the null value and display the empty value in date editor.

## C#

```
[DataType(DataType.Date)]
[Display(Name = "Birth Date")]
public DateTime? BirthDate { get; set; }
```





The image shows a mobile application interface for a data form. At the top is a blue header bar containing a checkmark icon on the left and status information (signal strength, 100% battery, and 5:44 PM) on the right. Below the header, the form consists of five labeled input fields, each with a horizontal line for text entry: "First Name", "Last Name", "Address", "Birth Date", and "Email". The form is set against a light gray background. At the bottom, there is a light gray navigation bar with four icons: a dot, a home icon, a square, and a back arrow.

*Customizing format in date editor*

In the `DatePicker`, short date will be shown by default. You can change the applied format by setting the `Format` property in `DataFormDateItem`.





**C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Date")
    {
        (e.DataFormItem as DataFormDateItem).Format = "ddd, MMM d, yyyy";
    }
}
```



100% 17:07

Total Amount	<u>1,000.00</u>
Discount	<u>\$10.00</u>
Date	<u>Wed, Feb 28, 2018</u>
Expense	<u>Enter expense</u>
Balance	<u></u>
Name	<u>Home</u>
ItemName	<u></u>
Time	<u>2/28/2018</u>
IsBillable	<input type="checkbox"/>

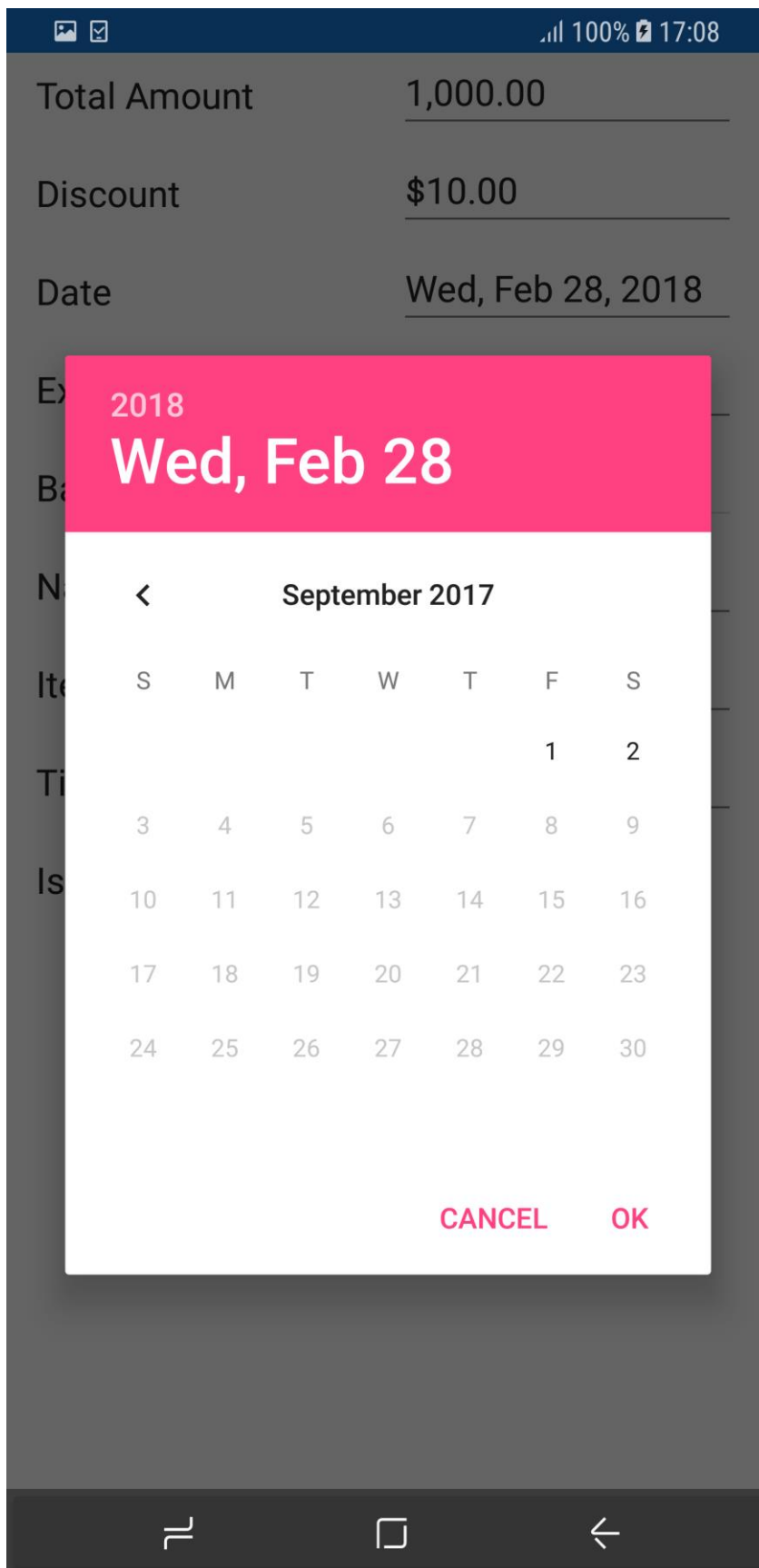


*Setting MaximumDate and MinimumDate in date editor*

You can customize the maximum and minimum allowable dates in the `DatePicker` by setting [MaximumDate](#) and [MinimumDate](#) in the [DataFormDateItem](#) respectively.

**C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Date")
    {
        (e.DataFormItem as DataFormDateItem).MinimumDate = new DateTime(2017, 5, 5);
        (e.DataFormItem as DataFormDateItem).MaximumDate = new DateTime(2017, 9, 2);
    }
}
```



### Time editor

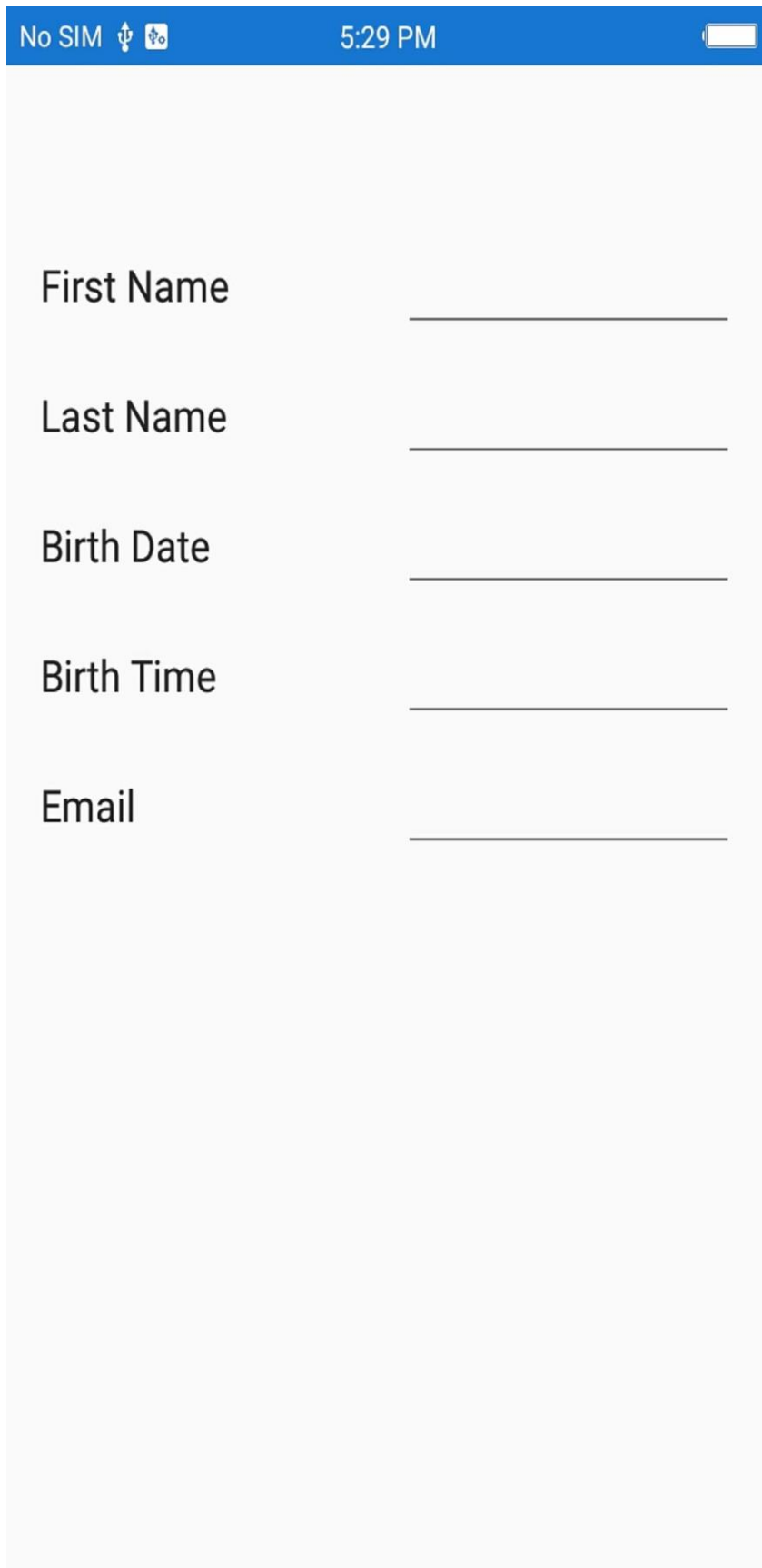
In the time editor, the [TimePicker](#) will be loaded.

#### Setting null value in time editor




In `TimePicker`, the default time value (12:00 AM) is displayed by default. You can also set the null value by adding nullable `DateTime` data type for the time picker property in data form, which allows you to set the null value and display the empty value in time editor.

#### C#

```
[DataType(DataType.Time)]  
[Display(Name = "Birth Time")]  
public DateTime? BirthTime { get; set; }
```



The image shows a mobile application interface with a blue status bar at the top. The status bar contains the text 'No SIM' followed by two small icons (a USB symbol and a SIM card symbol), the time '5:29 PM', and a battery level icon. Below the status bar is a light gray background. On this background, there are five text labels, each followed by a horizontal input line: 'First Name', 'Last Name', 'Birth Date', 'Birth Time', and 'Email'.

No SIM   5:29 PM 

First Name \_\_\_\_\_

Last Name \_\_\_\_\_

Birth Date \_\_\_\_\_

Birth Time \_\_\_\_\_

Email \_\_\_\_\_



### Customizing format in time editor


In the `TimePicker`, short time will be shown by default. You can change the applied format by setting the `Format` property in `DataFormTimeItem`.

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem != null && e.DataFormItem.Name == "BirthTime")  
        (e.DataFormItem as DataFormTimeItem).Format = "HH:mm";  
}
```



No SIM  

5:31 PM 

First Name	<input type="text" value="John"/>
Last Name	<input type="text" value="Peter"/>
Birth Date	<input type="text" value="1/13/1950"/>
Birth Time	<input type="text" value="17:30"/>
Email	<input type="text"/>

### Segment editor

In segment editor, the [SfSegmentedControl](#) is loaded, and DataForm Segment editor supports to enum and List data type properties.

To add Segment editor in DataForm, register the editor as Segment for the required property using the [RegisterEditor](#) method.

### Support for enum data type

By default, the [ItemsSource](#) for [SfSegmentedControl](#) is auto-generated for enum data types, if the property has been registered for Segment editor.

#### C#

```
dataForm.RegisterEditor("SaveTo", "Segment");
private Location saveTo;
[Display(Name = "Save To")]
public Location SaveTo
{
    get { return saveTo; }
    set { this.saveTo = value; }
}
public enum Location
{
    Sim,
    Phone
}
```

### Customizing ItemsSource of SfSegmentedControl

For List data types, you can set the [ItemsSource](#) for [SfSegmentedControl](#) by using the [SourceProvider](#) or using [ItemsSource](#) property of [DataFormSegmentItem](#).

### Using SourceProvider

#### C#

```
private string saveTo;
[Display(Name = "Save To")]
public string SaveTo
{
    get { return saveTo; }
    set { this.saveTo = value; }
}
public class SourceProviderExt : SourceProvider
{
    public override IList GetSource(string sourceName)
    {
        var list = new List<string>();
        if(sourceName == "SaveTo")
        {
            list.Add("Sim");
            list.Add("Phone");
        }
        return list;
    }
}
```


```
dataForm.RegisterEditor("SaveTo", "Segment");  
dataForm.SourceProvider = new SourceProviderExt();
```

### Using ItemsSource property of DataFormSegmentItem

You can also set `ItemsSource` for segment editor by using the [ItemsSource](#) property in the [DataFormSegmentItem](#).

#### C#

```
dataForm.RegisterEditor("SaveTo", "Segment");  
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem != null && e.DataFormItem.Name == "SaveTo")  
    {  
        var list = new List<string>();  
        list.Add("Sim");  
        list.Add("Phone");  
        (e.DataFormItem as DataFormSegmentItem).ItemsSource = list;  
    }  
}
```

 100% 9:19 AM

First Name

John

Last Name

Peter

Email


Contact Number


34752


Save To


Sim

Phone







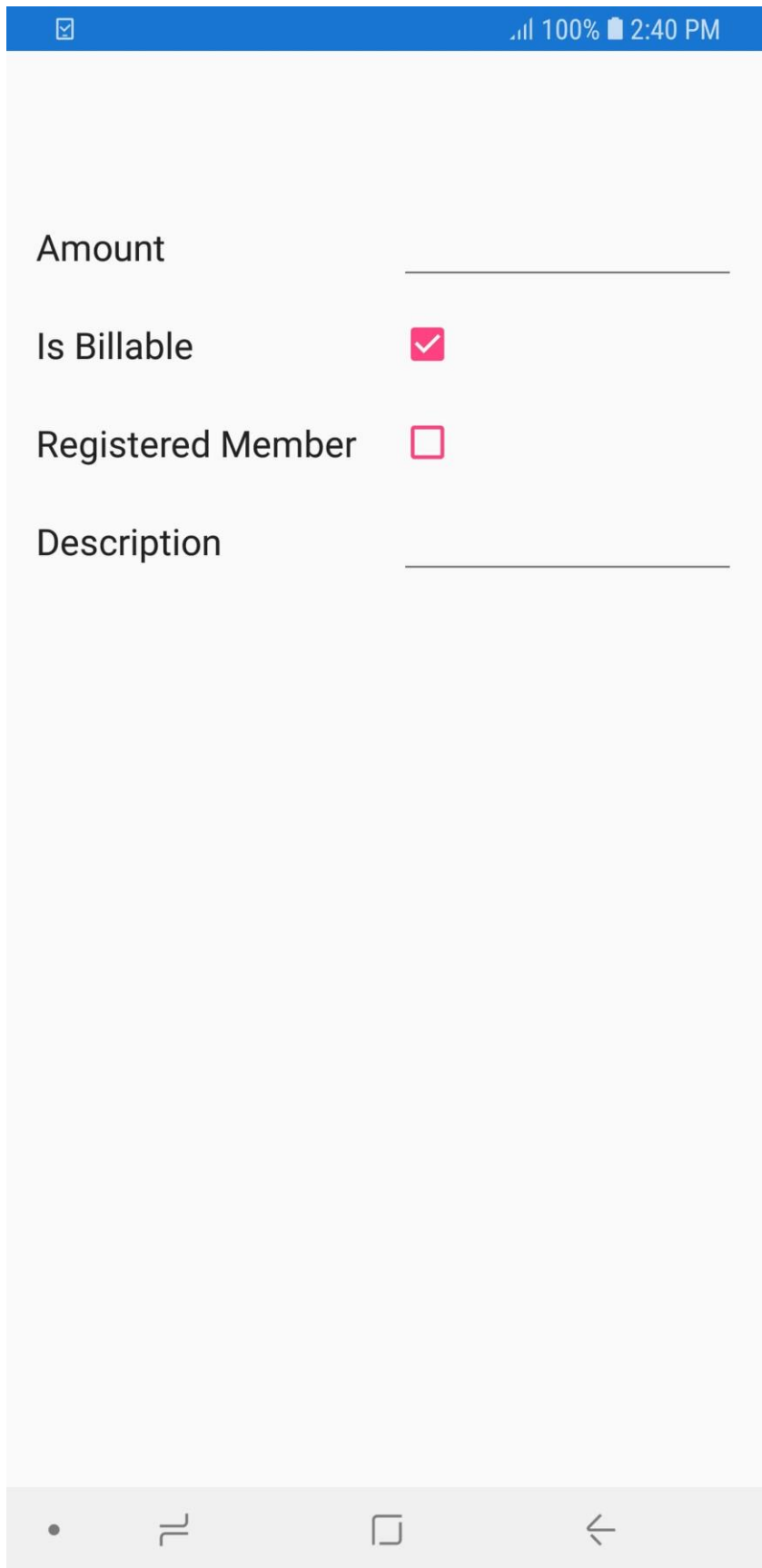


### CheckBox editor

In **CheckBox** editor, the [SfCheckBox](#) control is loaded. By default, for **bool** data type property, the **CheckBox** editor will be loaded in data form.

#### **C#**

```
[Display(Name = "Is Billable")]  
public bool IsBillable { get; set; } = true;  
[Display(Name = "Registered Member")]  
public bool RegisteredMember { get; set; }
```



The screenshot displays a mobile application interface with a blue status bar at the top. The status bar contains a checkmark icon, signal strength bars, 100% battery, and the time 2:40 PM. The main content area is a light gray form with four fields:

- Amount:** A text input field with a horizontal line below it.
- Is Billable:** A checkbox with a pink checkmark, indicating it is selected.
- Registered Member:** A checkbox with a pink outline, indicating it is not selected.
- Description:** A text input field with a horizontal line below it.

At the bottom of the screen is a white navigation bar with four icons: a dot, a double underline, a square, and a left arrow.

### Visual states of SfCheckBox

SfCheckBox support three visual states.

- Checked
- UnChecked
- Intermediate

### C#

```
[Display(Name = "Is Billable")]  
public bool IsBillable { get; set; } = true;  
[Display(Name = "Registered Member")]  
public bool RegisteredMember { get; set; } = false;  
[Display( Name = "Is Refundable")]  
public bool? IsRefundable { get; set; } = null;
```

Amount

Is Billable ☒

Registered Member ☐

Is Refundable ☒

Description

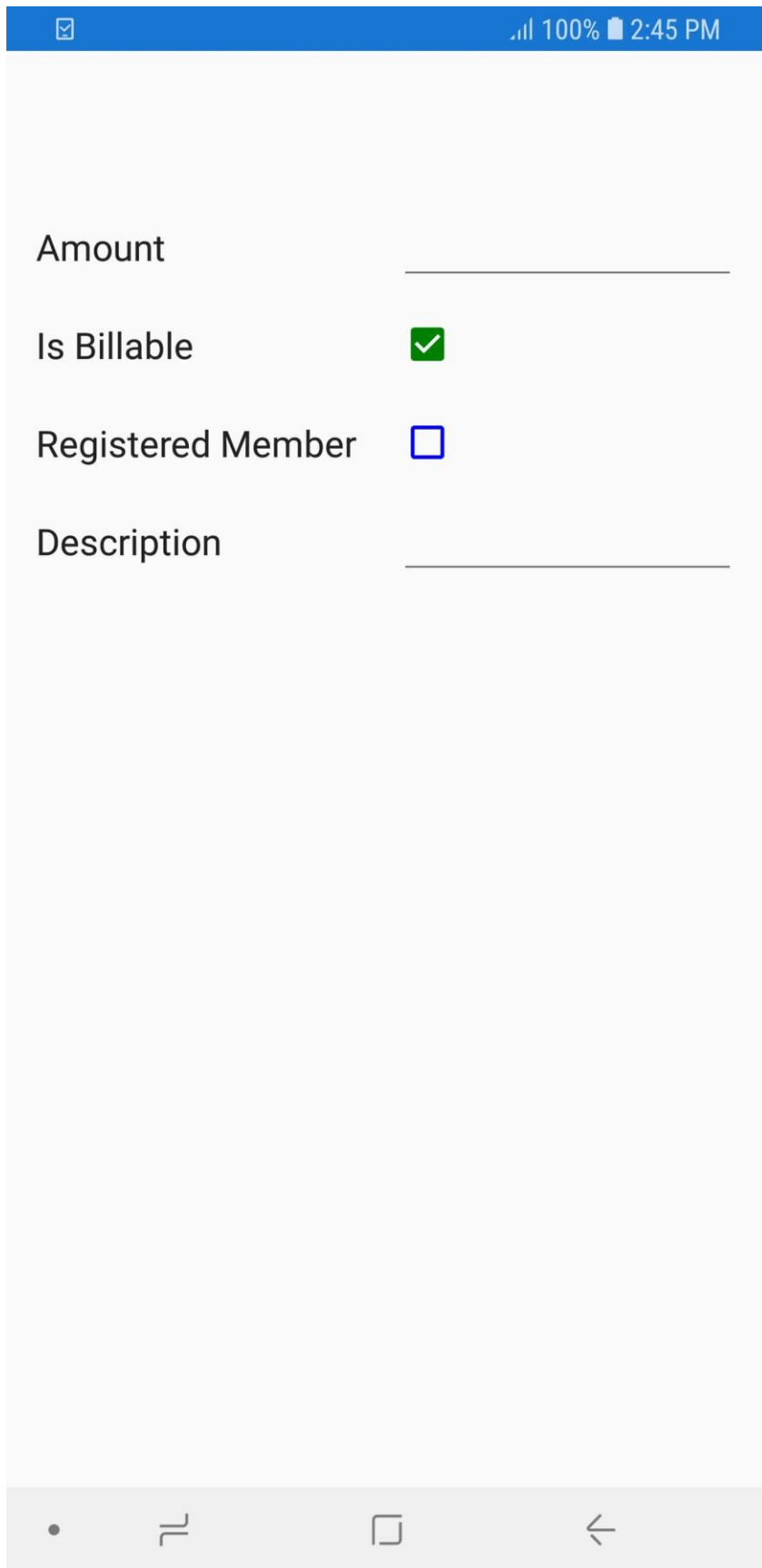


### Customizing the SfCheckBox state color

You can change the color of the checked/indeterminate state of SfCheckBox using the [CheckedColor](#) property and change the color of unchecked state using the [UnCheckedColor](#) property in [DataFormCheckBoxItem](#) in the [AutoGeneratingDataFormItem](#) event of data form.

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "IsBillable")
        {
            (e.DataFormItem as DataFormCheckBoxItem).CheckedColor = Color.Green;
        }
        if (e.DataFormItem.Name == "RegisteredMember")
        {
            (e.DataFormItem as DataFormCheckBoxItem).UnCheckedColor = Color.Blue;
        }
    }
}
```



The screenshot displays a mobile application interface with a blue header bar. The header bar contains a checkmark icon on the left and status information '100%' and '2:45 PM' on the right. The main content area is white and contains four form fields. The first field is labeled 'Amount' and has a horizontal line for input. The second field is labeled 'Is Billable' and has a green checkmark icon. The third field is labeled 'Registered Member' and has a blue square checkbox icon. The fourth field is labeled 'Description' and has a horizontal line for input. At the bottom of the screen, there is a grey bar with four icons: a dot, a double line, a square, and a left arrow.

Amount

Is Billable ☒

Registered Member ☐


Description

### Setting SfCheckBox caption text

You can customize the appearance of display text in SfCheckBox using the Text property of the DataFormCheckBoxItem.

#### C#

```
[DisplayOptions(ShowLabel = false)]
public bool Agree { get; set; }
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name.Equals("Agree"))
        {
            (e.DataFormItem as DataFormCheckBoxItem).IsThreeState = false;
            (e.DataFormItem as DataFormCheckBoxItem).Text = "I agree to the Terms &
Conditions";
        }
    }
}
```

 100% 3:09 PM

Amount

Is Billable

☒

Registered Member

☐

Is Refundable

☐

Description

☐ I agree to the Terms & Conditions

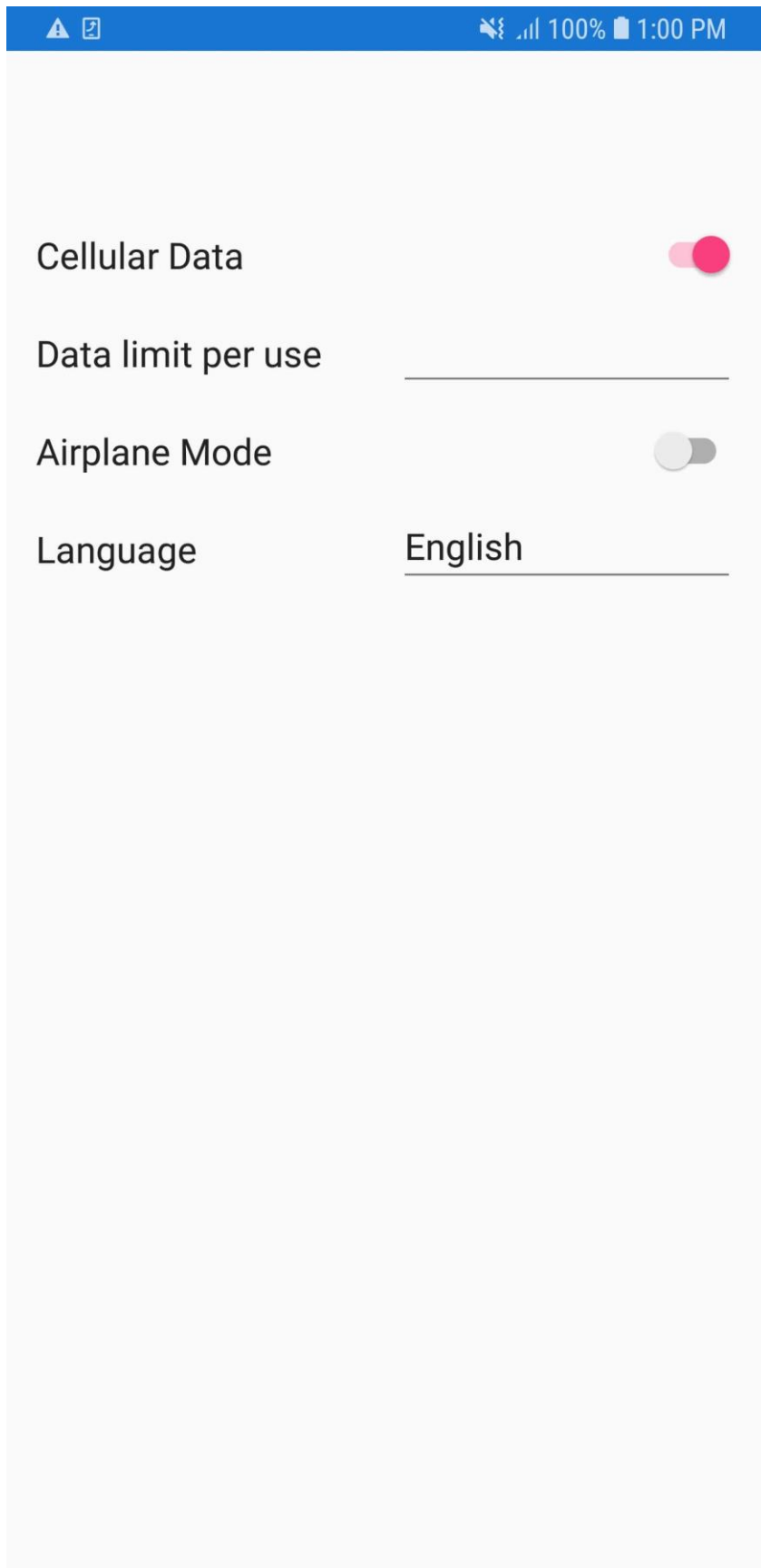
### Switch Editor

In switch editor, [Switch](#) is loaded, and DataForm **Switch** editor supports bool data type property.

To add **Switch** editor in DataForm, register the editor as **Switch** for the required property using the [RegisterEditor](#) method.

#### C#

```
dataForm.RegisterEditor("CellularData", "Switch");
dataForm.RegisterEditor("AirplaneMode", "Switch");
[Display(Name = "Cellular Data")]
public bool CellularData { get; set; } = true;
[Display(Name = "Airplane Mode")]
public bool AirplaneMode { get; set; }
```



A mockup of a mobile settings screen. At the top is a blue status bar with icons for a warning, a square, cellular signal, 100% battery, and the time 1:00 PM. Below the status bar is a light gray settings area. It contains four items: 'Cellular Data' with a red toggle switch turned on; 'Data limit per use' with a horizontal line below it; 'Airplane Mode' with a gray toggle switch turned off; and 'Language' with 'English' selected and a horizontal line below it.

Cellular Data ☒

Data limit per use \_\_\_\_\_

Airplane Mode ☐

Language English

### Drop down editor

In the drop down editor, the [SfComboBox](#) will be loaded.

#### *Customizing ItemsSource of SfComboBox*

By default, the `ItemsSource` for `SfComboBox` is auto-generated for enum types and collection type properties. For other types, you can set the `ItemsSource` by using the [SourceProvider](#).

#### Using SourceProvider

##### **C#**

```
private string _ItemName;
public string ItemName
{
    get
    {
        return _ItemName;
    }
    set
    {
        _ItemName = value;
    }
}

public class SourceProviderExt : SourceProvider
{
    public override IList GetSource(string sourceName)
    {
        var list = new List<string>();
        if (sourceName == "ItemName")
        {
            list.Add("Item1");
            list.Add("Item2");
            list.Add("Item3");
        }
        return list;
    }
}

dataForm.SourceProvider = new SourceProviderExt();
dataForm.RegisterEditor("ItemName", "DropDown");
```

#### Using ItemsSource property


You can also set the `ItemsSource` for drop down editor by using the [ItemsSource](#) property in the `DataFormDropDownItem`.

##### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Name")
    {
        var list = new List<string>();
        list.Add("Home");
        list.Add("Food");
        list.Add("Utilities");
        list.Add("Education");
    }
}
```

```
(e.DataFormItem as DataFormDropDownItem).ItemsSource = list;  
}  
}
```



 100% 12:34 AM

Name	<input type="text" value="John"/>
Expense	<input type="text" value="10089"/>
Balance	<input type="text" value="1,492"/>
Item Name	<div><div>▼</div><div><div>Item1</div><div>Item2</div><div>Item3</div></div></div>

*Changing ItemsSource of SfComboBox at run time*

You can also change the **ItemsSource** at runtime.

**C#**

```
private void Button_Click(object sender, EventArgs e)
{
    var dataFormItem = dataForm.ItemManager.DataFormItems["Name"];
    if (dataFormItem.Name == "Name")
    {
        var list = new List<string>();
        list.Add("Home");
        list.Add("Food");
        list.Add("Utilities");
        list.Add("Education");
        (dataFormItem as DataFormDropDownItem).ItemsSource = list;
    }
}
```

*Loading complex type property values in drop down editor*

You can display the complex type property values in drop down editor by using the [GetSource](#) override method of **SourceProvider** class, which is used to get source list as complex property values for drop down editor and set it to **SourceProvider** property of **SfDataForm**. You need to use **AutoGeneratingDataFormItem** event to set [DisplayMemberPath](#) and [SelectedValuePath](#) property value of **DataFormDropDownItem** for complex type property.

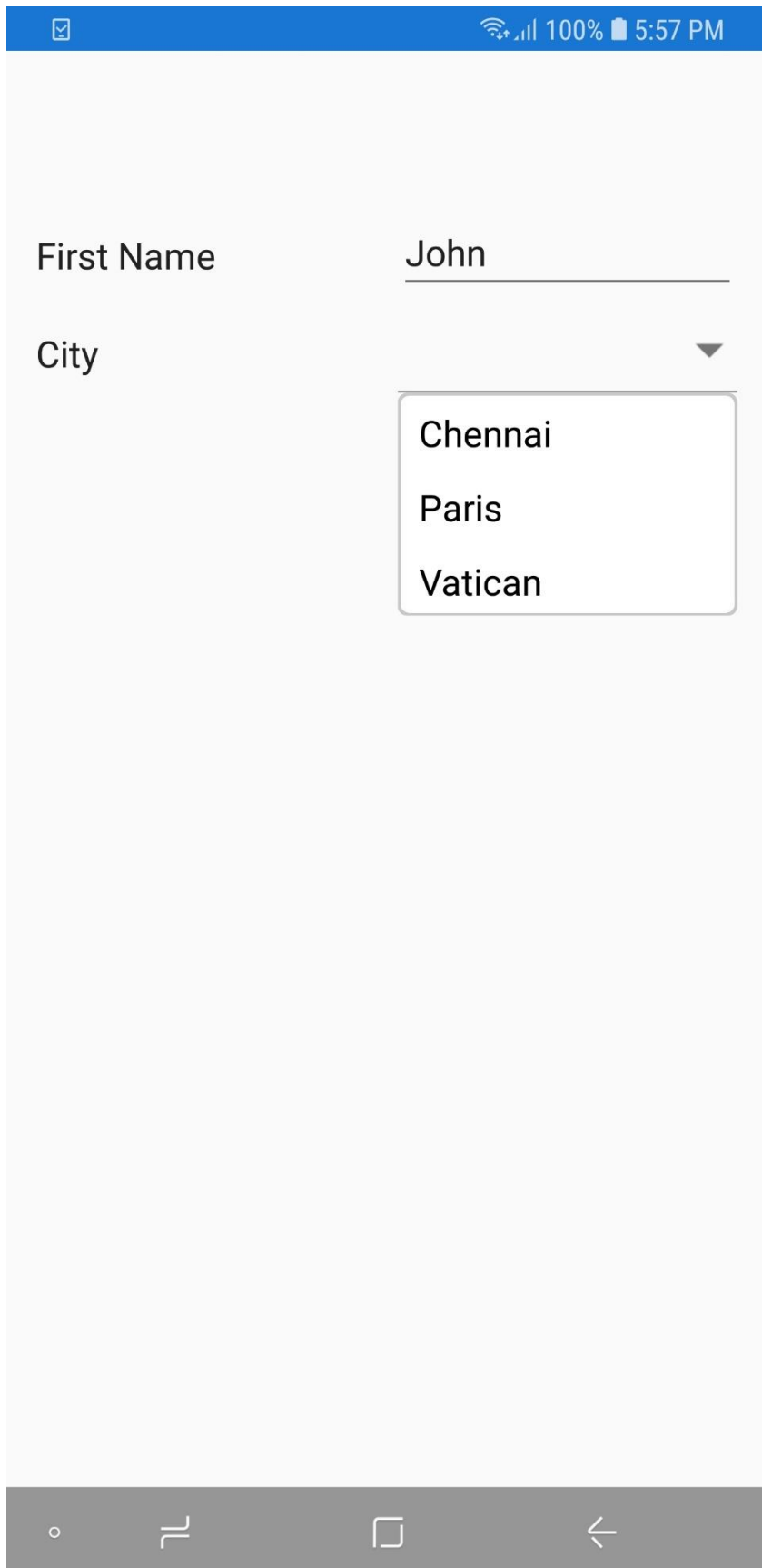
**Note:** Class cannot be directly set as data type for drop down editor in this complex type scenario.

**C#**

```
dataForm.SourceProvider = new SourceProviderExt();
dataForm.DataObject = new ContactInfo();
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
dataForm.RegisterEditor("City", "DropDown");
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "City")
    {
        if (Device.RuntimePlatform != Device.UWP)
        {
            (e.DataFormItem as DataFormDropDownItem).DisplayMemberPath = "City";
            (e.DataFormItem as DataFormDropDownItem).SelectedValuePath = "PostalCode";
        }
    }
}

public class SourceProviderExt : SourceProvider
{
    public override IList GetSource(string sourceName)
    {
        if (sourceName == "City")
        {
            List<Address> details = new List<Address>();
            details.Add(new Address() { City = "Chennai", PostalCode = 1 });
            details.Add(new Address() { City = "Paris", PostalCode = 2 });
            details.Add(new Address() { City = "Vatican", PostalCode = 3 });
        }
    }
}
```

```
return details;
}
return new List<string>();
}
}
public class ContactInfo
{
    [Display(Name = "First Name")]
    public String FirstName { get; set; }
    public string City { get; set; }
}
public class Address
{
    public int PostalCode { get; set; }
    public string City { get; set; }
}
```



The screenshot displays a mobile application interface with a blue status bar at the top. The status bar contains a checkmark icon, signal strength indicators, 100% battery, and the time 5:57 PM. The main content area is light gray and contains two form fields. The first field is labeled 'First Name' and has the text 'John' entered. The second field is labeled 'City' and has a dropdown menu open, showing three options: 'Chennai', 'Paris', and 'Vatican'. The dropdown menu is a white box with a gray border. At the bottom of the screen is a dark gray navigation bar with four icons: a circle, a square, a square, and a left arrow.

First Name John

City

- Chennai
- Paris
- Vatican

### Picker editor

In the picker editor, the [Picker](#) will be loaded.

#### *Changing title in the Picker*

You can show some text in the **Picker** popup by using the [Title](#) property in the [DataFormPickerItem](#).

#### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Name")
    {
        (e.DataFormItem as DataFormPickerItem).Title = "Expense Category";
    }
}
```

The screenshot displays an Android application interface for managing expenses. The background form contains the following fields:

- Total Amount: 1,000.00
- Discount: \$10.00
- Date: Wed, Feb 28, 2018
- Expense: Enter expense
- Balance: (empty)

An "Expense Category" modal dialog is overlaid on the form. It features a title bar, a list of categories, and two action buttons at the bottom:

- Home
- Entertainment
- CANCEL
- OK

The bottom of the screen shows the standard Android navigation bar with icons for back, home, and recent apps.

### Customizing ItemsSource of Picker

By default, the `ItemsSource` for picker is auto-generated for enum type and collection type properties. For other types, you can set the `ItemsSource` by using [SourceProvider](#).

#### Using SourceProvider

##### C#

```
private string _ItemName;
public string ItemName
{
    get
    {
        return _ItemName;
    }
    set
    {
        _ItemName = value;
    }
}

public class SourceProviderExt : SourceProvider
{
    public override IList GetSource(string sourceName)
    {
        var list = new List<string>();
        if (sourceName == "ItemName")
        {
            list.Add("Item1");
            list.Add("Item2");
            list.Add("Item3");
        }
        return list;
    }
}

dataForm.SourceProvider = new SourceProviderExt();
dataForm.RegisterEditor("ItemName", "Picker");
```

#### Using event

You can also set `ItemsSource` for picker editor by using the [ItemsSource](#) property in the [DataFormPickerItem](#).

##### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Name")
    {
        var list = new List<string>();
        list.Add("Home");
        list.Add("Food");
        list.Add("Utilities");
        list.Add("Education");
        (e.DataFormItem as DataFormPickerItem).ItemsSource = list;
    }
}
```

### *Changing ItemsSource of picker at run time*

You can also change the `ItemsSource` at runtime.

#### **C#**

```
private void Button_Click(object sender, EventArgs e)
{
    var dataFormItem = dataForm.ItemManager.DataFormItems["Name"];
    if (dataFormItem.Name == "Name")
    {
        var list = new List<string>();
        list.Add("Home");
        list.Add("Food");
        list.Add("Utilities");
        list.Add("Education");
        (dataFormItem as DataFormPickerItem).ItemsSource = list;
    }
}
```

### *Loading complex type property values in picker*

You can display the complex type property values in picker editor by using the [GetSource](#) override method of `SourceProvider` class, which is used to get source list as complex property values for picker editor and set it to `SourceProvider` property of `SfDataForm`. You need to use

`AutoGeneratingDataFormItem` event to set [DisplayMemberPath](#) and [ValueMemberPath](#) property value `DataFormPickerItem` for complex type property.

---

**Note:** Class cannot be directly set as data type for picker editor in this complex type scenario.

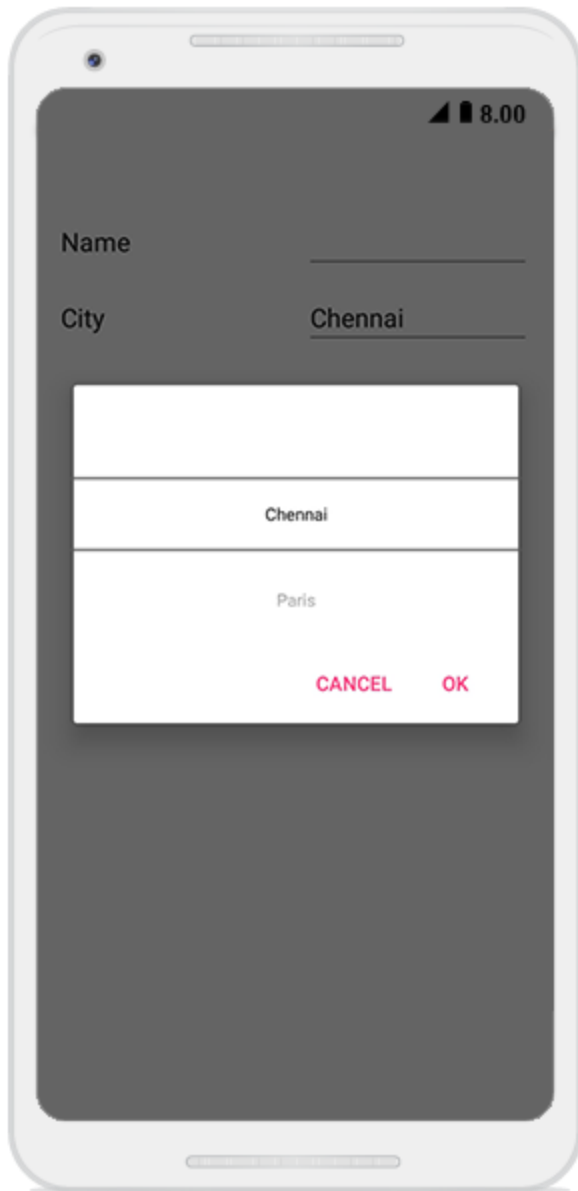
---

#### **C#**

```
dataForm.SourceProvider = new SourceProviderExt();
dataForm.RegisterEditor("City", "Picker");
dataForm.DataObject = new ContactInfo();
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "City")
    {
        if (Device.RuntimePlatform != Device.UWP)
        {
            (e.DataFormItem as DataFormPickerItem).DisplayMemberPath = "City";
            (e.DataFormItem as DataFormPickerItem).ValueMemberPath = "PostalCode";
        }
    }
}
public class SourceProviderExt : SourceProvider
{
    public override IList GetSource(string sourceName)
    {
        if (sourceName == "City")
        {
            List<Address> details = new List<Address>();
            details.Add(new Address() { City = "Chennai", PostalCode = 1 });
        }
    }
}
```



```
details.Add(new Address() { City = "Paris", PostalCode = 2 });
details.Add(new Address() { City = "Vatican", PostalCode = 3 });
return details;
}
return new List<string>();
}
}
public class ContactInfo
{
    public String FirstName { get; set; }
    public string City { get; set; }
}
public class Address
{
    public int PostalCode { get; set; }
    public string City { get; set; }
}
```



You can download the entire source code of this demo for Xamarin.Forms from here [DataFormPickerEditor](#)

---

**Note:** PickerEditor not supported in Xamarin.Forms.UWP.

---

NumericUpDown editor

In the numeric editor, the [SfNumericUpDown](#) will be loaded.



*Changing SpinButtonAlignment in NumericUpDown*

By default, up down button will be displayed in right side. You can change its alignment by using the [SpinButtonAlignment](#) property in the [DataFormNumericUpDownItem](#).

**C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
```

```
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Quantity")
        (e.DataFormItem as DataFormNumericUpDownItem).SpinButtonAlignment =
        SpinButtonAlignment.Both;
}
```



100% 17:06

Total Amount	<u>1,000.00</u>
Discount	<u>\$10.00</u>
Date	<u>Wed, Feb 28, 2018</u>
Expense	<u>Enter expense</u>
Balance	<u></u>
Name	<u>Home</u>
ItemName	<u></u>
Time	<u>2/28/2018</u>
IsBillable	<input type="checkbox"/>

### *Changing step value in numeric up down*

You can change the next increment and decrement values by using the [StepValue](#) property in the [DataFormNumericUpDownItem](#). The default value of step value is 1.

#### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Quantity")
        (e.DataFormItem as DataFormNumericUpDownItem).StepValue = 2;
}
```

---

**Note:** [StepValue](#) property not supported in Xamarin.FormsUWP.

---

### *Setting Maximum and Minimum value in numeric up down*

You can set minimum and maximum values for numeric up down by using [Minimum](#) and [Maximum](#) properties values respectively.

#### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Quantity")
    {
        (e.DataFormItem as DataFormNumericUpDownItem).Maximum = 100;
        (e.DataFormItem as DataFormNumericUpDownItem).Minimum = 0;
    }
}
```

### *Enabling auto reverse in numeric up down*

In the [SfNumericUpDown](#), once maximum and minimum values reached, the value will be unchanged. You can enable the cyclic behavior by setting the [AutoReverse](#) to [true](#) in the [DataFormNumericUpDownItem](#).

#### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Quantity")
    {
        (e.DataFormItem as DataFormNumericUpDownItem).AutoReverse = true;
    }
}
```



### *Changing CultureInfo in numeric up down and numeric text box*

You can change the culture in [SfNumericTextBox](#) and [SfNumericUpDown](#) by using the [CultureInfo](#) property in the [DataFormNumericItemBase](#).

SfNumericTextBox





**C#**

```
private double _discount = 10;
[DataType(DataType.Currency)]
public double Discount
{
    get
    {
        return _discount;
    }
    set
    {
        _discount = value;
        RaisePropertyChanged("Discount");
    }
}
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Discount")
    {
        (e.DataFormItem as DataFormNumericUpDownItem).CultureInfo = new
System.Globalization.CultureInfo("fr-FR");
    }
}
```



100% 17:17

Total Amount	<u>1,000.00</u>
Discount	<u>10,00 €</u>
Date	<u>Wed, Feb 28, 2018</u>
Expense	<u>Enter expense</u>
Balance	<u></u>
Name	<u>Home</u>
ItemName	<u></u>
Time	<u>2/28/2018</u>
IsBillable	<input type="checkbox"/>





## SfNumericUpDown

**C#**





```
dataForm.RegisterEditor("Discount", "NumericUpDown");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Discount")
    {
        (e.DataFormItem as DataFormNumericUpDownItem).FormatString = "c";
        (e.DataFormItem as DataFormNumericUpDownItem).CultureInfo = new
        System.Globalization.CultureInfo("fr-FR");
    }
}
```





100% 17:21

Total Amount	<u>1,000.00</u>
Discount	<u>- 10,00 € +</u>
Date	<u>Wed, Feb 28, 2018</u>
Expense	<u>Enter expense</u>
Balance	<u></u>
Name	<u>Home</u>
ItemName	<u></u>
Time	<u>2/28/2018</u>
IsBillable	<input type="checkbox"/>



## Password editor

In the password editor, the [Entry](#) is loaded.

### C#

```
private string password;
[Display(ShortName = "Transaction password", Prompt = "Enter password")]
[DataType(DataType.Password)]
public string Password
{
    get { return this.password; }
    set
    {
        this.password = value;
        RaisePropertyChanged("Password");
        this.RaiseErrorChanged("Password");
    }
}
```

100% 17:02

123456789

Re-enter account number

123456789

Account type

Savings

Name







Syncfusion

Amount

10,000.00

Transaction password

.....







1234567890

qwertyuiop

asdfghjkl

↑zxcvbnm↵

!#1, English (US) . Done



### RadioGroup editor

In the **RadioGroup** editor, the [SfRadioGroup](#) control is loaded.

The [items](#) for **SfRadioGroup** is generated for **enum** and **List** data type properties. In order to add **RadioGroup** editor in the DataForm, you need to register editor as **RadioGroup** for the required property by using the [RegisterEditor](#) method.

#### Support for enum data type

For **enum** data type property, **SfRadioGroup** [items](#) will be added based on specified property enum values.

#### C#

```
dataForm.RegisterEditor("Phone", "RadioGroup");
private Numbers phone;
public Numbers Phone
{
    get { return phone; }
    set { this.phone = value; }
}
public enum Numbers
{
    Home,
    Work,
    Other
}
```


#### Support for List data type

For **List** data type property, you have to set the **ItemsSource** by using the [SourceProvider](#), based on that **SfRadioGroup** [items](#) will be added.





#### C#

```
dataForm.RegisterEditor("Phone", "RadioGroup");
dataForm.SourceProvider = new SourceProviderExt();
private string phone;
public string Phone
{
    get { return phone; }
    set { this.phone = value; }
}
public class SourceProviderExt : SourceProvider
{
    public override IList GetSource(string sourceName)
    {
        var list = new List<string>();
        if(sourceName == "Phone")
        {
            list.Add("Home");
            list.Add("Work");
            list.Add("Other");
        }
        return list;
    }
}
```

```
}
```

 100% 9:22 AM

First Name	<input type="text" value="John"/>
Last Name	<input type="text" value="Peter"/>
Email	<input type="text"/>
Contact Number	<input type="text" value="34752"/>
Phone	<div><input checked="" type="radio"/> Home</div> <div><input type="radio"/> Work</div> <div><input type="radio"/> Other</div>



### MaskedEditText editor

In the MaskedEditText editor, the [SfMaskedEdit](#) control is loaded.

#### C#

```
[Display(Name = "Contact Number")]  
[DataType(DataType.PhoneNumber)]  
public string ContactNumber { get; set; }
```

### Setting the masked editor as int and double type

By default, the SfMaskedEdit includes prompt and literals along with your input value. The special characters are not allowed in int and double type, so you need to exclude prompt and literals using the [ValueMaskFormat](#) property in DataFormMaskedEditTextItem.

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
...  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem != null && e.DataFormItem.Name == "ContactNumber")  
    {  
        (e.DataFormItem as DataFormMaskedEditTextItem).ValueMaskFormat =  
        Syncfusion.XForms.MaskedEdit.MaskFormat.ExcludePromptAndLiterals;  
    }  
}
```

### Localizing special characters

The special symbols such as currency, date separator, decimal separator, and other symbols can be localized using the [CultureInfo](#) property of DataFormMaskedEditTextItem.

#### C#

```
(e.DataFormItem as DataFormMaskedEditTextItem).CultureInfo = new  
CultureInfo("fr-FR");
```

### Customizing the clipboard text

By default, when you perform cut or copy operation, the clipboard text will be included with prompt and literals along with your input value. You can modify this and allow the clipboard to hold the value with or without prompt and literals by setting the [CutCopyMaskFormat](#) property of the DataFormMaskedEditTextItem.

#### C#

```
(e.DataFormItem as DataFormMaskedEditTextItem).CutCopyMaskFormat =  
Syncfusion.XForms.MaskedEdit.MaskFormat.ExcludePromptAndLiterals;
```

### Mask and mask types

The mask and mask type of input can be customized using the [Mask](#) and [MaskType](#) properties of DataFormMaskedEditTextItem. Refer to this [link](#) to know more about the mask characters and mask types available in the masked editor.

**C#**

```
(e.DataFormItem as DataFormMaskedEditFormItem).Mask = @"+1\\(\\d{3}\\)\\d{6}";  
(e.DataFormItem as DataFormMaskedEditFormItem).MaskType =  
Syncfusion.XForms.MaskedEdit.MaskType.RegEx;
```


*Customizing prompt character*

The custom prompt character can be set using the [PromptChar](#) property of `DataFormMaskedEditFormItem`.

**C#**

```
(e.DataFormItem as DataFormMaskedEditFormItem).PromptChar = '#';
```



 100% 3:36 PM

First Name

Peter

Last Name

John

Contact Number


(347)52\_-\_\_


Email


Save To


Sim

Phone









## AutoComplete editor

In the autocomplete editor, the [SfAutoComplete](#) is loaded.

### *Customizing ItemsSource of autocomplete editor*

By default, the [ItemsSource](#) for **AutoComplete** editor is auto-generated for enum types. For other types, you can set [ItemsSource](#) using [SourceProvider](#).

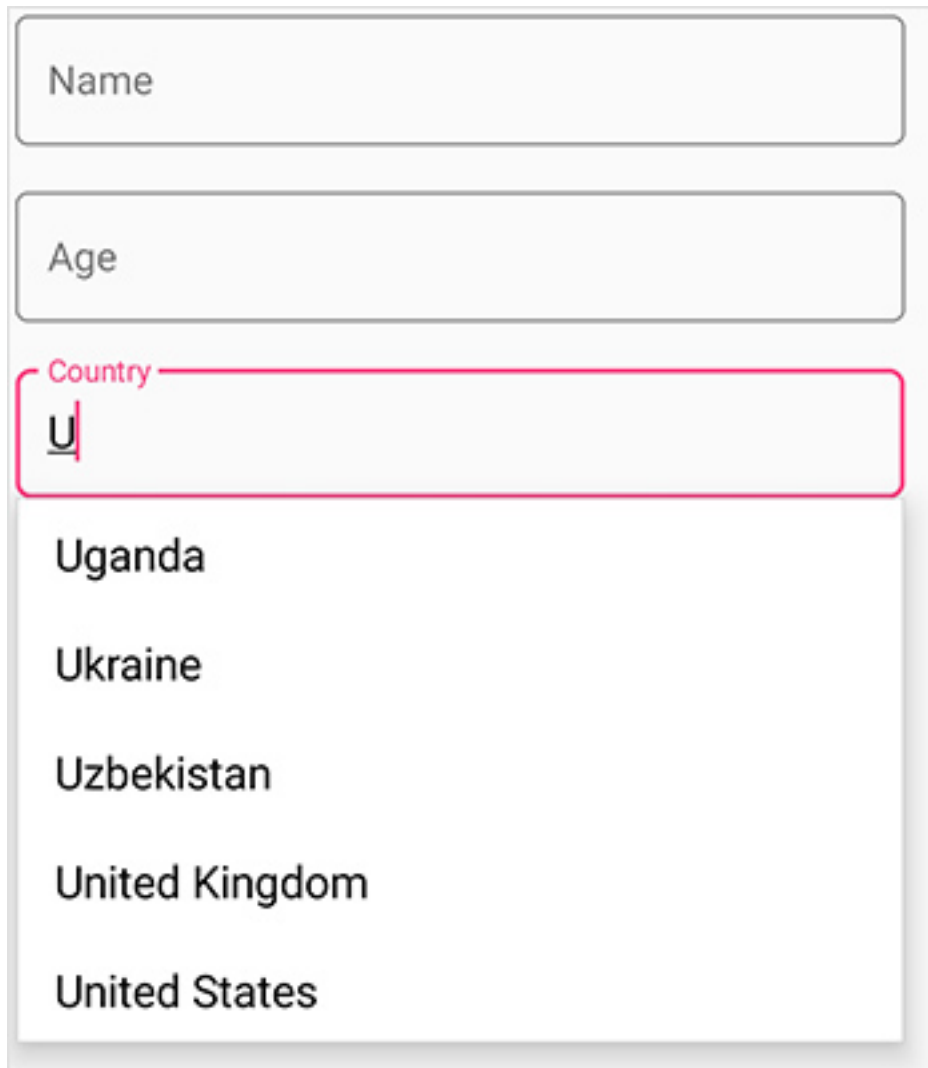
### Using SourceProvider

#### **C#**

```
private string country;
public string Country
{
    get
    {
        return country;
    }
    set
    {
        country = value;
    }
}

public class SourceProviderExt : SourceProvider
{
    public override IList GetSource(string sourceName)
    {
        var list = new List<string>();
        if (sourceName == "Country")
        {
            list.Add("Indonesia");
            list.Add("Italy");
            list.Add("India");
            list.Add("Iran");
            list.Add("Iraq");
            list.Add("Uganda");
            list.Add("Ukraine");
            list.Add("Canada");
            list.Add("Australia");
            list.Add("Uzbekistan");
            list.Add("France");
            list.Add("United Kingdom");
            list.Add("United States");
        }
        return list;
    }
}

dataForm.SourceProvider = new SourceProviderExt();
dataForm.RegisterEditor("Country", "AutoComplete");
```



The image shows a SfDataForm control with three input fields: 'Name', 'Age', and 'Country'. The 'Country' field is active, showing a dropdown list of suggestions. The suggestions are: Uganda, Ukraine, Uzbekistan, United Kingdom, and United States. The letter 'U' is entered in the 'Country' field, and a red border highlights the dropdown area.

Using [AutoGeneratingItem](#) event

You can also set `ItemsSource` for autocomplete editor by using [ItemsSource](#) property in the [DataFormAutoCompleteItem](#).

### C#

```
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var list = new List<string>();
        list.Add("Indonesia");
        list.Add("Italy");
        list.Add("India");
        list.Add("Iran");
        list.Add("Iraq");
        list.Add("Uganda");
        list.Add("Ukraine");
    }
}
```

```
list.Add("Canada");
list.Add("Australia");
list.Add("Uzbekistan");
list.Add("France");
list.Add("United Kingdom");
list.Add("United States");
(e.DataFormItem as DataFormAutoCompleteItem).ItemsSource = list;
}
```

Dynamically changing the ItemsSource of autocomplete editor

You can also change the [ItemsSource](#) at runtime.

### C#

```
private void Button_Click(object sender, EventArgs e)
{
    var dataFormItem = dataForm.ItemManager.DataFormItems["Country"];
    if (dataFormItem.Name == "Country")
    {
        var list = new List<string>();
        list.Add("Ukraine");
        list.Add("Canada");
        list.Add("Australia");
        list.Add("Uzbekistan");
        list.Add("France");
        list.Add("United Kingdom");
        list.Add("United States");
        (dataFormItem as DataFormAutoCompleteItem).ItemsSource = list;
    }
}
```

Loading complex type property values in autocomplete editor

You can display the complex type property values in autocomplete editor by using [GetSource](#) override method of the SourceProvider class, which is used to get source list as complex property values for autocomplete editor and set it to the SourceProvider property of SfDataForm. Use the [AutoGeneratingDataFormItem](#) event to set [DisplayMemberPath] and [SelectedValuePath] property values of AutoComplete for complex type property.

**Note:** Class cannot be directly set as data type for autocomplete editor in this complex type scenario.

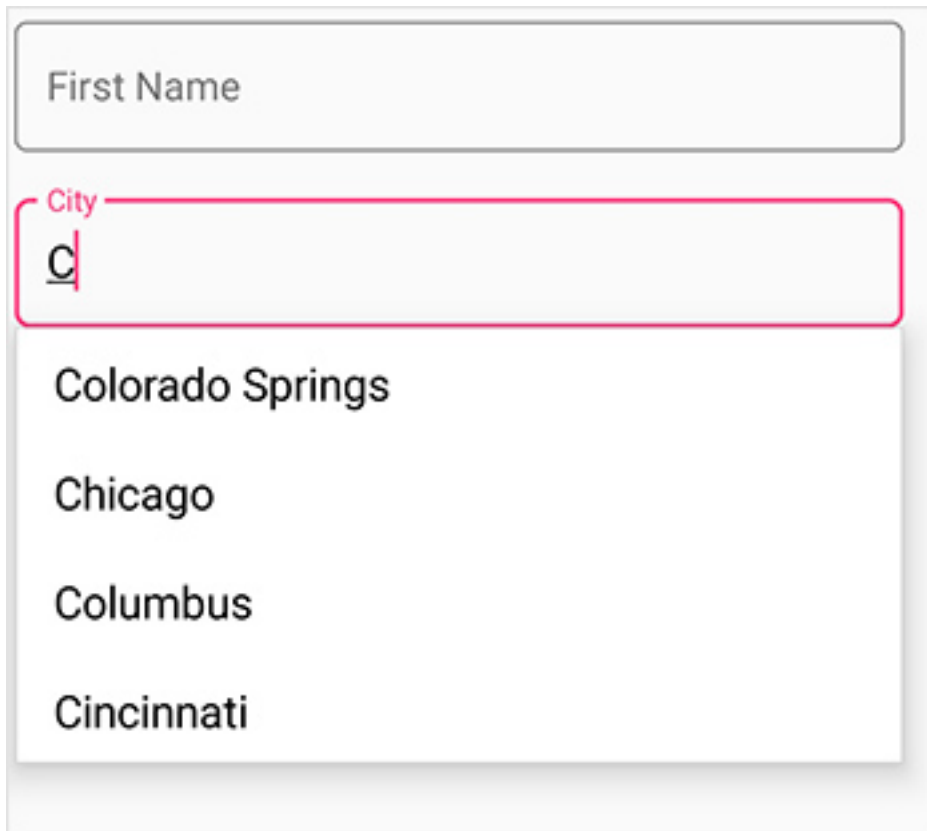
### C#

```
dataForm.SourceProvider = new SourceProviderExt();
dataForm.DataObject = new ContactInfo();
dataForm.RegisterEditor("City", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "City")
    {
        (e.DataFormItem as DataFormAutoCompleteItem).DisplayMemberPath = "City";
        (e.DataFormItem as DataFormAutoCompleteItem).SelectedValuePath =
"PostalCode";
    }
}
```

```
}
}
public class SourceProviderExt : SourceProvider
{
    public override IList GetSource(string sourceName)
    {
        if (sourceName == "City")
        {
            List<Address> details = new List<Address>();
            details.Add(new Address() { City = "Colorado Springs", PostalCode = 1 });
            details.Add(new Address() { City = "Chicago", PostalCode = 2 });
            details.Add(new Address() { City = "Columbus", PostalCode = 3 });
            details.Add(new Address() { City = "Portland", PostalCode = 4 });
            details.Add(new Address() { City = "Paris", PostalCode = 5 });
            details.Add(new Address() { City = "Las Vegas", PostalCode = 6 });
            details.Add(new Address() { City = "New York", PostalCode = 7 });
            details.Add(new Address() { City = "Cincinnati", PostalCode = 8 });
            details.Add(new Address() { City = "San Diego", PostalCode = 9 });
            return details;
        }
        return new List<string>();
    }
}

public class ContactInfo
{
    [Display(Name = "First Name")]
    public String FirstName { get; set; }
    public string City { get; set; }
}

public class Address
{
    public int PostalCode { get; set; }
    public string City { get; set; }
}
```



#### *Customizing the appearance of autocomplete editor*

##### *AutoComplete editor modes*

The [DataFormAutoCompleteEditor](#) provides the following three different ways to display the filtered suggestions.

- Suggest - Displays suggestions in drop-down list
- Append - Appends the first suggestion to text
- SuggestAppend - Performs both suggest and append.

The [AutoCompleteMode](#) property is used to choose the suggestion display mode in the [DataFormAutoCompleteItem](#) class. The default value is Suggest.

##### *Suggestion choices in list*

The filtered suggestions are displayed in a drop-down list. Users can pick an item from the list.

#### **C#**

```
dataForm.DataObject = new ContactInfo();
dataForm.SourceProvider = new SourceProviderExt();
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
```

```
autoCompleteItem.AutoCompleteMode = AutoCompleteMode.Suggest;
}
}
public class SourceProviderExt : SourceProvider
{
    public override IList GetSource(string sourceName)
    {
        var list = new List<string>();
        if (sourceName == "Country")
        {
            list.Add("Indonesia");
            list.Add("Italy");
            list.Add("India");
            list.Add("Iran");
            list.Add("Iraq");
            list.Add("Uganda");
            list.Add("Ukraine");
            list.Add("Canada");
            list.Add("Australia");
            list.Add("Uzbekistan");
            list.Add("France");
            list.Add("United Kingdom");
            list.Add("United States");
        }
        return list;
    }
}
```

The image shows a SfDataForm control with three input fields: 'Name', 'Age', and 'Country'. The 'Country' field is active, showing a dropdown list of suggestions. The suggestions are: Uganda, Ukraine, Uzbekistan, United Kingdom, and United States. The first suggestion, 'Uganda', is highlighted. The dropdown is open, and the text 'U' is visible in the input field.

#### [Appending suggestion to text](#)

The first item in the filtered suggestions is appended to autocomplete editor text. In this mode, the dropdown remains closed.

#### **C#**

```
dataForm.DataObject = new ContactInfo();
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
        autoCompleteItem.AutoCompleteMode = AutoCompleteMode.Append;
    }
}
```



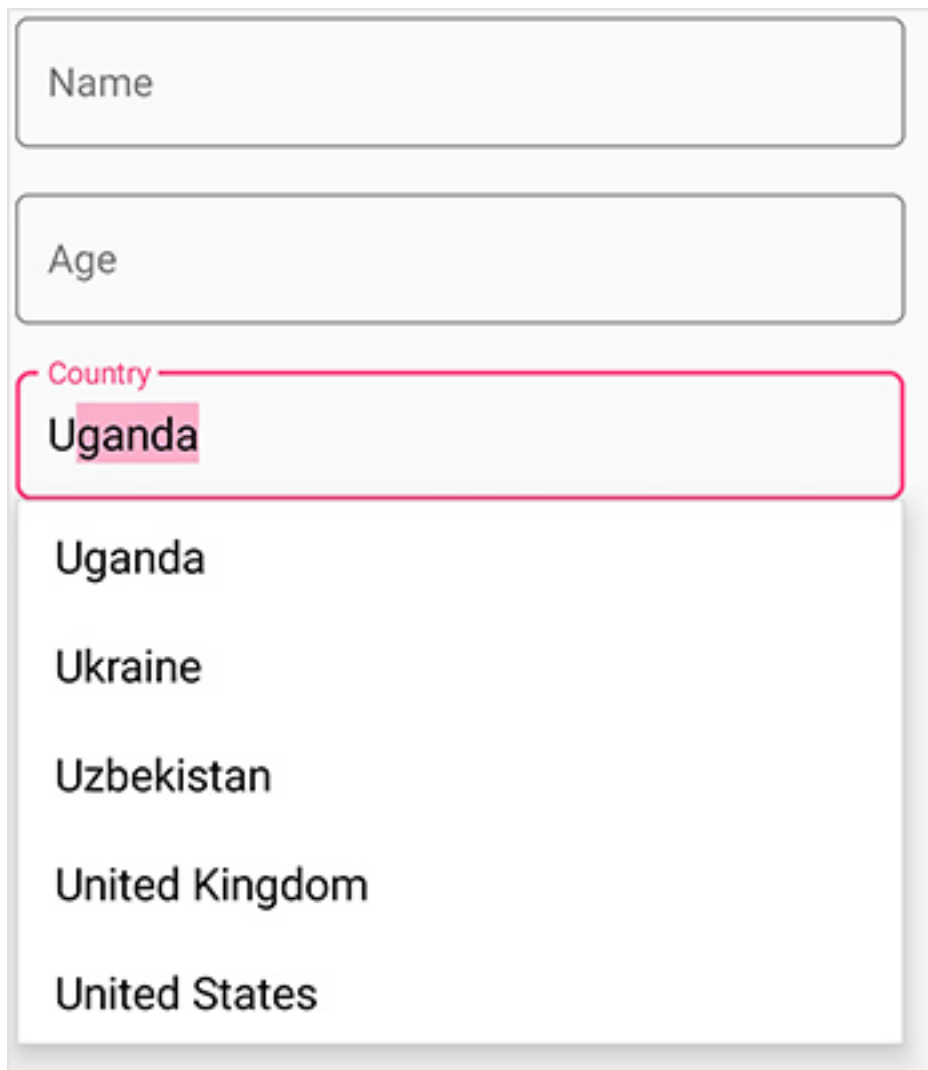


#### [Suggesting choices and appending suggestions to Text](#)

The text is appended to the first matched item in the suggestions collection, and the filtered suggestions are displayed in a drop-down list. The users can pick an item from a list directly or use the up and down keys for browsing the list.

#### **C#**

```
dataForm.DataObject = new ContactInfo();
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
        autoCompleteItem.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
    }
}
```



The image shows a form with three input fields: 'Name', 'Age', and 'Country'. The 'Country' field is highlighted with a red border and contains the text 'Uganda'. Below the 'Country' field, a dropdown menu is open, displaying a list of suggestions: 'Uganda', 'Ukraine', 'Uzbekistan', 'United Kingdom', and 'United States'. The 'Uganda' suggestion is highlighted with a pink background.

#### [AutoComplete editor suggestion 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. The available filtering modes are,

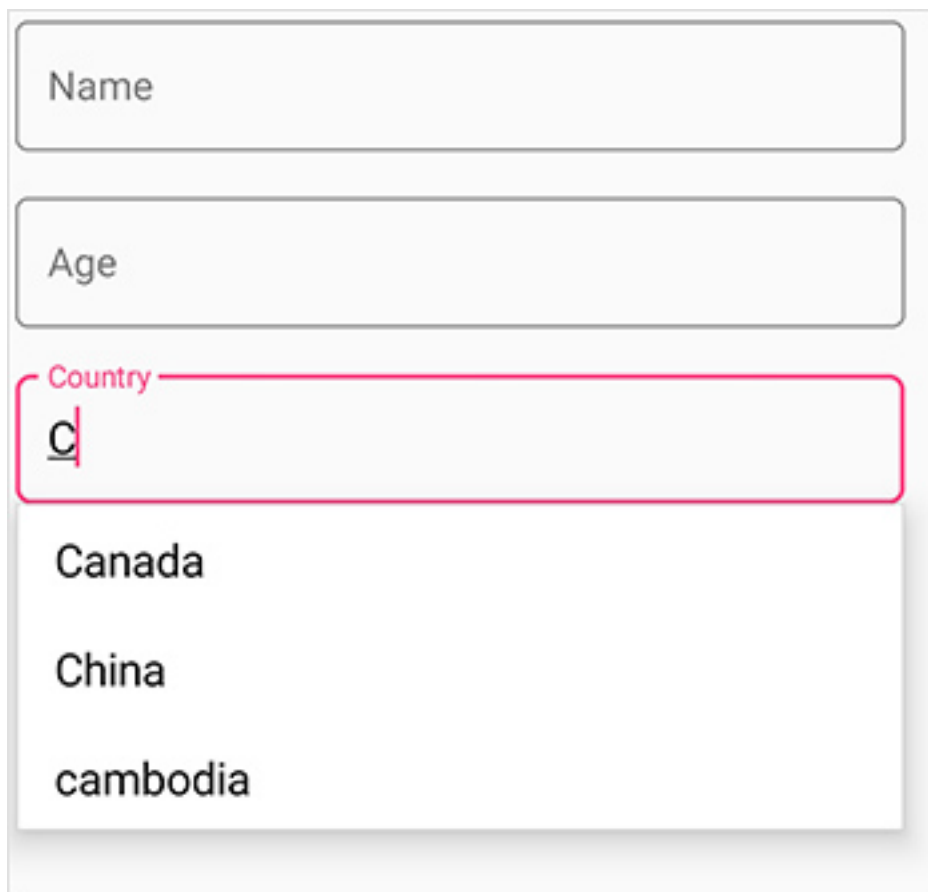
- StartsWith
- StartsWithCaseSensitive
- Contains
- ContainsWithCaseSensitive
- Equals
- EqualsWithCaseSensitive
- EndsWith
- EndsWithCaseSensitive

#### [Filtering words that starts with input text](#)

Displays all the matches that start with the typed characters in items source of autocomplete editor. This strategy is case in sensitive.

**C#**

```
dataForm.DataObject = new ContactInfo();
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
        autoCompleteItem.SuggestionMode =
        Syncfusion.XForms.DataForm.SuggestionMode.StartsWith;
    }
}
```



The screenshot shows a form with three input fields: 'Name', 'Age', and 'Country'. The 'Country' field is active, showing a red border and a list of suggestions: 'Canada', 'China', and 'cambodia'. The suggestions are filtered based on the input text 'C'.

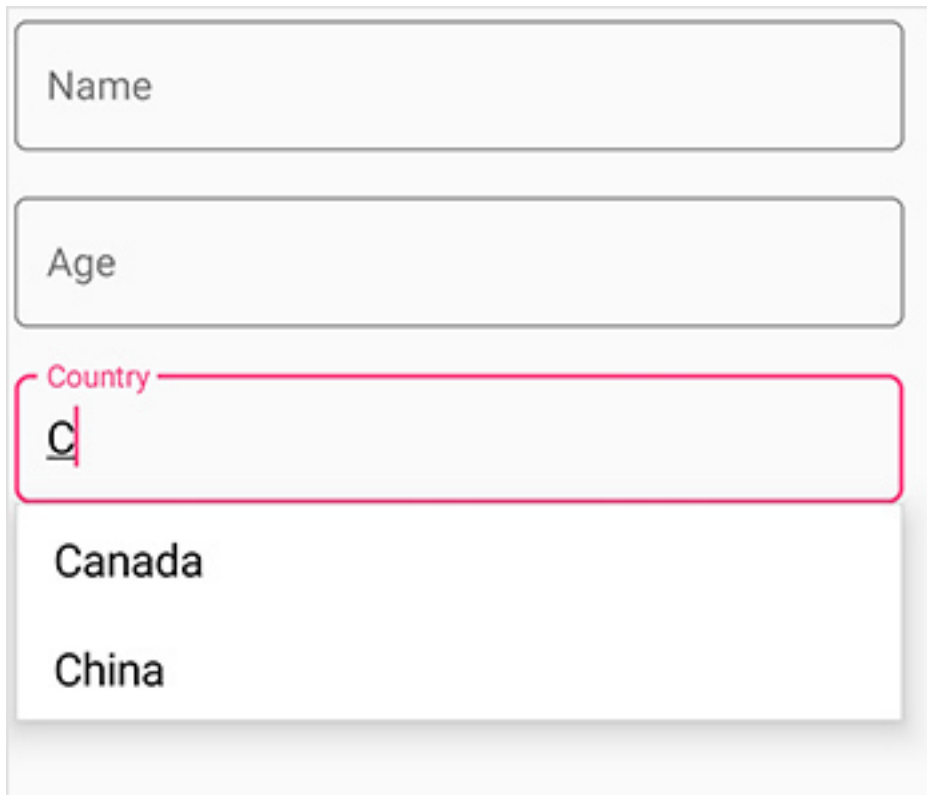
**Filtering words that starts with input text - CaseSensitive**

Displays all the matches that start with the typed characters in items source of autocomplete editor. This strategy is case sensitive.

**C#**

```
dataForm.DataObject = new ContactInfo();
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
```

```
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
        autoCompleteItem.SuggestionMode =
        Syncfusion.XForms.DataForm.SuggestionMode.StartsWithCaseSensitive;
    }
}
```

The image shows a vertical stack of four text input fields. The first field is labeled 'Name', the second 'Age', and the third 'Country'. The 'Country' field is active, with a red border and a red cursor. Below the 'Country' field, a dropdown menu is open, displaying two suggestions: 'Canada' and 'China'. The first suggestion, 'Canada', is highlighted with a light blue background. The input text in the 'Country' field is 'C'.

#### [Filtering words that contains the input text](#)

Displays all the matches that contain the typed characters in items source of autocomplete editor. This strategy is case in-sensitive.

#### [Filtering words that contains the input text - CaseSensitive](#)

Displays all the matches that contain the typed characters in items source of autocomplete editor. This strategy is case sensitive.

#### [Filtering words that equals the input text](#)

Displays all the words that completely matches with the typed characters in items source of autocomplete editor. This strategy is case in-sensitive.

#### [Filtering words that equal the input text - CaseSensitive](#)

Displays all the words that completely match with the typed characters in items source of autocomplete editor. This strategy is case sensitive.

#### Filtering words that end with the input text

Displays all the matches that end with the typed characters in items source of autocomplete editor. This strategy is case in-sensitive.

#### Filtering words that end with the input text - CaseSensitive

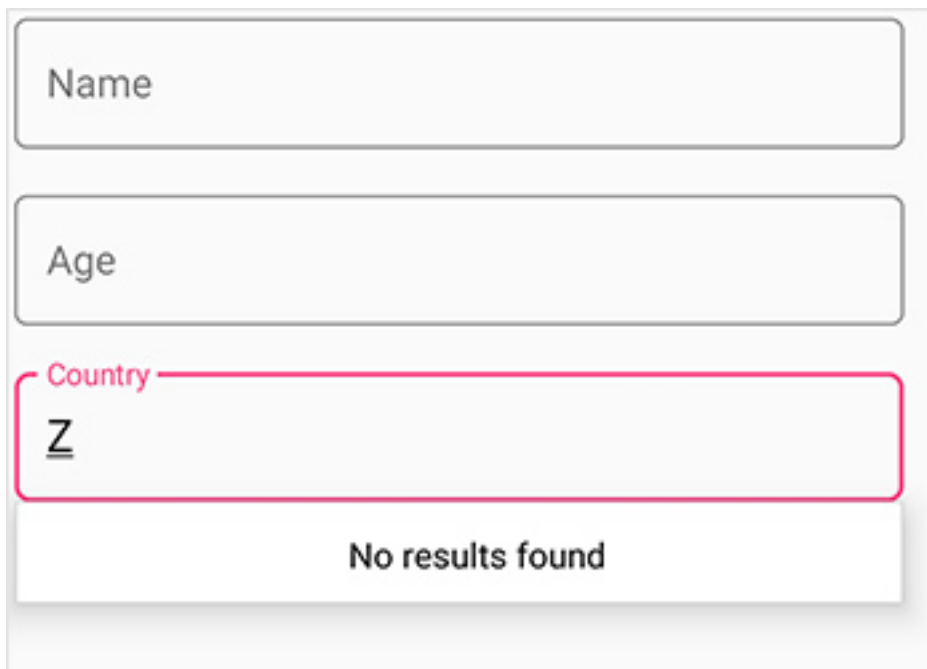
Displays all the matches that end with the typed characters in items source of autocomplete editor. This strategy is case sensitive.

#### No results found text

When the entered item is not in the suggestion list, SfAutoComplete displays a text that indicates there is no search results found. You can set the desire text to be displayed for indicating no results found with the [NoResultsFoundText](#) property.

#### C#

```
dataForm.DataObject = new ContactInfo();
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
        autoCompleteItem.NoResultsFoundText = "No result found";
    }
}
```



The image shows a user interface with three text input fields stacked vertically. The first field is labeled 'Name', the second 'Age', and the third 'Country'. The 'Country' field is currently active, indicated by a red border, and contains the letter 'Z'. Below the 'Country' field, a white box with a gray border contains the text 'No results found'.

**Note:** The [NoResultsFoundText](#) works by default in the UWP platform without setting any property by showing the text "No result found".

### Highlighting match text

Highlights matching characters in a suggestion list to pick an item with more clarity. The text highlight can be indicated with various customizing colors by enabling the following property.

- [HighlightedTextColor](#) - Sets the color of the highlighted text for differentiating the highlighted characters.

### TextHighlightMode

There are two ways to highlight the matching text:

- First occurrence

### C#

```
dataForm.DataObject = new ContactInfo();
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
        autoCompleteItem.SuggestionMode = SuggestionMode.StartsWith;
        autoCompleteItem.HighlightedTextColor = Color.Red;
        autoCompleteItem.TextHighlightMode = OccurrenceMode.FirstOccurrence;
        autoCompleteItem.ItemsSource = new List<string>
        {
            "Albania",
            "Algeria",
            "American Samoa",
            "Andorra"
        };
    }
}
```

Name

Age

Country

a

- Albania
- Algeria
- American Samoa
- Andorra

- Multiple occurrence

### C#

```
dataForm.DataObject = new ContactInfo();
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
        autoCompleteItem.SuggestionMode = SuggestionMode.Contains;
        autoCompleteItem.HighlightedTextColor = Color.Red;
        autoCompleteItem.TextHighlightMode = OccurrenceMode.MultipleOccurrence;
        autoCompleteItem.ItemsSource = new List<string>
        {
            "Albania",
            "Algeria",
            "American Samoa",
            "Andorra"
        };
    }
};
```

```
}  
}
```

Name

Age

Country

a

Albania

Algeria

American Samoa

Andorra

#### Maximum display item in dropdown column

Restrict the number of suggestions displayed and get the remaining items loaded by selecting LoadMore. You can restrict maximum suggestion to be displayed with the [MaximumSuggestion](#) property. You can set the desired text for displaying the Load more text using the [LoadMoreText](#) property.

#### C#

```
dataForm.DataObject = new ContactInfo();  
dataForm.RegisterEditor("Country", "AutoComplete");  
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")  
    {  
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);  
        autoCompleteItem.MaximumSuggestion = 3;  
        autoCompleteItem.LoadMoreText = "Load more";  
    }  
}
```



```
}
```

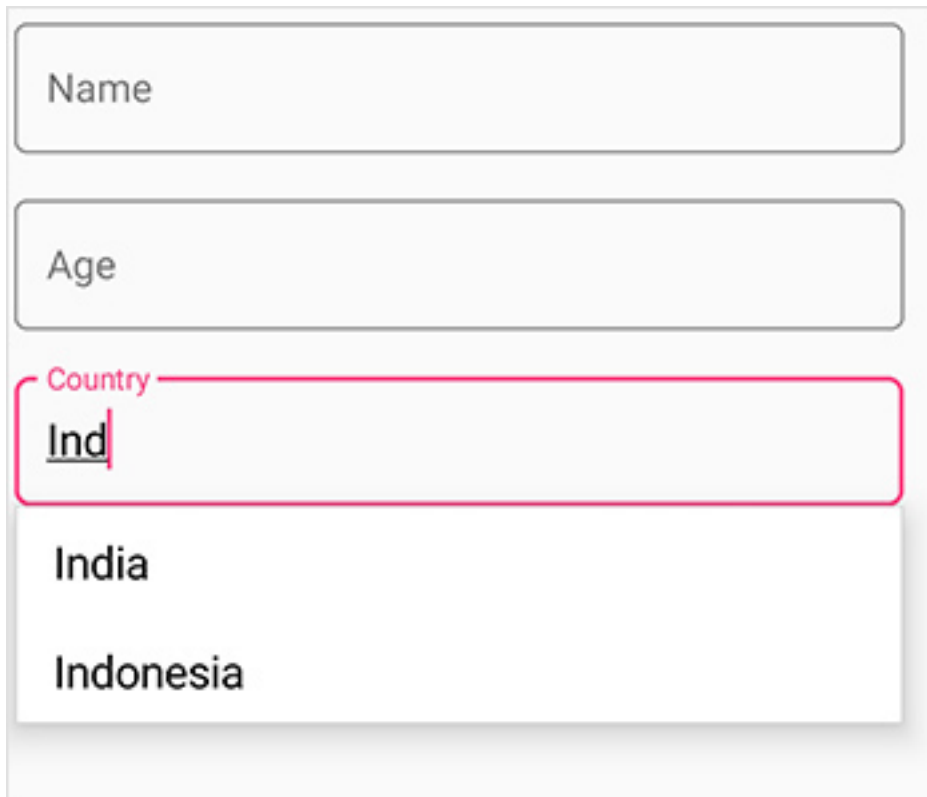
The image shows a user interface for a data form. It has three input fields: 'Name', 'Age', and 'Country'. The 'Country' field is active, showing a red border and a dropdown list of suggestions: 'Uganda', 'Ukraine', and 'Uzbekistan'. The first letter 'U' is entered in the field. Below the list is a 'Load more' button.

#### Minimum prefix character

Instead of displaying suggestion list on every character entry, matches can be filtered and displayed after a few character entries using the [MinimumPrefixCharacters](#) property. The default value is 1.

#### C#

```
dataForm.DataObject = new ContactInfo();
dataForm.RegisterEditor("Country", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Country")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
        autoCompleteItem.MinimumPrefixCharacters = 3;
    }
}
```



The image shows a SfDataForm control with three input fields: 'Name', 'Age', and 'Country'. The 'Country' field is active, showing the text 'Ind' with a red border. Below the input field, a suggestion list is displayed with two items: 'India' and 'Indonesia'.

#### Diacritic sensitivity

The control does not stick with one type of keyboard, so you can populate items from a language with letters containing diacritics, and search for them with the English characters from an en-US keyboard. Users can enable or disable the diacritic sensitivity using [IgnoreDiacritic](#) property. The following code example illustrates how to enable the diacritic sensitivity, so that items in the suggestion list can be populated by entering any diacritic character of that alphabet.

#### C#

```
dataForm.DataObject = new Queries();
dataForm.RegisterEditor("RelatedQuestions", "AutoComplete");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "RelatedQuestions")
    {
        var autoCompleteItem = (e.DataFormItem as DataFormAutoCompleteItem);
        autoCompleteItem.IgnoreDiacritic = false;
        autoCompleteItem.ItemsSource = new List<string>
        {
            "Hów tó gâin wéight?",
            "Hów tó drâw ân éléphânt?",
            "Whéré cân I buy â câmérá?",
            "Guidé mé âll thé wây",
        };
    }
}
public class Queries
```

```
{  
    [Display(ShortName = "Product Name")]  
    public string ProductName { get; set; }  
    public string Cost { get; set; }  
    [Display(ShortName = "Related questions asked")]  
    public string RelatedQuestions { get;set; }  
}
```

Product Name

Cost

Related questions asked

a

Hów tó gâin wéight?

Hów tó drâw ân éléphânt?

Whéré cân I buy â câmêrá?

Guidé mé âll thé wây

#### Custom editor

The custom editor can be added to DataForm by overriding the `DataFormEditor` class for business models. You can create custom editor using [Views](#) and [Layouts](#).

To add custom editor in DataForm, register the editor with custom registered type for the required property using `RegisterEditor` method. You can also customize editor settings by using specific override methods available in `DataFormEditor`.

- Creating custom editor using views.
- Creating custom editor using layouts.

*Creating custom editor using views*

Views such as labels, buttons, and sliders can be loaded to custom editor. Here, entry is loaded as custom editor for **ContactName** property.

**C#**

```
public class CustomTextEditor : DataFormEditor<Entry>
{
    public CustomTextEditor(SfDataForm dataForm) : base(dataForm)
    {
    }
    protected override Entry OnCreateEditorView(DataFormItem dataFormItem)
    {
        return new Entry();
    }
}
...
dataForm.RegisterEditor("CustomTextEditor", new CustomTextEditor(dataForm));
dataForm.RegisterEditor("ContactName", "CustomTextEditor");
```

*Creating custom editor using layouts*

Layouts such as Grid, StackLayout, ContentView, and ScrollView can be added as custom editor. Here, the label and image view in Grid are loaded as custom editors for **ContactName** property.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="DataForm_Forms.TextEditor">
    <ContentPage.Content>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="0.05*" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
            <Image Source="LabelContactName.png" />
            <Editor Grid.Column="1" />
        </Grid>
    </ContentPage.Content>
</ContentView>
```

**C#**

```
public class CustomTextEditor : DataFormEditor<TextEditor>
{
    public CustomTextEditor(SfDataForm dataForm) : base(dataForm)
    {
    }
    protected override TextEditor OnCreateEditorView(DataFormItem dataFormItem)
    {
        return new TextEditor();
    }
}
...
```

```
dataForm.RegisterEditor("CustomTextEditor", new CustomTextEditor(dataForm));  
dataForm.RegisterEditor("ContactName", "CustomTextEditor");
```

You should manually commit and validate the editor value of custom DataFormItem. Refer to this [link](#) to know more about custom editor.

## Validation

The data form validates the data and displays hints in the case of validation is not passed. In case of invalid data, the error message is shown at the bottom of the editor.

### Built in validations

The supported built in validations are as follows:

- INotifyDataErrorInfo
- Data annotation

### Using INotifyDataErrorInfo

You can validate the data by implementing the [INotifyDataErrorInfo](#) interface in the data object class.

### C#

```
public class EmployeeInfo : INotifyDataErrorInfo, INotifyPropertyChanged  
{  
    private int _EmployeeID;  
    private string _Name;  
    private string _Title;  
    public event PropertyChangedEventHandler PropertyChanged;  
    public event EventHandler<DataErrorsChangedEventArgs> ErrorsChanged;  
    public EmployeeInfo()  
    {  
    }  
    public int EmployeeID  
    {  
        get { return this._EmployeeID; }  
        set  
        {  
            this._EmployeeID = value;  
            this.RaisePropertyChanged("EmployeeID");  
        }  
    }  
    public string Name  
    {  
        get { return this._Name; }  
        set  
        {  
            this._Name = value;  
            this.RaisePropertyChanged("Name");  
        }  
    }  
    public string Title  
    {  
        get { return this._Title; }  
        set  
        {  
            this.Title = value;  
        }  
    }  
}
```

```

this.RaisePropertyChanged("Title");
}
}
[Display(AutoGenerateField = false)]
public bool HasErrors
{
    get
    {
        return false;
    }
}
private void RaisePropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
public IEnumerable GetErrors(string propertyName)
{
    var list = new List<string>();
    if (!propertyName.Equals("Title"))
        return list;
    if (this.Title.Contains("Marketing"))
        list.Add("Marketing is not allowed");
    return list;
}
}

```

#### Data annotations

You can validate the data using data annotation attributes.

The numeric type like Int, Double, Decimal properties can be validated using the [Range](#) attribute.

#### C#

```

private int _EmployeeID;
[Range(1000, 1500, ErrorMessage = "EmployeeID should be between 1000 and 1500")]
public int EmployeeID
{
    get { return this._EmployeeID; }
    set
    {
        this._EmployeeID = value;
        this.RaisePropertyChanged("EmployeeID");
    }
}

```

The String type property can be validated using [Required](#) and [StringLength](#) attributes.

#### C#

```

private string _Name;
[Required(AllowEmptyStrings = false, ErrorMessage = "Name should not be empty")]
[StringLength(10, ErrorMessage = "Name should not exceed 10 characters")]
public string Name

```

```
{
get { return this._Name; }
set
{
this._Name = value;
this.RaisePropertyChanged("Name");
}
}
```

#### Date range attribute

You can validate the date time value using date range attribute.

#### C#

```
private DateTime joinDate;
[DateRange(MinYear = 2010, MaxYear = 2017, ErrorMessage = "Join date is
invalid")]
public DateTime JoinDate
{
get
{
return joinDate;
}
set
{
joinDate = value;
}
}
```

The screenshot shows a mobile application interface with a blue header bar. The header bar contains a mail icon on the left and status information (signal strength, 100% battery, and time 15:16) on the right. Below the header, there is a form with five input fields. The first field, labeled 'EmployeeID', contains the value '0'. The second field, labeled 'Name', is empty. The third field, labeled 'CustomerID', is empty. The fourth field, labeled 'Title', is empty. The fifth field, labeled 'JoinDate', contains the value '7/19/0003'. Below the 'JoinDate' field, there is a red error message that reads 'Join date is invalid'. At the bottom of the screen, there is a grey navigation bar with four icons: a circle, a square, a square, and a triangle.

EmployeeID	0
Name	
CustomerID	
Title	
JoinDate	7/19/0003

Join date is invalid



## Validation mode

The [ValidationMode](#) determines when the value should be validated.

The supported validation modes are as follows:

- `LostFocus`
- `PropertyChanged`
- `Explicit`

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
  <ContentPage.Content>
    <dataForm:SfDataForm x:Name="dataForm" ValidationMode="LostFocus"/>
  </ContentPage.Content>
</ContentPage>
```

## C#

```
dataForm.ValidationMode = ValidationMode.LostFocus;
```

### *LostFocus*

If the commit mode is `LostFocus`, the value will be validated when the editor lost its focus.

### *PropertyChanged*

The value will be validated immediately when it is changed.

### *Explicit*

The value should be validated manually by calling the [SfDataForm.Validate](#) or [SfDataForm.Validate \(propertyName\)](#) method.

The following code validates the value of all the properties in the data object:

## C#

```
dataForm.Validate();
```

To validate the specific property value, pass the property name as argument.

## C#

```
dataForm.Validate("Name");
```

You can determine whether the data form or property is valid or not by using the `Validate` method.

## C#

```
bool isValid = dataForm.Validate();
bool isPropertyValid = dataForm.Validate("Property");
```

If the data form or property is valid, `true` will be returned. Or else `false` will be returned.

**Note:** For validating value, the new value should be committed in data object. So, `ValidationMode` takes higher priority than `CommitMode`.

#### Custom validation through events

You can validate the data using the [Validating](#) event of the data form.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-
namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm" Validating="DataForm_Validating" />
</Grid>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
dataForm.Validating += DataForm_Validating;
private void DataForm_Validating(object sender, ValidatingEventArgs e)
{
    if (e.PropertyName == "Name")
    {
        if (e.Value != null && e.Value.ToString().Length > 8)
        {
            e.IsValid = false;
            e.ErrorMessage = "Name should not exceed 8 characters";
        }
    }
}
```

You can get the notification after completing validation using the [Validated](#) event of the data form.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-
namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm" Validated="DataForm_Validated" />
</Grid>
```

```
</ContentPage.Content>
</ContentPage>
```

### C#

```
dataForm.Validated += DataForm_Validated;
private void DataForm_Validated(object sender, ValidatedEventArgs e)
{
    var isValid = e.IsValid;
    var propertyName = e.PropertyName;
}
```

You can get the details of invalid DataFormItems when validating the data form as **Explicit** validation mode using [ValidationCompleted](#) event. This event contains [ValidationCompletedEventArgs](#) argument, which holds a list of DataFormItem as errors.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm"
ValidationCompleted="DataForm_ValidationCompleted" />
</ContentPage.Content>
</ContentPage>
```

### C#

```
dataForm.ValidationCompleted += DataForm_ValidationCompleted;
...
private void DataForm_ValidationCompleted(object sender,
ValidationCompletedEventArgs e)
{
    var invalidItems = e.Errors;
}
```

### Valid or positive message

If the value meets the desired criteria, you can show the [valid or positive message](#). As error message, the valid message will also be displayed at the bottom of the editor.

### C#

```
private string _Name;
[DisplayOptions(ValidMessage = "Name length is enough")]
[StringLength(10, ErrorMessage = "Name should not exceed 10 characters")]
public string Name
{
    get { return this._Name; }
    set
```

```
{  
  this._Name = value;  
  this.RaisePropertyChanged("Name");  
}
```

The screenshot displays a mobile application interface with a blue status bar at the top showing icons for signal, Wi-Fi, 100% battery, and the time 15:21. Below the status bar is a form with five input fields. The first field, 'EmployeeID', contains the value '0'. The second field, 'Name', contains the value 'John' and has a green validation message 'Name length is enough' displayed below it. The third field, 'CustomerID', is empty. The fourth field, 'Title', is empty. The fifth field, 'JoinDate', contains the value '1/1/0001'. At the bottom of the screen is a grey navigation bar with four icons: a circle, a square, a square, and a left-pointing arrow.

EmployeeID	0
Name	John Name length is enough
CustomerID	
Title	
JoinDate	1/1/0001

How to validate the property value based on another value

To validate one property value based on another property value, use the [property changed event](#) and [Validate](#) methods.

Here, AccountNumber and AccountNumber1 fields are validated.

### C#

```
dataForm.DataObject = new RecipientInfo();
(dataForm.DataObject as INotifyPropertyChanged).PropertyChanged +=
DataFormGettingStarted_PropertyChanged;
private void DataFormGettingStarted_PropertyChanged(object sender,
PropertyChangedEventArgs e)
{
    if (e.PropertyName.Equals("AccountNumber"))
    {
        var value =
        (string)sender.GetType().GetProperty("AccountNumber1").GetValue(sender);
        if (!string.IsNullOrEmpty(value))
        dataForm.Validate("AccountNumber1");
    }
    else if (e.PropertyName.Equals("AccountNumber1"))
    {
        var value =
        (string)sender.GetType().GetProperty("AccountNumber").GetValue(sender);
        if (!string.IsNullOrEmpty(value))
        dataForm.Validate("AccountNumber");
    }
}
```

Customize validation message using DataTemplate

The default appearance of the validation message can be customized by using the [ValidationTemplate](#) property of the `DataForm`.

### XML

```
<dataForm:SfDataForm
x:Name="dataForm"
ValidationTemplate="{Binding ValidationTemplate}">
<dataForm:SfDataForm.BindingContext>
<local:ValidationDataTemplate/>
</dataForm:SfDataForm.BindingContext>
</dataForm:SfDataForm>
```

*Creating a DataTemplate*

### C#

```
public class ValidationDataTemplate :DataTemplate
{
    public DataTemplate ValidationTemplate { get; set; }
    public ValidationDataTemplate()
    {
        ValidationTemplate = new DataTemplate(() =>
        {
            return new Button

```

```
{
    Text = "Field should not be empty",
    TextColor = Color.White,
    BackgroundColor = Color.LightGreen
};
});
}
```

### Customize validation message using DataTemplateSelector

You can use **DataTemplateSelector** to choose a **DataTemplate** at runtime based on the value of a data-bound to [ValidationTemplate](#) property of DataForm. It lets you choose a different data template for each validation message, customizing the appearance of a particular validation message based on certain conditions.

### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<local:TemplateSelector x:Key="validationDataTemplateSelector" />
</ResourceDictionary>
</ContentPage.Resources>
<dataForm:SfDataForm Grid.Row="1" x:Name="dataForm"
ValidationTemplate="{StaticResource validationDataTemplateSelector}" />
```

### *Creating a DataTemplateSelector*

### C#

```
public class TemplateSelector : DataTemplateSelector
{
    public DataTemplate ValidMessageTemplate { get; set; }
    public DataTemplate InvalidMessageTemplate { get; set; }
    public DataTemplate LastNameTemplate { get; set; }
    public DataTemplate EmailTemplate { get; set; }
    public DataTemplate ContactNumberTemplate { get; set; }
    public TemplateSelector()
    {
        ValidMessageTemplate = new DataTemplate(typeof(ValidMessageTemplate));
        InvalidMessageTemplate = new DataTemplate(typeof(InvalidMessageTemplate));
        LastNameTemplate = new DataTemplate(typeof(LastNameTemplate));
        EmailTemplate = new DataTemplate(typeof(EmailTemplate));
        ContactNumberTemplate = new DataTemplate(typeof(ContactNumberTemplate));
    }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject container)
    {
        var dataform = (container as SfDataForm);
        if (dataform == null) return null;
        if (dataform != null)
        {
            if ((item as DataFormItem).LabelText == "First Name")
            {
                if (!(item as DataFormItem).IsValid)
                {
                    return InvalidMessageTemplate;
                }
            }
        }
    }
}
```

```

    }
    else
    {
        return ValidMessageTemplate;
    }
}
else if ((item as DataFormItem).LabelText == "Last Name")
{
    if (!(item as DataFormItem).IsValid)
    {
        return LastNameTemplate;
    }
}
else if ((item as DataFormItem).LabelText == "Email")
{
    if (!(item as DataFormItem).IsValid)
    {
        return EmailTemplate;
    }
}
else if ((item as DataFormItem).LabelText == "Contact Number")
{
    if (!(item as DataFormItem).IsValid)
    {
        return ContactNumberTemplate;
    }
}
return null;
}
else
return null;
}
}

```

Used **Button** inside a **Grid** to display the valid and invalid message in the view.

### XML

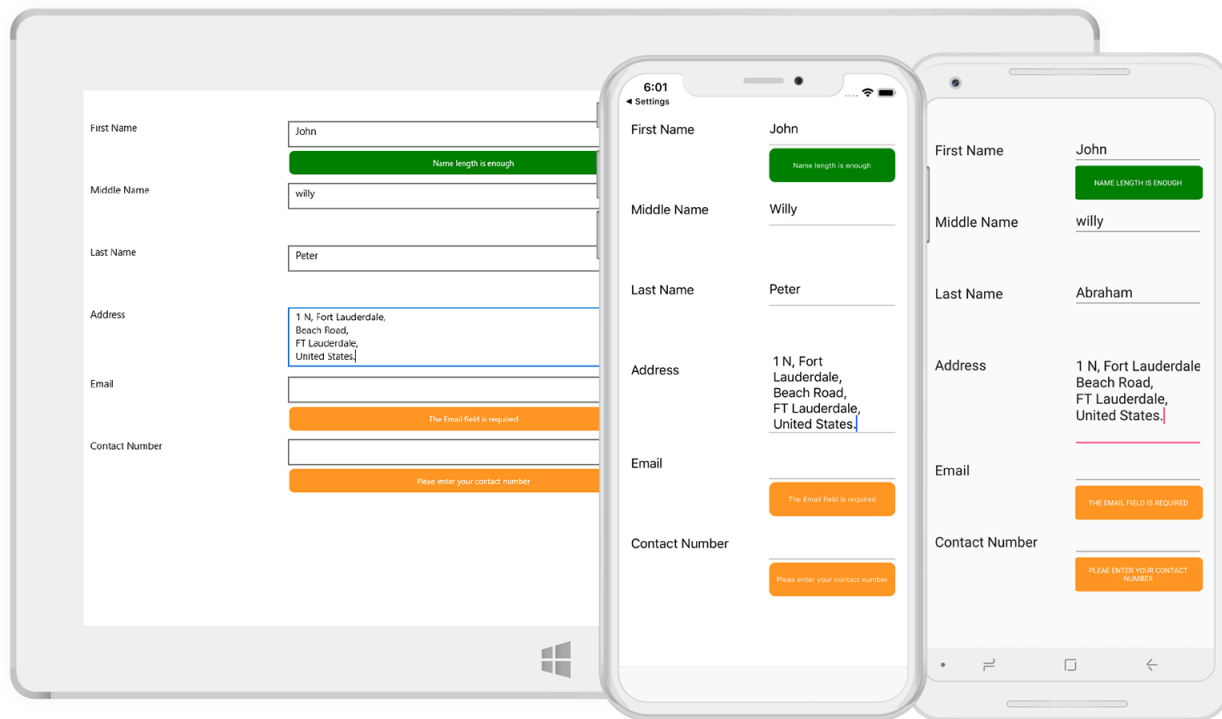
```

<Grid xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="DataForm_Validation.ValidMessageTemplate"
VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand">
    <Grid BackgroundColor="Transparent">
        <Button x:Name="maingrid" CornerRadius="8" Text="Name length is enough"
        FontSize="9" TextColor="Green" VerticalOptions="FillAndExpand"
        HorizontalOptions="FillAndExpand"/>
    </Grid>
</Grid>
...
<Grid xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="DataForm_Validation.InvalidMessageTemplate"
VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand">
    <Grid BackgroundColor="Transparent">

```



```
<Button x:Name="maingrid" CornerRadius="8" Text="Pleae enter your first  
name" FontSize="9" TextColor="White" VerticalOptions="FillAndExpand"  
HorizontalOptions="FillAndExpand"/>  
</Grid>  
</Grid>
```



## Layout



### Overview

The data form supports linear and grid layouts. The `DataFormLayoutManager` creates the [DataFormItemView](#), [DataFormGroupItemView](#), and manages layout of label, editor, and validation label.

### Linear layout support

By default, the data form arranges the fields one-by-one. It is applicable for both label positions: left and top.

When the label position is Left, the linear layout is shown as follows:

12:11 PM 100%  

First Name


Middle Name


Last Name


Contact Number


Email

Address











When the label position is Top, the linear layout is shown as follows:

12:05 PM  100% 

First Name





Middle Name

Last Name

Contact Number

Email

Address

## Grid layout support

By default, the data form arranges one data field per row. It is possible to have more than one data fields per row by setting the [ColumnCount](#) property which provides grid like layout for the data form.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-
namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm" ColumnCount="2"/>
</ContentPage.Content>
</ContentPage>
```

### C#

```
dataForm.ColumnCount = 2;
```

**Note:** Setting the `ColumnCount` property to `SfDataForm` does not arrange the data field in a group according to the column count. To set the column count for data fields in the data form group, refer to [loading different layout for data form group](#)

When the label position is Left, the grid layout is shown as follows:

The image shows a mobile application interface with a blue status bar at the top. The status bar contains icons for a gallery and email on the left, and a mute, Wi-Fi, cellular signal, 100% battery, and 16:17 time on the right. Below the status bar is a light gray data form. The form has six input fields arranged in two columns. The left column contains 'FirstNam', 'LastNam', and 'Email'. The right column contains 'MiddleNa', 'Contact N', and 'BirthDate'. Each field is represented by its label followed by a horizontal line. At the bottom of the screen is a dark gray navigation bar with four white icons: a circle, a stylized 'U' shape, a square, and a left-pointing arrow.

FirstNam	_____	MiddleNa	_____
LastNam	_____	Contact N	_____
Email	_____	BirthDate	_____

When the label position is Top, the grid layout is shown as follows:

The image shows a mobile application interface with a blue status bar at the top. The status bar contains icons for signal strength, Wi-Fi, and battery level (100%), along with the time 16:18. Below the status bar is a light gray data form with six input fields arranged in two columns. The left column contains fields for First Name, Last Name, and Email. The right column contains fields for Middle Name, Contact No, and BirthDate. Each field has a horizontal line indicating the input area. At the bottom of the screen is a dark gray navigation bar with four white icons: a circle, a square, a square with a circle inside, and a left-pointing arrow.

FirstName	MiddleName
_____	_____
LastName	Contact No
_____	_____
Email	BirthDate
_____	_____



### Label visibility

You can hide the label by defining the [DisplayOptions](#) attribute or by handling [AutoGeneratingDataFormItem](#) event. In this case, only the editor will be loaded.

#### Using attributes

##### C#

```
private double? percentage;
[DisplayOptions(ShowLabel = false)]
[Display(Prompt = "Enter percentage")]
public double? Percentage
{
    get
    {
        return percentage;
    }
    set
    {
        percentage = value;
        RaisePropertyChanged("Percentage");
    }
}
```

#### Using event

##### C#

```
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Percentage")
    {
        e.DataFormItem.PlaceholderText = "Enter percentage";
        e.DataFormItem.ShowLabel = false;
    }
}
```

The image shows a mobile application interface with a blue status bar at the top. The status bar contains icons for signal strength, Wi-Fi, and battery level (100%), along with the time 16:21. Below the status bar is a white header area with the text "Enter Name" and a horizontal line. The main content area is white and contains six text input fields, each with a label to its left and a horizontal line to its right. The labels are "MiddleName", "LastName", "Contact No", "Email", and "BirthDate". At the bottom of the screen is a gray navigation bar with four white icons: a circle, a square, a square, and a left arrow.

Enter Name

MiddleName

LastName

Contact No

Email

BirthDate

### Label position

Labels can be positioned either top or left side of the editor. By using the [LabelPosition](#) property, you can layout the label associated with editor.

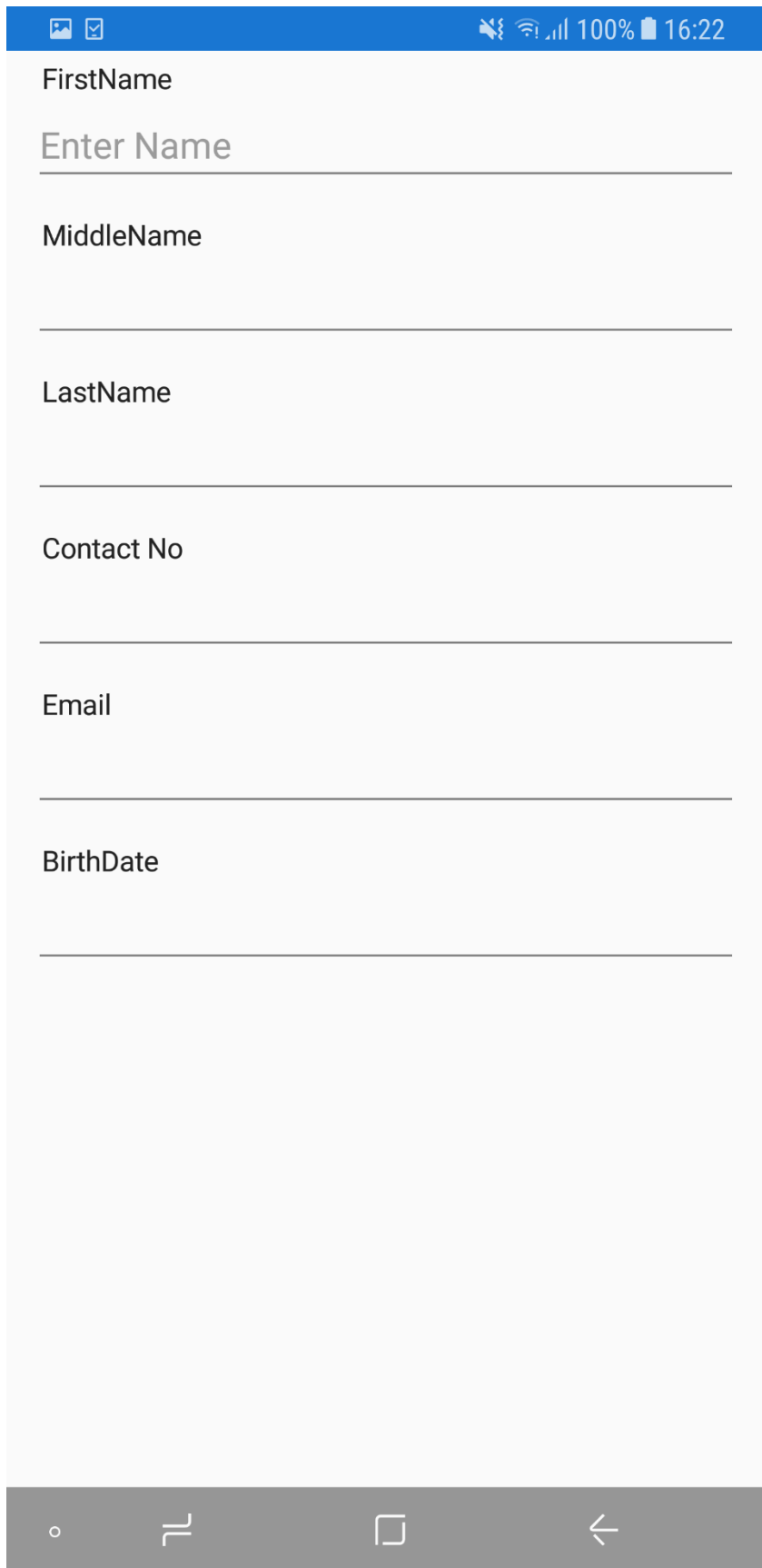
By default, the label will be positioned at left side of the editor.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-
namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<dataForm:SfDataForm x:Name="dataForm" LabelPosition="Top"/>
</ContentPage.Content>
</ContentPage>
```

### C#

```
dataForm.LabelPosition = LabelPosition.Top;
```



The image shows a mobile application interface with a blue status bar at the top. The status bar contains icons for a gallery, mail, and a back arrow, along with network and battery status indicators (100% battery, 16:22). Below the status bar is a light gray form with the following fields:

- FirstName**: A text input field with the placeholder text "Enter Name".
- MiddleName**: A text input field.
- LastName**: A text input field.
- Contact No**: A text input field.
- Email**: A text input field.
- BirthDate**: A text input field.

At the bottom of the screen is a dark gray navigation bar with four icons: a circle, a square, a square with a circle inside, and a left-pointing arrow.

### Changing label position of the DataFormItem

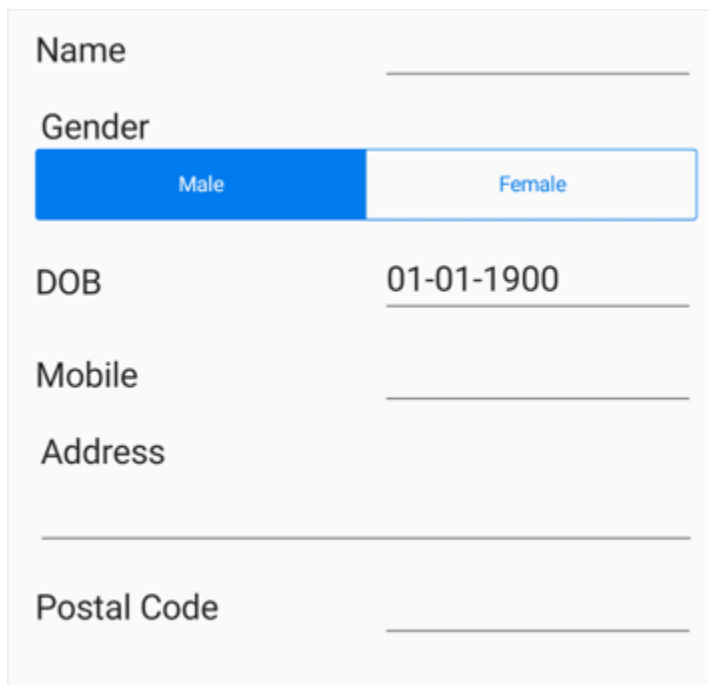
You can change the label position using the [LabelPosition](#) property in `DataFormItem`, and it will be handled in the `AutoGeneratingDataFormItem` event.

#### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

#### C#

```
dataForm.RegisterEditor("Gender", "Segment");
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "Gender")
        {
            e.DataFormItem.LabelPosition = LabelPosition.Top;
        }
        if (e.DataFormItem.Name == "Address")
        {
            e.DataFormItem.LabelPosition = LabelPosition.Top;
        }
    }
}
```



### Loading images for label

You can load image instead of label by defining attribute or by handling the `AutoGeneratingDataFormItem` event.

#### Using attributes

To show the image as label, use the `ImageName` property in `DisplayOptions` attribute. Images will be taken from ...\\Resources\\drawable folder in Xamarin.Forms.Android, ...\\Resources folder in Xamarin.Forms.iOS.

#### C#

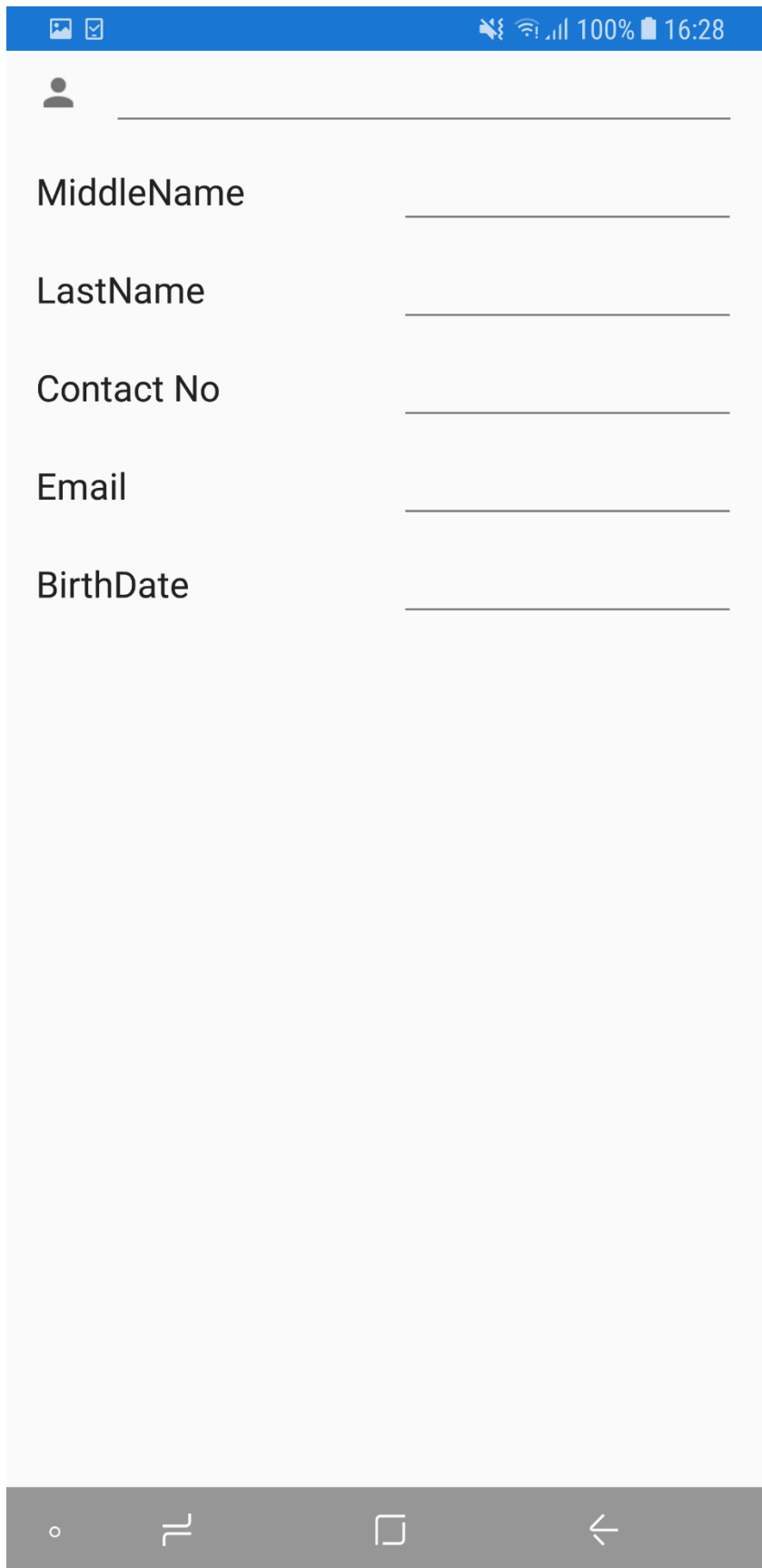
```
private string firstName;
[DisplayOptions(ImageName = "ContactInfo.png")]
public string FirstName
{
    get { return this.firstName; }
    set
    {
        this.firstName = value;
    }
}
```

#### Using event

By using the `ImageSource` property in the `DataFormItem`, you can load the image as label.


#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "FirstName")
        e.DataFormItem.ImageSource = ImageSource.FromFile("ContactInfo.png");
    }
}
```



The screenshot displays a mobile application interface with a blue header bar. On the left of the header are icons for a gallery and email. On the right are icons for signal strength, Wi-Fi, and battery level (100%), along with the time 16:28. Below the header is a light gray form area. It begins with a profile icon and a horizontal line for a name. This is followed by five labeled input fields: 'MiddleName', 'LastName', 'Contact No', 'Email', and 'BirthDate'. Each label is positioned to the left of a horizontal input line. At the bottom of the screen is a dark gray navigation bar containing four white icons: a circle, a square, a square with a circle, and a left-pointing arrow.

16:28 100% Wi-Fi Signal

 \_\_\_\_\_

MiddleName \_\_\_\_\_

LastName \_\_\_\_\_

Contact No \_\_\_\_\_

Email \_\_\_\_\_

BirthDate \_\_\_\_\_

Navigation icons: Circle, Square, Square with Circle, Left Arrow

### Changing order of the DataFormItem

You can change the order of the **DataFormItem** by using attributes or by handling **AutoGeneratingDataFormItem** event.

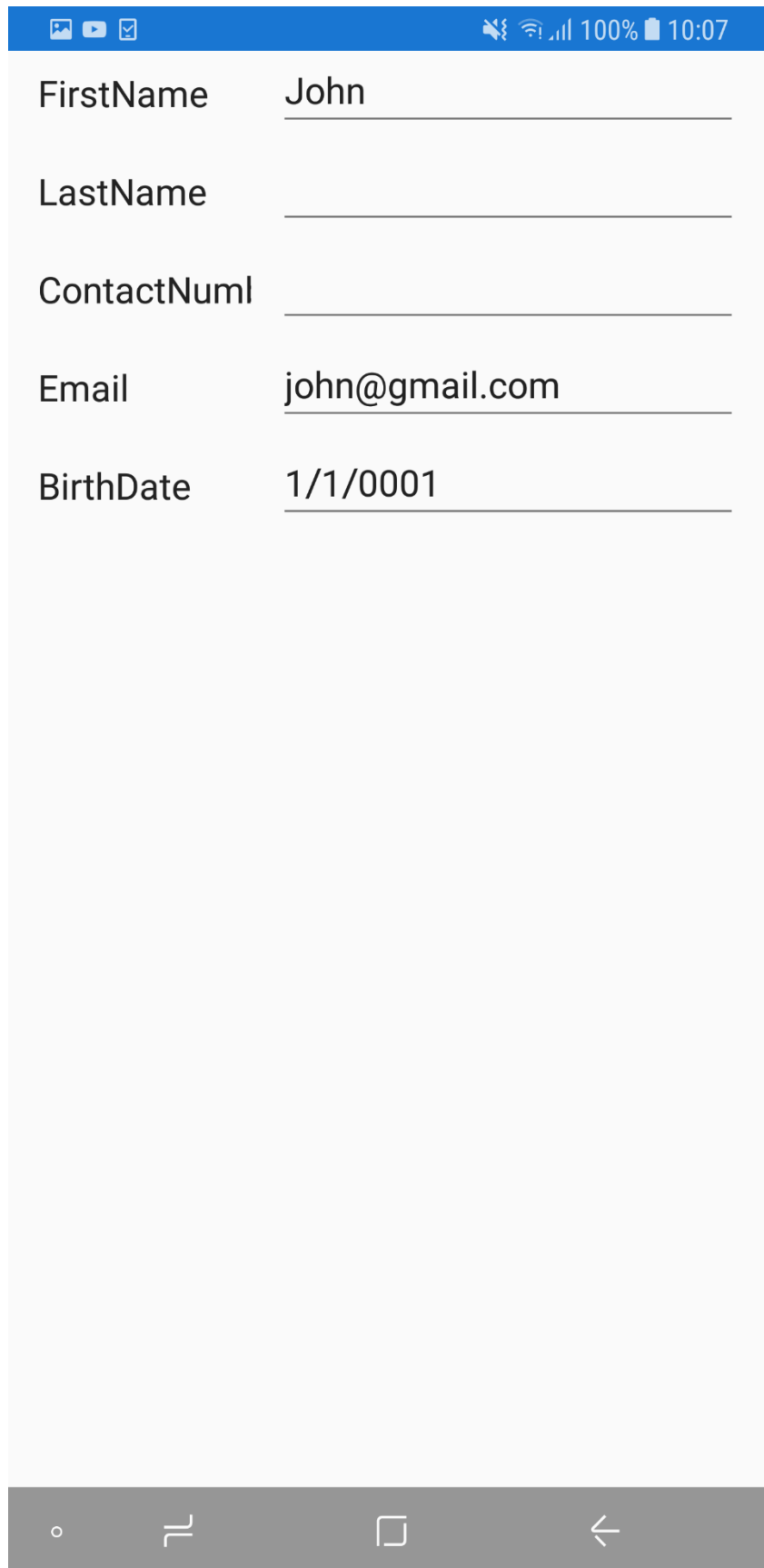
#### *Using attributes*

You can set the order by using the **Order** property in display attribute.

#### **C#**

```
public class ContactsInfo
{
    private string lastName;
    private string contactNo;
    public ContactsInfo()
    {
    }
    [Display(Order = 2)]
    public string ContactNumber
    {
        get { return contactNo; }
        set
        {
            this.contactNo = value;
        }
    }
    private string firstName;
    [Display(Order = 0)]
    public string FirstName
    {
        get { return this.firstName; }
        set
        {
            this.firstName = value;
        }
    }
    [Display(Order = 1)]
    public string LastName
    {
        get { return this.lastName; }
        set
        {
            this.lastName = value;
        }
    }
}
```





The image shows a mobile application interface with a data form. At the top is a blue status bar with icons for gallery, video, mail, and system status (signal, Wi-Fi, 100% battery, 10:07). The form has five rows, each with a label on the left and a text input field on the right. The inputs are: 'John' for 'FirstName', an empty field for 'LastName', an empty field for 'ContactNuml', 'john@gmail.com' for 'Email', and '1/1/0001' for 'BirthDate'. At the bottom is a grey navigation bar with four icons: a circle, a double line, a square, and a left arrow.

FirstName	John
LastName	
ContactNuml	
Email	john@gmail.com
BirthDate	1/1/0001

### Using event

You can change the fields order by using the [Order](#) property in the `DataFormItem`.

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "FirstName")
            e.DataFormItem.Order = 0;
    }
}
```

### Grouping data fields

It is possible to group some fields and set group name in the data form. You can expand or collapse the group by tapping the group item.

Grouping can be achieved by defining attributes or by handling the `AutoGeneratingDataFormItem` event.

### Using attributes

#### C#

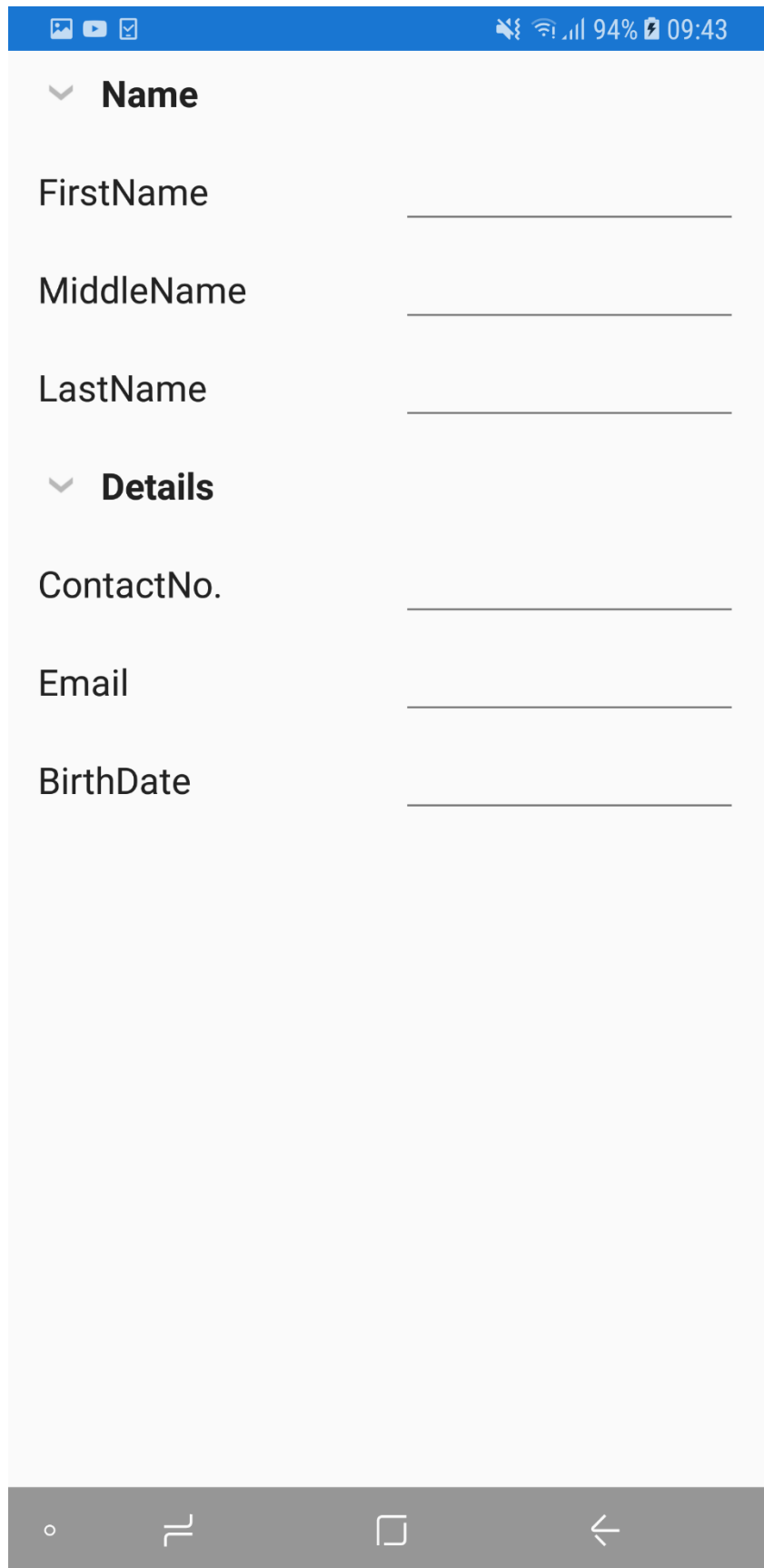
```
public class ContactsInfo
{
    private string lastName;
    private string contactNo;
    private string email;
    private DateTime? birthDate;
    public ContactsInfo()
    {
    }
    private string firstName;
    [Display(GroupName = "Name")]
    public string FirstName
    {
        get { return this.firstName; }
        set
        {
            this.firstName = value;
        }
    }
    private string middleName;
    [Display(GroupName = "Name")]
    public string MiddleName
    {
        get { return this.middleName; }
        set
        {
            this.middleName = value;
        }
    }
    [Display(GroupName = "Name")]
```

```
public string LastName
{
    get { return this.lastName; }
    set
    {
        this.lastName = value;
    }
}
[Display(GroupName = "Details", ShortName = "ContactNo.")]
public string ContactNumber
{
    get { return contactNo; }
    set
    {
        this.contactNo = value;
    }
}
[Display(GroupName = "Details")]
public string Email
{
    get { return email; }
    set
    {
        email = value;
    }
}
[Display(GroupName = "Details")]
public DateTime? BirthDate
{
    get { return birthDate; }
    set
    {
        birthDate = value;
    }
}
}
```

### Using event

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "FirstName" || e.DataFormItem.Name ==
"MiddleName" || e.DataFormItem.Name == "LastName")
            e.DataFormItem.GroupName = "Name";
        else
            e.DataFormItem.GroupName = "Details";
    }
}
```



**Name**

FirstName \_\_\_\_\_

MiddleName \_\_\_\_\_

LastName \_\_\_\_\_

**Details**

ContactNo. \_\_\_\_\_

Email \_\_\_\_\_

BirthDate \_\_\_\_\_

The image shows a mobile application interface with a blue header bar containing icons for gallery, video, and mail, and status information (signal, 94% battery, 09:43). Below the header is a form titled 'SfDataForm'. The form has two sections: 'Name' (expanded) and 'Details' (collapsed). The 'Details' section contains three input fields: 'ContactNo.', 'Email', and 'BirthDate'. At the bottom of the screen is a grey navigation bar with four icons: a circle, a square, a square, and a left arrow.

^ **Name**

▼ **Details**

ContactNo. \_\_\_\_\_

Email \_\_\_\_\_

BirthDate \_\_\_\_\_

### *Changing order of the DataFormGroupItem*

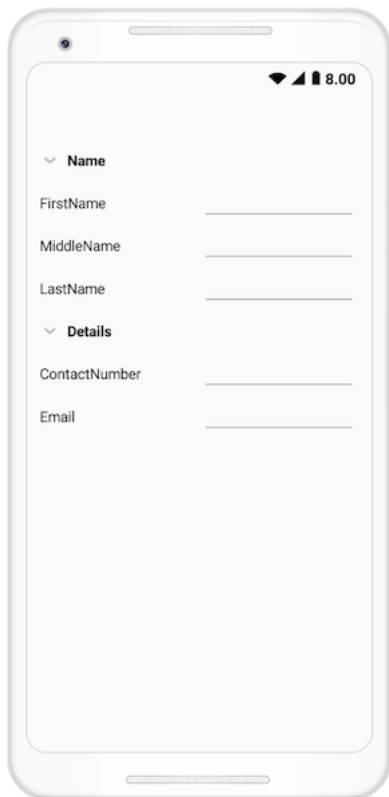
You can change the order of the `DataFormGroupItem` by using attributes. You can set the order of data form items in group by using the `Order` property along with `GroupName` property in display attribute.

#### **C#**

```
public class ContactInfo
{
    private string lastName;
    private string contactNo;
    public ContactInfo()
    {
    }
    private string firstName;
    [Display(Order = 0, GroupName = "Name")]
    public string FirstName
    {
        get { return this.firstName; }
        set
        {
            this.firstName = value;
        }
    }
    [Display(Order = 2, GroupName = "Name")]
    public string LastName
    {
        get { return this.lastName; }
        set
        {
            this.lastName = value;
        }
    }
    private string middleName;
    [Display(Order = 1, GroupName = "Name")]
    public string MiddleName
    {
        get { return this.middleName; }
        set
        {
            this.middleName = value;
        }
    }
    private string email;
    [Display(Order = 1, GroupName = "Details")]
    public string Email
    {
        get { return email; }
        set
        {
            this.email = value;
        }
    }
    [Display(Order = 0, GroupName = "Details")]
    public string ContactNumber
    {

```

```
get { return contactNo; }  
set  
{  
    this.contactNo = value;  
}  
}  
}
```



#### *Changing group name for group*

You can change the `GroupName` for the group in the `AutoGeneratingDataFormItem` event.

#### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormGroupItem != null && e.DataFormGroupItem.GroupName == "Name")  
        e.DataFormGroupItem.GroupName = "Name Group";  
}
```

#### *Loading different layout for group*

You can load linear or grid layout for the particular group by handling the `AutoGeneratingDataFormItem` event.

By setting the `ColumnCount` property in the data form, non-grouped items only will be arranged in the grid layout. To load the grid layout, set the `ColumnCount` for the `DataFormGroupItem`.

**C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormGroupItem != null && e.DataFormGroupItem.GroupName == "Name")  
        e.DataFormGroupItem.ColumnCount = 2;  
}
```



The image shows a mobile application interface with a blue header bar containing icons for gallery, video, and mail, and status information (signal, 95% battery, 09:46). Below the header is a form titled "Name" with a dropdown arrow. The form contains five input fields: "FirstNam" and "LastNam" (split into two columns), "ContactNumber", "Email", and "BirthDate". Each field has a corresponding input line. At the bottom is a grey navigation bar with four icons: a circle, a double line, a square, and a left arrow.

▼ **Name**

FirstNam \_\_\_\_\_ LastNam \_\_\_\_\_

ContactNumber \_\_\_\_\_

Email \_\_\_\_\_

BirthDate \_\_\_\_\_

Loading linear and grid layout for the group

### C#

```
public class ContactsInfo
{
    private string lastName;
    private string contactNo;
    private string email;
    private DateTime? birthDate;
    public ContactsInfo()
    {
    }
    private string firstName;
    [Display(GroupName = "Name")]
    public string FirstName
    {
        get { return this.firstName; }
        set
        {
            this.firstName = value;
        }
    }
    [Display(GroupName = "Name")]
    public string LastName
    {
        get { return this.lastName; }
        set
        {
            this.lastName = value;
        }
    }
    [Display(GroupName = "Details", ShortName = "ContactNo.")]
    public string ContactNumber
    {
        get { return contactNo; }
        set
        {
            this.contactNo = value;
        }
    }
    [Display(GroupName = "Details")]
    public string Email
    {
        get { return email; }
        set
        {
            email = value;
        }
    }
    [Display(GroupName = "Details")]
    public DateTime? BirthDate
    {
        get { return birthDate; }
        set
        {
            birthDate = value;
        }
    }
}
```

```
}  
}
```

### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormGroupItem != null && e.DataFormGroupItem.GroupName == "Name")  
        e.DataFormGroupItem.ColumnCount = 2;  
}
```

In the following image, for the **Name** group, the grid layout is loaded and for the **Details** group, linear layout is loaded:

The image shows a mobile application interface with a blue header bar containing icons for gallery, video, and mail, along with status icons for signal, Wi-Fi, 95% battery, and the time 09:46. Below the header is a form titled "Name" with a dropdown arrow. The form contains five input fields: "FirstNam" and "LastNam" are side-by-side; "ContactNumber", "Email", and "BirthDate" are stacked vertically. Each field has a horizontal underline. At the bottom is a grey navigation bar with four icons: a circle, a double line, a square, and a left arrow.

▼ **Name**

FirstNam \_\_\_\_\_ LastNam \_\_\_\_\_

ContactNumber \_\_\_\_\_

Email \_\_\_\_\_

BirthDate \_\_\_\_\_

### Setting different column count

You can also set different **ColumnCount** for each group.

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormGroupItem != null)
    {
        if (e.DataFormGroupItem.GroupName == "Name")
            e.DataFormGroupItem.ColumnCount = 2;
        else if (e.DataFormGroupItem.GroupName == "Details")
            e.DataFormGroupItem.ColumnCount = 3;
    }
}
```

▼ **Name**

FirstNam \_\_\_\_\_ LastNam \_\_\_\_\_

▼ **Details**

Contæ \_\_\_\_\_ Email \_\_\_\_\_ Birth[ \_\_\_\_\_

### *Loading group in collapsed state*

By default, the group will be loaded in expanded state. You can collapse the group by setting the [IsExpanded](#) property to false in the [DataFormGroupItem](#).

### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormGroupItem != null && e.DataFormGroupItem.GroupName == "Name")
        e.DataFormGroupItem.IsExpanded = false;
}
```

### *Restricting the group expanding and collapsing*

You can set restrict the group being expanded or collapsed by setting the [AllowExpandCollapse](#) to false in the [DataFormGroupItem](#).

In this case, the group will be shown without expander.

The image displays a mobile application interface for a data form. At the top, a blue status bar contains icons for gallery, video, and mail on the left, and signal, Wi-Fi, 96% battery, and 09:48 on the right. Below the status bar, the form is titled "Name" in bold. It features six input fields, each with a label to its left and a horizontal line for text entry: "FirstName", "LastName", "ContactNumber", "Email", and "BirthDate". The form is set against a light gray background. At the bottom, a dark gray navigation bar contains four white icons: a circle, a stylized "N" shape, a square, and a left-pointing arrow.

**Name**

FirstName \_\_\_\_\_

LastName \_\_\_\_\_

ContactNumber \_\_\_\_\_

Email \_\_\_\_\_

BirthDate \_\_\_\_\_



### *Programmatically expand or collapse group*

You can expand or collapse the group programmatically by using [ExpandGroup](#) and [CollapseGroup](#) methods respectively.

#### **C#**

```
dataForm.ExpandGroup("Group1");  
dataForm.CollapseGroup("Group1");
```

### *Changing DataFormGroupItem visibility*

You can change the [DataFormGroupItem](#) visibility by using the [IsVisible](#) property in the [DataFormGroupItem](#).

#### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if(e.DataFormGroupItem != null )  
    {  
        if (e.DataFormGroupItem.GroupName == "Details")  
            e.DataFormGroupItem.IsVisible = false;  
    }  
}
```

Here, the **Details** group will be hidden.

### *Customizing DataFormLayoutManager*

To customize the layout, override the [DataFormLayoutManager](#) and assign to the [SfDataForm.LayoutManager](#) property.

#### **C#**

```
public class DataFormLayoutManagerExt : DataFormLayoutManager  
{  
    public DataFormLayoutManagerExt(SfDataForm dataForm) : base(dataForm)  
    {  
    }  
}  
dataForm.LayoutManager = new DataFormLayoutManagerExt(dataForm);
```

### *Changing editor padding*

You can change the editor padding by overriding the [GetLeftPaddingForEditor](#) method.

#### **C#**

```
public class ContactsInfo  
{  
    private string lastName;  
    public ContactsInfo()  
    {  
    }  
    private string firstName;  
    [DisplayOptions(ImageSource = Resource.Drawable.Name)]
```

```
public string FirstName
{
    get { return this.firstName; }
    set
    {
        this.firstName = value;
    }
}
[Display(Prompt = "Enter last name")]
[DisplayOptions(ShowLabel = false)]
public string LastName
{
    get { return this.lastName; }
    set
    {
        this.lastName = value;
    }
}
}
```

### **C#**

```
dataForm.LayoutManager = new DataFormLayoutManagerExt(dataForm);
public class DataFormLayoutManagerExt : DataFormLayoutManager
{
    public DataFormLayoutManagerExt(SfDataForm dataForm) : base(dataForm)
    {
    }
    protected override int GetLeftPaddingForEditor(DataFormItem dataFormItem)
    {
        if (dataFormItem.Name == "LastName")
            return 35;
        return base.GetLeftPaddingForEditor(dataFormItem);
    }
}
```

Here, the LastName padding is customized.

The image displays a mobile application interface. At the top, there is a blue header bar containing three icons: a camera, a video camera, and an envelope. Below the header bar is a status bar showing various system icons (mute, Wi-Fi, cellular signal) and the text '100%' followed by a battery icon and the time '10:04'. The main content area is a light gray rectangle. It features a form with four horizontal input lines. The first line contains the placeholder text 'Enter last name'. The bottom of the screen is occupied by a dark gray navigation bar with four white icons: a circle, a double line, a square, and a left-pointing arrow.

### *Customizing label and editor*

By using `DataFormLayoutManager` class, you can customize the generated label by overriding the `GenerateViewForLabel` method and also you can customize the editor by overriding the `OnEditorCreated` method. Here, `BackgroundColor` and `TextColor` of label and editor is customized.

#### **C#**

```
public class ContactsInfo
{
    public ContactsInfo()
    {
    }
    private string firstName = "Iris";
    public string FirstName
    {
        get { return this.firstName; }
        set
        {
            this.firstName = value;
        }
    }
    private string lastName;
    public string LastName
    {
        get { return this.lastName; }
        set
        {
            this.lastName = value;
        }
    }
}
```

#### **C#**

```
dataForm.LayoutManager = new DataFormLayoutManagerExt(dataForm);
public class DataFormLayoutManagerExt : DataFormLayoutManager
{
    public DataFormLayoutManagerExt(SfDataForm dataForm) : base(dataForm)
    {
    }
    protected override View GenerateViewForLabel(DataFormItem dataFormItem)
    {
        var label = base.GenerateViewForLabel(dataFormItem);
        if (label is Label)
        {
            (label as Label).BackgroundColor = Color.FromHex("#ff9522");
            (label as Label).TextColor = Color.White;
        }
        return label;
    }
    protected override void OnEditorCreated(DataFormItem dataFormItem, View editor)
    {
        if (editor is Entry)
        {
            (editor as Entry).TextColor = Color.White;
            editor.BackgroundColor = Color.FromHex("#0073dc");
        }
    }
}
```

```
}  
}
```

The screenshot displays a mobile application interface with a light gray background. At the top, there is a blue status bar containing the text "100% 16:55". Below the status bar, there is a data form with two rows. The first row has an orange input field labeled "FirstName" and a blue input field labeled "Iris". The second row has an orange input field labeled "LastName" and a blue input field. At the bottom of the screen, there is a gray navigation bar with four icons: a circle, a double line, a square, and a left arrow.

FirstName	Iris
LastName	

### Label width customization

You can set label and editor width proportionally by using [LabelWidth](#) and [EditorWidth](#) properties.

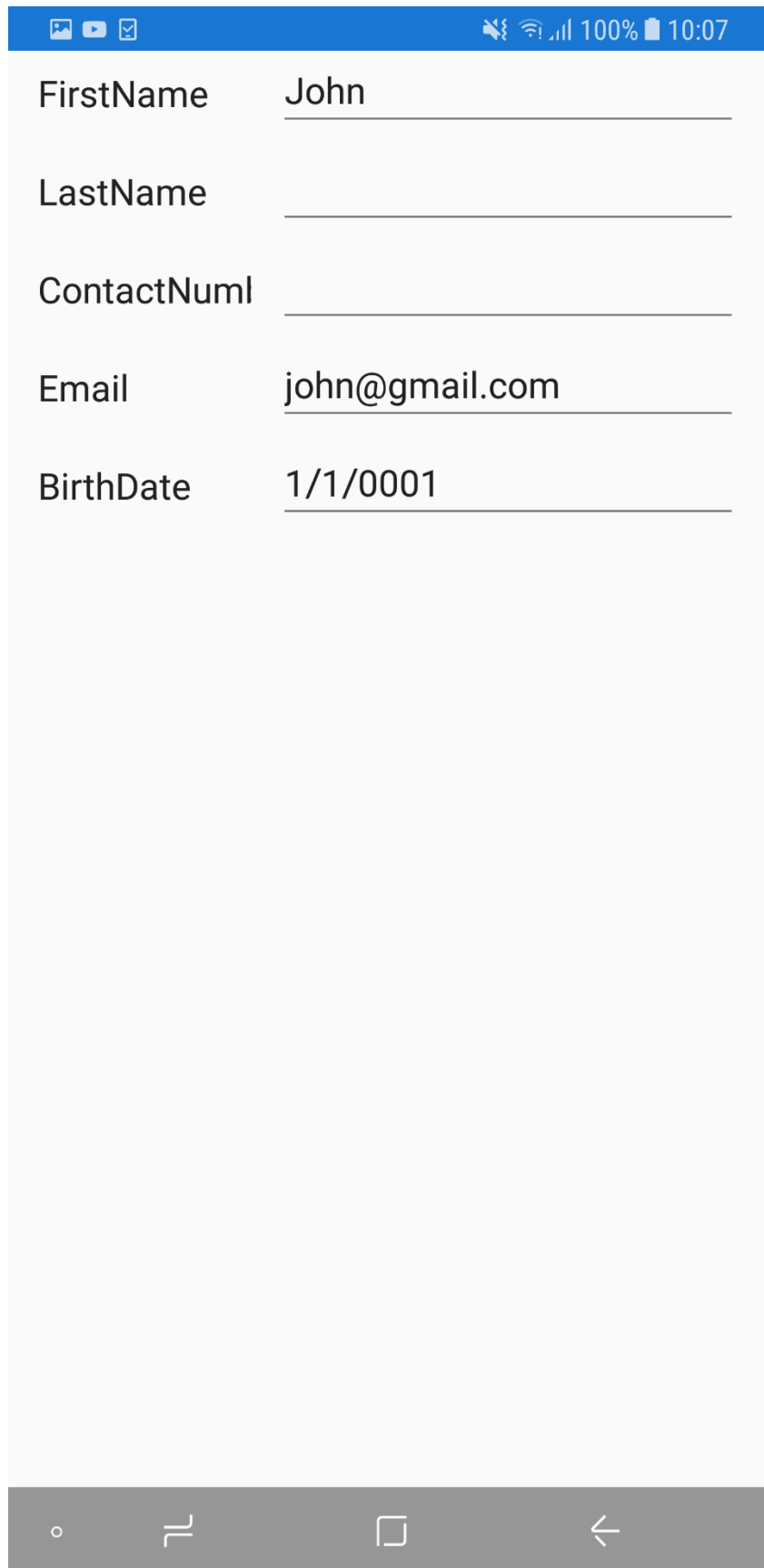
#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:dataForm="clr-namespace:Syncfusion.XForms.DataForm;assembly=Syncfusion.SfDataForm.XForms"
x:Class="GettingStarted.MainPage">
  <ContentPage.Content>
    <dataForm:SfDataForm x:Name="dataForm" LabelWidth="1" EditorWidth="2"/>
  </ContentPage.Content>
</ContentPage>
```

#### C#

```
dataForm.LabelWidth = 1;
dataForm.EditorWidth = 2;
```

Here, the available width is divided into proportionally for editor (2) and label (1).



The image shows a mobile application interface with a data form. At the top is a blue status bar with icons for gallery, video, mail, and system status (signal, Wi-Fi, 100% battery, 10:07). The form has five rows, each with a label on the left and a text input field on the right. The inputs are: 'John' for 'FirstName', an empty field for 'LastName', an empty field for 'ContactNuml', 'john@gmail.com' for 'Email', and '1/1/0001' for 'BirthDate'. At the bottom is a grey navigation bar with four icons: a circle, a double line, a square, and a left arrow.

FirstName	John
LastName	
ContactNuml	
Email	john@gmail.com
BirthDate	1/1/0001



---

**Note:** It is applicable only when `LabelPosition` is Left.

---

By default, the available width is divided equally for editor and label.

#### Spanning rows and columns

You can increase row height and column width by defining the `DisplayOptions` attribute.

#### Row span

You can increase the row height by using the `RowSpan` property in the `DisplayOptions` attribute.

#### C#

```
private string firstName;
[DisplayOptions(RowSpan = 2)]
public string FirstName
{
    get { return this.firstName; }
    set
    {
        this.firstName = value;
    }
}
```

Here, `FirstName` field's row height is increased.

A screenshot of a mobile application interface. At the top is a blue status bar with icons for gallery, video, and mail on the left, and mute, Wi-Fi, cellular signal, 100% battery, and 10:08 on the right. Below the status bar is a light gray form area. The form contains five labels on the left, each followed by a horizontal input line on the right: 'FirstName', 'LastName', 'ContactNuml', 'Email', and 'BirthDate'. At the bottom of the screen is a dark gray navigation bar with four white icons: a circle, a horizontal line with a hook, a square, and a left-pointing arrow.

FirstName

LastName

ContactNuml

Email

BirthDate

### *Column span*

When the grid layout is used, you can increase the column width by using the [ColumnSpan](#) property in the `DisplayOptions` attribute.

#### **C#**

```
dataForm.ColumnCount = 2;
```

#### **C#**

```
private string firstName;  
[DisplayOptions(ColumnSpan = 2)]  
public string FirstName  
{  
    get { return this.firstName; }  
    set  
    {  
        this.firstName = value;  
    }  
}
```

The image shows a mobile application interface with a blue status bar at the top. The status bar contains icons for gallery, video, and mail on the left, and icons for silent mode, Wi-Fi, cellular signal, 100% battery, and the time 10:09 on the right. Below the status bar is a light gray data form. The form has five input fields arranged in two rows. The first row has a single field labeled 'FirstName'. The second row has two fields: 'LastName' on the left and 'ContactN' on the right. The third row also has two fields: 'Email' on the left and 'BirthDate' on the right. At the bottom of the screen is a dark gray navigation bar with four white icons: a circle, a home button, a square, and a back arrow.

FirstName \_\_\_\_\_

LastName \_\_\_\_\_ ContactN \_\_\_\_\_

Email \_\_\_\_\_ BirthDate \_\_\_\_\_

### Change DataFormItem visibility at runtime

You can change the field visibility by using the [IsVisible](#) property in the `DataFormItem`.

#### C#

```
var dataFormItem = dataForm.ItemManager.DataFormItems["Name"];  
if (dataFormItem.Name == "Name")  
{  
    dataFormItem.IsVisible = false;  
}
```

Here, the `Name` field will be hidden.

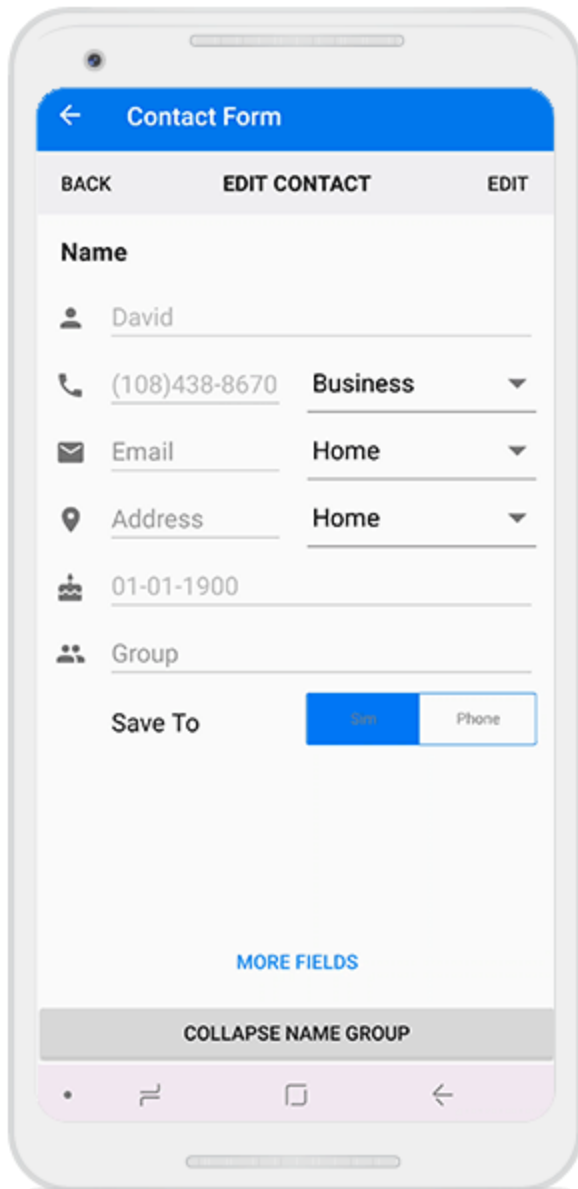
### Change DataFormGroupItem visibility at runtime

You can change the [DataFormGroupItem](#) visibility by using the [IsVisible](#) property in the `DataFormGroupItem`.

#### C#

```
var dataFormGroupItem = dataForm.ItemManager.GetDataFormGroupItem("Name");  
if (dataFormGroupItem.GroupName == "Name")  
{  
    dataFormGroupItem.IsVisible = false;  
}
```

Here, the `Name` group will be hidden.

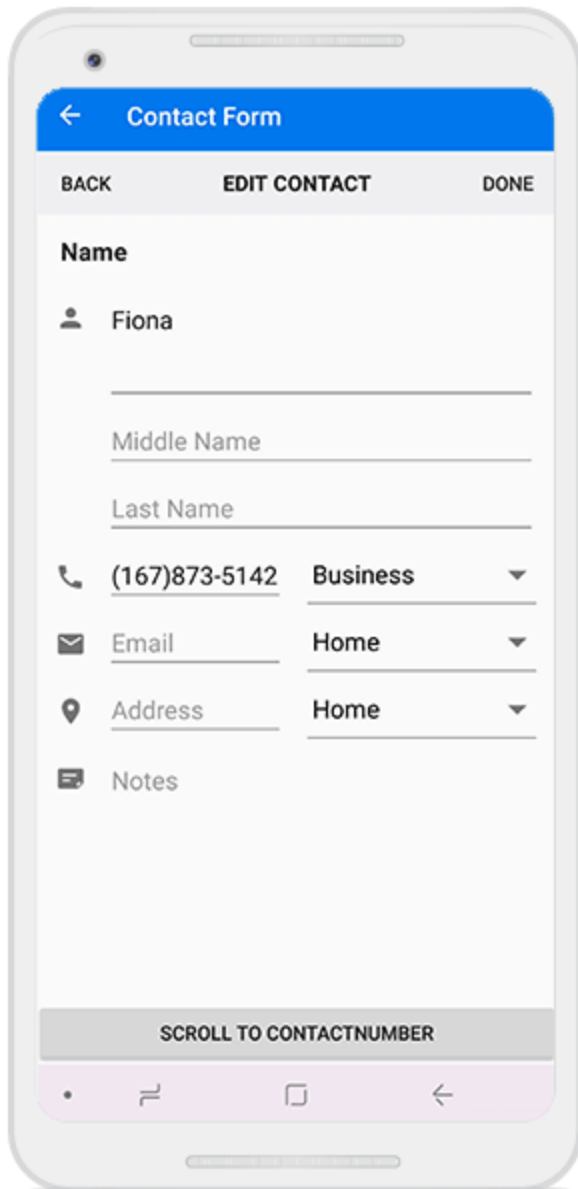


Programmatically scroll to specific editor

You can programmatically scroll to specific editor using the [ScrollTo](#) method by passing the **property name**.

**C#**

```
dataForm.ScrollTo("ContactNumber")
```



### Changing the height of DataFormItem.

You can define the height of each `DataFormItem` using the [Height](#) property, and it will be handled in the `AutoGeneratingDataFormItem` event.

You can define the `Height` as described as follows.

- You can directly set the exact `Height` value; it considers the `GridLength` as `GridUnitType.Absolute`.
- You can use the `GridLength.Auto` to size the height of `DataFormItem`, so that it fits to the label text that it contains.
- You can use the `GridLength.Star` to display the default `DataFormItem` height.

**C#**

```

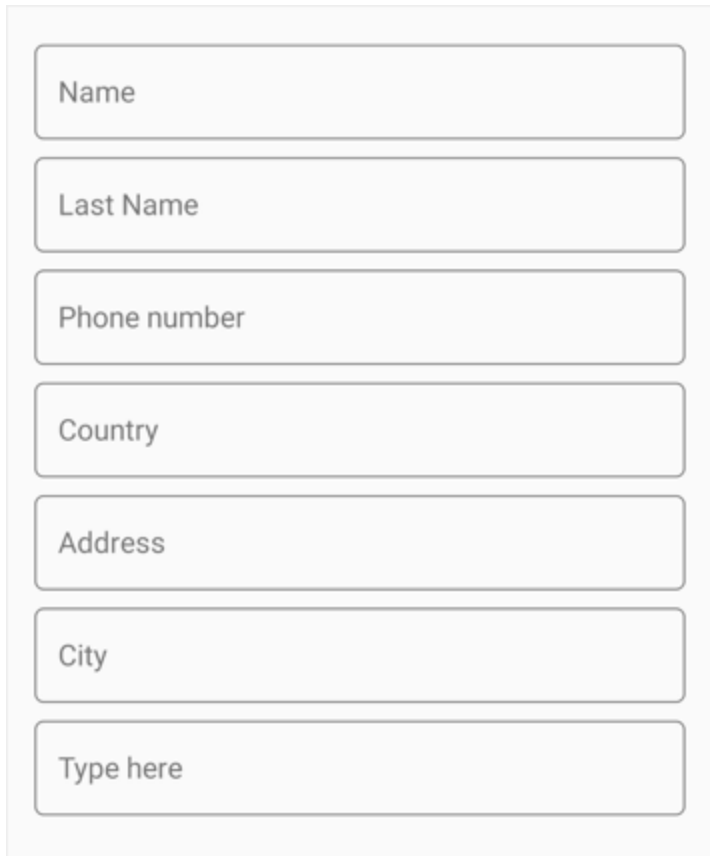
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object
sender, AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "Experience" || e.DataFormItem.Name == "Rating"
|| e.DataFormItem.Name == "Recommend")
        {
            e.DataFormItem.Height = GridLength.Auto;
        }
        if (e.DataFormItem.Name == "Improvement")
        {
            e.DataFormItem.Height = 200;
        }
    }
}

```

**Floating label layout**

The Dataform supports floating label layout with support for assistive labels, leading and trailing icons, and a password toggle icon to show or hide a password. It provides three type of containers such as filled, outlined and none. Floating label layout can be enabled by setting `DataForm.LayoutOptions` to `TextInputLayout`.





### Layout options

By default, the dataform arranges the editors and their labels corresponding to the fields in stack layout. However, to enable the floating label layout set [LayoutOptions](#) property of `DataForm` or [DataFormItem](#) to `TextInputLayout`.

### XML

```
<dataForm:SfDataForm x:Name="dataForm" LayoutOptions="TextInputLayout">
</dataForm:SfDataForm/>
```

### C#

```
dataForm.LayoutOptions = LayoutOptions.TextInputLayout;
```

### Changing layout options of the DataFormItem

You can change layout option by using the [LayoutOptions](#) property in the `DataFormItem` and it will be handled in the `AutoGeneratingDataFormItem` event.

### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem != null)  
    {  
        if (e.DataFormItem.Name == "Name" || e.DataFormItem.Name == "PhoneNumber" ||  
            (e.DataFormItem.Name == "Address" || e.DataFormItem.Name == "City"))  
        {  
            e.DataFormItem.LayoutOptions = LayoutOptions.TextInputLayout;  
        }  
    }  
}
```

Name

Last Name

Phone number

Country

Address

City

### Supported editors

Dataform supports the floating label layout for the following editors.

- Text editor
- Password editor
- MaskedEditText editor
- Numeric editor
- MultilineText editor
- Date editor
- Time editor
- Picker editor
- DropDown editor
- AutoComplete editor.

### Container types

Containers enhance the perspective of dataform editor views and provide some contrast between editor view and assistive labels. Their border and assistive label color will be changed based on the dataform field validation.

By default, the container type is **Outlined**. By using the [ContainerType](#) property in **DataForm** or in **DataFormItem**, you can modify the container type to **Filled** or **None**.

#### Filled

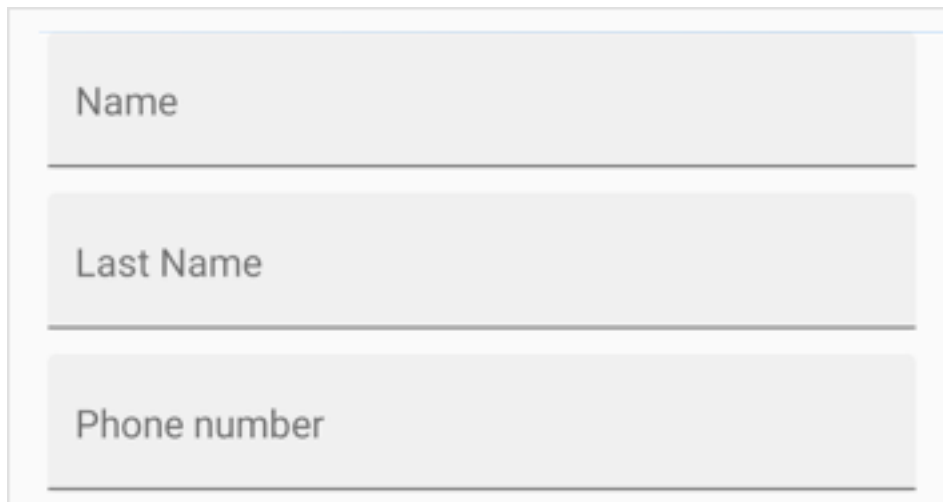
The background color of dataform editor view will be filled with container color and it can be enabled by setting the [ContainerType](#) property to **Filled** in **DataForm** or in **DataFormItem**.

#### XML

```
<dataForm:SfDataForm x:Name="dataForm" ContainerType="Filled">
</dataForm:SfDataForm>
```

#### C#

```
dataForm.LayoutOptions = ContainerType.Filled;
```



#### Outlined

To enable the outlined container type, you can set [ContainerType](#) property to **Outlined** in **DataForm** or in **DataFormItem** which covers the editor view with rounded-corner.

#### XML

```
<dataForm:SfDataForm x:Name="dataForm" ContainerType="Outlined">
</dataForm:SfDataForm>
```

#### C#

```
dataForm.LayoutOptions = ContainerType.Outlined;
```



### None

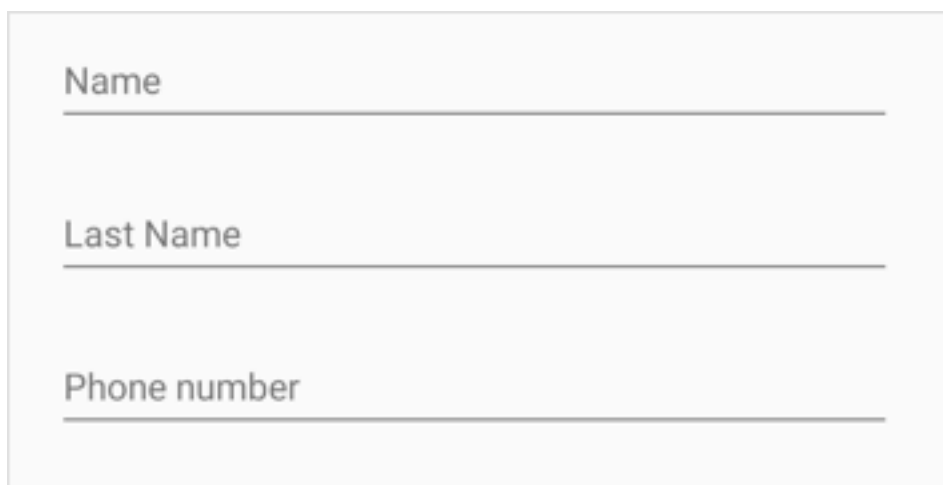
When setting the [ContainerType](#) property to **None** in **DataForm** or in **DataFormItem** container, it will have empty background and enough spacing.

### XML

```
<dataForm:SfDataForm x:Name="dataForm" ContainerType="None">
</dataForm:SfDataForm>
```

### C#

```
dataForm.LayoutOptions = ContainerType.None;
```



### Changing container type of the DataFormItem

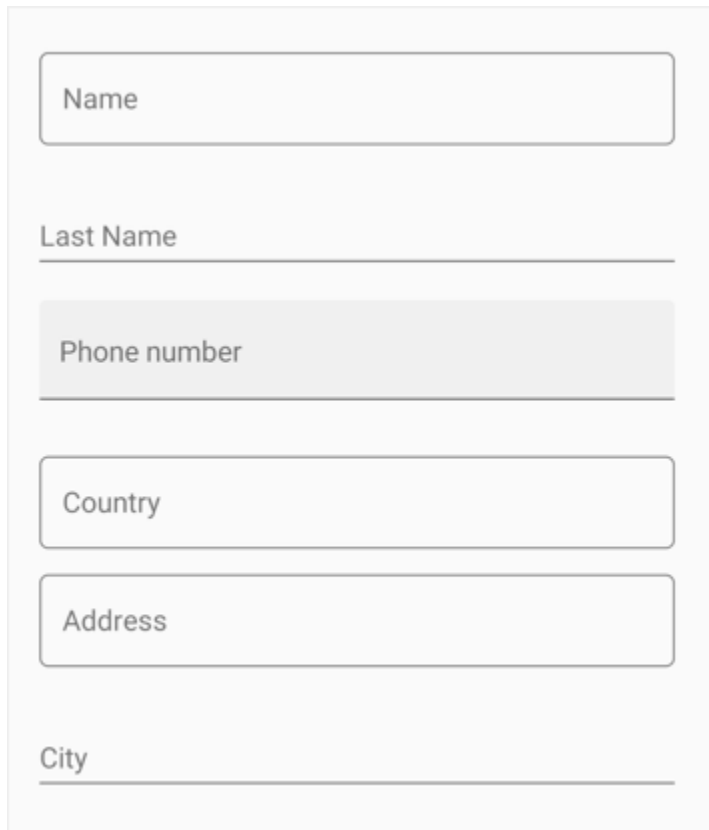
You can change the container type of the [DataFormItem](#) by using the [ContainerType](#) property in the [TextInputLayoutSettings](#) of **DataFormItem** and it will be handled by the **AutoGeneratingDataFormItem** event.

### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

## C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "PhoneNumber")
        {
            e.DataFormItem.ContainerType = ContainerType.Filled;
        }
        if (e.DataFormItem.Name == "LastName")
        {
            e.DataFormItem.ContainerType = ContainerType.None;
        }
    }
}
```



## Leading view

Floating label layout supports leading view, which shows an icon view to the left of editor.

Unicode or font icons for the labels can be displayed as icons. By setting the [LeadingView](#) property in `TextInputLayoutSettings` of `DataFormItem`, a label can be added as a leading icon for editor view. By setting the [LeadingViewPosition](#) property, it can be placed either inside or outside the container. It is positioned outside by default.

**Note:** Refer to the following links to learn more about font icons:

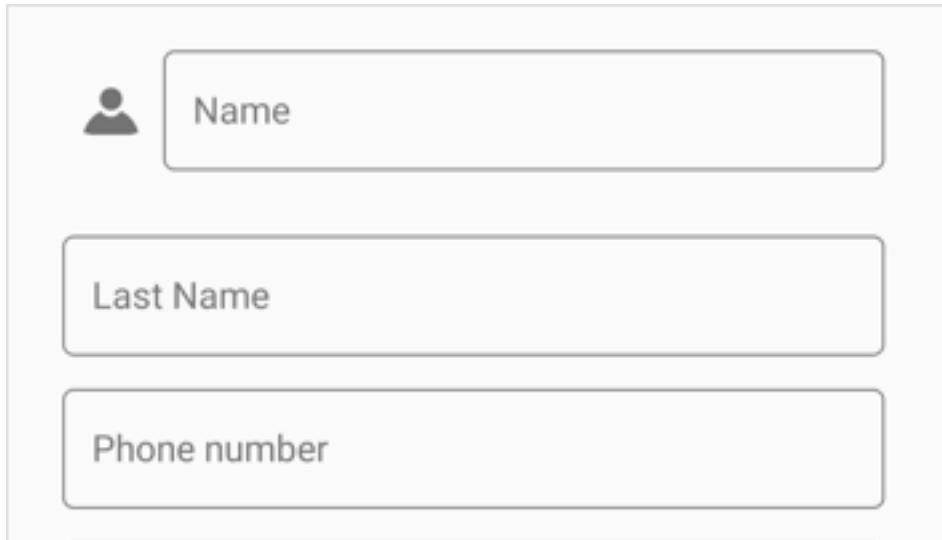
- [How to create font icons using our metro studio and export as ttf?](#)
- [How to set font family for the custom fonts in labels?](#)

### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem1(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "Name")
        {
            e.DataFormItem.TextInputLayoutSettings = new
            Syncfusion.SfDataForm.XForms.TextInputLayoutSettings()
            {
                LeadingView = new Label()
                {
                    VerticalTextAlignment = Device.RuntimePlatform == Device.iOS ?
                    TextAlignment.Center : TextAlignment.Start,
                    FontSize = Device.RuntimePlatform == Device.iOS ? 18 : 24,
                    Text = "A",
                    FontFamily = Device.RuntimePlatform == Device.iOS ? "ContactsIcons" :
                    Device.RuntimePlatform == Device.Android ? "ContactsIcons.ttf#ContactsIcons"
                    : "Assets/Fonts/ContactsIcons.ttf#ContactsIcons"
                }
            };
        }
    }
}
```



### Trailing view

Floating label layout supports trailing view, which shows an icon view to the right of editor.

Unicode or font icons for the labels can be displayed as icons. By setting the [TrailingView](#) property in [TextInputLayoutSettings](#) of [DataFormItem](#), a label can be added as a trailing icon for editor view. By setting the [TrailingViewPosition](#) property, it can be placed either inside or outside the container. It is positioned outside by default.

**Note:** Refer to the following links to learn more about font icons:

- [How to create font icons using our metro studio and export as ttf?](#)
- [How to set font family for the custom fonts in labels?](#)

### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem1(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "PhoneNumber")
        {
            e.DataFormItem.TextInputLayoutSettings = new
            Syncfusion.SfDataForm.XForms.TextInputLayoutSettings()
            {
                TrailingView = new Label()
                {
                    VerticalTextAlignment = Device.RuntimePlatform == Device.iOS ?
                    TextAlignment.Center : TextAlignment.Start,
```

```
FontSize = Device.RuntimePlatform == Device.iOS ? 18 : 24,
Text = "I",
FontFamily = Device.RuntimePlatform == Device.iOS ? "ContactsIcons" :
Device.RuntimePlatform == Device.Android ? "ContactsIcons.ttf#ContactsIcons"
: "Assets/Fonts/ContactsIcons.ttf#ContactsIcons"
}
};
}
}
}
```

Enable password visibility toggle for password editor

Password toggle visibility in floating label layout is used to show or hide the visibility of characters in the dataform password editor. You can enable this toggle by setting the [EnablePasswordVisibilityToggle](#) property to true in [DataFormTextItem](#).

#### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"/>
```

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Password")
    {
        (e.DataFormItem as DataFormTextItem).EnablePasswordVisibilityToggle = true;
    }
}
```



### Assistive label

Assistive labels comprise of floating label or hint label, helper label, counter label and validation label.

### *Reserve space for assistive labels*

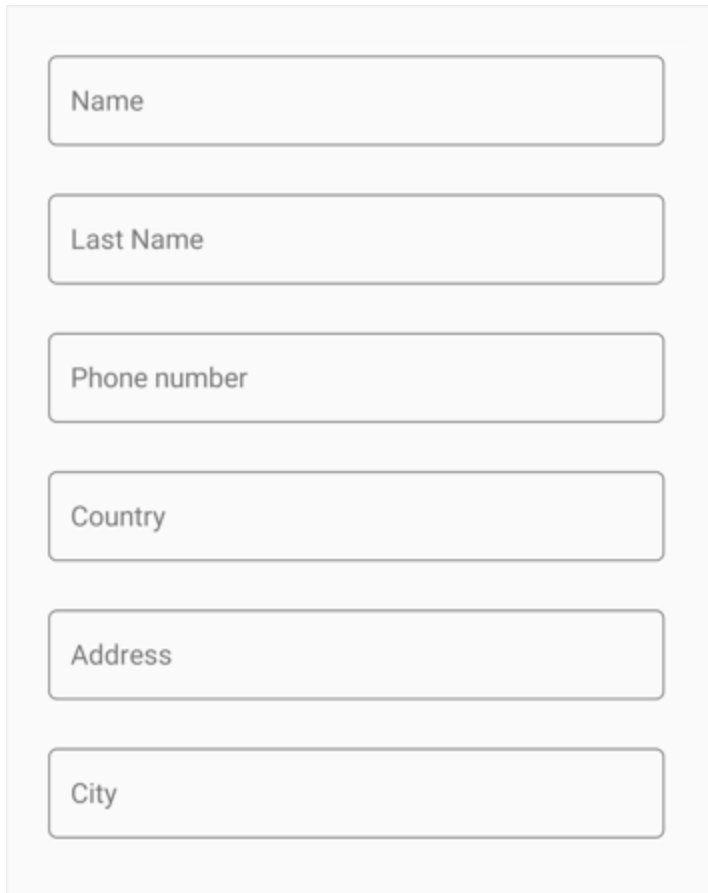
The reserved spaces for assistive labels can be removed by setting the [ReserveSpaceForAssistiveLabels](#) property to false or it can be set to false automatically if there is no prompt value or water mark provider for `DataFormItem` and/or no counter label added.

### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem1(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        e.DataFormItem.TextInputLayoutSettings = new
Syncfusion.SfDataForm.XForms.TextInputLayoutSettings()
        {
            ReserveSpaceForAssistiveLabels= true
        };
    }
}
```



#### *Hint label*

The data field caption will be displayed as floating or hint label for the editor, and it will be enabled by using the `Name` or [LabelText](#) property.

#### *Helper label*

The helper label is used to display the water mark for the editor to provide hint for users and it can be set using [Prompt](#).

The visibility of helper text in DataForm floating label layout can be collapsed by setting [ShowHelperText](#) to false in `DataForm` or [TextInputLayoutSettings](#). In this case, DataForm Prompt text will be displayed as editor view hint text if the hint label visibility is collapsed using the `ShowLabel` property of [DisplayOption](#) or `DataFormItem`.

#### **XML**

```
<dataForm:SfDataForm x:Name="dataForm" LayoutOptions="TextInputLayout"
ShowHelperText="False">
</dataForm:SfDataForm/>
```

#### **C#**

```
dataForm.ShowHelperText = false;
```

### Changing helper text visibility of DataFormItem

You can remove the helper label of [DataFormItem](#) by setting the [ShowHelperText](#) property to false in [TextInputLayoutSettings](#), and it will be handled using the [AutoGeneratingDataFormItem](#) event.

#### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "PhoneNumber")
        {
            e.DataFormItem.TextInputLayoutSettings = new
            Syncfusion.SfDataForm.XForms.TextInputLayoutSettings()
            {
                ShowHelperText = false
            };
        }
    }
}
```



### Validation label

The validation label used to display the dataform validation messages such as valid or invalid data. Refer [validation](#) to learn more about dataform validation.

*Counter label*

The counter label is can be used to notify the character count limitation in given validation. It can be enabled by setting the [ShowCharCount](#) property to `true`. Character limit can be set using the `StringLength` attribute.

**XML**

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

**C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null && e.DataFormItem.Name == "Password")
    {
        e.DataFormItem.ShowCharCount = true;
    }
}
private string _Password;
[Required(AllowEmptyStrings = false, ErrorMessage = "Password should not be
empty")]
[StringLength(10, ErrorMessage = "Password should not exceed 10
characters")]
public string Password
{
    get { return this._Password; }
    set
    {
        this._Password = value;
        this.RaisePropertyChanged("Password");
    }
}
```

### Appearance customization

The assistive labels and border color can be customized based on the editor view states and data validation.

#### Changing outline corner radius

When setting the [OutlineCornerRadius](#) property to double value, the corner radius of the container will be changed.

#### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem1(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        e.DataFormItem.TextInputLayoutSettings = new
Syncfusion.SfDataForm.XForms.TextInputLayoutSettings()
        {
            OutlineCornerRadius = 30
        };
    }
}
```



#### Focused color

When the given editor view is focused, the [FocusedColor](#) property value will be applied to the label text and border.

#### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

**C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem1(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "Name")
        {
            e.DataFormItem.FocusedColor = Color.Violet;
        }
    }
}
```

*Unfocused color*

When the given editor view is unfocused, the [UnfocusedColor](#) property value will be applied to the label text and border.

**XML**

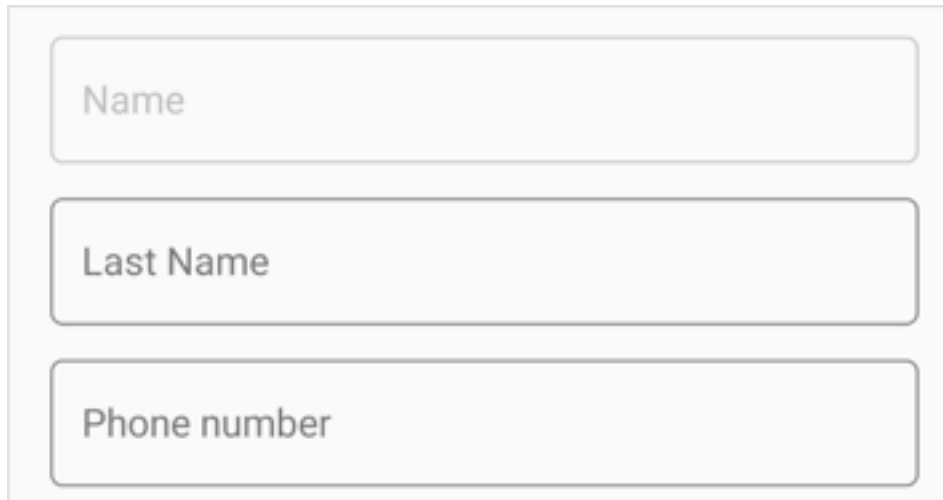
```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

**C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem1(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "Name")
        {

```

```
{  
    e.DataFormItem.UnfocusedColor = Color.Silver;  
}  
}
```



#### *Error message color*

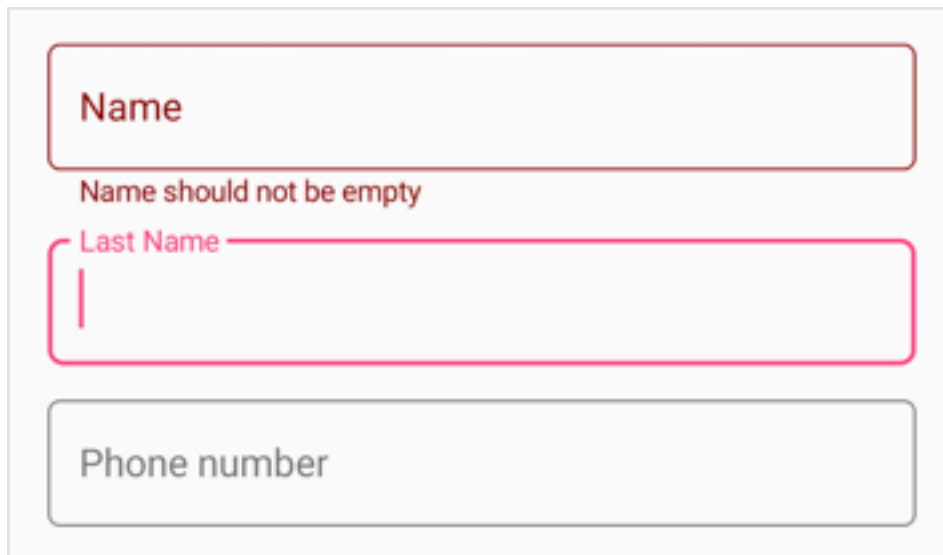
In case of invalid data, the [ErrorMessageColor](#) will be applied to the error label, hint label and border.

#### **XML**

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"  
    AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">  
</dataForm:SfDataForm>
```

#### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem1(object sender,  
    AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem != null)  
    {  
        if (e.DataFormItem.Name == "Name")  
        {  
            e.DataFormItem.ErrorMessageColor = Color.DarkRed;  
        }  
    }  
}
```



#### *Valid message color*

In case of valid data, the [ValidMessageColor](#) will be applied to the valid message label, hint label and border.

#### **XML**

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

#### **C#**

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem1(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "Name")
        {
            e.DataFormItem.ValidMessageColor = Color.BlueViolet;
        }
    }
}
```



Customize the background color of editor view

The floating label layout editor view background color can be customized by setting the [ContainerBackgroundColor](#) property of `DataForm` or [TextInputLayoutSettings](#). It is applicable when the [ContainerType](#) is set to `Filled` or `Outlined`.

#### XML

```
<dataForm:SfDataForm x:Name="dataForm" LayoutOptions="TextInputLayout"
ContainerBackgroundColor="Silver">
</dataForm:SfDataForm/>
```

#### C#

```
dataForm.ContainerBackgroundColor = Color.Silver;
```

*Customize the editor view background color of DataFormItem*

You can change the editor view background color of [DataFormItem](#) using the [ContainerBackgroundColor](#) property in [TextInputLayoutSettings](#), and it will be handled using the `AutoGeneratingDataFormItem` event.

#### XML

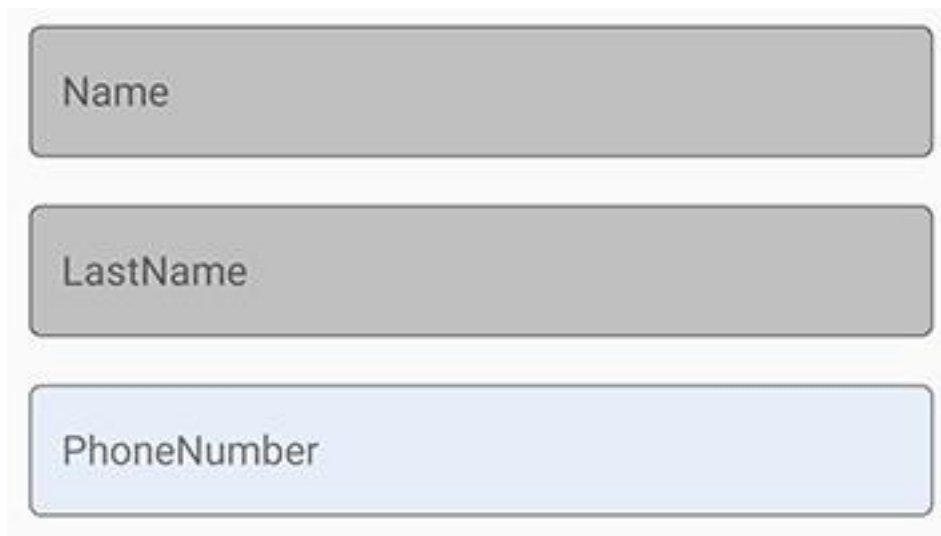
```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"
AutoGeneratingDataFormItem="DataForm_AutoGeneratingDataFormItem">
</dataForm:SfDataForm>
```

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.Name == "PhoneNumber")
        {

```

```
e.DataFormItem.TextInputLayoutSettings = new  
Syncfusion.SfDataForm.XForms.TextInputLayoutSettings()  
{  
    ContainerBackgroundColor = Color.FromHex("#E6EEF9")  
};  
}  
}
```



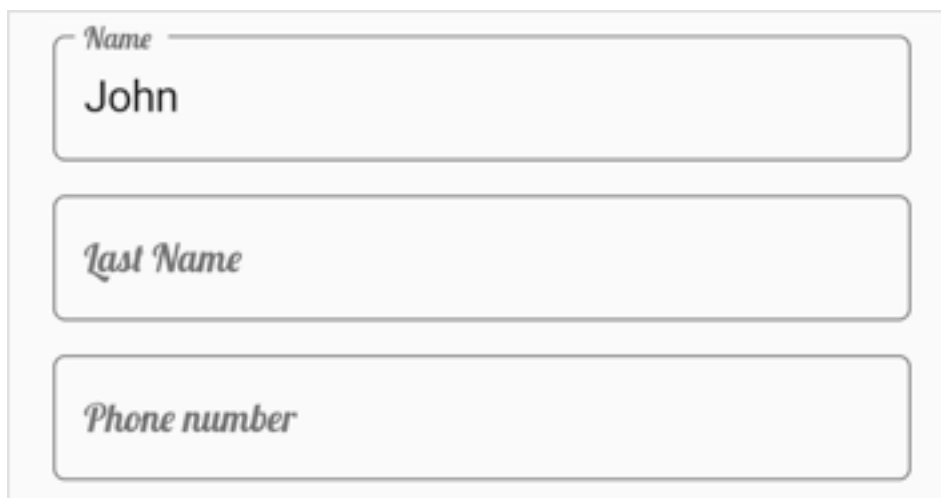
The image shows three text input fields stacked vertically. Each field has a light gray background and rounded corners. The first field is labeled 'Name', the second 'LastName', and the third 'PhoneNumber'. The labels are positioned at the top left of each field, partially overlapping the input area.

#### Font customization

You can customize the font of assistive labels by setting the `FontFamily`, `FontSize`, and `FontAttributes` properties of the `LabelStyle` property.

#### Hint label style

You can customize the text of hint label by setting the `FontFamily`, `FontSize`, and `FontAttributes` properties of [HintLabelStyle](#) in `DataFormItem`.



The image shows three text input fields stacked vertically. Each field has a light gray background and rounded corners. The first field is labeled 'Name' with the text 'John' entered. The second field is labeled 'Last Name' and is empty. The third field is labeled 'Phone number' and is empty. The labels are positioned at the top left of each field, partially overlapping the input area.

#### Helper label style

You can customize the text of helper label by setting the `FontFamily`, `FontSize`, and `FontAttributes` properties of [HelperLabelStyle](#) in `DataFormItem`.



#### Validation label style

You can customize the text of validation label by setting the `FontFamily`, `FontSize`, and `FontAttributes` properties of [ValidationLabelStyle](#) in `DataFormItem`.

#### XML

```
<dataForm:SfDataForm x:Name="dataForm" DataObject="{Binding ContactsInfo}"/>
```

#### C#

```
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;  
private void DataForm_AutoGeneratingDataFormItem(object sender,  
AutoGeneratingDataFormItemEventArgs e)  
{  
    if (e.DataFormItem != null && e.DataFormItem.Name == "Name")  
    {  
        e.DataFormItem.HintLabelStyle = new LabelStyle() { FontFamily =  
            Device.OnPlatform("Lobster-Regular", "Lobster-Regular.ttf#Lobster-Regular",  
                "Assets/Fonts/Lobster-Regular.ttf#Lobster"), FontSize = 16};  
    }  
}
```

### Limitations

- It is recommended not to provide background color for editor view while customizing existing editor inside floating label layout.
- It is recommended not to use non-editable custom editor views inside floating label layout such as range slider, etc.
- It is recommended not to use the [LayoutOptions](#) attribute for setting default layout for data fields.
- It is recommended not to use null values in [Date Editor](#) and [Time Editor](#) with text input layout in initial loading time and runtime in iOS platform.

### Unsupported editors

Floating label layout do not support for the following non-editable editors.

- RadioGroup editor
- Segment editor
- CheckBox editor
- Switch editor

---

**Note:** While using unsupported editors with `DataForm` layout option as `TextInputLayout` we can set layout options for unsupported editors' data fields using this [link](#)

---

You can download the [DataForm with floating labels](#) sample in GitHub.

### Data Annotations

The data form supports the following attribute, and these attributes can be accessible using `System.ComponentModel.DataAnnotations` assembly.

#### Display attribute

Property	Details
----------	---------

<a href="#">Name</a>	Specifies the label text.
<a href="#">GroupName</a>	Specifies the group name which groups the fields in the data form. Refer to <a href="#">here</a> for more details.
<a href="#">ShortName</a>	Specifies the label text. It takes higher priority than Name.
<a href="#">AutoGenerateField</a>	Specifies whether the field should be auto generated or not. Refer to <a href="#">here</a> for more details.
<a href="#">Prompt</a>	Specifies watermark text for the editor. Refer to <a href="#">here</a> for more details.
<a href="#">Order</a>	Specifies the order of field in the data form. Refer to <a href="#">here</a> for more details.

#### Validation attributes

Property	Details
MinLength	Specifies the required minimum length. Refer to <a href="#">[here](https://help.syncfusion.com/xamarin/sfdataform/validation#data-annotations)</a> for more details.
MaxLength	Specifies the required maximum length. Refer to <a href="#">{{'here'  markdownify }}</a> for more details.
Required	Specifies the required data field value. Refer to <a href="#">[here](https://help.syncfusion.com/xamarin/sfdataform/validation#data-annotations)</a> for more details.
Range	Specifies the maximum and minimum values. Refer to <a href="#">{{'here'  markdownify }}</a> for more details.

#### Bindable attribute

It specifies whether the field should be auto generated or not. Refer to [here](#) for more details.

#### Editable attribute

It specifies whether the data field is editable or not.

#### ReadOnly attribute

It specifies whether the data field is read only or not. Refer to [here](#) for more details.

#### EnumDataType attribute

It specifies enum type for the data field.

#### DataType attribute

It specifies data type for the field.

Supported data types are Text, MultilineText, Date, DateTime, Time, and Currency.

Refer to [here](#) for more details.

### DisplayFormat attribute

It specifies how data fields are displayed and formatted.

Supported data types are currency, percentage, date, time.

Property	Details
DataFormatString	Specifies the display format of the field value.

Editor	FormatString
Currency	{0:C}
Percentage	{0:P}
Date	{0:D}
Time	{0:T}

### C#

```

dataForm.DataObject = new ContactInfo();
public class ContactInfo
{
    private int markPercentage;
    [DisplayFormat(DataFormatString = "{0:P}")]
    public String MarkPercentage
    {
        get
        {
            return markPercentage;
        }
        set
        {
            markPercentage = value;
        }
    }
}

```

### CustomDataType attribute

The Percent data type is supported. Refer to [here](#) for more details.

### Custom attribute

The data form supports the following custom attribute, and these attributes can be accessible using `Syncfusion.SfDataForm.XForms`.

### DisplayOptions attribute

Property	Details
<a href="#">RowSpan</a>	Specifies the row span for the data form item. Refer to <a href="#">here</a> for more details.

<a href="#">ColumnSpan</a>	Specifies the column span for the data form item. Refer to <a href="#">here</a> for more details.
<a href="#">ValidMessage</a>	Specifies positive message to be shown when validation is passed. Refer to <a href="#">here</a> for more details.
<a href="#">ImageSource</a>	Specifies the image source for loading image instead of label. Refer to <a href="#">here</a> for more details.
<a href="#">ShowLabel</a>	Specifies whether the label should be visible or not. Refer to <a href="#">here</a> for more details.

*Converter attribute*

Property	Details
<a href="#">ConverterType</a>	Specifies the Converter type which converts the original value in different format or as different value. Refer to <a href="#">here</a> for more details.

*DateRange attribute*

Property	Details
<a href="#">MinYear</a>	Specifies the required minimum year.
<a href="#">MinMonth</a>	Specifies the required minimum month.
<a href="#">MinDay</a>	Specifies the required minimum day.
<a href="#">MaxYear</a>	Specifies the required maximum year.
<a href="#">MaxMonth</a>	Specifies the required maximum month.
<a href="#">MaxDay</a>	Specifies the required maximum day.

Refer to [here](#) for more details.

*Localization*

You can localize the DataFormItem [Display](#) attribute values and validation ([Required](#), [StringLength](#)) attributes values by using `ResourceType` display attribute or using the `AutoGeneratingDataFormItem` event.

Please refer the [Localization](#) document to localize the application.

Based on the culture specifies the corresponding culture string value of display attribute in Resource (.Resx) file as mentioned in document.

*Localizing data form item display values*

Here, the display attributes or data form item display values get localized based on culture from Localization Resource File (.Resx).

### Using attribute

**ResourceType** [Display](#) attribute specifies the Resources File (.Resx) which is used to localize the Display attribute of **Name**, **ShortName**, **GroupName** and **Prompt** values.

### C#

```
[Display(Name = "FirstName", Prompt = "EnterFirstName", GroupName = "Name",
ResourceType = typeof(Localization))]
public String FirstName { get; set; }
```

### Using event

You can also localize the DataFormItem **LabelText**, **PlaceholderText**, **GroupName** in the **AutoGeneratingDataFormItem** event of SfDataForm by using the Resources (.Resx) file.

Here, string member of .resx file will be accessed through the class (in resxFilename.Designer.cs) which was auto-generated when .resx file created and static string members get localized using [ResourceManager](#) based on culture.

### C#

```
[Display(GroupName = "Name")]
public String FirstName { get; set; }
dataForm.AutoGeneratingDataFormItem += DataForm_AutoGeneratingDataFormItem;
private void DataForm_AutoGeneratingDataFormItem(object sender,
AutoGeneratingDataFormItemEventArgs e)
{
    if (e.DataFormItem != null)
    {
        if (e.DataFormItem.LabelText == "FirstName")
        {
            e.DataFormItem.LabelText = Localization.FirstName;
            e.DataFormItem.PlaceholderText = Localization.EnterFirstName;
        }
        if (e.DataFormItem.GroupName == "Name")
        {
            e.DataFormItem.GroupName = Localization.Name;
        }
    }
}
```

### Localizing validation error messages

Here, the validation (**Required**, **StringLength**) attributes or data form error messages get localized based on culture from Localization Resource File (.Resx).

### Using attribute

The **Required** and **StringLength** attributes error message can be localized using [ErrorMessageResourceType](#) and [ErrorMessageResourceName](#) properties which are used to get a localized error messages from Localization Resource File (.Resx) based on culture.

### C#

```
[Required(AllowEmptyStrings = false, ErrorMessageResourceType =
typeof(Localization), ErrorMessageResourceName = "FirstNameEmpty")]
```



```
[StringLength(25,MinimumLength =5,ErrorMessageResourceType =  
typeof(Localization), ErrorMessageResourceName = "FirstNameLength")]  
public String FirstName { get; set; }
```

### Using event

You can also localize the data form error message in the **Validating** event of SfDataForm by using the Resources (.Resx) file.

Here, string member of .resx file will be accessed through the class (in resxFilename.Designer.cs) which was auto-generated when .resx file created and static string members get localized using [ResourceManager](#) based on culture.

### C#

```
dataForm.Validating += DataForm_Validating;  
private void DataForm_Validating(object sender, ValidatingEventArgs e)  
{  
    if (e.PropertyName == "FirstName")  
    {  
        if (e.Value.ToString().Length == 0)  
        {  
            e.IsValid = false;  
            e.ErrorMessage = Localization.FirstNameEmpty;  
        }  
        else if (e.Value != null && e.Value.ToString().Length > 0 &&  
            !(e.Value.ToString().Length >= 5))  
        {  
            e.IsValid = false;  
            e.ErrorMessage = Localization.FirstNameLength;  
        }  
    }  
}
```



You can download the entire source code of this demo for Xamarin.Forms using attributes from here [LocalizationThroughAttribute](#) and using event from here [LocalizationThroughEvent](#).

### Right to left (RTL)

SfDataForm supports to change the layout direction of the control in the right-to-left direction by setting the [FlowDirection](#) to `RightToLeft` or by changing the device language.

#### XML

```
<dataForm:SfDataForm FlowDirection="RightToLeft">  
</dataForm:SfDataForm>
```

#### C#

```
dataForm.FlowDirection = FlowDirection.RightToLeft;
```

**Note**

For implementing the `FlowDirection` in the control, Xamarin.Forms package version must be 3.0 and above. Please refer [RightToLeft](#) to get more details about `RightToLeft` flow direction in Xamarin.Forms.

*Android*

For Android, add `android:supportsRtl="true"` in your application tag of `AndroidManifest.xml` file, and make sure your `MinSDKVersion` is 17+. By changing the device language / enabling the device's `Force RTL layout` can achieve the `RightToLeft` layout direction in DataForm.

**XML**

```
<manifest ... >
<uses-sdk android:minSdkVersion="17" ... />
<application ... android:supportsRtl="true">
</application>
</manifest>
```

*iOS*

For iOS, add the `RightToLeft` language in the `CFBundleLocalizations` section of your `Info.plist` file, and make sure you're targeting iOS 9+.

**XML**

```
<resources>
<key>CFBundleDevelopmentRegion</key>
<string>en</string>
<key>CFBundleLocalizations</key>
<array>
<string>en</string>
<string>ar</string>
</array>
</resources>
```

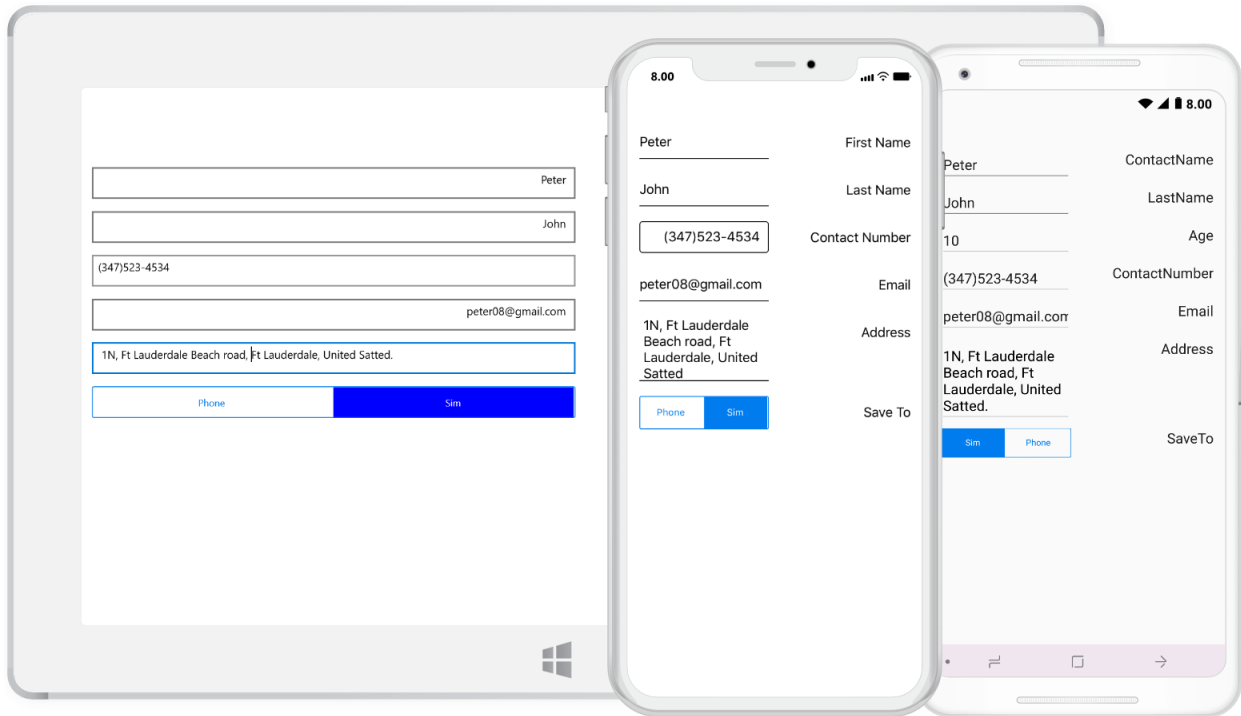
Localization native development region	String en
▼ Localizations	Array (2 items)
	String en
	String ar

*UWP*

For UWP, you need to set `FlowDirection` to `RightToLeft` in the `MainPage.cs` file of the `UWP` project.

**C#**

```
public MainPage ()
{
    ...
    this.FlowDirection = FlowDirection.RightToLeft;
    LoadApplication (new App ());
    ...
}
```



### AutomationId

The **SfDataForm** control has built-in **AutomationId** support for inner elements. Please find the following table of Automation IDs for inner elements.

Editor	AutomationId Format	Example
Text editor	"Focus" + LabelText	FocusFirstName
Multiline text editor	"Focus" + LabelText	FocusAddress
Password Editor	"Focus" + LabelText	FocusPassword
Switch Editor	"Focus" + LabelText	FocusRegistered
Picker	"Focus" + LabelText	FocusName
Date Editor	"Focus" + LabelText	FocusBirth Date
Time editor	"Focus" + LabelText	FocusBirth Time
Dropdown Editor	"Focus" + LabelText	FocusTeam
AutoComplete Editor	"Focus" + LabelText	FocusCountry
Numeric Editor	"Focus" + LabelText	FocusNumeric
NumericUpDown Editor	"Focus" + LabelText	FocusValue
Checkbox Editor	"Focus" + LabelText	FocusCheckBox
Masked Editor	"Focus" + LabelText	FocusContact

To keep unique `AutomationId`, these inner elements' `AutomationIds` are updated based on the control's `AutomationId`. For example, if you set `SfDataForm AutomationId` as `SfDataForm.AutomationId = ContactsInfo`, then the Automation framework will interact with the text editor as `ContactsInfoFocus FirstName`. The following screenshots denote the `AutomationIds` for inner elements.

FirstName	<input type="text"/>	ContactsInfoFocusFirstName
Birth Date	12/6/1994	ContactsInfoFocusBirthDate
Birth Time	08:30 AM	ContactsInfoFocusBirthTime
Address	<input type="text"/>	ContactsInfoFocusAddress
Country	United States of...	ContactsInfoFocusCountry
Registered	<input type="radio"/>	ContactsInfoFocusRegistered
TrackHours	<input type="checkbox"/>	ContactsInfoFocusTrackHours
Team	<input type="text"/>	ContactsInfoFocusTeam

Name	Raju	ContactsInfoFocusName
Contact	<input type="text"/>	ContactsInfoFocusContact
Password	<input type="text"/>	ContactsInfoFocusPassword
Amount	10 - +	ContactsInfoFocusAmount
Value	0.000	ContactsInfoFocusValue

## SfDataGrid

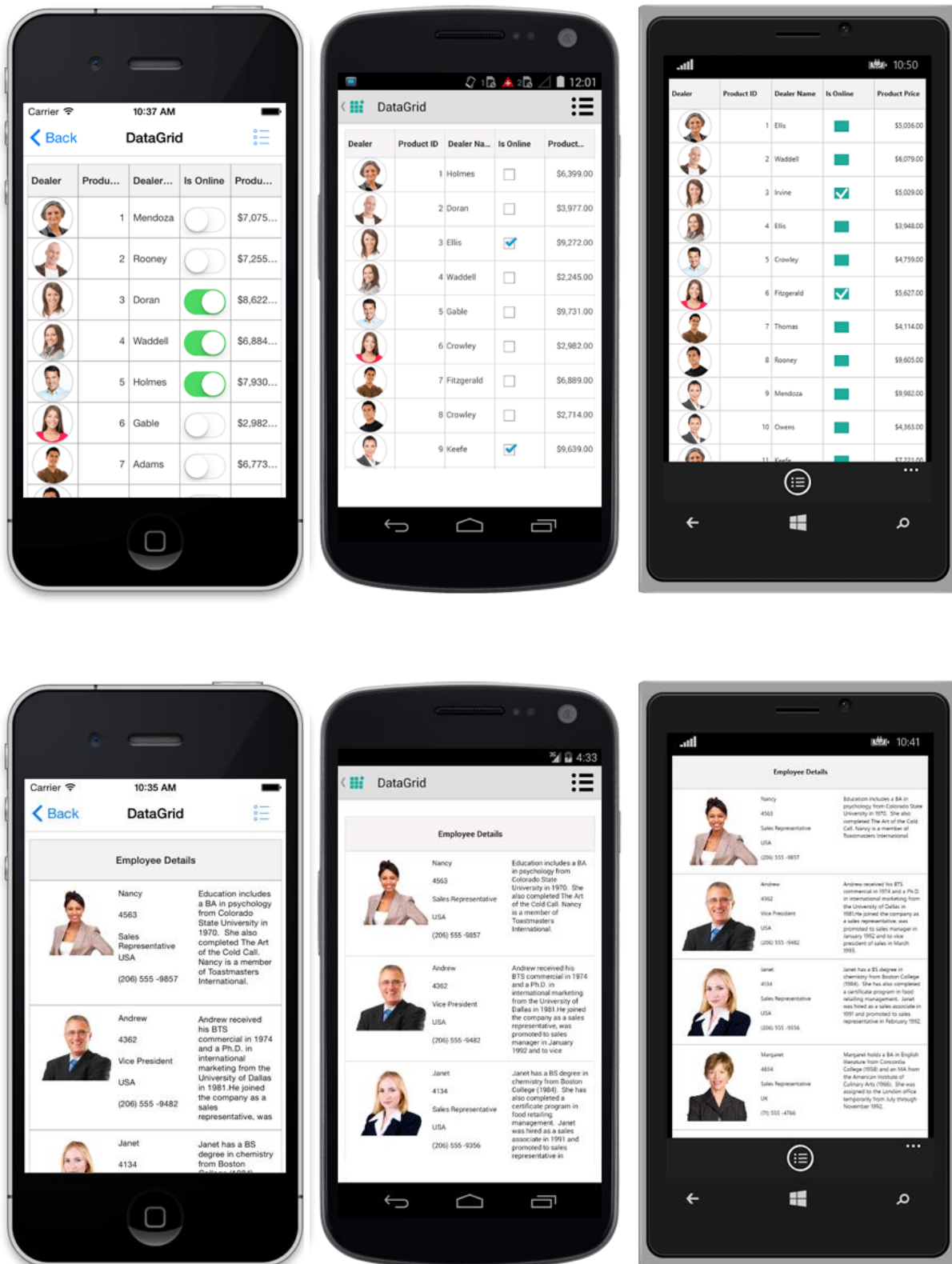
### SfDataGrid

The `DataGrid` control is available in `Xamarin.Forms`, `Xamarin.Android`, and `Xamarin.iOS`. This helps to create entirely customizable features and is used to display and manipulate large amounts of data in a tabular view. It is built from the ground up to achieve the best possible performance even when loading

large data sets. The following table lists the key features of this control in Xamarin.Forms, Xamarin.Android, and Xamarin.iOS.

**Information:** Xamarin.Forms is unique and offers a single language(C#) and runtime that works across the platforms like Android, iOS, UWP and MacOS platform as well.

Features	Xamarin.Forms (Android, iOS ,UWP and MacOS)	Xamarin.Android	Xamarin.iOS
Diagonal scrolling			
Sorting			
Custom sorting			
Grouping			
Custom grouping			
Summaries			
Filtering			
Editing			
Resizing columns			
Selection			
Load more			
Pull to refresh			
Swiping			
Column drag and drop			
Row drag and drop			
Template column			
Custom cell			
Styles			
Conditional styles			
Exporting			
Row Freezing			
Column freezing			
Row height customization			



## Getting Started

This section provides a quick overview for working with the SfDataGrid for Xamarin.Forms. Walk through the entire process of creating a real world of this control.

**Note:** Watch the video: [Xamarin.Forms DataGrid - Getting Started \(YouTube\)](#)

### Assembly deployment

After installing Essential Studio for Xamarin, find all the required assemblies in the installation folders {Syncfusion Essential Studio Installed location}\Essential Studio\{{ site.releaseversion }}\Xamarin\lib.

Eg: C:\Program Files (x86)\Syncfusion\Essential Studio\{{ site.releaseversion }}\Xamarin\lib.

**Note:** Assemblies can be found in an unzipped package location in Mac.

### NuGet configuration

To install the required NuGet for the SfDataGrid control in the application, configure the NuGet packages of the Syncfusion components.

Refer to the following KB to configure the NuGet packages of the Syncfusion components:

[How to configure package source and install Syncfusion NuGet packages in an existing project?](#)

The following NuGet package should be installed to use the SfDataGrid control in the application.

Project	Required package
Xamarin.Forms	Syncfusion.Xamarin.SfDataGrid

### Adding SfDataGrid reference

You can add SfDataGrid reference using one of the following methods:

#### Method 1: Adding SfDataGrid reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](#). To add SfDataGrid to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfDataGrid](#), and then install it.

![Adding SfDataGrid reference from NuGet](Getting-started\_images/Adding SfDataGrid reference.png)

**Note:** Install the same version of SfDataGrid NuGet in all the projects.

#### Method 2: Adding SfDataGrid reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfDataGrid control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfDataGrid assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Core.XForms.dll Syncfusion.Data.Portable.dll Syncfusion.GridCommon.Portable.dll Syncfusion.SfDataGrid.XForms.dll
-----	--



	Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.Licensing.dll Syncfusion.SfComboBox.XForms.dll
Android	Syncfusion.Data.Portable.dll Syncfusion.GridCommon.Portable.dll Syncfusion.SfDataGrid.XForms.dll Syncfusion.SfDataGrid.XForms.Android.dll Syncfusion.SfNumericTextBox.Android.dll Syncfusion.SfNumericTextBox.XForms.Android.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll Syncfusion.SfComboBox.XForms.dll Syncfusion.SfComboBox.XForms.Android.dll
iOS	Syncfusion.Data.Portable.dll Syncfusion.GridCommon.Portable.dll Syncfusion.SfDataGrid.XForms.dll Syncfusion.SfDataGrid.XForms.iOS.dll Syncfusion.SfNumericTextBox.iOS.dll Syncfusion.SfNumericTextBox.XForms.iOS.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll Syncfusion.SfComboBox.XForms.dll Syncfusion.SfComboBox.XForms.iOS.dll
UWP	Syncfusion.Data.Portable.dll Syncfusion.GridCommon.Portable.dll Syncfusion.SfDataGrid.XForms.dll Syncfusion.SfDataGrid.XForms.UWP.dll Syncfusion.SfInput.UWP.dll Syncfusion.SfShared.UWP.dll Syncfusion.SfNumericTextBox.XForms.UWP.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll Syncfusion.SfComboBox.XForms.dll Syncfusion.SfComboBox.XForms.UWP.dll
macOS	Syncfusion.Data.Portable.dll Syncfusion.GridCommon.Portable.dll Syncfusion.SfDataGrid.XForms.dll Syncfusion.SfDataGrid.XForms.macOS.dll Syncfusion.Core.XForms.dll

	Syncfusion.Core.XForms.macOS.dll Syncfusion.Licensing.dll
--	--

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the SfDataGrid on each platform

To use the SfDataGrid inside an application, each platform application must initialize the SfDataGrid renderer. This initialization step varies from platform to platform and is discussed in the following sections:

#### Android

The Android launches the SfDataGrid without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch the SfDataGrid in iOS, call the `SfDataGridRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework initialization and before the `LoadApplication` is called as demonstrated in the following code example:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.SfDataGrid.XForms.iOS.SfDataGridRenderer.Init ();
    LoadApplication (new App ());
    ...
}
```

#### Universal Windows Platform (UWP)

To launch the SfDataGrid in UWP, call the `SfDataGridRenderer.Init()` in the `MainPage` constructor before the `LoadApplication` is called as demonstrated in the following code example.

#### C#

```
public MainPage ()
{
    ...
    Syncfusion.SfDataGrid.XForms.UWP.SfDataGridRenderer.Init ();
    LoadApplication (new App ());
    ...
}
```

### Release mode issue in UWP platform

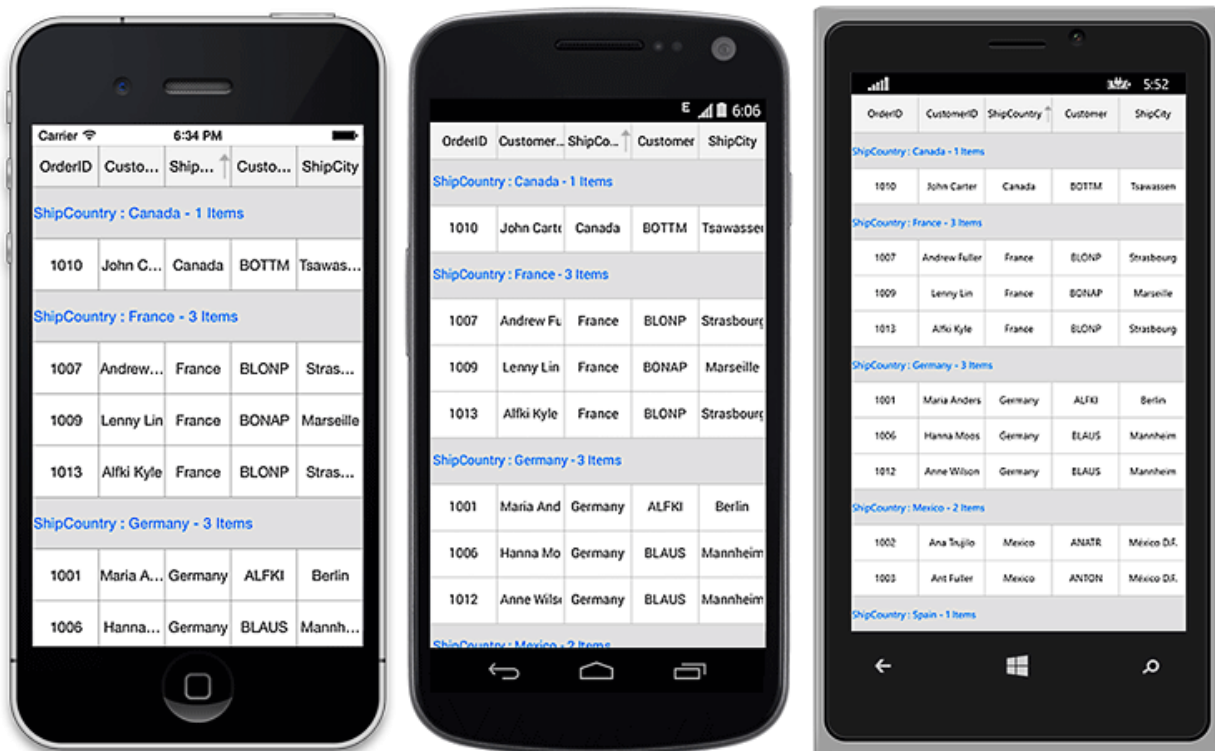
The known Framework issue in UWP platform is the custom controls will not render when deployed the application in **Release Mode**. It can be resolved by initializing the SfDataGrid assemblies in **App.xaml.cs** in UWP project as in the following code snippet.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you should add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.SfDataGrid.XForms.UWP.SfDataGridRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(Syncfusion.SfNumericTextBox.XForms.UWP.SfNumericTextBoxRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Create a simple SfDataGrid

This section explains how to create a SfDataGrid and configure it. The SfDataGrid control can be configured entirely in C# code or using XAML markup. This is how the final output will look like on iOS, Android, and Windows Phone devices.



---

**Note:** You can download the complete project of this demo from [GitHub](#).

---

In this walk through, a new application can be created that contains the SfDataGrid which includes the following topics:

- [Creating the project](#)
- [Adding SfDataGrid in Xamarin.Forms](#)
- [Create data model](#)
- [Binding data](#)
- [Defining columns](#)
- [Sorting](#)
- [Grouping](#)
- [Selection](#)

### Creating the project

Create a new BlankApp (Xamarin.Forms.Portable) application in Xamarin Studio or Visual Studio for Xamarin.Forms.

### Adding SfDataGrid in Xamarin.Forms

1. Add the required assembly references to the pcl and renderer projects as discussed in the [Assembly deployment](#) section.
2. Import the SfDataGrid control namespace as `xmlns:syncfusion="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"` in XAML Page.
3. Set the SfDataGrid control as content to the ContentPage.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
xmlns:syncfusion="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
x:Class="GettingStarted.Sample">
  <ContentPage.Content>
    <syncfusion:SfDataGrid x:Name="dataGrid" />
  </ContentPage.Content>
</ContentPage>
```

### C#

```
using Syncfusion.SfDataGrid.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public class App : Application
    {
        SfDataGrid dataGrid;
        public App ()
```

```
{
    dataGrid = new SfDataGrid();
    MainPage = new ContentPage { Content = dataGrid };
}
}
```

### Create DataModel for the SfDataGrid

The SfDataGrid is a data-bound control. Hence, a data model should be created to bind it to the control.

Create a simple data source as shown in the following code example in a new class file. Save it as OrderInfo.cs file:

#### C#

```
public class OrderInfo
{
    private int orderID;
    private string customerID;
    private string customer;
    private string shipCity;
    private string shipCountry;
    public int OrderID {
        get { return orderID; }
        set { this.orderID = value; }
    }
    public string CustomerID {
        get { return customerID; }
        set { this.customerID = value; }
    }
    public string ShipCountry {
        get { return shipCountry; }
        set { this.shipCountry = value; }
    }
    public string Customer {
        get { return this.customer; }
        set { this.customer = value; }
    }
    public string ShipCity {
        get { return shipCity; }
        set { this.shipCity = value; }
    }
    public OrderInfo (int orderId, string customerId, string country, string customer, string shipCity)
    {
        this.OrderID = orderId;
        this.CustomerID = customerId;
        this.Customer = customer;
        this.ShipCountry = country;
        this.ShipCity = shipCity;
    }
}
```

**Note:** If you want your data model to respond to property changes, implement `INotifyPropertyChanged` interface in your model class.

Create a model repository class with OrderInfo 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 OrderInfoRepository.cs file:

### C#

```
public class OrderInfoRepository
{
    private ObservableCollection<OrderInfo> orderInfo;
    public ObservableCollection<OrderInfo> OrderInfoCollection {
        get { return orderInfo; }
        set { this.orderInfo = value; }
    }
    public OrderInfoRepository ()
    {
        orderInfo = new ObservableCollection<OrderInfo> ();
        this.GenerateOrders ();
    }
    private void GenerateOrders ()
    {
        orderInfo.Add (new OrderInfo (1001, "Maria Anders", "Germany", "ALFKI", "Berlin"));
        orderInfo.Add (new OrderInfo (1002, "Ana Trujillo", "Mexico", "ANATR", "Mexico D.F.));
        orderInfo.Add (new OrderInfo (1003, "Ant Fuller", "Mexico", "ANTON", "Mexico D.F.));
        orderInfo.Add (new OrderInfo (1004, "Thomas Hardy", "UK", "AROUT", "London"));
        orderInfo.Add (new OrderInfo (1005, "Tim Adams", "Sweden", "BERGS", "London"));
        orderInfo.Add (new OrderInfo (1006, "Hanna Moos", "Germany", "BLAUS", "Mannheim"));
        orderInfo.Add (new OrderInfo (1007, "Andrew Fuller", "France", "BLONP", "Strasbourg"));
        orderInfo.Add (new OrderInfo (1008, "Martin King", "Spain", "BOLID", "Madrid"));
        orderInfo.Add (new OrderInfo (1009, "Lenny Lin", "France", "BONAP", "Marsielle));
        orderInfo.Add (new OrderInfo (1010, "John Carter", "Canada", "BOTTM", "Lenny Lin"));
        orderInfo.Add (new OrderInfo (1011, "Laura King", "UK", "AROUT", "London"));
        orderInfo.Add (new OrderInfo (1012, "Anne Wilson", "Germany", "BLAUS", "Mannheim"));
        orderInfo.Add (new OrderInfo (1013, "Martin King", "France", "BLONP", "Strasbourg"));
        orderInfo.Add (new OrderInfo (1014, "Gina Irene", "UK", "AROUT", "London"));
    }
}
```

### Binding data to the SfDataGrid

To bind the data source to the SfDataGrid, set the [SfDataGrid.ItemsSource](#) property as follows. You can bind the data source of the SfDataGrid either from XAML or in code.

The following code example binds the collection created in previous step to `SfDataGrid.ItemsSource` property.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
xmlns:syncfusion="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
x:Class="GettingStarted.Sample">
<ContentPage.BindingContext>
<local:OrderInfoRepository x:Name="viewModel" />
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrderInfoCollection}">
</syncfusion:SfDataGrid>
</ContentPage.Content>
</ContentPage>
```

**C#**

```
OrderInfoRepository viewModel = new OrderInfoRepository ();
dataGrid.ItemsSource = viewModel.OrderInfoCollection;
```

Run the application to render the following output.



## Defining columns

By default, the SfDataGrid automatically creates columns for all the properties in the data source. The type of the column generated depends on the type of data in the column. When the columns are auto-generated, handle the [SfDataGrid.AutoGeneratingColumn](#) event to customize or cancel the columns before they are added to the columns collection in the SfDataGrid.

The columns can be manually defined by setting the [SfDataGrid.AutoGenerateColumns](#) property to false and by adding the [GridColumn](#) objects to the [SfDataGrid.Columns](#) collection. It can be done from both XAML and code. The following code example illustrates this:

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    ColumnSizer="Star"
    AutoGenerateColumns="False"
    ItemsSource="{Binding OrderInfoCollection}">
    <syncfusion:SfDataGrid.Columns x:TypeArguments="syncfusion:Columns">
    <syncfusion:GridTextColumn HeaderText="Order ID"
    MappingName="OrderID" />
    <syncfusion:GridTextColumn HeaderText="Customer ID"
    MappingName="CustomerID" />
    <syncfusion:GridTextColumn MappingName="Customer" />
    <syncfusion:GridTextColumn HeaderText="Ship Country"
    MappingName="ShipCountry" />
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
dataGrid.AutoGenerateColumns = false;
GridColumn orderIdColumn = new GridTextColumn ();
orderIdColumn.MappingName = "OrderID";
orderIdColumn.HeaderText = "Order ID";
GridColumn customerIdColumn = new GridTextColumn ();
customerIdColumn.MappingName = "CustomerID";
customerIdColumn.HeaderText = "Customer ID";
GridColumn customerColumn = new GridTextColumn ();
customerColumn.MappingName = "Customer";
customerColumn.HeaderText = "Customer";
GridColumn countryColumn = new GridTextColumn ();
countryColumn.MappingName = "ShipCountry";
countryColumn.HeaderText = "Ship Country";
dataGrid.Columns.Add (orderIdColumn);
dataGrid.Columns.Add (customerIdColumn);
dataGrid.Columns.Add (customerColumn);
dataGrid.Columns.Add (countryColumn);
```

## Sorting

The SfDataGrid allows sorting on its data by setting the [SfDataGrid.AllowSorting](#) property to true.

### XML

```
<syncfusion:SfDataGrid AllowSorting="True" />
```



**C#**

```
dataGrid.AllowSorting = true;
```

Run the application and touch the header cell to sort the data and the following output will be displayed.



Sorting can also be configured by adding the column to the [SfDataGrid.SortColumnDescriptions](#) collection as follows.

**XML**

```
<syncfusion:SfDataGrid.SortColumnDescriptions>
<syncfusion:SortColumnDescription ColumnName="CustomerID" />
</syncfusion:SfDataGrid.SortColumnDescriptions>
```

**C#**

```
dataGrid.SortColumnDescriptions.Add (new SortColumnDescription ()
{ ColumnName = "CustomerID" });
```

**Grouping**

The SfDataGrid allows grouping a column by adding the column to the [SfDataGrid.GroupColumnDescriptions](#) collection as follows.

**XML**

```
<syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:GroupColumnDescription ColumnName="ShipCountry" />
```

```
</syncfusion:SfDataGrid.GroupColumnDescriptions>
```

## C#

```
dataGrid.GroupColumnDescriptions.Add (new GroupColumnDescription ()
{ ColumnName = "ShipCountry" });
```

Run the application to render the following output.



## Selection

The SfDataGrid allows selecting the row or rows by setting the [SfDataGrid.SelectionMode](#) property. You can set the [SfDataGrid.SelectionMode](#) property to single, multiple, single deselect, or none. Information about the row or rows selected can be tracked using the [SfDataGrid.SelectedItem](#) and [SfDataGrid.SelectedItems](#) properties.

The selection operations can be handled with the help of the [SelectionChanging](#) and [SelectionChanged](#) events of the SfDataGrid.

## Launching the SfDataGrid inside a StackLayout

The StackLayout positions the child element one after the other. They are adding either horizontally or vertically. Space of the [StackLayout](#) depends on the HorizontalOptions and VerticalOptions properties are set. But by default, the [StackLayout](#) will try to use the entire screen.

The SfDataGrid control can be loaded inside any layout such as [Grid](#), [StackLayout](#), etc., When loading SfDataGrid inside a [StackLayout](#), set the Horizontal and/or VerticalOptions of the SfDataGrid and its parent to "LayoutOptions.FillAndExpand" based on the orientation of the container in which the SfDataGrid is loaded.

Refer to the following code example to load the SfDataGrid control inside a `StackLayout`. The `VerticalOptions` of the `StackLayout` and the SfDataGrid alone is set as “FillAndExpand” as the default orientation of the `StackLayout` is vertical.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GridInForms"
xmlns:sfgrid="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
x:Class="GridInForms.MainPage">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout VerticalOptions="FillAndExpand">
<SearchBar Placeholder="UserName" TextChanged="searchBar_TextChanged" />
<sfgrid:SfDataGrid x:Name="dataGrid"
ColumnSizer="Star"
ItemsSource="{Binding OrderInfoCollection}"
VerticalOptions="FillAndExpand" />
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

### C#

```
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        private StackLayout stackLayout;
        private SfDataGrid dataGrid;
        private ViewModel viewModel;
        private SearchBar searchBar;
        public MainPage()
        {
            InitializeComponent();
            stackLayout = new StackLayout();
            dataGrid = new SfDataGrid();
            viewModel = new ViewModel();
            searchBar = new SearchBar();
            searchBar.Placeholder = "UserName";
            dataGrid.ItemsSource = viewModel.OrdersInfo;
            dataGrid.ColumnSizer = ColumnSizer.Star;
            stackLayout.VerticalOptions = LayoutOptions.FillAndExpand;
            stackLayout.HorizontalOptions = LayoutOptions.FillAndExpand;
            stackLayout.Children.Add(searchBar);
            stackLayout.Children.Add(dataGrid);
            this.Content = stackLayout;
        }
    }
}
```

Refer to the following screenshot for the outcome.



**Note:** In case, if the orientation of the `StackLayout` is horizontal, set the `HorizontalOptions` instead. In some cases, set both the “`VerticalOptions`” and “`HorizontalOptions`” of the `SfDataGrid` based on its parent.

Loading the `SfDataGrid` with customized height and width

The `SfDataGrid` can be load with specific height and width inside different layouts using the [SfDataGrid.HeightRequest](#) and [SfDataGrid.WidthRequest](#) properties.

The following code example illustrates how this can be done.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
xmlns:local="clr-namespace:GridInForms"
x:Class="GridInForms.MainPage"
xmlns:sfgrid="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
" >
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
HeightRequest="290"
WidthRequest="200"
VerticalOptions="CenterAndExpand"
HorizontalOptions="Center"/>
</ContentPage>
```

## C#

```
public MainPage()
{
    InitializeComponent();
    viewModel = new ViewModel();
    dataGrid = new SfDataGrid();
    dataGrid.ItemsSource = viewModel.OrdersInfo;
    dataGrid.HeightRequest = 290;
    dataGrid.WidthRequest = 200;
    dataGrid.VerticalOptions = LayoutOptions.CenterAndExpand;
    dataGrid.HorizontalOptions = LayoutOptions.Center;
    this.Content = dataGrid;
}
```

The following screenshot shows how the SfDataGrid is loaded with specific height and width with VerticalOptions and HorizontalOptions.

![DataGrid with customized height and width](SfDataGridimages>Loadingwith specificheightand\_width.png)

---

**Note:** Set the [HorizontalOptions](#) and [VerticalOptions](#) for the grid accordingly.

---

### [Linker issue in Xamarin.Forms.iOS](#)

There are some known Framework issues in Xamarin.Forms.iOS platform.

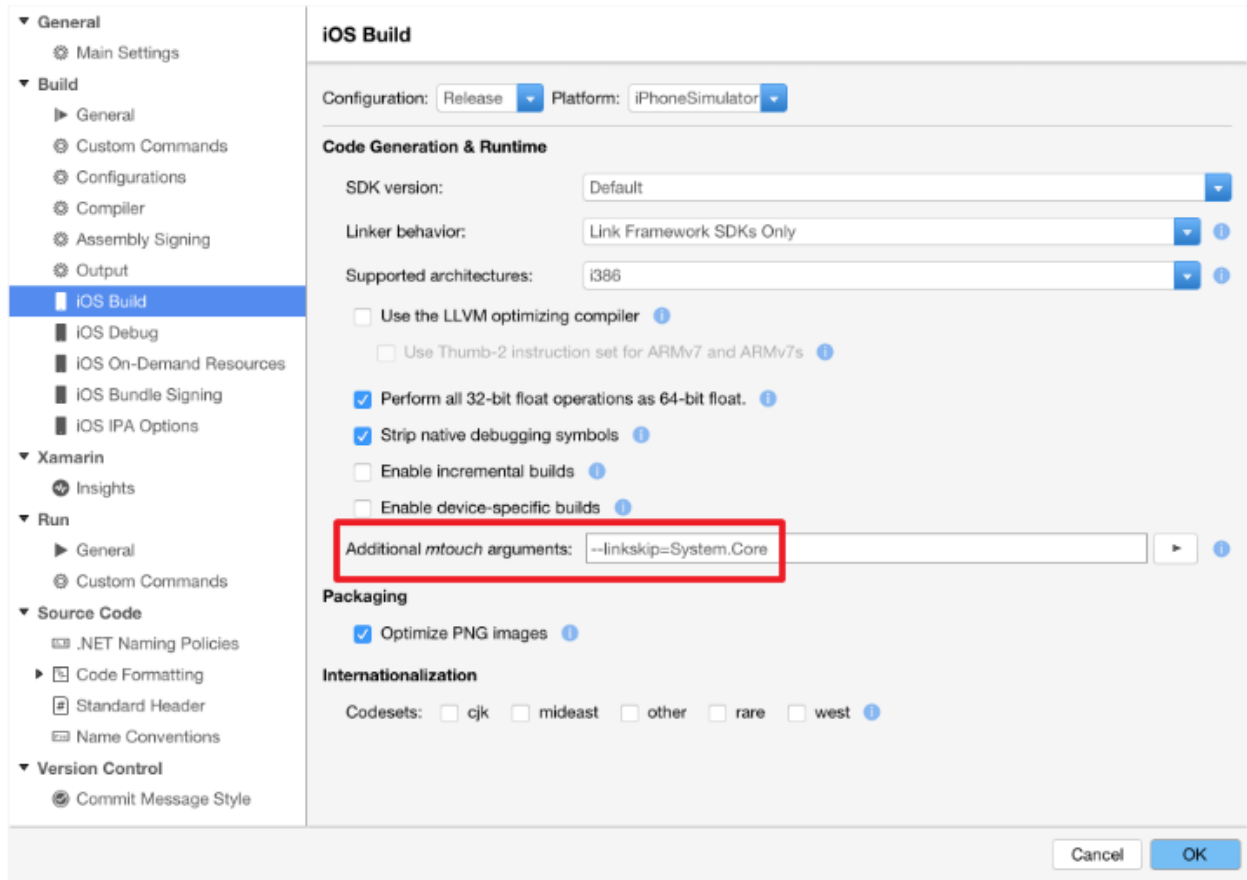
When creating the SfDataGrid in `Xamarin.Forms` with `Linker behavior` in `iOS` renderer project as “Link Framework SDKs only”, sometimes `System.MethodMissingException` or `No method Count exists on type System.Linq.Queryable` exception will be thrown.

The above exceptions can be resolved by using the following workaround:

#### **Workaround:**

The above exceptions can be resolved in two ways.

1. By setting `LinkerBehavior` as “Do not Link”. 2. By setting custom linker argument in `iOS` renderer project as in the following screenshot.



## Data Binding

The SfDataGrid is bound to an external data source to display the data. It supports data sources such as [List](#), [ObservableCollection](#), and so on. The [SfDataGrid.ItemsSource](#) property helps to bind this control with collection of objects.

In order to bind data source of the SfDataGrid, set the [SfDataGrid.ItemsSource](#) property as follows. Such that each row in the SfDataGrid would bind to an object in data source. Each column would bind to a property in the data model object.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:DataGridDemo;assembly=DataGridDemo"
xmlns:syncfusion="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
x:Class="DataGridDemo.Sample">
<ContentPage.BindingContext>
<local:OrderInfoRepository x:Name="viewModel" />
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrderInfoCollection}">
</syncfusion:SfDataGrid>
```

```
</ContentPage.Content>
</ContentPage>
```

**C#**

```
OrderInfoRepository viewModel = new OrderInfoRepository ();
dataGrid.ItemsSource = viewModel.OrderInfoCollection;
```

If the data source implements `ICollectionChanged` interface, then the SfDataGrid will automatically refresh the view when an item is added, removed, or cleared. When you add or remove an item in `ObservableCollection`, it automatically refreshes the view as the `ObservableCollection`. That implements the [INotifyCollectionChanged](#). But when you do the same in `List`, it will not refresh the view automatically.

If the data model implements the [INotifyPropertyChanged](#) interface, then the SfDataGrid responds to the property change at runtime to update the view.

---

**Note:** The SfDataGrid does not supports `DataTable` binding in `Xamarin.Forms` since `System.Data` is inaccessible in `Portable Class Library`.

---

**Binding with IEnumerable**

The SfDataGrid control supports to bind any collection that implements the [IEnumerable](#) interface. All the data operations such as sorting, grouping, and filtering are supported when binding collection derived from `IEnumerable`.

**Binding with DataTable**

SfDataGrid control supports to bind the [DataTable](#). SfDataGrid control automatically refresh the UI when binding `DataTable` as `ItemsSource` when rows are added, removed or cleared.

**XML**

```
<sfGrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Table}">
</sfGrid:SfDataGrid>
```

**C#**

```
DataTable Table = this.GetDataTable();
this.sfDataGrid1.ItemsSource = Table;
```

Below are the limitations when binding `DataTable` as `ItemsSource` to SfDataGrid.

- Custom sorting is not supported.
- `SfDataGrid.View.Filter` is not supported.
- Advanced Filtering does not support Case Sensitive filtering.
- [GridUnboundColumn.Expression](#) is not supported. This can be achieved by using the [DataColumn](#) of `DataTable` by setting [DataColumn.Expression](#) property.
- [SfDataGrid.LiveDataUpdateMode](#) is not supported.

### Binding complex properties

The SfDataGrid control supports to bind complex property to its columns. To bind complex property to the [GridColumn](#), set the complex property path to the [MappingName](#).

### XML

```
<sfGrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfGrid:GridTextColumn MappingName="OrderID.Order" />
<sfGrid:GridTextColumn MappingName="CustomerID" />
<sfGrid:GridTextColumn MappingName="Freight" />
<sfGrid:GridTextColumn MappingName="Country" />
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

### C#

```
this.dataGrid.Columns.Add(new GridTextColumn() { MappingName =
"OrderID.Order" });
```

### View

The DataGrid has the [View](#) property of type [ICollectionViewAdv](#) interface that implements [ICollectionView](#) interface. [View](#) is responsible for maintaining and manipulating data and other advanced operations, like [Sorting](#), [Grouping](#), and etc.,

When you bind collection to the [ItemsSource](#) property of the SfDataGrid, then [View](#) will be created and maintains the operations on [Data](#) such as [Grouping](#), [Sorting](#), [Insert](#), [Delete](#), and [Modification](#).

**Note:** The DataGrid creates different types of view derived from [ICollectionViewAdv](#) interface based on the [ItemsSource](#).

**Information:** [View](#) related properties can be used only after creating [SfDataGrid](#) view. Hence changes related to view can be done in [SfDataGrid.GridViewCreated](#) or [SfDataGrid.GridLoaded](#) event or in runtime only.

The following property is associated with [View](#)

#### LiveDataUpdateMode

The SfDataGrid supports to update the view during data manipulation operations and property changes using the [LiveDataUpdateMode](#). It allows to update the view based on the [SfDataGrid.View.LiveDataUpdateMode](#) property.

LiveDataUpdateMode	Description
Default	Data operations are not updated.
AllowSummaryUpdate	Summaries are updated during data manipulation change.
AllowDataShaping	Data operations like sorting, grouping, and filtering are updated during data manipulation change.



**C#**

```
dataGrid.GridViewCreated += DataGrid_GridViewCreated;  
private void DataGrid_GridViewCreated(object sender,  
GridViewCreatedEventArgs e)  
{  
    dataGrid.View.LiveDataUpdateMode = LiveDataUpdateMode.Default;  
}
```

The following events are associated with **View**.

*RecordPropertyChanged*

The [RecordPropertyChanged](#) event is raised when **DataModel** property value is changed, if **DataModel** implements the [INotifyPropertyChanged](#) interface. The event receives with two arguments namely sender that handles the **DataModel** and the [PropertyChangedEventArgs](#) as argument.

**PropertyChangedEventArgs** has the following property:

[PropertyName](#): It denotes the **PropertyName** of the changed value.

*CollectionChanged*

The [CollectionChanged](#) event is raised when changes in **Records/DisplayElements** collection. The event receives two arguments namely sender that handles **View** object and the [NotifyCollectionChangedEventArgs](#) as argument.

**NotifyCollectionChangedEventArgs** has the following 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*

The [SourceCollectionChanged](#) event is raised when making changes in **SourceCollection**. For example, adding or removing the collection. The event receives two arguments namely sender that handles **GridQueryableCollectionViewWrapper** object and the [NotifyCollectionChangedEventArgs](#) as argument.

**NotifyCollectionChangedEventArgs** has the following properties:

[Action](#): It contains the current action. (i.e.) **Add**, **Remove**, **Move**, **Replace**, **Reset**.

[NewItems](#): It contains the list of new items involved in the change.

[OldItems](#): It contains the list of old items affected by the **Action**.

[NewStartingIndex](#): It contains the index at which the change occurred.

[OldStartingIndex](#): It contains the index at which the **Action** occurred.

The following methods are associated with **View** which can be used to defer refresh the view:

Method Name	Description
DeferRefresh	Enter the defer cycle so that you can perform all data operations in view and update once.
BeginInit & EndInit	When BeginInit method is called it suspends all the updates until EndInit method is called. You can suspend and resume all the operations in these methods and update the view at once.

### Data Virtualization

Data grid provides support to handle the large amount of data through built-in virtualization feature. With Data virtualization, the record entries will be created in the runtime only upon scrolling to the vertical end which increases the performance of grid loading time.

To set SfDataGrid.EnableDataVirtualization property to true, follow the code example:

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding EmployeeDetails}"
EnableDataVirtualization="True">
```

### C#

```
datagrid.EnableDataVirtualization = true;
```

### NotificationSubscriptionMode

Data grid exposed [SfDataGrid.NotificationSubscriptionMode](#) property that allows you to set whether the underlying source collection items can listen to the INotifyCollectionChanged or INotifyPropertyChanging events. You can handle the property change or collection change by setting the NotificationSubscriptionMode property.

NotificationSubscriptionMode	Description
CollectionChange	Denotes a view that listens System.Collections.Specialized.INotifyCollectionChanged.CollectionChanged event of the SourceCollection.
None	Denotes System.ComponentModel.INotifyPropertyChanging.PropertyChanging, System.ComponentModel.INotifyPropertyChanged.PropertyChanged, and System.Collections.Specialized.INotifyCollectionChanged.CollectionChanged events will not be listened.
PropertyChange	Denotes a view that listens the System.ComponentModel.INotifyPropertyChanging.PropertyChanging and System.ComponentModel.INotifyPropertyChanged.PropertyChanged events of the data object.

To set the NotificationSubscriptionMode property, follow the code example.

#### **XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding EmployeeDetails}"
NotificationSubscriptionMode="CollectionChange">
```

#### **C#**

```
dataGrid.NotificationSubscriptionMode =
NotificationSubscriptionMode.CollectionChange;
```

#### *Binding SfDataGrid.SelectedIndex property*

You can bind any int value to the bindable property [SfDataGrid.SelectedIndex](#) which gets or sets the lastly selected row's index in the SfDataGrid.

Refer the below code to bind the [SfDataGrid.SelectedIndex](#) from the ViewModel.

#### **XML**

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderInfoCollection}"
SelectionMode="Multiple"
SelectedIndex="{Binding SelectedIndex}">
</sfgrid:SfDataGrid>
```

#### **C#**

```
//ViewModel.cs code
private int _selectedIndex;
public int SelectedIndex
{
    get { return _selectedIndex; }
    set { this._selectedIndex = value;RaisePropertyChanged("SelectedIndex"); }
}
public ViewModel()
{
    this.SelectedIndex = 5;
}
```

#### *Binding SfDataGrid.SelectedItem property*

You can bind any object value to the bindable property [SfDataGrid.SelectedItem](#) which gets or sets the selected item in the SfDataGrid.

Refer the below code to bind the [SfDataGrid.SelectedItem](#) from the ViewModel.

#### **XML**

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderInfoCollection}"
```

```
SelectionMode="Multiple"
SelectedItem="{Binding SelectedItem}">
</sfgrid:SfDataGrid>
```

**C#**

```
//ViewModel.cs code
private object _selectedItem;
public object SelectedItem
{
    get { return _selectedItem; }
    set { this._selectedItem = value; RaisePropertyChanged("SelectedItem"); }
}
public ViewModel()
{
    this.SelectedItem = this.OrderInfoCollection[8];
}
```

*Binding SfDataGrid.SelectedItems property*

You can bind any object type collection to the bindable property SfDataGrid [SfDataGrid.SelectedItems](#) which gets or sets the collection of [SelectedItem](#) collection in the SfDataGrid.

Refer the below code to bind the [SfDataGrid.SelectedItems](#) from the ViewModel.

**XML**

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderInfoCollection}"
SelectionMode="Multiple"
SelectedItem="{Binding SelectedItems}">
</sfgrid:SfDataGrid>
```

**C#**

```
//ViewModel.cs code
private ObservableCollection<object> _selectedItems;
public ObservableCollection<object> SelectedItems
{
    get { return _selectedItems; }
    set { this._selectedItems = value; RaisePropertyChanged("SelectedItems"); }
}
public ViewModel()
{
    this.SelectedItems.Add(OrderInfoCollection[1]);
    this.SelectedItems.Add(OrderInfoCollection[5]);
    this.SelectedItems.Add(OrderInfoCollection[8]);
}
```

*Binding GridColumn properties*

You can also assign value via binding to the properties of the [GridColumn](#) such as [HeaderCellTextSize](#), [CellTextSize](#), [FontAttribute](#), [RecordFont](#), [HeaderFont](#) etc.

Refer the below code to bind the GridColumn properties from the ViewModel.

## XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderInfoCollection}">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn MappingName="Customer" CellTextSize="{Binding
CellTextSize,Source={x:Reference viewModel}}"/>
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
```

## C#

```
//ViewModel.cs code
private double _cellTextSize;
public double CellTextSize
{
    get { return _cellTextSize; }
    set { this._cellTextSize = value; RaisePropertyChanged("CellTextSize"); }
}
public ViewModel()
{
    this.CellTextSize = 20;
}
```

### *Binding GridPickerColumn ItemsSource from ViewModel*

You can assign any object typed collection to the [GridPickerColumn.ItemsSource](#) to display a list of items in the [GridPickerColumn](#) when entering edit mode.

Refer the below code to bind the ItemsSource of GridPickerColumn from the ViewModel.

## XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderInfoCollection}">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridPickerColumn BindingContext="{x:Reference viewModel}"
MappingName="ShipCity"
ItemsSource="{Binding CustomerNames}" HeaderText="PickerColumn"/>
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
```

## C#

```
//ViewModel.cs code
private ObservableCollection<string> _customerNames;
public ObservableCollection<string> CustomerNames
{
    get { return _customerNames; }
    set { this._customerNames = value; RaisePropertyChanged("CustomerNames"); }
}
public ViewModel()
{
    this.CustomerNames = customerNames.ToObservableCollection();
}
```

```
}
string[] customerNames = { "Thomas", "John", "Hanna", "Laura", "Gina" };
```

#### *Binding the ItemsSource in ViewModel to the Picker loaded inside template*

The `ItemsSource` of a picker which is loaded inside the `GridTemplateColumn` can also be assigned any value via binding by passing the binding context as the `Source` to the `ItemsSource` property.

Refer the below code to bind the `ItemsSource` of Picker loaded inside the `GridTemplateColumn` from the `ViewModel`.

#### **XML**

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderInfoCollection}">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTemplateColumn MappingName="Customer" HeaderText="Picker">
<sfgrid:GridTemplateColumn.CellTemplate>
<DataTemplate>
<Picker ItemsSource="{Binding SelectedModels,Source={x:Reference
viewModel}}" SelectedIndex="0"/>
</DataTemplate>
</sfgrid:GridTemplateColumn.CellTemplate>
</sfgrid:GridTemplateColumn>
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
```

#### **C#**

```
//ViewModel.cs code
private List<String> _vehicleModel;
public List<String> SelectedModels
{
get { return _vehicleModel; }
set { this._vehicleModel = value;RaisePropertyChanged("SelectedCars"); }
}
public ViewModel()
{
this.SelectedModels = selectedModels.ToList();
}
string [] selectedModels = { "Select Car", "Audi", "Bentley", "Mercedes
Benz", "Porsche" };
```

#### *Binding the button command in template column to ViewModel*

You can also assign any value to the `Command` property of the `Button` loaded inside the `GridTemplateColumn` via binding.

Refer the below code to bind the `command` property of `Button` loaded inside the `GridTemplateColumn` from the `ViewModel`.

#### **XML**

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderInfoCollection}">
```

```

<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTemplateColumn MappingName="Customer">
<sfgrid:GridTemplateColumn.CellTemplate>
<DataTemplate>
<Button Text="Template" Command="{Binding ButtonCommand, Source={x:Reference
viewModel}}"/>
</DataTemplate>
</sfgrid:GridTemplateColumn.CellTemplate>
</sfgrid:GridTemplateColumn>
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>

```

## C#

```

//ViewModel.cs code
private Command _buttonCommand;
public Command ButtonCommand
{
    get { return _buttonCommand; }
    set { this._buttonCommand = value;RaisePropertyChanged("ButtonCommand"); }
}
public ViewModel()
{
    this.ButtonCommand = new Command(CustomMethod);
}
public void CustomMethod()
{
    // Customize your code here
}

```

You can download the source code of binding the SfDataGrid properties sample [here](#)

## Columns

The SfDatagrid allows to create and add columns in the following two ways:

- Automatic columns generation based on the underlying collection.
- Manually defining columns in XAML or C#.

### Automatic columns generation

The SfDataGrid creates columns automatically based on the bindable property [SfDataGrid.AutoGenerateColumns](#). Columns are generated based on type of individual properties in the underlying collection which is set as ItemsSource. For example, [GridNumericColumn](#) is added for int type property. Below table shows the column type created for the respective data type. For remaining types, [GridTextColumn](#) will be created.

Data Tye	Column
string, object	GridTextColumn
int, float, double, decimal and it's respective nullable types	GridNumericColumn
DateTime	GridDateTimeColumn

bool	GridSwitchColumn
enum	GridPickerColumn
ImageSource	GridImageColumn

You can refer the sample from [here](#) to get to know the codes for defining properties in the Model class and populating data for generating different types of column automatically.

#### *AutoGenerateColumns with different modes*

The auto generation of the columns in SfDataGrid is based on the [SfDataGrid.AutoGenerateColumnsMode](#) property.

[SfDataGrid.AutoGenerateColumnsMode](#) decides a way to create columns when [SfDataGrid.AutoGenerateColumns](#) is set to `true`. It also decides to retain grouping and sorting when the [ItemsSource](#) changed.

The [SfDataGrid.AutoGenerateColumnsMode](#) is of [AutoGenerateColumnsMode](#) type which has the following five options:

Modes	Description
<a href="#">None</a>	Stores only the columns that are defined in SfDataGrid.Columns collection. When changing the ItemsSource, the grouping and sorting for explicitly defined SfDataGrid.Columns alone will be retained.
<a href="#">Reset</a>	Retains the columns defined explicitly in the application level and creates columns newly for all the other properties in a data source. When changing the ItemsSource, the grouping and sorting for explicitly defined SfDataGrid.Columns alone will be retained.
<a href="#">ResetAll</a>	When changing the ItemsSource, the columns for the previous data source are cleared and the columns will be created newly for the new data source. Even when columns are explicitly defined it does not consider the defined columns and creates the column based on the underlying collection. Further when changing the ItemsSource, the grouping and sorting for all the columns will be cleared.
<a href="#">RetainOld</a>	When changing the ItemsSource, creates columns for all fields in a data source when the Grid does not have any explicit definition for columns. When columns are defined explicitly, then the defined columns alone are retained and new columns are not created.



	Similarly when changing the ItemsSource and when the Grid have any explicit definition for columns, the grouping and sorting are retained as it is.
<a href="#">SmartReset</a>	Retains the columns defined explicitly in application level and the columns with MappingName identical to the properties in the new data source. Creates columns newly for all the other properties in the data source. Similarly it retains the grouping and sorting of the columns that are defined explicitly in application level and the columns with MappingName identical to the properties in new data source.

The default value of `SfDataGrid.AutoGenerateColumns` property is `true` and `SfDataGrid.AutoGenerateColumnsMode` is `Reset`. By default, SfDataGrid creates columns automatically for every non-explicitly defined public property in the underlying collection bound to its `ItemsSource` property.

**Note:** When you change items source for the SfDataGrid during run time, then the columns are generated on the basis of option set for `SfDataGrid.AutoGenerateColumnsMode`.

#### *Auto generate columns for custom type*

By default columns are also auto generated for custom type properties and for parent properties of complex properties in the data object. To prevent such columns from being auto generated set the `SfDataGrid.AutoGenerateColumnsForCustomType` property as `False`.

In case of complex properties, use the `SfDataGrid.AutoGenerateColumnsModeForCustomType` to auto generate columns for either parent property, inner properties of the parent or both parent and inner properties.

#### **XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo,Mode=TwoWay}"
AutoGenerateColumnsForCustomType="True"
AutoGenerateColumnsModeForCustomType="Both"
AllowEditing="True"
AllowSorting="True"
NavigationMode="Cell"
SelectionMode="Single">
</syncfusion:SfDataGrid>
```

#### **C#**

```
this.dataGrid.AutoGenerateColumnsForCustomType = true;
this.dataGrid.AutoGenerateColumnsModeForCustomType =
AutoGenerateColumnsModeForCustomType.Both;
```

### *Customize automatically generated columns*

When `SfDataGrid.AutoGenerateColumns` is `true`, the `SfDataGrid.AutoGeneratingColumn` event is raised for each `GridColumn`. This event receives two arguments namely sender which is the `SfDataGrid` and the `AutoGeneratingColumnEventArgs`.

The `AutoGeneratingColumnEventArgs` object contains the following properties:

- `Column`: This property returns the created column which can be customized.
- `Cancel`: This property cancels the column creation.
- `PropertyType`: This property provides the type of underlying model property for which the column is created.

You can skip generating a column by handling the `SfDataGrid.AutoGeneratingColumn` event as shown as follows:

#### **C#**

```
dataGrid.AutoGeneratingColumn += GridAutoGeneratingColumns;
void GridAutoGeneratingColumns(object sender,
AutoGeneratingColumnEventArgs e)
{
    if (e.Column.MappingName == "EmployeeID")
        e.Cancel = true;
}
```

Formatting for auto generated columns can be applied as follows:

#### **C#**

```
void GridAutoGeneratingColumns(object sender, AutoGeneratingColumnEventArgs
e)
{
    if (e.Column.MappingName == "Freight") {
        e.Column.Format = "C";
        e.Column.CultureInfo = new CultureInfo ("en-US");
    } else if (e.Column.MappingName == "ShippingDate")
        e.Column.Format = "dd/MM/yyyy";
}
```

You can perform any desired operation based on the property type of the underlying model object as follows.

#### **C#**

```
void GridAutoGeneratingColumns(object sender, AutoGeneratingColumnEventArgs
e)
{
    if(e.PropertyType == typeof(string))
    {
        // Here we have hidden the columns if the underlying property type is
        string.
        e.Column.IsHidden = true;
    }
}
```

You can also customize header text, sorting, alignment, padding, etc., of a column by handling the `SfDataGrid.AutoGeneratingEvent`.

### Manually generate columns

The `SfDataGrid` also allows to define the columns manually by adding the `GridColumn` objects to the [SfDataGrid.Columns](#) collection. If you want only the manually defined columns to be in view, you can achieve it by setting the [SfDataGrid.AutoGenerateColumns](#) property to `false`. There are different types of columns available. Any column can be created based on the requirements from both XAML and code.

The following code example illustrates about creating columns manually:

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    ColumnSizer="Star"
    AutoGenerateColumns="False"
    ItemsSource="{Binding OrderInfoCollection}">
    <syncfusion:SfDataGrid.Columns x:TypeArguments="syncfusion:Columns">
        <syncfusion:GridTextColumn HeaderText="Order ID"
            MappingName="OrderID" />
        <syncfusion:GridTextColumn HeaderText="Customer ID"
            MappingName="CustomerID" />
        <syncfusion:GridTextColumn MappingName="Customer" />
        <syncfusion:GridTextColumn HeaderText="Ship Country"
            MappingName="ShipCountry" />
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
dataGrid.AutoGenerateColumns = false;
GridColumn orderIdColumn = new GridTextColumn ();
orderIdColumn.MappingName = "OrderID";
orderIdColumn.HeaderText = "Order ID";
GridColumn customerIdColumn = new GridTextColumn ();
customerIdColumn.MappingName = "CustomerID";
customerIdColumn.HeaderText = "Customer ID";
GridColumn customerColumn = new GridTextColumn ();
customerColumn.MappingName = "Customer";
GridColumn countryColumn = new GridTextColumn ();
countryColumn.MappingName = "ShipCountry";
countryColumn.HeaderText = "Ship Country";
dataGrid.Columns.Add (orderIdColumn);
dataGrid.Columns.Add (customerIdColumn);
dataGrid.Columns.Add (customerColumn);
dataGrid.Columns.Add (countryColumn);
```

### Resizing columns

The `SfDataGrid` allows to resize the columns by tapping and dragging the right border of the column headers. Resizing can be enabled or disabled by setting the [SfDataGrid.AllowResizingColumn](#) property. A resizing indicator is displayed while resizing a column.

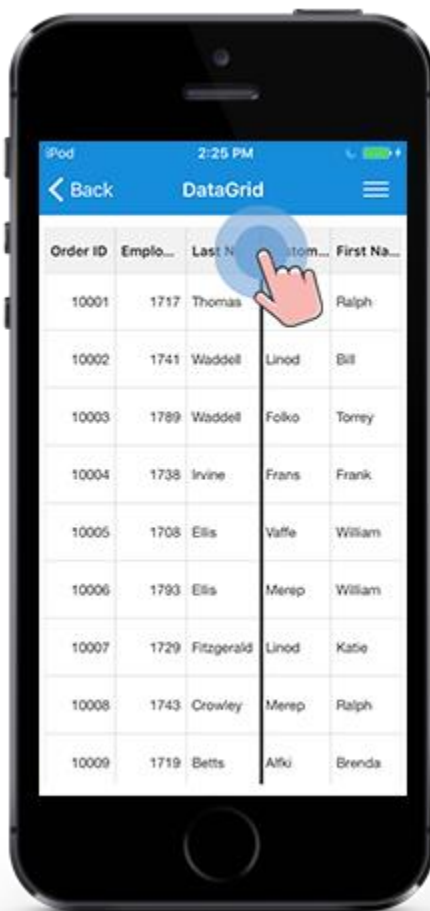
**Note:** Resizing considers [GridColumn.MinimumWidth](#) and [GridColumn.MaximumWidth](#) of the column and will not resize the minimum and maximum width constraints.

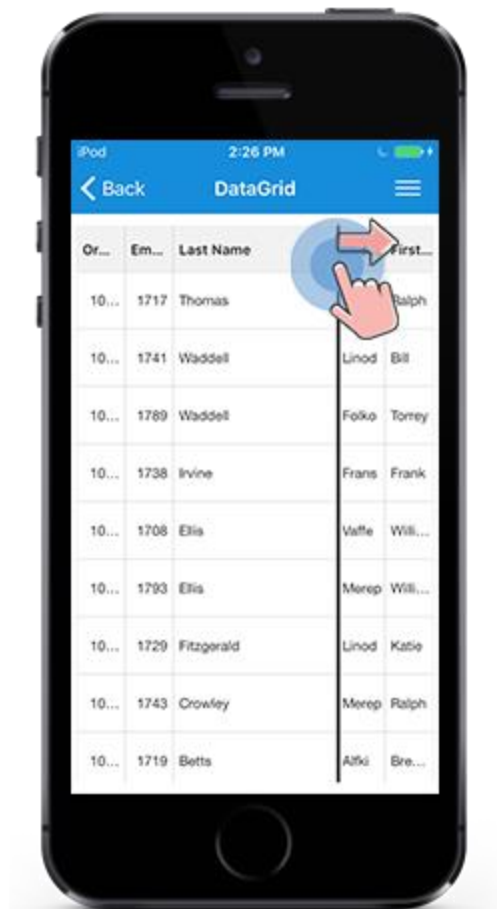
### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowResizingColumn="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}" />
```

### C#

```
dataGrid.AllowResizingColumn = true;
```





The column width can be changed by tapping and dragging the resizing indicator.

---

**Note:** The resizing indicator appears while tapping the right corner of the column header.

---

To interactively hide a column, set the `GridColumn.MinimumWidth` property to zero. Resize the column to a width less than 0.

#### *Resizing modes*

The SfDataGrid allows two modes of resizing by setting the [SfDataGrid.ResizingMode](#) property. The resizing modes are as follows:

- **OnMoved:** The resizing indicator is moved based on the touch point. The width of the column is updated as the resizing indicator moves.
- **OnTouchUp:** The resizing indicator is moved based on the touch point. However the width of the column is updated only on a touch up operation.

---

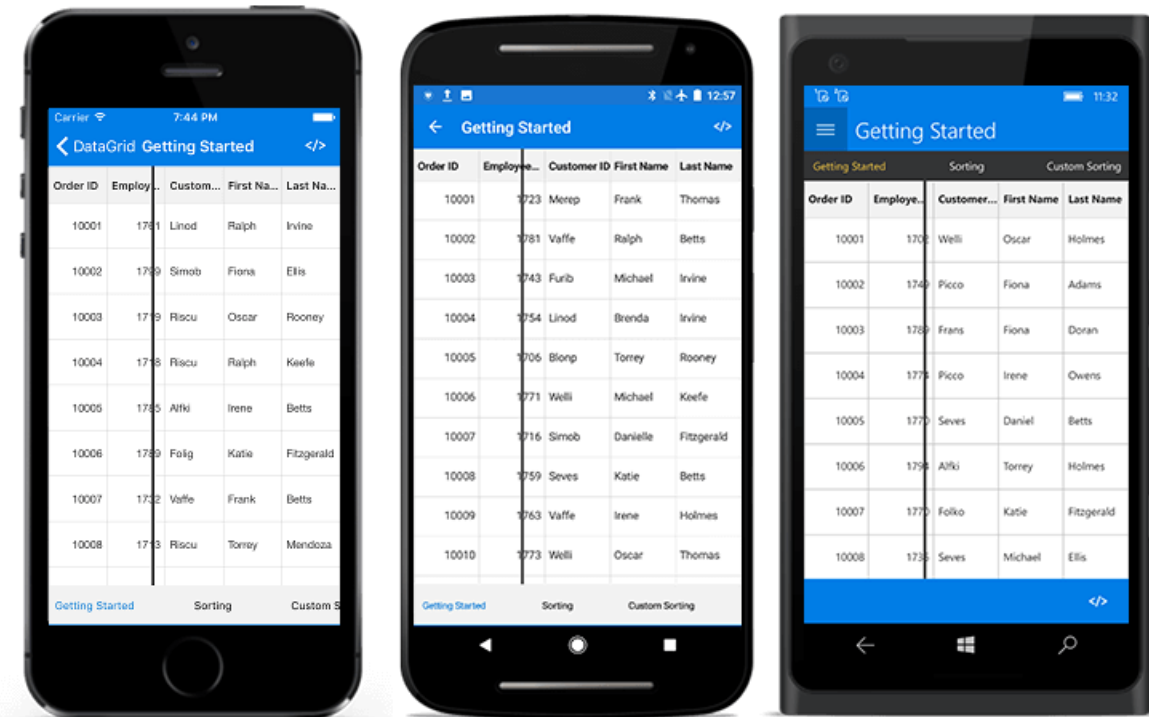
**Note:** The default resizing mode is OnMoved.

---

The following image shows resizing mode OnMoved:



The following image shows resizing mode OnTouchUp:



### Resizing events

Based on the requirement resizing operation can be handled using the [SfDataGrid.ColumnResizing](#) event. The `SfDataGrid.ColumnResizing` event is fired while resizing a column. It will be continuously fired till the resizing operation ends.

By handling the `SfDataGrid.ColumnResizing` event, the resizing of a particular column can be canceled.

The `SfDataGrid.ColumnResizing` event provides the following properties through [GridResizingEventArgs](#).

- [Index](#) - Returns the index of the column currently being resized.
- [NewValue](#) - Returns the current width of the column being resized.
- [ResizingState](#) - Returns the current state of the user-interaction through a value from the `ProgressStates` enum.
- [Cancel](#) - A Boolean property to cancel the event and the resizing operation.

### Cancel resizing for a column

To cancel resizing a particular column, use the `SfDataGrid.ColumnResizing` event. Based on the different arguments provided in the `GridResizingEventArgs`, the resizing operation of a column can be canceled.

To cancel resizing a column using the `SfDataGrid.ColumnResizing` event using the `Index` value, follow the code example:

#### C#

```
this.dataGrid.ResizingColumns += dataGrid_ResizingColumns;
private void DataGrid_ColumnResizing(object sender, GridResizingEventArgs e)
{
    //Code to end resizing if ColumnIndex is 2
    if (e.Index == 2)
        e.Cancel = true;
}
```

To cancel resizing a column using the `SfDataGrid.ColumnResizing` event using the `NewValue` value, follow the code example:

#### C#

```
this.dataGrid.ResizingColumns += dataGrid_ResizingColumns;
private void DataGrid_ColumnResizing(object sender, GridResizingEventArgs e)
{
    //Code to end resizing if Column's Width is >= 100
    if (e.NewValue >= 100 ||)
        e.Cancel = true;
}
```

To cancel resizing a column using the `SfDataGrid.ColumnResizing` event using the `ProgressStates` value, follow the code example:

#### C#

```
this.dataGrid.ResizingColumns += dataGrid_ResizingColumns;
```

```
private void DataGrid_ColumnResizing(object sender, GridResizingEventArgs e)
{
    //Code to end resizing if interaction state is Progressing
    if (e.ResizingState == ProgressStates.Progressing)
    {
        e.Cancel = true;
    }
}
```

## Column Types

The SfDataGrid contains different types of columns. The functionalities of the column can be implied by its name. Based on the requirements any column can be used.

The following table describes the types of columns and its usage:

Column Type	Renderer	Description
<a href="#">GridTextColumn</a>	<a href="#">GridCellTextViewRenderer</a>	To display string or numbers in each row.
<a href="#">GridSwitchColumn</a>	<a href="#">GridCellSwitchRenderer</a>	To display switch in each row.
<a href="#">GridImageColumn</a>	<a href="#">GridImageCellRenderer</a>	To display an image in each row.
<a href="#">GridTemplateColumn</a>	<a href="#">GridCellTemplateRenderer</a>	To customize the column based on the requirements.
<a href="#">GridNumericColumn</a>	<a href="#">GridCellNumericRenderer</a>	To display a numeric data.
<a href="#">GridPickerColumn</a>	<a href="#">GridCellPickerRenderer</a>	To display the IEnumerable data using Picker.
<a href="#">GridComboBoxColumn</a>	<a href="#">GridCellComboBoxRenderer</a>	To display the IEnumerable data using Picker.
<a href="#">GridDateTimeColumn</a>	<a href="#">GridDateTimeColumn</a>	To display the date and time value.
<a href="#">GridUnboundColumn</a>	<a href="#">GridUnboundCellTextBoxRenderer</a>	To add additional columns that are not bound with data object from the underlying data source.

## GridColumn

The [GridColumn](#) is the base column types of all columns. Hence its properties are used by all the columns. The following sub-sections explain the properties and customizations of GridColumn:

### Binding options

Display content of the GridColumn is determined from the [GridColumn.DisplayBinding](#) property. It gets or sets display binding that associates the GridColumn with a property in the data source.

The actual bound value of the GridColumn is determined from [GridColumn.ValueBinding](#) property. It gets or sets value binding that associates the GridColumn with a property in the data source.



### Mapping column to particular property

The [GridColumn.MappingName](#) associates the GridColumn with a property available in the underlying data source. While setting MappingName alone to the SfDataGrid, the `GridColumn.DisplayBinding` will be automatically generated based on the MappingName. Data manipulation operations like sorting, filtering, and grouping will be done based on the MappingName property.

To format cell content, use the converter of the `GridColumn.DisplayBinding` to customize the cell content. The following code example appends the text "Customer" along with the Customer ID:

### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<local:DisplayBindingConverter x:Key="displayBindingConverter" />
</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="CustomerID"
DisplayBinding="{Binding CustomerID,
Converter={StaticResource displayBindingConverter}}" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
public class DisplayBindingConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        if (value != null)
            return "Customer:" + value.ToString();
        return null;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return value.ToString().Substring(9);
    }
}
```

### Header customizations

#### HeaderCellTextSize

The FontSize can be customized using the [GridColumn.HeaderCellTextSize](#) property. The default font size is 14.

#### HeaderFont

The FontFamily can be customized using the [GridColumn.HeaderFont](#) property. The default font value is HelveticaNeue LT 55 Roman.

### HeaderFontAttribute

The FontAttribute can be customized using the [GridColumn.HeaderFontAttribute](#) property. The default value of this property is **None**. It can be customized as **Bold** or **Italic**.

### HeaderText

To customize the display content of the header cell, use the [GridColumn.HeaderText](#) property. It specifies the text displayed in the column header. If header text is not defined, then **GridColumn.MappingName** will be assigned to the header text and will be displayed as column header.

### HeaderTextAlignment

To get or set the TextAlignment of the header cell, use the [GridColumn.HeaderTextAlignment](#) property. The default alignment is **Center**. It can be customized as **Start** or **End**.

### HeaderTemplate

Based on the requirement, the header cell can be customized using the [GridColumn.HeaderTemplate](#) property. To customize header cell by loading a template in the header cell, follow the code example:

#### XML

```
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID">
<syncfusion:GridTextColumn.HeaderTemplate>
<DataTemplate>
<Label x:Name="OrderID" Text="OrderID" TextColor="Black"
Background="Yellow" YAlign="Center" />
</DataTemplate>
</syncfusion:GridTextColumn.HeaderTemplate>
</syncfusion:GridTextColumn>
</syncfusion:SfDataGrid.Columns>
```

### Setting manual column width

SfDataGrid allows you to customize the width of each GridColumn in the [SfDataGrid.Columns](#) collection. To customize column width, use the [GridColumn.Width](#) property. By default, this property will not be assigned any value. The GridColumn renders in view based on the value of the [DefaultColumnWidth](#) property.

**Note:** Set the IsHidden property to True instead of setting column width as 0 to hide a column.

Customize the width for auto generated columns in both XAML and code as follows:

#### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="false"
<sfgrid:SfDataGrid.Columns x:TypeArguments="sfgrid:Columns">
<sfgrid:GridTextColumn MappingName="OrderID"
Width="100"/>
</sfgrid:SfDataGrid.Columns >
</sfgrid:SfDataGrid>
```

#### C#

```
// AutoGenerated Column
dataGrid.AutoGeneratingColumn += DataGrid_AutoGeneratingColumn;
```

```
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs e) {
    if (e.Column.MappingName == "OrderID") {
        e.Column.Width = 100;
    }
}
// Manually generated column
dataGrid.Columns.Add(new GridTextColumn() { MappingName = "OrderID", Width = 100 });
```

### Hiding a column

To hide a particular column, use the [GridColumn.IsHidden](#) property. The default value of the `IsHidden` property is `False`.

**Note:** Set the `IsHidden` property to `True` instead of setting column width as `0` to hide a column.

To hide column using the `IsHidden` property, follow the code example:

### XML

```
<syncfusion:GridTextColumn MappingName="OrderID" IsHidden = "True"/>
```

### C#

```
// AutoGenerate Column
dataGrid.AutoGeneratingColumn += DataGrid_AutoGeneratingColumn;
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs e) {
    if (e.Column.MappingName == "OrderID") {
        e.Column.IsHidden = true;
    }
}
// Manually generated column
dataGrid.Columns.Add(new GridTextColumn() { MappingName = "OrderID", IsHidden = true });
```

### LoadUIView

The [GridColumn.LoadUIView](#) property indicates whether to load `UIElement` inside the `GridCell` or to draw cell value directly in the `canvas` of the `GridCell` in Android platform.

- When `LoadUIView` is set to `false`, cell value of the column is directly drawn in the `canvas` of the grid cells to improve performance.
- While setting the `LoadUIView` to `true`, a `UIElement` ([SfLabel](#)) is loaded in the `GridCells`.

The default value of the `LoadUIView` is `false` for `Xamarin.Forms.Android`, and `true` for other platforms in `Xamarin.Forms`.

### C#

```
GridTextColumn customerID = new GridTextColumn();
customerID.MappingName = "Description";
customerID.LoadUIView = true;
```

### Padding

SfDataGrid allows the user to set padding for the cells by using the property [GridColumn.Padding](#).

### XML

```
<syncfusion:GridTextColumn MappingName="OrderID" TextAlignment="Start"
    Padding="10,0,0,0"/>
```

### C#

```
GridTextColumn orderID = new GridTextColumn();
orderID.MappingName = "OrderID";
orderID.TextAlignment = TextAlignment.Start;
orderID.Padding = new Thickness(10, 0, 0, 0);
```

**Note:** `GridTemplateColumn` ignores the `GridTemplateColumn.Padding` values set in the sample since it takes default padding values based on the [DataGridStyle.GetGridLinesVisibility](#), so that the borders are visible. Hence to set custom padding values, set margin for the base view of your `DataTemplate`.

### GridTextColumn

`GridTextColumn` inherits all the properties of `GridColumn`. It is used to host the textual content in the record cells. Each of the record cell displays text based on the `MappingName` that associates the column with a property in the data source.

The following code example creates `GridTextColumn`:

### XML

```
<syncfusion:GridTextColumn MappingName="OrderID" />
```

### C#

```
dataGrid.Columns.Add(new GridTextColumn() { MappingName = "OrderID" });
```

The following topics explain the customizations done in the `GridTextColumn`:

### Formatting

To format values displayed in the `GridColumn`, use the [GridColumn.Format](#) property.

#### Format column using StringFormat

Assign the format of string to the `GridColumn.Format` property based on the bound data type of the property, the `GridColumn` is associated to format the value. You can use different [StringFormats](#) to customize values displayed in the record cells.

To apply formatting for a `GridTextColumn`, follow the code example:

### XML

```
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="Freight" Format="C" />
<syncfusion:GridTextColumn MappingName="ShippingDate" Format="dd/MM/yyyy" />
</syncfusion:SfDataGrid.Columns>
```

### C#

```
dataGrid.Columns.Add (new GridTextColumn () {  
    MappingName = "Freight",  
    Format = "C"  
});  
dataGrid.Columns.Add (new GridTextColumn () {  
    MappingName = "ShippingDate",  
    Format = "dd/MM/yyyy"  
});
```

#### Format column using converter

Using converter, set the format of the column.

To set the format using converter, follow the code example:

### XML

```
<sfgrid:SfDataGrid.Columns>  
<sfgrid:GridTextColumn MappingName="Salary" DisplayBinding="{Binding Salary,  
Converter={StaticResource SummaryConverter}}" />  
</sfgrid:SfDataGrid.Columns>
```

### C#

```
public object Convert(object value, Type targetType, object parameter,  
    CultureInfo culture)  
{  
    var formattedString = string.Format("$ {0}", value);  
    return formattedString;  
}
```

**Note:** For AutoGenerated columns formatting can be applied by handling the [SfDataGrid.AutoGeneratingColumn](#) event.

#### Formatting GridTextColumn with different culture

To apply different [CultureInfo](#) for GridColumns, use the [GridColumn.CultureInfo](#) property. Assign format of the string to this property. Based on the type of the property the column is associated to format the value. You can use different [StringFormat](#)s to customize values displayed in the record cells.

To apply different cultures for the GridColumns, follow the code example:

### C#

```
dataGrid.Columns.Add (new GridTextColumn () {  
    MappingName = "Freight",  
    Format = "C",  
    CultureInfo = new CultureInfo("en-US")  
});  
dataGrid.Columns.Add (new GridTextColumn () {  
    MappingName = "OrderID",  
    Format = "C",  
    CultureInfo = new CultureInfo("en-GB")  
});
```

For auto generated columns, this is achievable by handling the [SfDataGrid.AutoGeneratingColumn](#) event. To apply different cultures for auto generated GridColumns, follow the code example:

### C#

```
void GridAutoGeneratingColumns(object sender, AutoGeneratingColumnArgs e)
{
    if (e.Column.MappingName == "Freight") {
        e.Column.Format = "C";
        e.Column.CultureInfo = new CultureInfo ("en-US");
    } else if (e.Column.MappingName == "OrderID") {
        e.Column.Format = "C";
        e.Column.CultureInfo = new CultureInfo ("en-GB");
    }
}
```

### Font and alignment options

#### CellTextSize

The FontSize for the content of record cells can be customized using the [GridColumn.CellTextSize](#) property. The default font size of the record cells is 14.

#### RecordFont

The FontFamily for the content of record cell can be customized using the [GridColumn.RecordFont](#) property. The default font value is **Helvetica Neue**.

#### FontAttribute

The FontAttribute for the content of record cells can be customized using the [\[GridColumn.FontAttribute\]](#). The record cells text can be customized as **Bold**, or **Italic**, or **None**. The default value of this property is **None**.

To set font attribute for a column, follow the code example:

### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid" ItemsSource="{Binding OrdersInfo}">
  <sfgrid:SfDataGrid.Columns>
    <sfgrid:GridTextColumn MappingName="Freight" FontAttribute="Bold" />
  </sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
```

### C#

```
dataGrid.Columns.Add (new GridTextColumn ()
{
    MappingName = "Freight",
    FontAttribute = FontAttribute.Bold,
});
```

Run the application to render the following output:



### TextAlignment

To get or set `TextAlignment` of the header cell, use the [GridColumn.TextAlignment](#) property. The default alignment of the record cell is `Center`. It can be customized as `Start` or `End`.

### LineBreakMode

When the text for the record cells exceeds the content area, wrap the record cell by setting the [GridColumn.LineBreakMode](#) as `LineBreakMode.WordWrap`.

To use `LineBreakMode`, follow the code example:

#### C#

```
dataGrid.Columns[0].LineBreakMode=LineBreakMode.WordWrap;
```

### GridSwitchColumn

`GridSwitchColumn` inherits all the properties of `GridColumn`. It loads a switch as the content of record cells in the column and responds to value changes in it. The underlying data source can be changed that it toggles the values shown in the switch. The `SfDataGrid` automatically generates `GridSwitchColumn` if the property in the underlying collection of type set to `bool`.

To use `GridSwitchColumn`, follow the code example:

#### XML

```
<ContentPage.BindingContext>
<local:ViewModel />
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid">
```

```
AutoGenerateColumns="True"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfGrid:GridSwitchColumn MappingName="IsClosed" />
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

## C#

```
// Model class
public class Model
{
    private bool _isClosed;
    public bool IsClosed
    {
        get { return _isClosed; }
        set
        {
            this._isClosed = value;
        }
    }
}

// ViewModel class
public class ViewModel
{
    public ViewModel()
    {
        GetOrderDetails(50);
    }
    #region ItemsSource
    private ObservableCollection<OrderInfo> ordersInfo;
    public ObservableCollection<OrderInfo> OrdersInfo
    {
        get { return ordersInfo; }
        set { this.ordersInfo = value; }
    }
    #endregion
    #region ItemSource Generator
    public void GetOrderDetails(int count)
    {
        var orderDetails = new ObservableCollection<OrderInfo>();
        for (int i = 1; i <= count; i++)
        {
            var order = new OrderInfo()
            {
                IsClosed = (i % 2) == 0 ? true : false
            };
            orderDetails.Add(order);
        }
        ordersInfo = orderDetails;
    }
    #endregion
}
```



### Editing for switch column

To edit the switch column, set the [AllowEditing](#) property to `true`. By default, `AllowEditing` is `true`. If `AllowEditing` is set to `false`, you cannot check or uncheck the Switch column.

To set the `AllowEditing` property, follow the code example:

#### XML

```
<sfgrid:GridSwitchColumn MappingName="IsClosed" AllowEditing="true"/>
```

#### C#

```
GridSwitchColumn column = new GridSwitchColumn();  
column.MappingName = "IsClosed";  
column.AllowEditing = true;
```

{% endtabs%}

### GridImageColumn

`GridImageColumn` is derived from `GridColumn`. Hence, it inherits all the properties of `GridColumn`. It displays image as cell content of a column. To create `GridImageColumn`, the property corresponding to the column in the underlying collection must be `ImageSource` type.

In `GridImageColumn`, it is possible to load images in any of the following four ways:

- **FromFile:** Required to specify the path of the file.
- **FromResource:** Required to set image as embedded resource.
- **FromStream:** Required to load image from `byte[]` array.
- **FromURI:** Required to set image from a web service or website.

To load image (embedded resource) in `GridImageColumn`, follow the code example:

#### XML

```
<ContentPage.BindingContext>  
<local:ViewModel />  
</ContentPage.BindingContext>  
<sfGrid:SfDataGrid x:Name="dataGrid"  
AutoGenerateColumns="True"  
ItemsSource="{Binding OrdersInfo}">  
<sfGrid:SfDataGrid.Columns>  
<sfGrid:GridImageColumn MappingName="DealerImage" />  
</sfGrid:SfDataGrid.Columns>  
</sfGrid:SfDataGrid>
```

#### C#

```
// Model class  
public class Model  
{  
    private ImageSource _dealer;  
    public ImageSource DealerImage  
    {  
        get { return _dealer; }  
    }  
}
```

```
set
{
    this._dealer = value;
}
}
}
// ViewModel class
public class ViewModel
{
    public ViewModel()
    {
        GetOrderDetails(50);
    }
    #region ItemsSource
    private ObservableCollection<OrderInfo> ordersInfo;
    public ObservableCollection<OrderInfo> OrdersInfo
    {
        get { return ordersInfo; }
        set { this.ordersInfo = value; }
    }
    #endregion
    #region ItemSource Generator
    public void GetOrderDetails(int count)
    {
        var orderDetails = new ObservableCollection<OrderInfo>();
        for (int i = 1; i <= count; i++)
        {
            var order = new OrderInfo()
            {
                DealerImage = ImageSource.FromResource("DataGridDemo.Buchanan.png") // Need
                to give the image path properly. Here, DataGridDemo denotes the project name
                and Buchanan denotes the image name.
            };
            orderDetails.Add(order);
        }
        ordersInfo = orderDetails;
    }
    #endregion
}
```



### Aspect

SfDataGrid allows you to set the **Aspect** to size the loaded images within the bounds of the grid cell (whether to stretch, crop or letterbox) using the [GridImageColumn.Aspect](#) property. The supported aspects are described below, the default value is AspectFit.

**AspectFill:** Clips the image so that it fills the display area while preserving the aspect (no distortion).

**AspectFit:** Letterboxes the image (if required) so that the entire image fits into the display area, with blank space added to the top/bottom or sides depending on whether the image is wide or tall.

**Fill:** Stretches the image to completely and exactly fill the display area. This may result in the image being distorted.

To set **Aspect** to images loaded inside **GridImageColumn**, refer the below code snippet.

### XML

```
<ContentPage.BindingContext>
<local:ViewModel />
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfGrid:GridImageColumn MappingName="DealerImage" Aspect="AspectFit"/>
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

### GridTemplateColumn

The GridTemplateColumn is derived from GridColumn. Hence, it inherits all the properties of GridColumn. It allows you to extend the functionality of GridColumn with own view by creating the **CellTemplate** or **CellTemplateSelector**.

The following table provides the list of properties in GridTemplateColumn:

Property	Type	Description	Default Value
CellTemplate	DataTemplate	Gets or sets the template that is used to display the contents of the record cells.	Null
CellTemplateSelector	DataTemplateSelector	Gets or sets the template selector that is used to display the contents of the record cells.	Null

### CellTemplate

Underlying records will be the BindingContext for the **CellTemplate**. The following code example shows templating of GridTemplateColumn:

#### XML

```
<syncfusion:GridTemplateColumn MappingName="CustomerID">
  <syncfusion:GridTemplateColumn.CellTemplate>
    <DataTemplate>
      <Label Text="{Binding CustomerID}" TextColor="Blue"
        XAlign="Center" YAlign="Center" />
    </DataTemplate>
  </syncfusion:GridTemplateColumn.CellTemplate>
</syncfusion:GridTemplateColumn>
```

#### C#

```
GridTemplateColumn templateColumn = new GridTemplateColumn()
{
    MappingName = "CustomerID",
    Width = 50,
};
var dataTemplate = new DataTemplate(() =>
{
    var label = new Label()
    {
        TextColor = Color.Blue,
        VerticalOptions = LayoutOptions.Center,
        HorizontalOptions = LayoutOptions.Center
    };
    label.SetBinding(Label.TextProperty, "CustomerID");
    return label;
});
templateColumn.CellTemplate = dataTemplate;
```

The following code example illustrates how template column can be used to load a stock cell inside it:

**XML**

```

<ContentPage.Resources>
<ResourceDictionary>
<local:ImageConverter x:Key="imageConverter" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.ContentView>
<syncfusion:SfDataGrid x:Name="dataGrid"
ColumnSizer="Star">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn HeaderText="Stock Change"
MappingName="StockChange">
<syncfusion:GridTemplateColumn.CellTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Image Grid.Column="0"
Source="{Binding StockChange,
Converter={StaticResource imageConverter}}" />
<Label x:Name="changeValue" Grid.Column="1"
Text="{Binding StockChange}" TextColor="Black"
XAlign="Center" YAlign="Center">
</Label>
</Grid>
</DataTemplate>
</syncfusion:GridTemplateColumn.CellTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
</ContentPage.ContentView>

```

In order to get the above code example working, write a converter to load images inside the [GridCell](#) based on the CellValue. The images that have to be loaded in the [GridCell](#) must be added as EmbeddedResource.

The converter code for loading images in a template column is shown in the following code:

**C#**

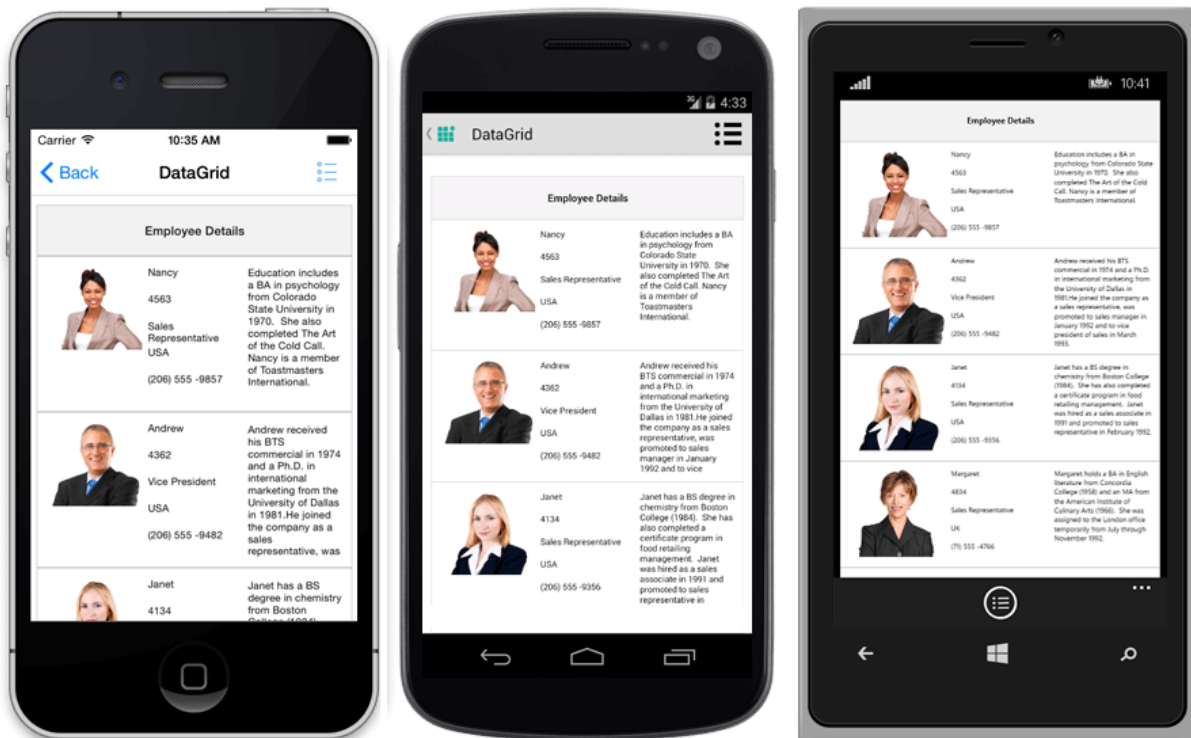
```

public class ImageConverter: IValueConverter
{
public object Convert (object value, Type targetType, object parameter, CultureInfo culture)
{
var data = value as double?;
if (data != null && data > 0)
return ImageSource.FromResource ("DataGridSample.Icons.Green.png");
else
return ImageSource.FromResource ("DataGridSample.Icons.Red.png");
}
public object ConvertBack (object value, Type targetType, object parameter, CultureInfo culture)

```

```
{
    throw new NotImplementedException ();
}
```

The following screenshot shows the different types of columns in the SfDataGrid:



### CellTemplateSelector

Underlying records will be the BindingContext for the CellTemplateSelector. The following code example shows templating of the GridTemplateColumn using the CellTemplateSelector property:

### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="low" >
<Label Text="{Binding Freight}"
TextColor="White"
BackgroundColor="Red"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center" />
</DataTemplate>
<DataTemplate x:Key="average" >
<Label Text="{Binding Freight}"
TextColor="Black"
BackgroundColor="Yellow"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center" />
</DataTemplate>
```

```
</DataTemplate>
<DataTemplate x:Key="high" >
<Label Text="{Binding Freight}"
TextColor="White"
BackgroundColor="Green"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center" />
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<sfgrid:GridTemplateColumn MappingName="Freight" >
<sfgrid:GridTemplateColumn.CellTemplateSelector>
<local:FreightTemplateSelector High="{StaticResource high}"
Average="{StaticResource average}"
Low="{StaticResource low}" />
</sfgrid:GridTemplateColumn.CellTemplateSelector>
</sfgrid:GridTemplateColumn>
```

## C#

```
// FreightTemplateSelector implementation
public class FreightTemplateSelector : DataTemplateSelector
{
    public DataTemplate Low { get; set; }
    public DataTemplate Average { get; set; }
    public DataTemplate High { get; set; }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject
container)
    {
        var value = double.Parse((item as OrderInfo).Freight);
        if (value > 750)
            return High;
        else if (value > 500)
            return Average;
        else
            return Low;
    }
}
```



### Getting row index of a row in GridTemplateColumn

The SfDataGrid provides various resolving methods to resolve the row index of grid rows based on certain criteria. The actual row index of a row can be resolved by using the `ResolveToRowIndex(recordRowIndex)` method.

The row index of a grid row can be obtained in GridTemplateColumn by retrieving the record index of the row using the bound data from its `BindingContext` and by resolving the `recordRowIndex` using the `SfDataGrid.ResolveToRowIndex(recordRowIndex)` method.

### XML

```
// MainPage.Xaml
<sfgrid:GridTemplateColumn HeaderText="ShipCity" MappingName="ShipCity">
  <sfgrid:GridTemplateColumn.CellTemplate>
    <DataTemplate>
      <Button Clicked="button_Clicked" WidthRequest="120" Text="{Binding ShipCity}"/>
    </DataTemplate>
  </sfgrid:GridTemplateColumn.CellTemplate>
</sfgrid:GridTemplateColumn>
```

### C#

```
// MainPage.cs
public partial class MainPage : ContentPage
{
  public MainPage ()
  {
  }
```



```
InitializeComponent();
}
private void button_Clicked(object sender, EventArgs e)
{
    var button = sender as Button;
    var record = button.BindingContext as OrderInfo;
    var recordRowIndex = viewModel.OrderInfoCollection.IndexOf(record);
    var rowIndex = sfGrid.ResolveToRowIndex(recordRowIndex);
}
}
```

**Note:** The row index of the row can also be retrieved by using the [GridTapped](#), [GridDoubleTapped](#), and [GridLongPressed events](#). When using complex layout in a [DataTemplate](#), set the [InputTransparent](#) property of the views loaded in the [DataTemplate](#) of the [GridTemplateColumn](#) to [True](#).

#### *Loading DatePicker and TimePicker together*

Currently Xamarin.Forms does not provide a view that combines both the [DatePicker](#) and the [TimePicker](#) as one control. However, the two controls are available individually.

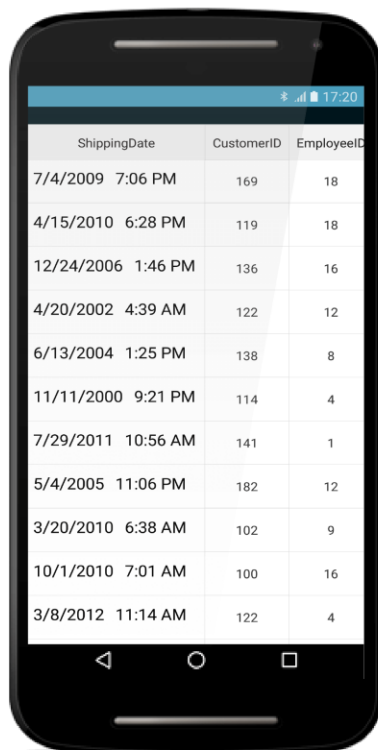
The SfDataGrid supports [DatePicker](#) and [TimePicker](#) in the same column. It can be achieved by loading the [DatePicker](#) and [TimePicker](#) in [StackLayout](#) in the [GridTemplateColumn](#).

To load [DatePicker](#) and [TimePicker](#) together, follow the code example:

#### **XML**

```
<sfgrid:GridTemplateColumn MappingName="ShippingDate">
  <sfgrid:GridTemplateColumn.CellTemplate>
    <DataTemplate>
      <StackLayout Orientation="Horizontal">
        <DatePicker Date="{Binding ShippingDate}" TextColor="Black"/>
        <TimePicker Time="{Binding ShippingTime}" TextColor="Black"/>
      </StackLayout>
    </DataTemplate>
  </sfgrid:GridTemplateColumn.CellTemplate>
</sfgrid:GridTemplateColumn>
```

The following screenshot shows that how [DatePicker](#) and [TimePicker](#) are viewed together:



### GridDateTimeColumn

The [SfDataGrid.GridDateTimeColumn](#) inherits all the properties of the [SfDataGrid.GridColumn](#). It displays the date information as the content of a column. To create the [SfDataGrid.GridDateTimeColumn](#), the property corresponding to the column in the underlying collection must be of type `DateTime`. To enable or disable editing for the particular column, set the [GridColumn.AllowEditing](#) property to true or false. In the editing mode it displays a customized `DatePicker` element which is derived from the [Xamarin.Forms.DatePicker](#). The [SfDatePicker](#) enables to scroll through a list of dates between the [GridDateTimeColumn.MinimumDate](#) and [GridDateTimeColumn.MaximumDate](#) and selects one from it.

### XML

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfgrid:GridDateTimeColumn Format="d"
HeaderText="Shipped Date"
MappingName="ShippedDate" />
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

### C#

```
dataGrid = new SfDataGrid();
GridDateTimeColumn dateColumn = new GridDateTimeColumn()
{
```

```
MappingName = "ShippedDate",
HeaderText = "Shipped Date",
Format = "d"
};
dataGrid.Columns.Add(dateColumn);
```

## C#

```
// Model class
public class Model
{
    private DateTime shippedDate;
    public DateTime ShippedDate
    {
        get { return shippedDate; }
        set
        {
            shippedDate = value;
            RaisePropertyChanged("ShippedDate");
        }
    }
}

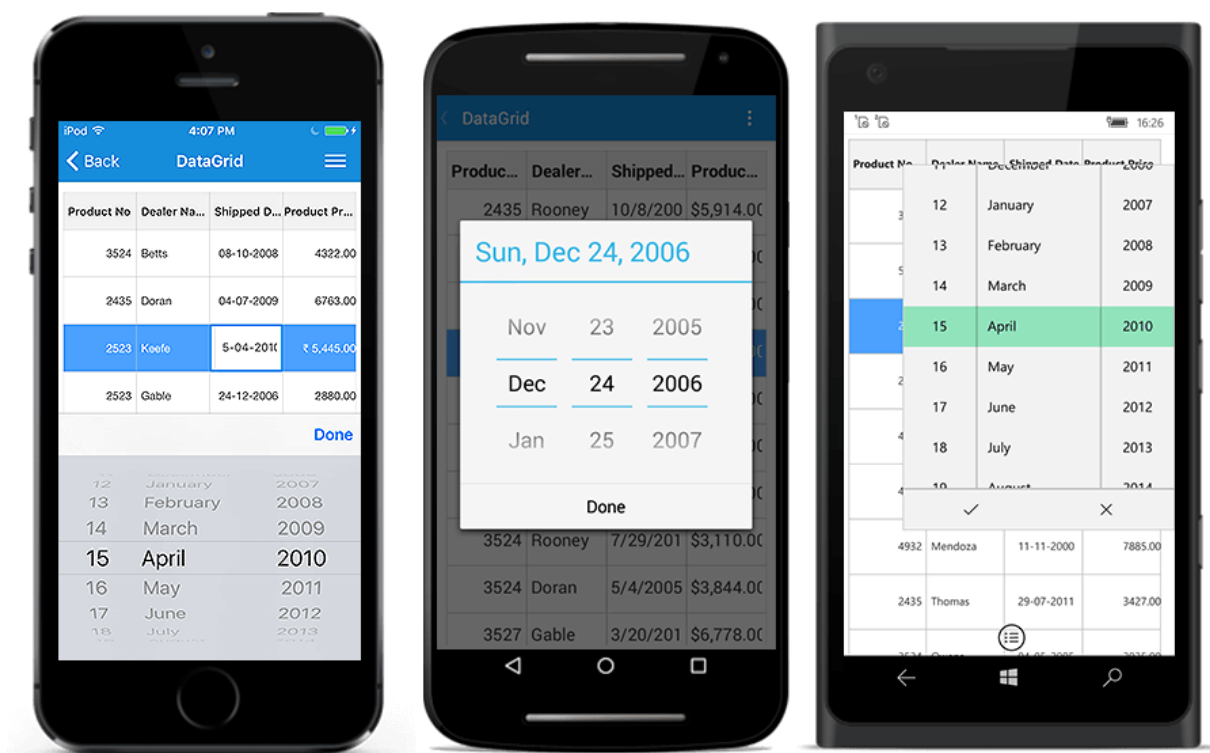
// ViewModel class
public class ViewModel
{
    private List<DateTime> OrderedDates;
    public ViewModel()
    {
        GetOrderDetails(50);
    }
    #region ItemsSource
    private ObservableCollection<OrderInfo> ordersInfo;
    public ObservableCollection<OrderInfo> OrdersInfo
    {
        get { return ordersInfo; }
        set { this.ordersInfo = value; }
    }
    #endregion
    #region ItemSource Generator
    private List<DateTime> GetDateBetween(int startYear, int endYear, int count)
    {
        List<DateTime> date = new List<DateTime>();
        Random d = new Random(1);
        Random m = new Random(2);
        Random y = new Random(startYear);
        for (int i = 0; i < count; i++)
        {
            int year = y.Next(startYear, endYear);
            int month = m.Next(3, 13);
            int day = d.Next(1, 31);
            date.Add(new DateTime(year, month, day));
        }
        return date;
    }
    public void GetOrderDetails(int count)
    {

```

```

var orderDetails= new ObservableCollection<OrderInfo>();
this.OrderedDates = GetDateBetween(2000, 2014, count);
for (int i = 1; i <= count; i++)
{
    var order = new OrderInfo()
    {
        ShippedDate = this.OrderedDates[i - 1],
    };
    orderDetails.Add(order);
}
ordersInfo = orderDetails;
}
#endregion
}

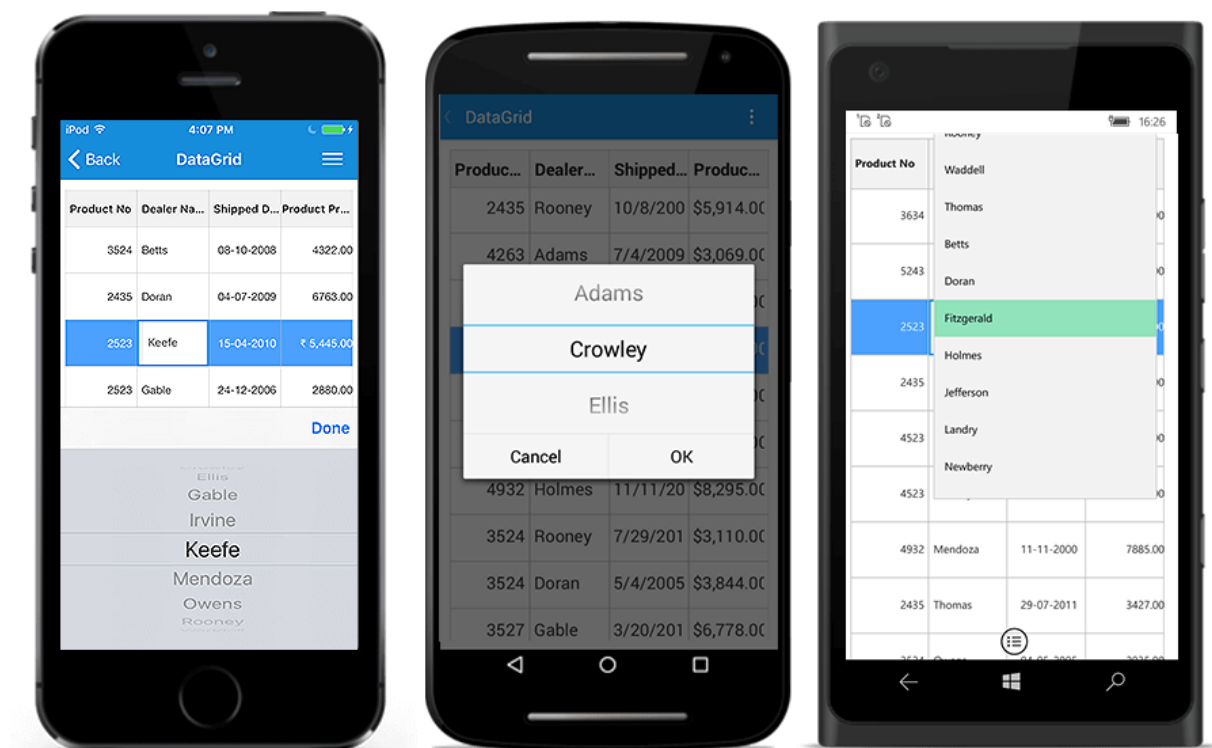
```



### GridPickerColumn

The [GridPickerColumn](#) inherits all the properties of the [SfDataGrid.GridColumn](#). It displays a list of items in the form of a picker as the content of a column. To enable or disable editing for the particular column, set the [GridColumn.AllowEditing](#) property to true or false. In the editing mode it displays a customized [Picker](#) element which is derived from the [Xamarin.Forms.Picker](#). The data source to [Picker](#) can be set by using the [GridPickerColumn.ItemsSource](#) property. The picker column can be populated with data by the following ways:

- Collection of primitive types
- Collection of user-defined types (Custom objects)



### Collection of primitive types

To display the collection of items in the picker drop down, create a `GridPickerColumn` and set its `ItemsSource` property to a simple collection.

To load the `GridPickerColumn` with a simple string collection, follow the code example:

### XML

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfGrid:GridPickerColumn BindingContext="{x:Reference viewModel}"
HeaderText="Dealer Name"
ItemsSource="{Binding CustomerNames}"
MappingName="DealerName"/>
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

### C#

```
dataGrid = new SfDataGrid();
GridPickerColumn pickerColumn = new GridPickerColumn()
{
    BindingContext = viewModel,
    MappingName = "DealerName",
```

```
ItemsSource = viewModel.CustomerNames,
HeaderText = "Dealer Name"
};
dataGrid.Columns.Add(pickerColumn);
```

**C#**

```
// ViewModel class
public class ViewModel
{
    public ObservableCollection<string> CustomerNames { get; set; }
    public ViewModel()
    {
        this.CustomerNames = Customers.ToObservableCollection();
    }
    internal string[] Customers = new string[]
    { "Adams", "Crowley", "Ellis", "Gable", "Irvine", "Keefe", "Mendoza", "Owens", "Rooney", "Wadded", };
}
```

*Collection of user-defined types*

To display a list of user-defined items in the picker drop down, create a `SfDataGrid.GridPickerColumn` and set its `ItemsSource` property to a user-defined collection. Initially, the picker column will be displayed with the values from the `GridColumn.MappingName` property of the column if the `DisplayMemberPath` and `ValueMemberPath` are not set.

*DisplayMemberPath*

Displays a value by comparing values of the properties set as `GridColumn.MappingName` and `ValueMemberPath` in their respective underlying collections. If the values of `ValueMemberPath` property contains the current value of `MappingName` property, its corresponding value of `DisplayMemberPath` property is displayed in the `GridCell`. Or else the `GridCell` appears blank. However, in edit mode the values of the `DisplayMemberPath` property are displayed as picker items.

*ValueMemberPath*

Once editing completed, the column having the `MappingName` equal to the `ValueMemberPath` has its data changed to the corresponding `ValueMemberPath` value for the selected `DisplayMemberPath` value in the picker.

*Customization of picker dropdown values*

To customize the picker data using `DisplayMemberPath` and `ValueMemberPath`, follow the code example:

**XML**

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn HeaderText="Order ID"
MappingName="OrderID"/>
<sfgrid:GridPickerColumn BindingContext="{x:Reference viewModel}"
```

```

HeaderText="Picker Column"
DisplayMemberPath="EmployeeID"
ValueMemberPath="OrderID"
ItemsSource="{Binding PickerInfo}"
MappingName="OrderID"/>
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>

```

**C#**

```

sfGrid = new SfDataGrid();
viewModel = new ViewModel();
sfGrid.ItemsSource = viewModel.OrdersInfo;
GridTextColumn orderIDColumn = new GridTextColumn();
orderIDColumn.MappingName = "OrderID";
orderIDColumn.HeaderText = "Order ID";
GridPickerColumn pickerColumn = new GridPickerColumn();
pickerColumn.MappingName = "OrderID";
pickerColumn.HeaderText = "Picker Column";
pickerColumn.DisplayMemberPath = "EmployeeID";
pickerColumn.ValueMemberPath = "OrderID";
pickerColumn.ItemsSource = viewModel.PickerInfo;
sfGrid.Columns.Add(orderIDColumn);
sfGrid.Columns.Add(pickerColumn);
// ViewModel class
public class ViewModel
{
    public class ViewModel : INotifyPropertyChanged
    {
        public ViewModel ()
        {
            SetRowsToGenerate (100);
            this.PickerInfo = OrdersInfo.ToList();
        }
        #region ItemsSource
        private OrderInfoRepository order;
        private ObservableCollection<OrderInfo> ordersInfo;
        public ObservableCollection<OrderInfo> OrdersInfo
        {
            get { return ordersInfo; }
            set { this.ordersInfo = value; RaisePropertyChanged("OrdersInfo"); }
        }
        public List<OrderInfo> PickerInfo
        {
            get;
            set;
        }
        #endregion
        #region ItemSource Generator
        public void SetRowsToGenerate (int count)
        {
            order = new OrderInfoRepository ();
            ordersInfo = order.GetOrderDetails (count);
        }
        #endregion
        public ObservableCollection<OrderInfo> GetOrderDetails(int count)
    }
}

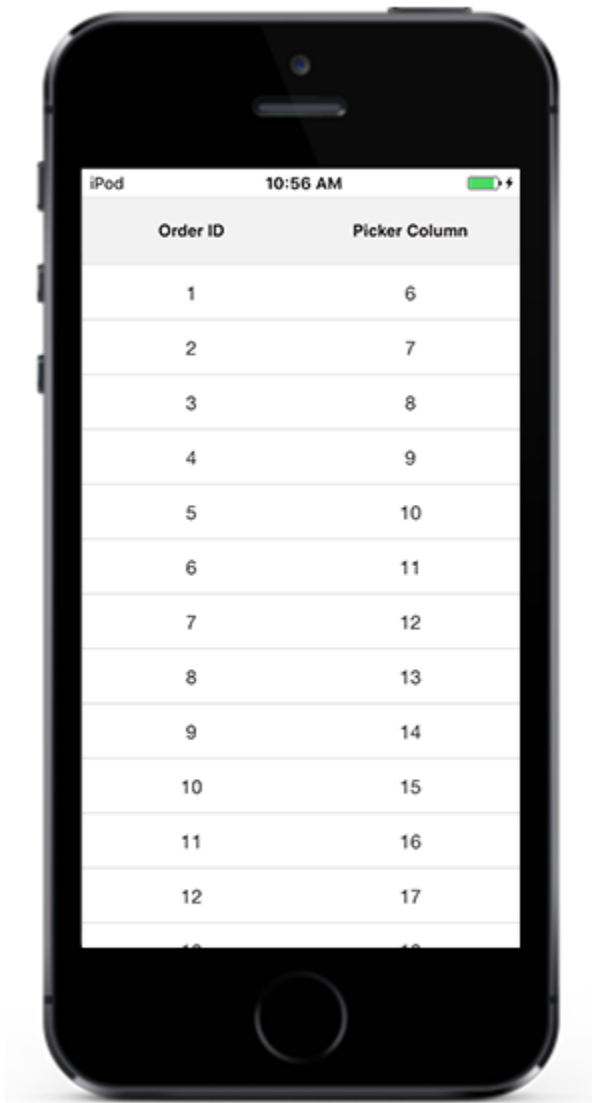
```

```
{
ObservableCollection<OrderInfo> orderDetails = new
ObservableCollection<OrderInfo> ();
for (int i = 1; i <= count; i++)
{
var order = new OrderInfo ()
{
OrderID = i,
EmployeeID = i+5,
};
orderDetails.Add (order);
}
return orderDetails;
}
#region INotifyPropertyChanged implementation
public event PropertyChangedEventHandler PropertyChanged;
private void RaisePropertyChanged(String name)
{
if (PropertyChanged != null)
this.PropertyChanged(this, new PropertyChangedEventArgs (name));
}
#endregion
}
```

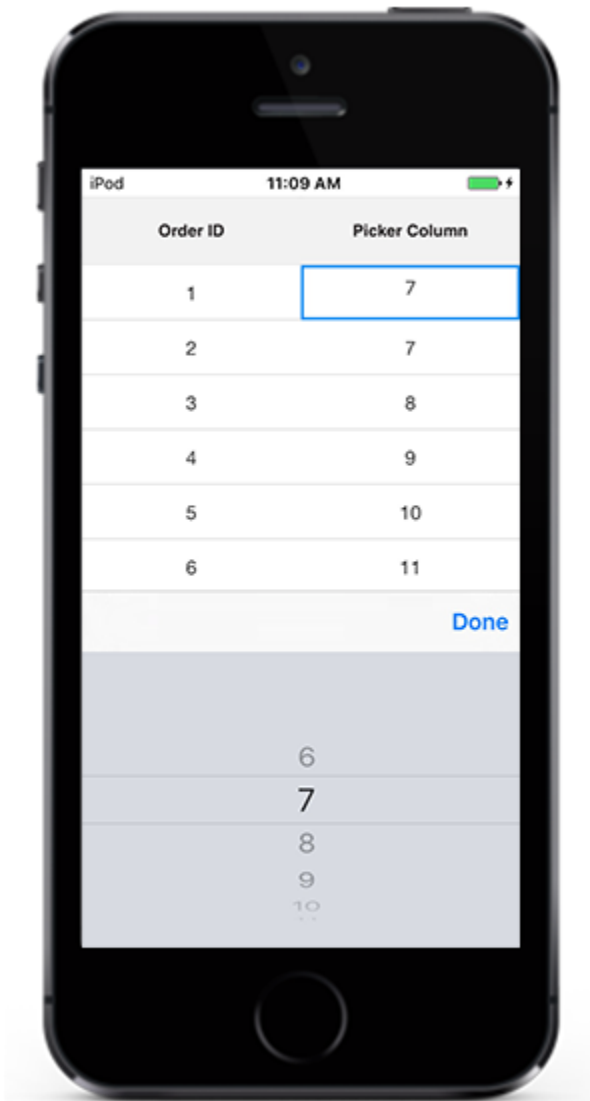
The following screenshots explain the above code and show the working of the **PickerColumn** with **ValueMemberPath** and **DisplayMemberPath** properties set:

In the above code example underlying collection has two properties (OrderID,EmployeeID). A **GridPickerColumn** has been created with MappingName = OrderID, DisplayMemberPath = EmployeeID, ValueMemberPath = OrderID. EmployeeID has the values 6,7,8,9,10.... and OrderID has the values 1,2,3,4,5.... . Initially, the GridCells of the **PickerColumn** will be displayed with the values 6,7,8,9,10.... in row wise order based on the **DisplayMemberPath**.

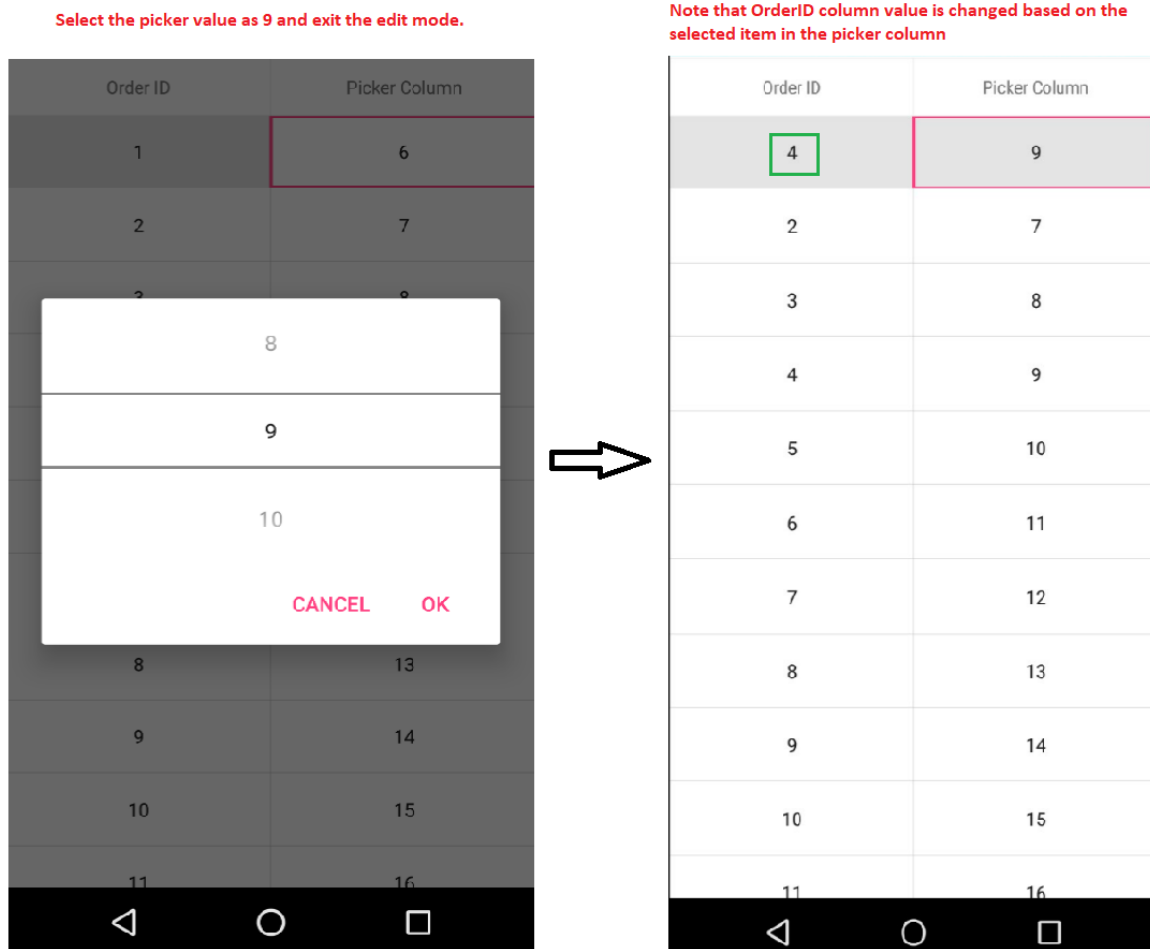




Upon entering the edit mode at `RowColumnIndex(1,1)`, the Picker pop up opens with the picker items as 6,7,8,9,10.... again based on the `DisplayMemberPath`.



When edit mode is exited by selecting a value(9) from the Picker pop up, the `GridCell` at RowColumn index(0,1) displays the corresponding OrderID value for the selected EmployeeID value which is 4.



#### Loading Different ItemSource for each row of GridPickerColumn

You can load the different ItemsSource to each row of GridPickerColumn by setting [GridPickerColumn.ItemsSourceSelector](#) property.

#### Implementing ItemsSourceSelector

[GridPickerColumn.ItemsSourceSelector](#) needs to implement `ItemsSourceSelector` interface which requires you to implement `GetItemsSource` method which receives the below parameters,

- Record – data object associated with row.
- Data Context – Binding context of data grid.

In the below code, ItemsSource for ShipCity column returned based on ShipCountry column value using the record and Binding context of data grid passed to `GetItemsSource` method.

#### XML

```
<ContentPage.Resources>
  <ResourceDictionary>
    <local:ItemSourceSelector x:Key="converter"/>
  </ResourceDictionary>
</ContentPage.Resources>
<sfgrid:SfDataGrid x:Name="dataGrid"
```

```
ItemsSource="{Binding DealerInformation}"
AllowEditing="True"
AutoGenerateColumns="false"
NavigationMode="Cell"
EditTapAction="OnDoubleTap"
SelectionMode="Single">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridPickerColumn BindingContext="{x:Reference viewModel}"
ItemsSource="{Binding CountryList}"
MappingName="ShipCountry"
LoadUIView="True">
</sfgrid:GridPickerColumn>
<sfgrid:GridPickerColumn ItemsSourceSelector="{StaticResource converter}"
MappingName="ShipCity"
LoadUIView="True">
</sfgrid:GridPickerColumn>
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
```

## C#

```
public class ItemSourceSelector : IItemsSourceSelector
{
    public IEnumerable GetItemsSource(object record, object dataContext)
    {
        if (record == null)
        {
            return null;
        }
        var orderinfo = record as DealerInfo;
        var countryName = orderinfo.ShipCountry;
        var viewModel = dataContext as EditingViewModel;
        // Returns ShipCity collection based on ShipCountry.
        if (viewModel.ShipCities.ContainsKey(countryName))
        {
            string[] shipcities = null;
            viewModel.ShipCities.TryGetValue(countryName, out shipcities);
            return shipcities.ToList();
        }
        return null;
    }
}
```

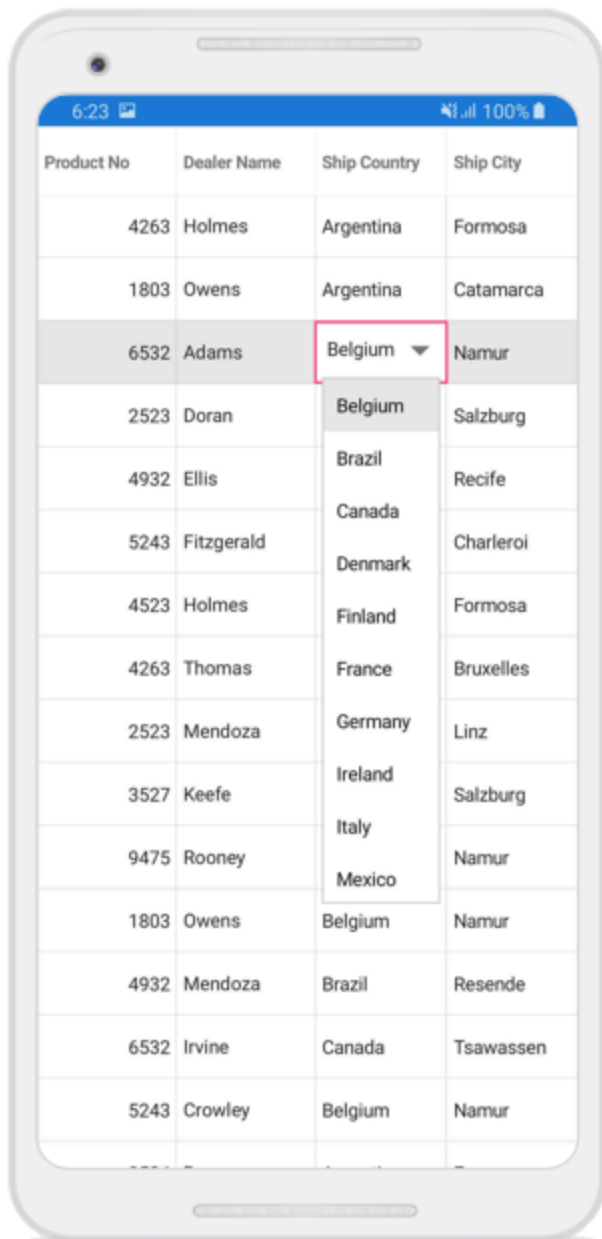
Product No	Dealer Name	Ship Country	Ship City
2435	Holmes	Austria	Graz
4932	Thomas	Argentina	Rosario
2523	Irvine	Canada	Alberta
6532	Adams	Canada	Montral
1803	Owens	Belgium	Namur
<div>Salzburg</div> <div>Linz</div> <div>Wels</div> <div>CANCEL OK</div>			
3524	Rooney	Brazil	Resende
4263	Waddell	Argentina	Formosa
4263	Ellis	Canada	Tsawassen
9475	Gable	Canada	Montral
9475	Thomas	Belgium	Charleroi
9475	Betts	Austria	Linz

Product No	Dealer Name	Ship Country	Ship City
2435	Holmes	Austria	Graz
4932	Thomas	Argentina	Rosario
2523	Irvine	Canada	Alberta
6532	Adams	Canada	Montral
1803	Owens	Belgium	Namur
<div>Alberta</div> <div>Montral</div> <div>Tsawassen</div> <div>CANCEL OK</div>			
3524	Rooney	Brazil	Resende
4263	Waddell	Argentina	Formosa
4263	Ellis	Canada	Tsawassen
9475	Gable	Canada	Montral
9475	Thomas	Belgium	Charleroi
9475	Betts	Austria	Linz

### GridComboBoxColumn

The [GridComboBoxColumn](#) inherits all the properties of the [SfDataGrid.GridColumn](#). It displays a list of items in the form of a [SfComboBox](#) as the content of a column. To enable or disable editing for the particular column, set the [GridColumn.AllowEditing](#) property to true or false. In the editing mode it displays a [SfComboBox](#) element. The data source to SfComboBox can be set by using the [GridColumn.AllowEditing](#) property to true or false. In the editing mode it displays a [SfComboBox](#) element. The data source to SfComboBox can be set by using the [GridComboBoxColumn.ItemsSource](#) property. The combobox column can be populated with data by the following ways:

- Collection of primitive types
- Collection of user-defined types (Custom objects)



### Collection of primitive types

To display the collection of items in the comboBox drop down, create a [GridComboBoxColumn](#) and set its [GridComboBoxColumn.ItemsSource](#) property to a simple collection.

To load the [GridComboBoxColumn](#) with a simple string collection, follow the code example:

### XML

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
```

```
<sfgrid:GridComboBoxColumn BindingContext="{x:Reference viewModel}"
HeaderText="Dealer Name"
ItemsSource="{Binding CustomerNames}"
MappingName="DealerName"/>
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

**C#**

```
dataGrid = new SfDataGrid();
GridComboBoxColumn comboBoxColumn = new GridComboBoxColumn()
{
    BindingContext = viewModel,
    MappingName = "DealerName",
    ItemsSource = viewModel.CustomerNames,
    HeaderText = "Dealer Name"
};
dataGrid.Columns.Add(comboBoxColumn);
```

**C#**

```
// ViewModel class
public class ViewModel
{
    public ObservableCollection<string> CustomerNames { get; set; }
    public ViewModel()
    {
        this.CustomerNames = Customers.ToObservableCollection();
    }
    internal string[] Customers = new string[]
    { "Adams", "Crowley", "Ellis", "Gable", "Irvine", "Keefe", "Mendoza", "Owens", "Roone
y", "Wadded", };
}
```

*Collection of user-defined types*

To display a list of user-defined items in the combo-box drop down, create a [GridComboBoxColumn](#) and set its [GridComboBoxColumn.ItemsSource](#) property to a user-defined collection. Initially, the combo-box column will be displayed with the values from the [GridColumn.MappingName](#) property of the column if the [DisplayMemberPath](#) and [ValueMemberPath](#) are not set.

For more details about the DisplayMemberPath and ValueMemberPath, Please refer [here](#).

*Customizing GridComboBoxColumn*

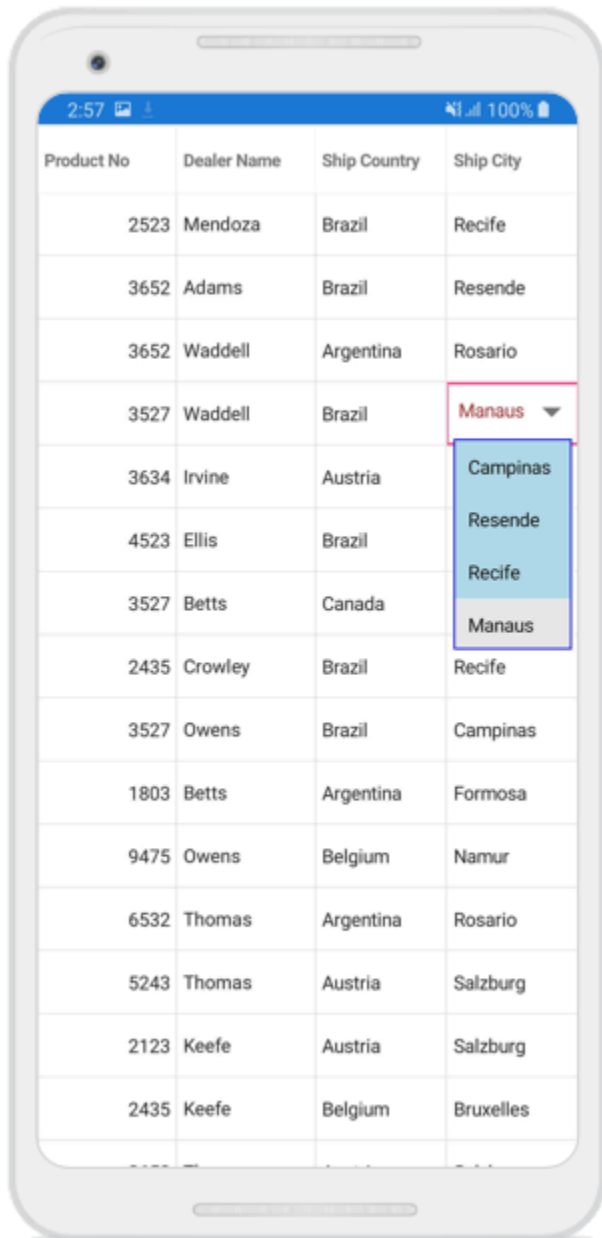
Editable element of the [GridComboBoxColumn](#) can be Customizing by using the custom [GridCellComboBoxRenderer](#).

Replace the default renderer with custom renderer in the SfDataGrid.CellRenderers collection.

**C#**

```
this.dataGrid.CellRenderers.Remove("ComboBox");
this.dataGrid.CellRenderers.Add("ComboBox", new CustomComboBoxRenderer());
public class CustomComboBoxRenderer : GridCellComboBoxRenderer
{
}
```

```
public override void OnInitializeEditView(DataColumnBase dataColumn,
GridComboBox view)
{
    base.OnInitializeEditView(dataColumn, view);
    view.DropDownBackgroundColor = Color.LightBlue;
    view.DropDownBorderColor = Color.Blue;
    view.TextColor = Color.Brown;
}
}
```





*Loading Different ItemSource for each row of GridComboBoxColumn*

You can load the different ItemsSource to each row of GridComboBoxColumn by setting [GridComboBoxColumn.ItemsSourceSelector](#) property.

*Implementing IItemsSourceSelector*

[GridComboBoxColumn.ItemsSourceSelector](#) needs to implement IItemsSourceSelector interface which requires you to implement GetItemsSource method which receives the below parameters,

- Record – data object associated with row.
- Data Context – Binding context of data grid.

In the below code, ItemsSource for ShipCity column returned based on ShipCountry column value using the record and Binding context of data grid passed to GetItemsSource method.

**XML**

```
<ContentPage.Resources>
<ResourceDictionary>
<local:ItemSourceSelector x:Key="converter"/>
</ResourceDictionary>
</ContentPage.Resources>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding DealerInformation}"
AllowEditing="True"
AutoGenerateColumns="false"
NavigationMode="Cell"
EditTapAction="OnDoubleTap"
SelectionMode="Single">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridComboBoxColumn BindingContext="{x:Reference viewModel}"
ItemsSource="{Binding CountryList}"
MappingName="ShipCountry"
LoadUIView="True">
</sfgrid:GridComboBoxColumn>
<sfgrid:GridComboBoxColumn ItemsSourceSelector="{StaticResource converter}"
MappingName="ShipCity"
LoadUIView="True">
</sfgrid:GridComboBoxColumn>
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
```

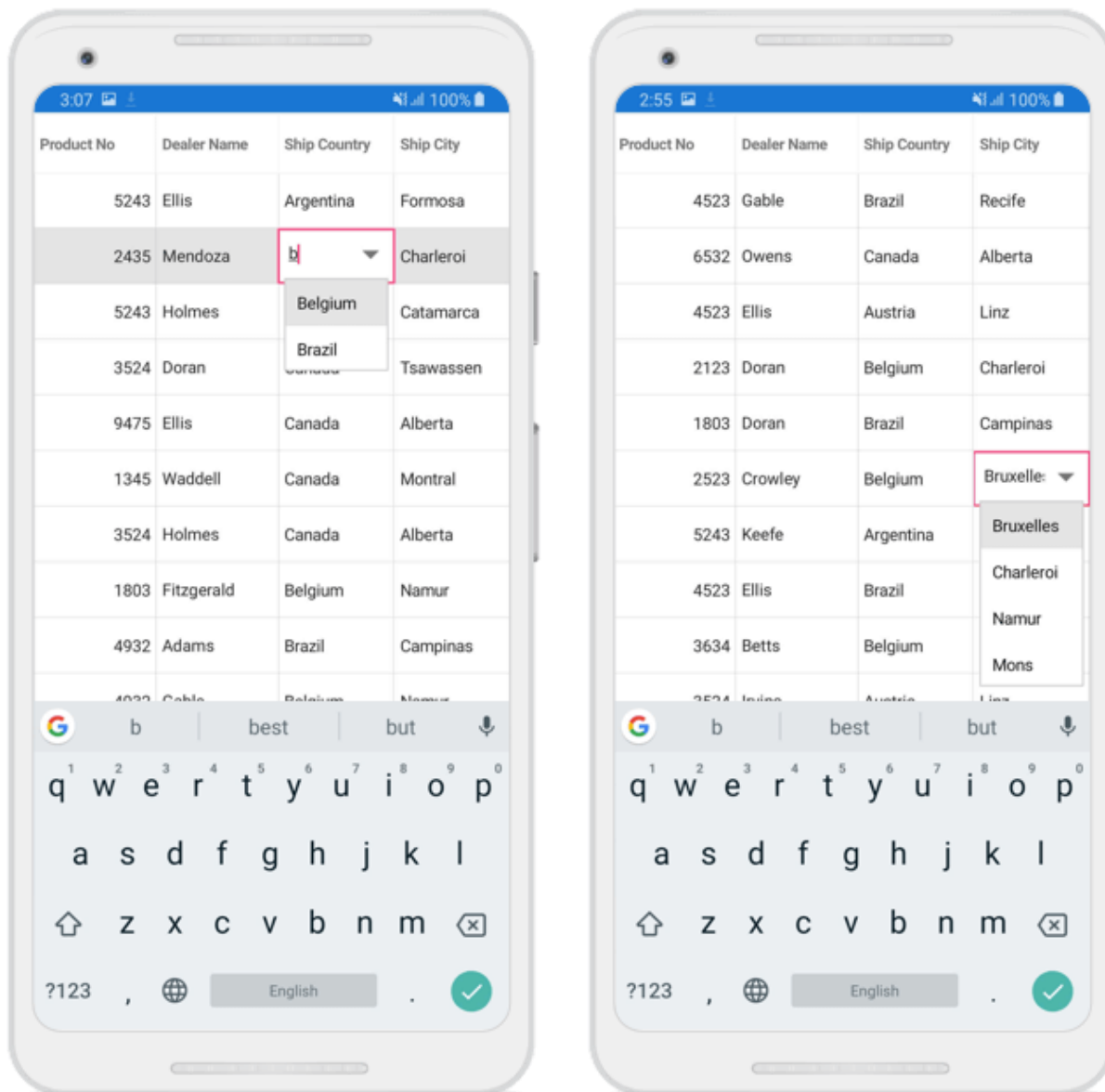
**C#**

```
public class ItemSourceSelector : IItemsSourceSelector
{
    public IEnumerable GetItemsSource(object record, object dataContext)
    {
        if (record == null)
        {
            return null;
        }
        var orderinfo = record as DealerInfo;
        var countryName = orderinfo.ShipCountry;
        var viewModel = dataContext as EditingViewModel;
        // Returns ShipCity collection based on ShipCountry.
```

```

if (viewModel.ShipCities.ContainsKey(countryName))
{
    string[] shipcities = null;
    viewModel.ShipCities.TryGetValue(countryName, out shipcities);
    return shipcities.ToList();
}
return null;
}
}

```



#### Editing the combo box

The [GridComboBoxColumn](#) supports both editable and non-editable text box to choose selected items in given data source. Users can select one item from the suggestion list.

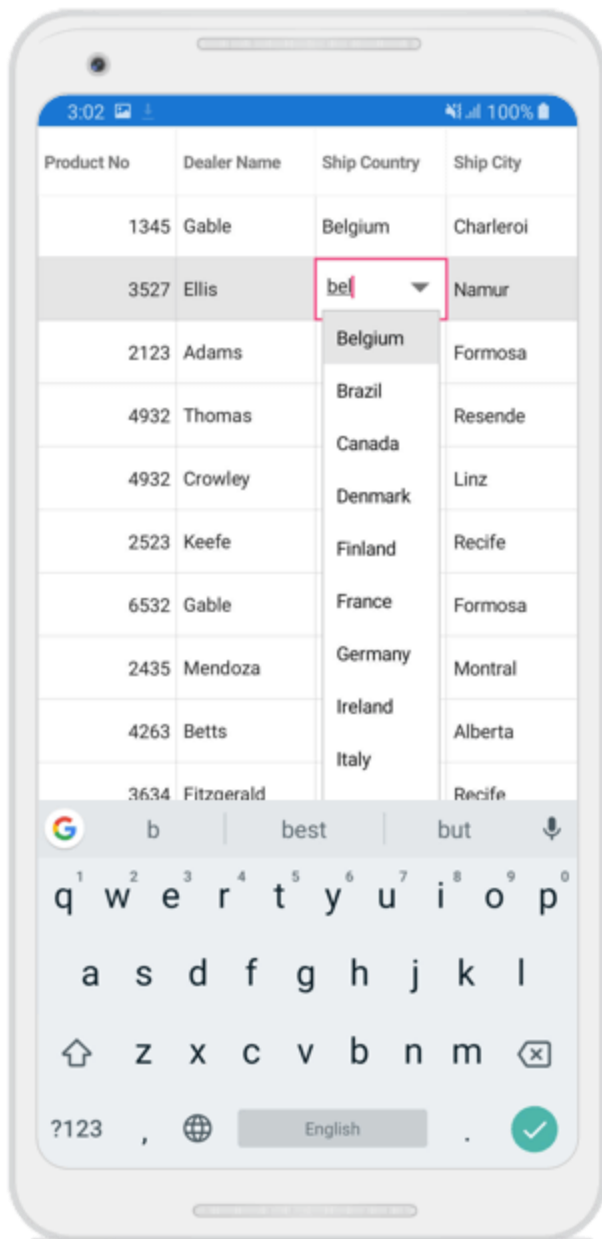
[IsEditableMode](#) property is used to enable the user input in [GridComboBoxColumn](#). The default value is false.

#### XML

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfgrid:GridComboBoxColumn BindingContext="{x:Reference viewModel}"
HeaderText="Dealer Name"
IsEditableMode="True"
ItemsSource="{Binding CustomerNames}"
MappingName="DealerName"/>
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

#### C#

```
dataGrid = new SfDataGrid();
GridComboBoxColumn comboBoxColumn = new GridComboBoxColumn()
{
    BindingContext = viewModel,
    MappingName = "DealerName",
    ItemsSource = viewModel.CustomerNames,
    IsEditableMode = True,
    HeaderText = "Dealer Name"
};
dataGrid.Columns.Add(comboBoxColumn);
```



#### Auto completing on edit mode

The auto completion on the edit mode can be enabled by using the [GridComboBoxColumn.AutoCompleteMode](#) property. Default value is Suggest. Following types of auto complete modes are available,

- Append
- Suggest
- SuggestAppend

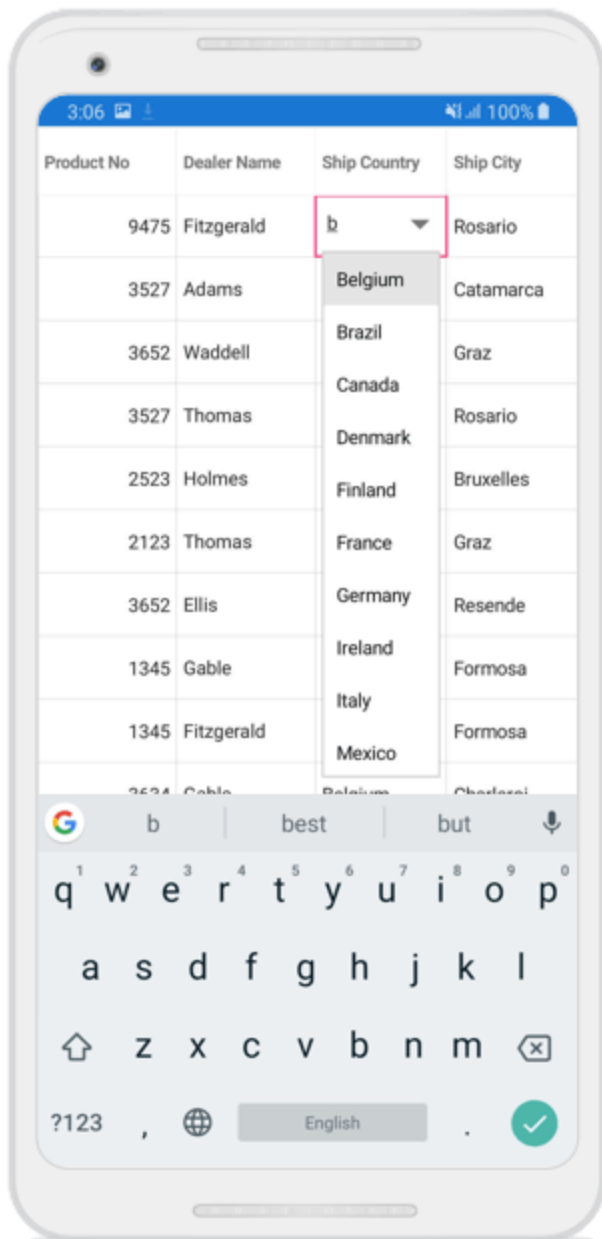
#### XML

```
<ContentPage.BindingContext>
```

```
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfgrid:GridComboBoxColumn BindingContext="{x:Reference viewModel}"
HeaderText="Dealer Name"
IsEditableMode="True"
AutoCompleteMode="Suggest"
ItemsSource="{Binding CustomerNames}"
MappingName="DealerName"/>
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

### C#

```
dataGrid = new SfDataGrid();
GridComboBoxColumn comboBoxColumn = new GridComboBoxColumn()
{
    BindingContext = viewModel,
    MappingName = "DealerName",
    ItemsSource = viewModel.CustomerNames,
    IsEditableMode = True,
    AutoCompleteMode = ComboBoxMode.Suggest,
    HeaderText = "Dealer Name"
};
dataGrid.Columns.Add(comboBoxColumn);
```



### Auto suggesting on edit mode

By default, auto suggestion in the dropdown will display the value based on the [StartsWith](#) filter condition. This can be changed to retrieve the matches with the Contains condition by using the [SuggestionMode](#) property.

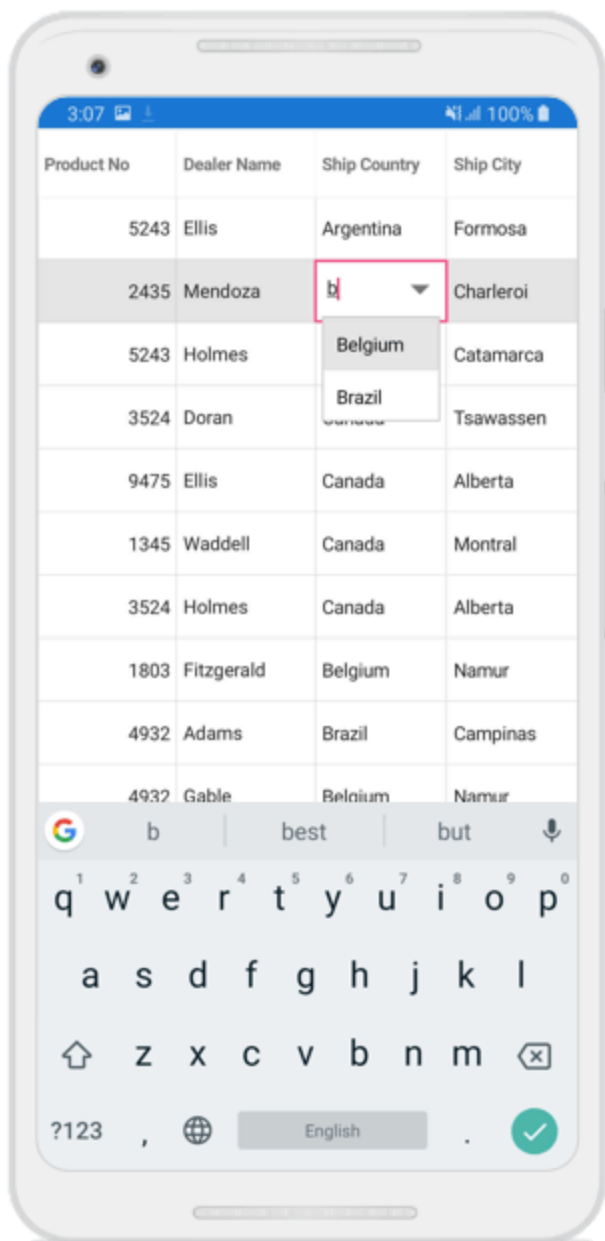
### XML

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfgrid:GridComboBoxColumn BindingContext="{x:Reference viewModel}"
```

```
HeaderText="Dealer Name"
IsEditableMode="True"
CanFilterSuggestions="True"
AutoCompleteMode="Suggest"
SuggestionMode="Contains"
ItemsSource="{Binding CustomerNames}"
MappingName="DealerName"/>
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

## C#

```
dataGrid = new SfDataGrid();
GridComboBoxColumn comboBoxColumn = new GridComboBoxColumn()
{
    BindingContext = viewModel,
    MappingName = "DealerName",
    ItemsSource = viewModel.CustomerNames,
    IsEditableMode = True,
    CanFilterSuggestions = true;
    AutoCompleteMode = ComboBoxMode.Suggest,
    SuggestionMode = SuggestionMode.Contains,
    HeaderText = "Dealer Name"
};
dataGrid.Columns.Add(comboBoxColumn);
```



### GridNumericColumn

The [GridNumericColumn](#) inherits all the properties of [GridColumn](#). It is used to display numeric data. To create [GridNumericColumn](#), the property corresponding to the column in the underlying collection must be a numeric type(int, double, float, etc.). To enable or disable editing a particular column, set the [GridColumn.AllowEditing](#) property to true or false. In the editing mode it displays the [SfNumericTextBox](#) element from the [View](#). To create a [GridNumericColumn](#), follow the code example:

### XML

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfGrid:SfDataGrid x:Name="dataGrid"
```



```
ItemsSource="{Binding OrdersInfo}">
<sfGrid:SfDataGrid.Columns>
<sfgrid:GridNumericColumn NumberDecimalDigits="0"
HeaderText="Product No"
MappingName="ProductNo"/>
</sfGrid:SfDataGrid.Columns>
</sfGrid:SfDataGrid>
```

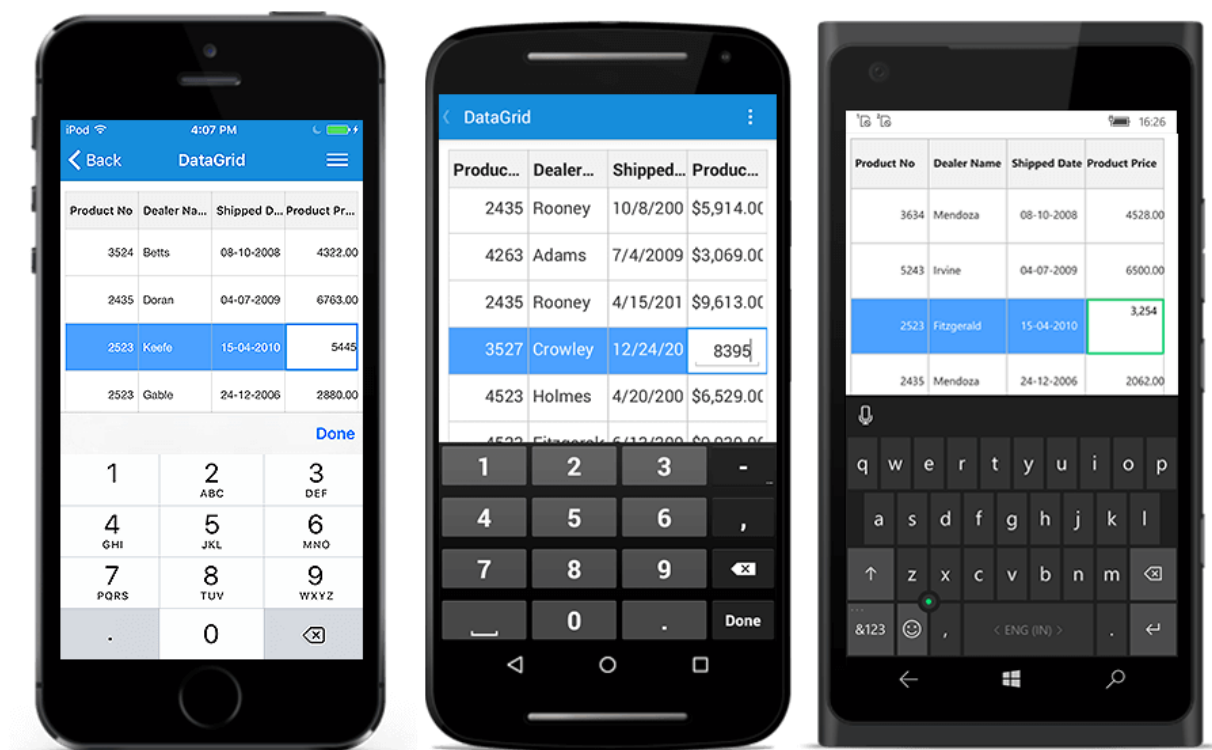
## C#

```
dataGrid = new SfDataGrid();
GridNumericColumn numericColumn = new GridNumericColumn()
{
    MappingName = "ProductNo",
    HeaderText = "Product No",
    NumberDecimalDigits = 0
};
dataGrid.Columns.Add(numericColumn);
```

### Number formatting

The `GridNumericColumn` allows formatting the numeric data with culture-specific information.

- [NumberDecimalDigits](#): To change the number of decimal digits to be displayed after the decimal point, use the `GridNumericColumn.NumberDecimalDigits` property.
- [NumberDecimalSeparator](#): By default, the dot(.) operator separates the decimal part of numeric value. Any operator can be used as decimal separator using the `GridNumericColumn.NumberDecimalSeparator` property.
- [NumberGroupSeparator](#): By default, the comma(,) separates group of digits before the decimal point. Any operator can be used as group separator using the `GridNumericColumn.NumberGroupSeparator` property.
- [NumberGroupSizes](#): To change the number of digits in each group before the decimal point on numeric values, use the `GridNumericColumn.NumberGroupSizes` property.
- [NumberNegativePattern](#): To format the pattern of negative numeric values, use `GridNumericColumn.NumberNegativePattern`.
- [MinValue](#): To set the minimum value for the numeric column, use the `GridNumericColumn.MinValue` property.
- [MaxValue](#): To set the maximum value for the numeric column, use the `GridNumericColumn.MaxValue` property.



### Row header

Row header is a special column placed as first cell of each row, and it will be always frozen. To enable the row header, set the [SfDataGrid.ShowRowHeader](#) to true.

Further, the SfDataGrid allows customizing the row header width using the [SfDataGrid.RowHeaderWidthProperty](#). The default value of `SfDataGrid.RowHeaderWidth` is 20.

To enable and customize the row header, follow the code example:

### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
    ColumnSizer="Star"
    ShowRowHeader="True"
    RowHeaderWidth="50"
    ItemsSource="{Binding OrdersInfo}">
</sfgrid:SfDataGrid>
```

### C#

```
dataGrid.ShowRowHeader = true;
dataGrid.RowHeaderWidth = 50;
```

### Bind a view model property inside header template?

The SfDataGrid allows binding the view model property to the [HeaderTemplate](#) by setting the BindingContext of the the [GridColumn](#) as ViewModel.

To bind a view model property inside `HeaderTemplate`, follow the code example:

#### **XML**

```
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
AutoGenerateColumns="False"
ColumnSizer="Star"
SelectionMode="Single">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn MappingName="OrderID">
<sfgrid:GridColumn.HeaderTemplate>
<DataTemplate>
<Label BindingContext="{StaticResource viewModel}"
VerticalTextAlignment="Center"
HorizontalTextAlignment="Center"
Text="{Binding HeaderText}" TextColor="Blue" IsVisible="{Binding
_Visibility}"/>
</DataTemplate>
</sfgrid:GridColumn.HeaderTemplate>
</sfgrid:GridTextColumn>
<sfgrid:GridTextColumn MappingName="EmployeeID"/>
<sfgrid:GridTextColumn MappingName="FirstName"/>
<sfgrid:GridTextColumn MappingName="ShipCity"/>
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
```



## ColumnSizer

The SfDataGrid allows to apply **ColumnSizer** for the [GridColumn](#) by setting the [SfDataGrid.ColumnSizer](#) property.

To apply **ColumnSizer** in the SfDataGrid, follow the code example:

### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ColumnSizer="None">
```

### C#

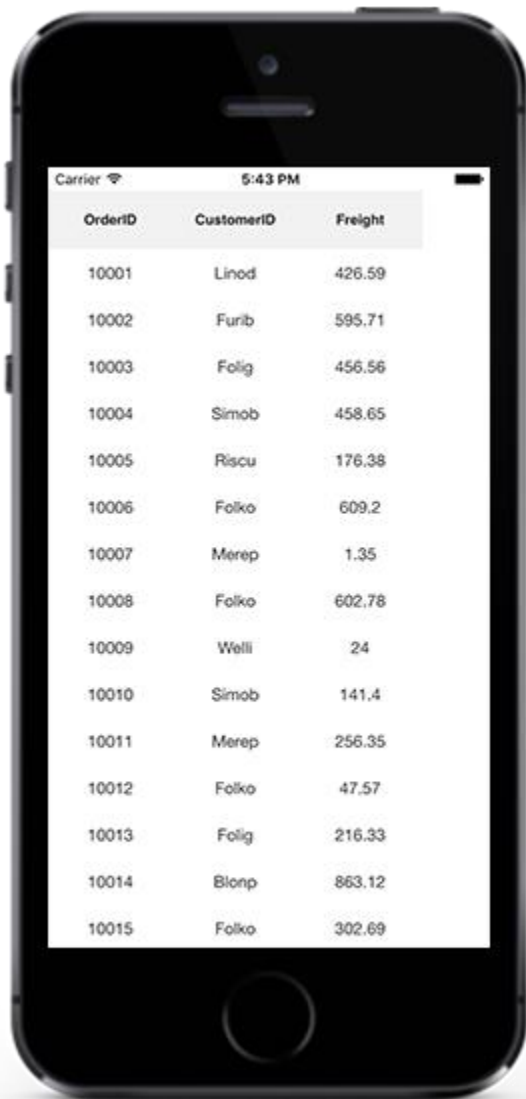
```
dataGrid.ColumnSizer = ColumnSizer.None;
```

The SfDataGrid applies width for all the `GridColumn`s in the [SfDataGrid.Columns](#) collection based on the `SfDataGrid.ColumnSizer` property. Following lists of options are available to set width of the column:

- None
- LastColumnFill
- Star
- Auto

#### `ColumnSizer.None`

No column sizing is applied when the [SfDataGrid.ColumnSizer](#) set to [None](#). Columns are arranged in view based on the [SfDataGrid.DefaultColumnWidth](#) property. This is the default value of the `SfDataGrid.ColumnSizer` property.



OrderID	CustomerID	Freight
10001	Linod	426.59
10002	Furib	595.71
10003	Folig	456.56
10004	Simob	458.65
10005	Riscu	176.38
10006	Folko	609.2
10007	Merep	1.35
10008	Folko	602.78
10009	Welli	24
10010	Simob	141.4
10011	Merep	256.35
10012	Folko	47.57
10013	Folig	216.33
10014	Blonp	863.12
10015	Folko	302.69

### ColumnSizer.LastColumnFill

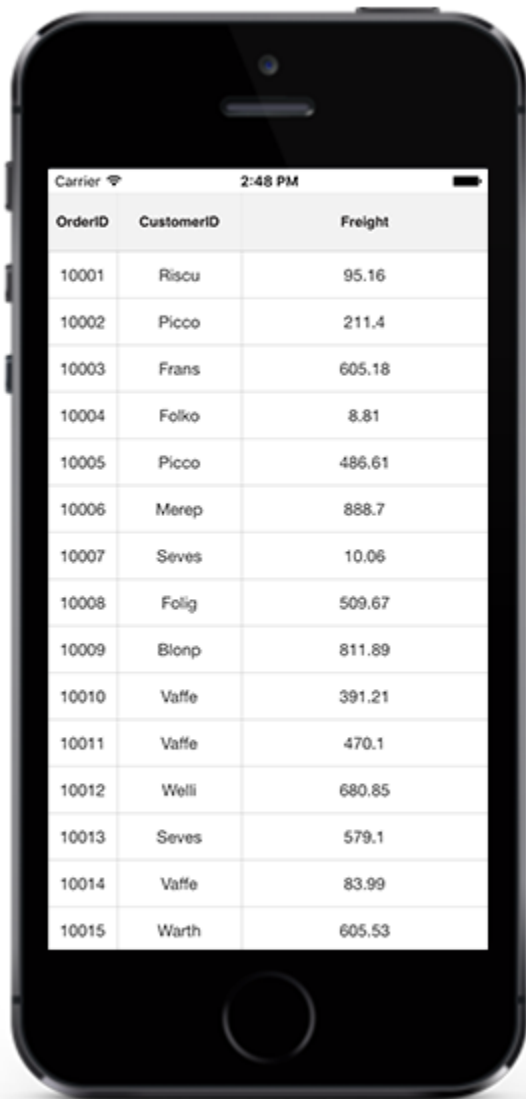
When the [SfDataGrid.ColumnSizer](#) is [LastColumnFill](#), the column width of the [GridColumns](#) are adjusted with respect to [SfDataGrid.DefaultColumnWidth](#) property. In case the columns does not fill the entire view space, width of the last column fills the unoccupied space in the view.

### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ColumnSizer="LastColumnFill">
```

### C#

```
dataGrid.ColumnSizer = ColumnSizer.LastColumnFill;
```



OrderID	CustomerID	Freight
10001	Riscu	95.16
10002	Picco	211.4
10003	Frans	605.18
10004	Folko	8.81
10005	Picco	486.61
10006	Merep	888.7
10007	Seves	10.06
10008	Folig	509.67
10009	Blonp	811.89
10010	Vaffe	391.21
10011	Vaffe	470.1
10012	Welli	680.85
10013	Seves	579.1
10014	Vaffe	83.99
10015	Warth	605.53

### ColumnSizer.Star

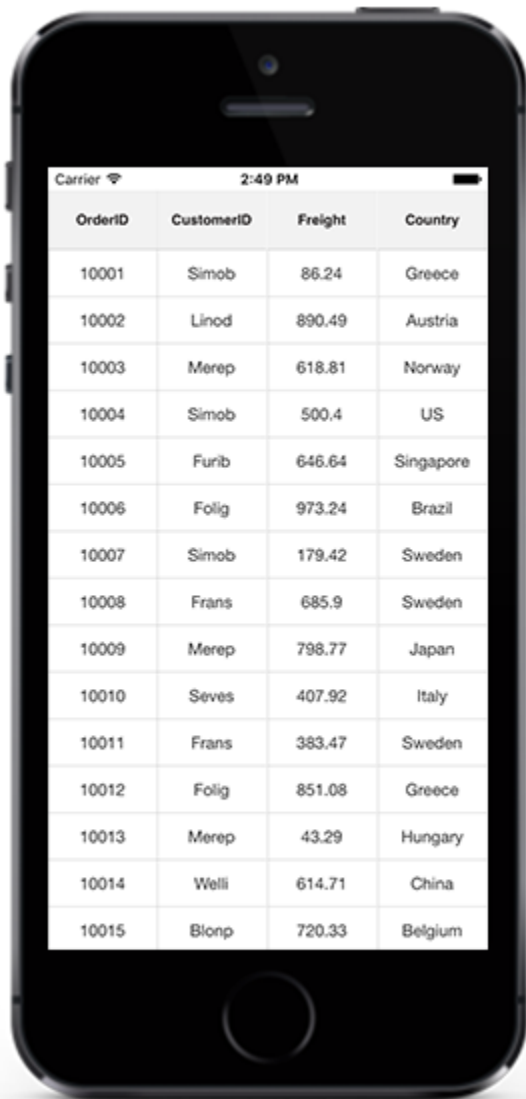
When the [SfDataGrid.ColumnSizer](#) is [Star](#), all the [GridColumn](#)s are adjusted to an equal column width to fit within the view. Setting [ColumnSizer](#) to [Star](#) will disable the [HorizontalScrolling](#) in the [SfDataGrid](#).

### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ColumnSizer="Star">
```

### C#

```
dataGrid.ColumnSizer = ColumnSizer.Star;
```



OrderID	CustomerID	Freight	Country
10001	Simob	86.24	Greece
10002	Linod	890.49	Austria
10003	Merep	618.81	Norway
10004	Simob	500.4	US
10005	Furib	646.64	Singapore
10006	Folig	973.24	Brazil
10007	Simob	179.42	Sweden
10008	Frans	685.9	Sweden
10009	Merep	798.77	Japan
10010	Seves	407.92	Italy
10011	Frans	383.47	Sweden
10012	Folig	851.08	Greece
10013	Merep	43.29	Hungary
10014	Welli	614.71	China
10015	Blonp	720.33	Belgium

### ColumnSizer.Auto

When the [SfDataGrid.ColumnSizer](#) is [Auto](#), the width of the `GridColumn`s are adjusted based on the header text or cell contents.

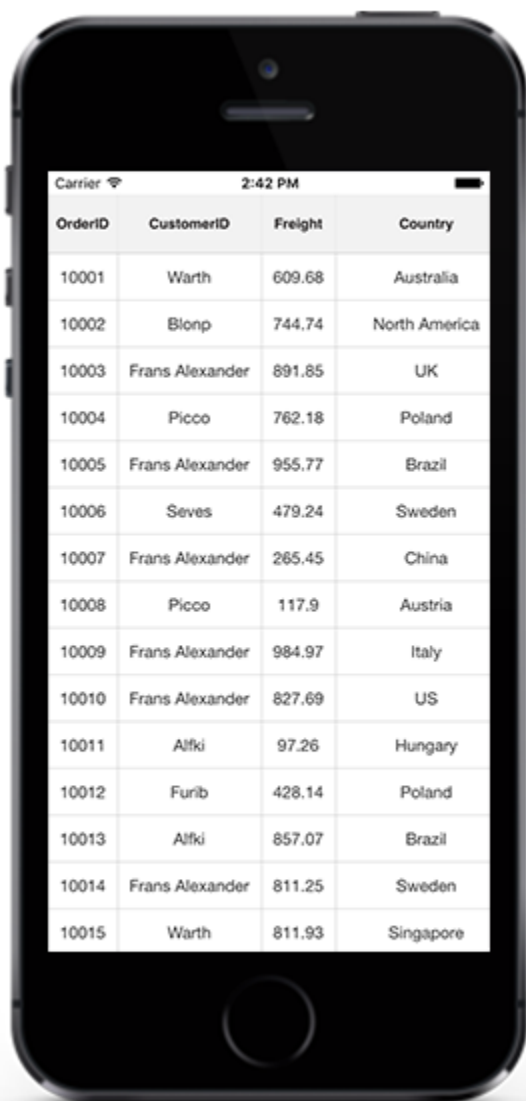
### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ColumnSizer="Auto">
```

### C#

```
dataGrid.ColumnSizer = ColumnSizer.Auto;
```





OrderID	CustomerID	Freight	Country
10001	Warth	609.68	Australia
10002	Blonp	744.74	North America
10003	Frans Alexander	891.85	UK
10004	Picco	762.18	Poland
10005	Frans Alexander	955.77	Brazil
10006	Seves	479.24	Sweden
10007	Frans Alexander	265.45	China
10008	Picco	117.9	Austria
10009	Frans Alexander	984.97	Italy
10010	Frans Alexander	827.69	US
10011	Alfki	97.26	Hungary
10012	Furib	428.14	Poland
10013	Alfki	857.07	Brazil
10014	Frans Alexander	811.25	Sweden
10015	Warth	811.93	Singapore

---

**Note:** If any column explicitly specified a width using the [GridColumn.Width](#) property, that column is not considered ColumnSizing width and skipped while applying the ColumnSizer for the grid columns.

---

How to

*Apply ColumnSizer for a particular column*

To apply column sizing to individual column, use the [GridColumn.ColumnSizer](#) property. The `GridColumn.ColumnSizer` property is also a type of the [ColumnSizer](#). If the `GridColumn.ColumnSizer` is not explicitly set to a value, then it takes the value of the [SfDataGrid.ColumnSizer](#) and applies width to the columns accordingly.

To apply `ColumnSizer` for a particular column, follow the code example:

**XML**

```
<sfgrid:SfDataGrid x:Name="dataGrid"
```

```

AutoGenerateColumns="True"
ColumnSizer="None">
<syncfusion:GridTextColumn MappingName="OrderID"
ColumnSizer = "Auto" />

```

**C#**

```

GridTextColumn textColumn = new GridTextColumn();
textColumn.MappingName = "CustomerID";
textColumn.HeaderText = "Full Name";
textColumn.ColumnSizer = ColumnSizer.Auto;

```

*Fill remaining width for any column*

The **SfDataGrid** allows to fill the remaining width in view for any column using [GridColumn.ColumnSizer](#) property.

The **GridColumn.ColumnSizer** has higher priority than the [SfDataGrid.ColumnSizer](#) property. Hence, individual columns having the **GridColumn.ColumnSizer** property set will not be included in the column size calculations of the **SfDataGrid**. To fill the column with remaining width in view, set the **GridColumn.ColumnSizer** property as [ColumnSizer.LastColumnFill](#). Refer to the following code example to achieve the same:

In the below code snippet, the **SfDataGrid** is applied with **ColumnSizer.Star** and the second column is applied with **ColumnSizer.LastColumnFill**. Hence the second column will take up the remaining space after other columns are rendered with star size.

**XML**

```

<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrdersInfo}">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn MappingName="OrderID"/>
<sfgrid:GridTextColumn MappingName="CustomerID"
ColumnSizer="LastColumnFill"/>
<sfgrid:GridTextColumn MappingName="Salary"/>
<sfgrid:GridTextColumn MappingName="Country"/>
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>

```

**C#**

```

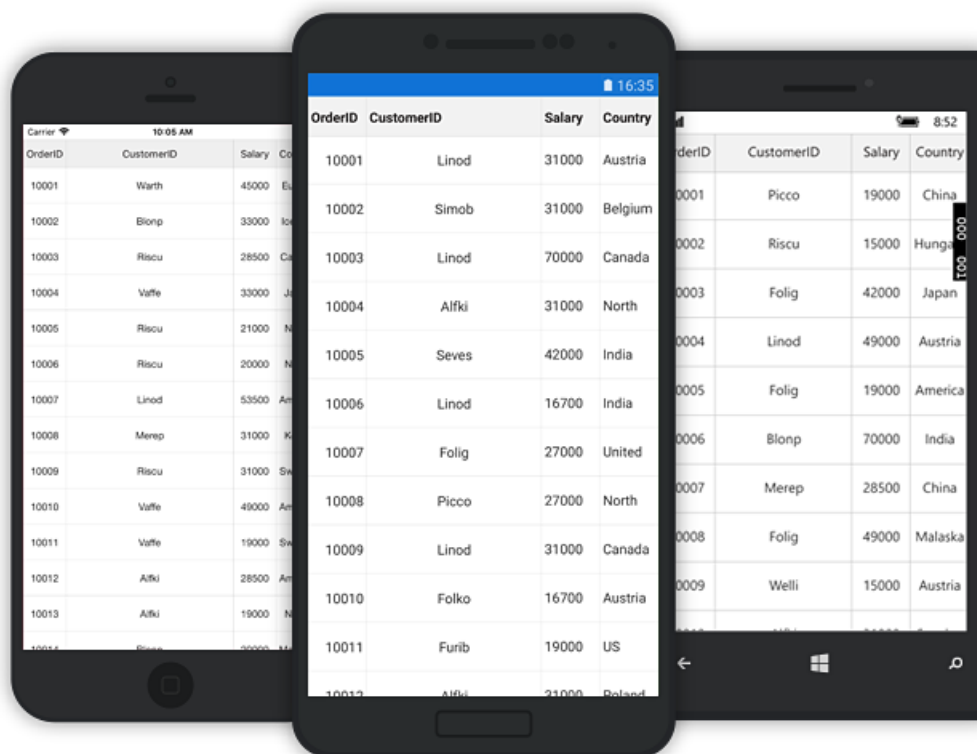
SfDataGrid dataGrid;
ViewModel viewModel;
public MainPage()
{
InitializeComponent();
dataGrid = new SfDataGrid();
viewModel = new ViewModel();
dataGrid.AutoGenerateColumns = false;
dataGrid.ItemsSource = viewModel.OrdersInfo;
GridTextColumn orderIDColumn = new GridTextColumn();
orderIDColumn.MappingName = "OrderID";
GridTextColumn customerIDColumn = new GridTextColumn();

```

```

customerIDColumn.MappingName = "CustomerID";
customerIDColumn.ColumnSizer = ColumnSizer.LastColumnFill;
GridTextColumn salaryColumn = new GridTextColumn();
salaryColumn.MappingName = "Salary";
GridTextColumn countryColumn = new GridTextColumn();
countryColumn.MappingName = "Country";
dataGrid.Columns.Add(orderIDColumn);
dataGrid.Columns.Add(customerIDColumn);
dataGrid.Columns.Add(salaryColumn);
dataGrid.Columns.Add(countryColumn);
}

```



### Refreshing ColumnSizer at runtime

To refresh the column sizing for `SfDataGrid.Columns` at runtime, use the [SfDataGrid.GridColumnSizer.Refresh\(\)](#) method.

Consider that `ColumnSizer.Auto` is applied to the `SfDataGrid`. If the underlying values are changed at run time, refresh the column sizer as follows:

### XML

```

<StackLayout HorizontalOptions="Center"
Orientation="Vertical">
<Button x:Name="button"
Text="Refresh ColumnSizer"
HeightRequest="100"
HorizontalOptions="Center"
Clicked="ColumnSizerChanged"/>
<sfgrid:SfDataGrid x:Name="dataGrid"

```

```

AutoGenerateColumns="True"
AllowEditing="True"
ColumnSizer="Auto"
ItemsSource="{Binding OrdersInfo}">
</sfgrid:SfDataGrid>
</StackLayout>

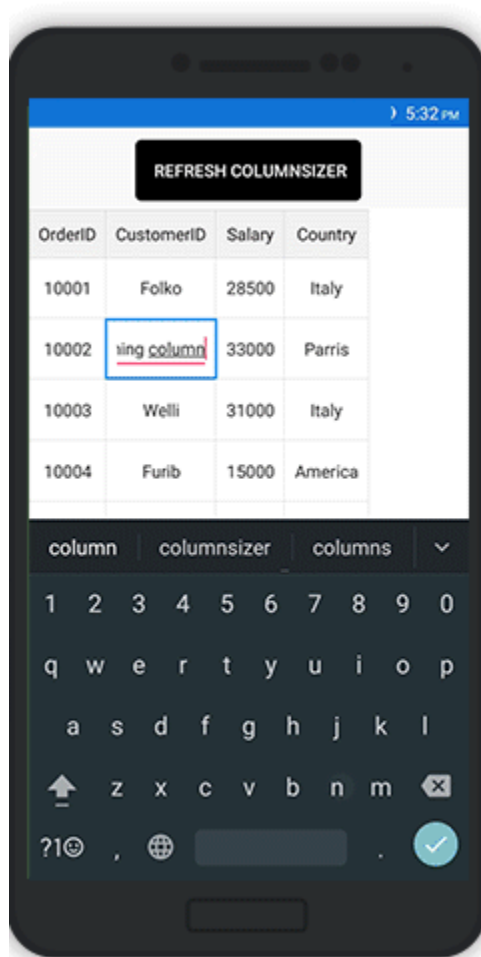
```

## C#

```

private void ColumnSizerChanged(object sender, EventArgs e)
{
    //Refreshes the column sizer of the SfDataGrid
    dataGrid.GridColumnSizer.Refresh(true);
}

```



### Resetting column width to apply ColumnSizer

By default, columns having the [GridColumn.Width](#) property set will not be included for column sizer calculations of the SfDataGrid. To include the width columns and reset the column sizer at runtime, set the [GridColumn.Width](#) property to double.NaN before calling the [SfDataGrid.GridColumnSizer.Refresh\(\)](#) method. Refer to the following code example to achieve the same:


## XML

```
<StackLayout
Orientation="Vertical">
<Button x:Name="button"
Text=" Reset ColumnWidth"
TextColor="White"
HeightRequest="50"
BackgroundColor="Black"
HorizontalOptions="Center"
Clicked="ColumnSizerChanged"/>
<sfgrid:SfDataGrid x:Name="dataGrid"
AllowEditing="True"
AutoGenerateColumns="False"
ItemsSource="{Binding OrdersInfo}">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn MappingName="OrderID" Width="20"/>
<sfgrid:GridTextColumn MappingName="CustomerID"/>
<sfgrid:GridTextColumn MappingName="Salary"/>
<sfgrid:GridTextColumn MappingName="Country"/>
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
</StackLayout>
```

## C#


```
private void ColumnSizerChanged(object sender, EventArgs e)
{
    //Resets the widths for the columns having GridColumn.Width property set
    ResetColumnsWidth();
    dataGrid.GridColumnSizer.Refresh(true);
}

private void ResetColumnsWidth()
{
    foreach (var column in dataGrid.Columns)
    {
        // Setting NaN values to columns for which width is applied
        if (!double.IsNaN(column.Width))
        {
            column.Width = double.NaN;
        }
    }
}
```




RESET COLUMNWIDTH

OrderID	CustomerID	Salary	Country
10001	Alfki	70000	Korea
10002	Folig	19000	Nepal
10003	Blonp	24500	Belgium
10004	Merep	15000	Canada
10005	Seves	49000	India
10006	Folig	28500	Hungary
10007	Folig	24500	Brazil
10008	Riscu	24500	Belgium
10009	Folko	37500	Japan
10010	Frans	37500	Austria



After resetting  
column width and  
refreshing the  
columnSizer



RESET COLUMNWIDTH

OrderID	CustomerID	Salary	Country
10001	Alfki	70000	Korea
10002	Folig	19000	Nepal
10003	Blonp	24500	Belgium
10004	Merep	15000	Canada
10005	Seves	49000	India
10006	Folig	28500	Hungary
10007	Folig	24500	Brazil
10008	Riscu	24500	Belgium
10009	Folko	37500	Japan
10010	Frans	37500	Austria

#### Customize auto width calculation for a column

For cases, where a column might require more width than the applied auto width or if you want to apply your own custom logic to calculate the auto width of a column, return a desired width in the [GetColumnAutoSizeWidth\(\)](#) override of the custom written column-sizer class derived from [GridColumnSizer](#) and assign it to the [SfDataGrid.GridColumnSizer](#) property.

If in case you want to modify the auto calculations of a column's header cell alone, return a desired width in the [GetHeaderCellWidth\(\)](#) override of your custom column-sizer class.

#### C#

```
public class CustomColumnSizer : GridColumnSizer
{
    protected override double GetColumnAutoSizeWidth(GridColumn column)
    {
        if (column.MappingName == "OrderID")
        {
            // return width based on your logic
        }
        else
        {
            return base.GetColumnAutoSizeWidth(column);
        }
    }
    protected override double GetHeaderCellWidth(GridColumn column)
```

```

{
    if (column.MappingName == "CustomerID")
    {
        // return width based on your logic
    }
    else
    {
        return base.GetHeaderCellWidth(column);
    }
}
}

```

## XML

```

<ContentPage.Resources>
<local:CustomColumnSizer x:Key="CustomColumnSizer"/>
</ContentPage.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo,Mode=TwoWay}"
ColumnSizer="Auto"
GridColumnSizer="{x:StaticResource CustomColumnSizer}">
</syncfusion:SfDataGrid>

```

## C#

```
dataGrid.GridColumnSizer = new CustomColumnSizer();
```

## Star column sizer ratio support

To customize the `ColumnSizer.Star` width calculation, write a custom `GridColumnSizer` class derived from [GridColumnSizer](#) and assign it to the `SfDataGrid.ColumnSizer` property. To implement your own logic to divide column width in different ratios, override the [SetStarWidthForColumns](#) method in your custom `GridColumnSizer` class.

To set star sizer ratio for individual column, follow the code example:

## C#

```

<sfgrid:SfDataGrid.Columns >
<sfgrid:GridTextColumn HeaderText="OrderID" MappingName="OrderID"
local:StarSizerRatioHelpers.ColumnRatio="2"/>
<sfgrid:GridTextColumn HeaderText="CustomerID" MappingName="CustomerID"
local:StarSizerRatioHelpers.ColumnRatio="3"/>
<sfgrid:GridTextColumn HeaderText="Salary" MappingName="Salary"/>
<sfgrid:GridTextColumn HeaderText="Country" MappingName="Country"/>
</sfgrid:SfDataGrid.Columns>

```

The following code example demonstrates how the width is calculated for column using the `SetStarWidthForColumns` method based on the `ColumnRatio` property of the `StarSizerRatioHelpers` class:

## C#

```

// Assign custom GridColumnSizer to datagrid GridColumnSizer
dataGrid.GridColumnSizer = new CustomColumnSizer(this.dataGrid);

```

```

public class CustomColumnSizer : GridColumnSizer
{
    public CustomColumnSizer(SfDataGrid grid) : base(grid)
    {
    }
    protected override void SetStarWidthForColumns(double columnsWidth,
    IEnumerable<GridColumn> columns)
    {
        var removedColumn = new List<GridColumn>();
        var column = columns.ToList();
        var totalRemainingStarValue = columnsWidth;
        double removedWidth = 0;
        bool isRemoved;
        while (column.Count > 0)
        {
            isRemoved = false;
            removedWidth = 0;
            var columnsCount = 0;
            foreach (var data in column)
            {
                columnsCount += StarSizerRatioHelpers.GetColumnRatio(data);
            }
            double starWidth = Math.Floor((totalRemainingStarValue / columnsCount));
            var getColumn = column.First();
            //Calculate the ColumnSizer ratio for every column
            starWidth *= StarSizerRatioHelpers.GetColumnRatio(getColumn);
            var columnSizer = DataGrid.GridColumnSizer;
            var method = columnSizer.GetType().GetRuntimeMethods().FirstOrDefault(x =>
            x.Name == "SetColumnWidth");
            var width = method.Invoke(columnSizer, new object[] { getColumn, starWidth
            });
            double computeWidth = (double)width;
            if (starWidth != computeWidth && starWidth > 0)
            {
                isRemoved = true;
                column.Remove(getColumn);
                foreach (var remColumn in removedColumn)
                {
                    if (!column.Contains(remColumn))
                    {
                        removedWidth += remColumn.ActualWidth;
                        column.Add(remColumn);
                    }
                }
                removedColumn.Clear();
                totalRemainingStarValue += removedWidth;
            }
            totalRemainingStarValue = totalRemainingStarValue - computeWidth;
            if (!isRemoved)
            {
                column.Remove(getColumn);
                if (!removedColumn.Contains(getColumn))
                removedColumn.Add(getColumn);
            }
        }
    }
}

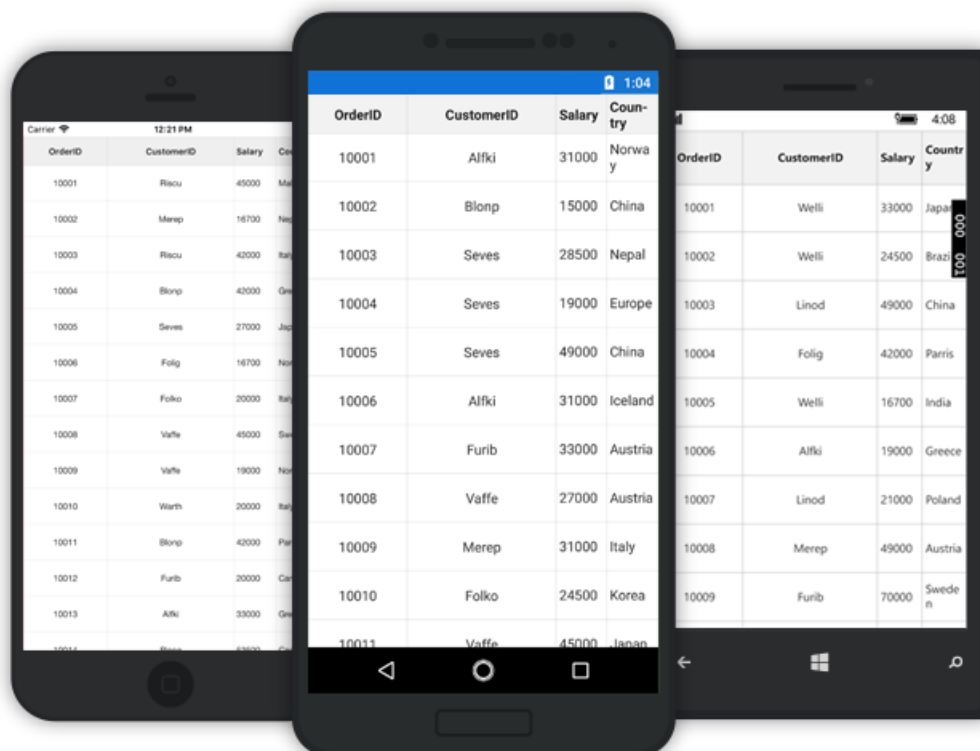
```



The following code example explains the `StarSizerRatioHelpers` class:

**C#**

```
public static class StarSizerRatioHelpers
{
    public static int GetColumnRatio(BindableObject obj)
    {
        return (int)obj.GetValue(ColumnRatioProperty);
    }
    public static void SetColumnRatio(BindableObject obj, int value)
    {
        obj.SetValue(ColumnRatioProperty, value);
    }
    public static readonly BindableProperty ColumnRatioProperty =
        BindableProperty.Create("ColumnRatio", typeof(int),
            typeof(StarSizerRatioHelpers), 1, BindingMode.TwoWay);
    public static void OnColumnSizerChanged(BindableObject bindable, object
        oldValue, object newValue)
    {
    }
}
```



## Freeze Panes

The SfDataGrid allows to freeze the rows and columns when scrolling the grid.

### Freeze rows

The SfDataGrid provides extensive support to freeze the rows at the top of the view below the header row by setting the [SfDataGrid.FrozenRowCount](#) property.

The following code example illustrates freezing two rows:

#### C#

```
//Setting number of rows to freeze in SfDataGrid  
dataGrid.FrozenRowCount = 2;
```

#### Limitation

- **FrozenRowCount** should be lesser than the number of rows displayed in view. For example, If you have 10 rows in view, then you set **FrozenRowCount** to a maximum value of 9.

---

**Note:** Header row is frozen by default and works regardless of the **FrozenRowCount** property.

---

### Freeze columns

The SfDataGrid also supports to freeze the columns at the left of the view by setting the [SfDataGrid.FrozenColumnsCount](#) property.

The following code example illustrates freezing two columns:

#### C#

```
//Setting number of columns to freeze in SfDataGrid  
dataGrid.FrozenColumnsCount = 2;
```

#### Limitation

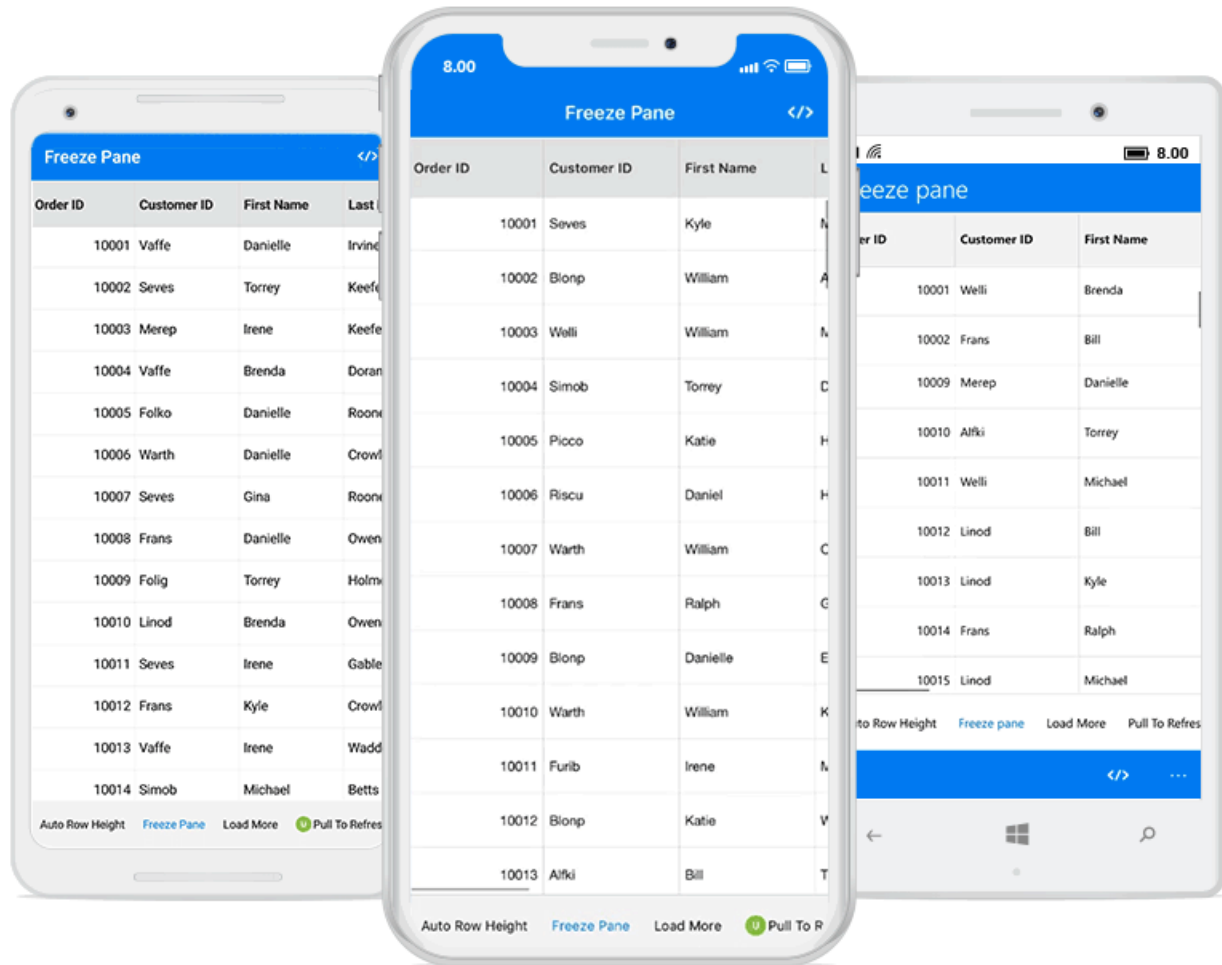
- **FrozenColumnsCount** should be lesser than number of columns displayed in view. For example, If you have 5 columns in view, then you can set **FrozenColumnsCount** to a maximum value of 4.

---

**Note:** RowHeader is frozen by default and works regardless of the **FrozenColumnsCount** property.

---

The following GIF illustrates FrozenRows and FrozenColumns.



## Grid Events

### GridTapped event

This event will be triggered while tapping the SfDataGrid with [GridTappedEventArgs](#) properties as follows:

- [RowIndex](#): Gets row index of the tapped row.
- [ColumnIndex](#): Gets column index of the tapped column.
- [RowData](#): Gets row data of the tapped row.

To hook the `GridTapped` event, and to get the tapped row and column details, follow the code example:

#### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    GridTapped="DataGrid_GridTapped"
    ItemsSource="{Binding OrdersInfo}" />
```

#### C#

```
private void DataGrid_GridTapped(object sender, GridTappedEventArgs e)
{
    var rowIndex = e.RowColumnIndex.RowIndex;
    var rowData = e.RowData;
    var columnIndex = e.RowColumnIndex.ColumnIndex;
}
```

### GridDoubleTapped event

This event will be triggered while double tapping the SfDataGrid with [GridDoubleTappedEventArgs](#) properties as follows:

- [RowIndex](#): Gets row index of the double tapped row.
- [ColumnIndex](#): Gets column index of the double tapped column.
- [RowData](#): Gets row data of the double tapped row.

To hook the `GridDoubleTapped` event, and to get the double tapped row and column details, follow the code example:

#### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    GridDoubleTapped="DataGrid_GridDoubleTapped"
    ItemsSource="{Binding OrdersInfo}" />
```

#### C#

```
private void DataGrid_GridDoubleTapped(object sender,
    GridDoubleTappedEventArgs e)
{
    var rowIndex = e.RowColumnIndex.RowIndex;
    var rowData = e.RowData;
    var columnIndex = e.RowColumnIndex.ColumnIndex;
}
```

### GridLongPressed event

This event will be triggered while long pressing the SfDataGrid with [GridLongPressedEventArgs](#) properties as follows:

- [RowIndex](#): Gets row index of the long pressed row.
- [ColumnIndex](#): Gets column index of the long pressed column.
- [RowData](#): Gets row data of the long pressed row.

To hook the `GridLongPressed` event, and to get the long pressed row and column details, follow the code example:

#### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    GridLongPressed="DataGrid_GridLongPressed"
    ItemsSource="{Binding OrdersInfo}" />
```

### C#

```
private void DataGrid_GridLongPressed(object sender,
GridLongPressedEventArgs e)
{
    var rowIndex = e.RowColumnIndex.RowIndex;
    var rowData = e.RowData;
    var columnIndex = e.RowColumnIndex.ColumnIndex;
}
```

### GridViewCreated event

This event will be triggered once the [SfDataGrid.View](#) is created. This event gives any operation only after the creation of [SfDataGrid.View](#) by handling the [GridViewCreatedEventArgs](#).

To hook the [GridViewCreated](#) event, and to set alternate row colors, follow the code example:

### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
GridViewCreated="DataGrid_GridViewCreated"
ItemsSource="{Binding OrdersInfo}" />
```

### C#

```
private void DataGrid_GridViewCreated(object sender,
GridViewCreatedEventArgs e)
{
    (sender as SfDataGrid).GridStyle = new CustomGridStyle();
}
internal class CustomGridStyle : DataGridStyle
{
    public override Color GetAlternatingRowBackgroundColor()
    {
        return Color.Aqua;
    }
}
```

### GridLoaded event

This event will be triggered once components in the [SfDataGrid](#) initialized and rendered. This event gives any operation only after loading the grid by handling the [GridLoadedEventArgs](#).

To hook the [GridLoaded](#) event, and to show the [ActivityIndicator](#) until the grid comes to view, follow the code example:

### XML

```
<Grid x:Name="grid"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
    <sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
GridLoaded="DataGrid_GridLoaded"/>
```

```
HorizontalOptions="FillAndExpand"  
ItemsSource="{Binding OrdersInfo}"  
VerticalOptions="FillAndExpand" />
```

### C#

```
private void DataGrid_GridLoaded(object sender, GridLoadedEventArgs e)  
{  
    ActivityIndicator indicator = new ActivityIndicator();  
    indicator.IsRunning = true;  
    indicator.IsVisible = true;  
    indicator.BackgroundColor = Color.Gray;  
    grid.Children.Add(indicator);  
    await Task.Delay(2000);  
    indicator.IsRunning = false;  
    indicator.IsVisible = false;  
}
```

### ValueChanged event

The [SfDataGrid.ValueChanged](#) event will be triggered whenever the current cell's value has been changed in the GridTextBoxColumn, GridNumericColumn or GridSwitchColumn. This event handler contains the parameter of type [ValueChangedEventArgs](#) that contains the following properties.

- [Column](#) : Gets the current [GridColumn](#) that contains the grid cell for which value is edited or changed.
- [NewValue](#) : The newly edited value to be committed.
- [RowColumnIndex](#) : The current [RowColumnIndex](#) of the grid cell undergoing the value change.
- [RowData](#) : The [RowData](#) of the row that contains the grid cell undergoing the value change.
- [CellValue](#) : The initial value when current cell entered edit mode.

### C#

```
dataGrid.ValueChanged += DataGrid_ValueChanged;  
private void DataGrid_ValueChanged(object sender, ValueChangedEventArgs e)  
{  
    var column = e.Column;  
    var newValue = e.NewValue;  
    var rowColIndex = e.RowColIndex;  
    var rowData = e.RowData;  
}
```

### ItemsSource changed event

The [SfDataGrid.ItemsSourceChanged](#) event will be triggered whenever the [SfDataGrid.ItemsSource](#) property is changed in the grid during both the runtime changes and initial loading of the DataGrid. This event handler contains the parameter of type [GridItemsSourceChangedEventArgs](#) that contains the following properties:

- [OldItemSource](#): Gets the previous ItemsSource collection as object. Always null when the grid is initially loaded.
- [NewItemSource](#): Gets the current ItemsSource collection as object.

- **OldView**: Gets the old [SfDataGrid.View](#) associated with the **OldItemSource**. Always null when the grid is initially loaded.
- **NewView**: Gets the new [SfDataGrid.View](#) associated with the **NewItemSource**.

The following code example shows how to hook the **SfDataGrid.ItemsSourceChanged** event and get the **ItemsSource** collection details.

#### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding OrdersInfo}"
ItemsSourceChanged="DataGrid_ItemsSourceChanged"/>
```

#### C#

```
private void DataGrid_ItemsSourceChanged(object sender,
GridItemsSourceChangedEventArgs e)
{
    var newItemSource = e.NewItemSource;
    var oldItemSource = e.OldItemSource;
    var newView = e.NewView;
    var oldView = e.OldView;
}
```

#### Create custom context menu using grid events

The SfDataGrid allows you to display any custom view, like context menu. This menu acts similar to pop-up by using the **GridLongPressed** and **GridTapped** events.

To create custom context menu using grid events, follow the code example:

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ContextMenuSupport"
x:Class="ContextMenuSupport.MainPage"
xmlns:sfgrid="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
">
<ContentPage.BindingContext>
<local:ViewModel />
</ContentPage.BindingContext>
<RelativeLayout x:Name="relativeLayout">
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Collection}"
ColumnSizer="Star"
RelativeLayout.WidthConstraint="{ConstraintExpression
Type=RelativeToParent,Property=Width,Factor=1,Constant=0}"
RelativeLayout.HeightConstraint="{ConstraintExpression
Type=RelativeToParent,Property=Height,Factor=1,Constant=0}"
GridLongPressed="DataGrid_GridLongPressed"
/>
</RelativeLayout>
</ContentPage>
```

**C#**

```
public partial class MainPage : ContentPage
{
    StackLayout contextMenu;
    Button sortButton;
    Button clearSortButton;
    private bool isContextMenuDisplayed = false;
    private string currentColumnName;
    public MainPage()
    {
        InitializeComponent();
        // Creates the view for the ContextMenu
        CreateContextMenu();
        dataGrid.AllowSorting = true;
        dataGrid.GridTapped += DataGrid_GridTapped;
    }
    private void DataGrid_GridTapped(object sender, GridTappedEventArgs e)
    {
        // Hides the context menu when SfDataGrid is tapped anywhere outside the context menu view
        relativeLayout.Children.Remove(contextMenu);
        isContextMenuDisplayed = false;
    }
    public void CreateContextMenu()
    {
        contextMenu = new StackLayout();
        sortButton = new Button();
        sortButton.Text = "Sort";
        sortButton.BackgroundColor = Color.Black;
        sortButton.TextColor = Color.White;
        sortButton.Clicked += SortButton_Clicked;
        clearSortButton = new Button();
        clearSortButton.Text = "Clear sort";
        clearSortButton.BackgroundColor = Color.Black;
        clearSortButton.TextColor = Color.White;
        clearSortButton.Clicked += ClearSortButton_Clicked;
        // A custom view hosting two buttons are now created
        contextMenu.Children.Add(sortButton);
        contextMenu.Children.Add(clearSortButton);
    }
    // Removes the sorting applied to the SfDataGrid
    private void ClearSortButton_Clicked(object sender, EventArgs e)
    {
        relativeLayout.Children.Remove(contextMenu);
        isContextMenuDisplayed = false;
        dataGrid.SortColumnDescriptions.Clear();
    }
    // Sorts the SfDataGrid data based on the column selected in the context menu
    private void SortButton_Clicked(object sender, EventArgs e)
    {
        relativeLayout.Children.Remove(contextMenu);
        isContextMenuDisplayed = false;
        dataGrid.SortColumnDescriptions.Clear();
    }
}
```

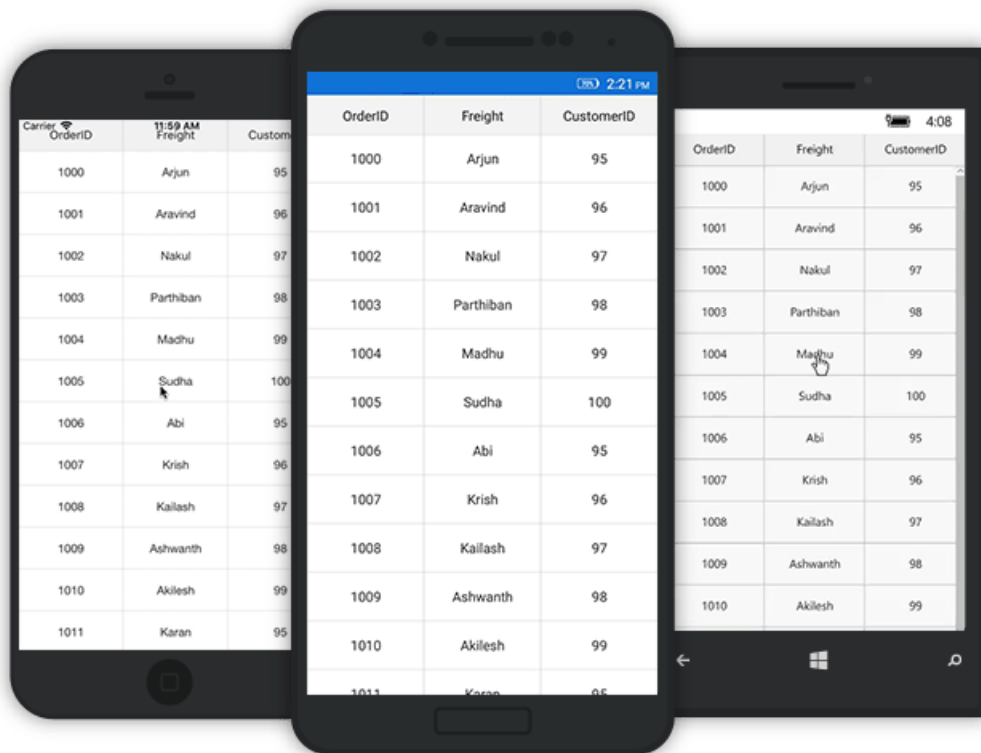


```

dataGrid.SortColumnDescriptions.Add(new SortColumnDescription()
{
    ColumnName = currentColumnName
});
}
public void DataGrid_GridLongPressed(object sender,
GridLongPressedEventArgs e)
{
    if (!isContextMenuDisplayed)
    {
        currentColumnName =
dataGrid.Columns[e.RowColumnIndex.ColumnIndex].MappingName;
var point = dataGrid.RowColumnIndexToPoint(e.RowColumnIndex);
// Display the ContextMenu when the SfDataGrid is long pressed
relativeLayout.Children.Add(contextMenu, Constraint.Constant(point.X),
Constraint.Constant(point.Y));
isContextMenuDisplayed = true;
    }
    else
    {
        // Hides the context menu when SfDataGrid is long pressed when the context
menu is already visible in screen
relativeLayout.Children.Remove(contextMenu);
isContextMenuDisplayed = false;
    }
}
}

```

Refer to the following GIF for final rendering on execution of above code example:



## Commands

### *GridTapped command*

The [SfDataGrid.GridTappedCommand](#) will be executed when tapping the SfDataGrid. You can directly assign any [ICommand](#) type property to the [SfDataGrid.GridTappedCommand](#) or write a class derived from ICommand interface, and assign your CustomClass type property to the [SfDataGrid.GridTappedCommand](#) property. By configuring this, the data grid comes handy if you have your own logics determining whether to execute the command or not.

### XML

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
GridTappedCommand="{Binding TappedCommandAction}"/>
```

### C#

```
// viewModel.cs
public ViewModel ()
{
    // assigning command action to ICommand type property
    TappedCommandAction = new Command(add);
    // assigning command action to your custom command type property
    TappedCommandAction1 = new DerivedTappedCommand();
}
// ICommand type property for binding with DataGrid.TappedCommandAction
public ICommand TappedCommandAction
{
    get;
    set;
}
// Custom command type property for binding with
DataGrid.TappedCommandAction
public DerivedTappedCommand TappedCommandAction1
{
    get;
    set;
}
public void add()
{
    //your logics here
}
// below codes for writing custom command derived from ICommand.
public class DerivedTappedCommand : ICommand
{
    public event EventHandler CanExecuteChanged;
    public DerivedTappedCommand( )
    {
    }
    public bool CanExecute(object parameter)
    {
        return true;
    }
}
```

```
public void Execute(object parameter)
{
    // your logics here.
}
}
```

You can download the sample demo [here](#).

#### *GridDoubleTapped command*

The [SfDataGrid.GridDoubleTappedCommand](#) will be executed when double tapping the SfDataGrid. You can directly assign any [ICommand](#) type property to the [SfDataGrid.GridTappedCommand](#) or write a class derived from [ICommand](#) interface, and assign your CustomClass type property to the [SfDataGrid.GridTappedCommand](#) property. By configuring this, the data grid comes handy if you have your own logics determining whether to execute the command or not.

#### XML

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
GridDoubleTappedCommand="{Binding DoubleTappedCommandAction}" />
```

#### C#

```
// viewModel.cs
public ViewModel()
{
    // assigning command action to ICommand type property
    DoubleTappedCommandAction = new Command(add);
    // assigning command action to your custom command type property
    DoubleTappedCommandAction1 = new DerivedDoubleTappedCommand();
}
// ICommand type property for binding with DataGrid.DoubleTappedCommand
public ICommand DoubleTappedCommandAction
{
    get;
    set;
}
// Custom command type property for binding with
DataGrid.DoubleTappedCommand
public DerivedDoubleTappedCommand DoubleTappedCommandAction1
{
    get;
    set;
}
public void add()
{
    //your logics here
}
// below codes for writing custom command derived from ICommand.
public class DerivedDoubleTappedCommand : ICommand
{
    public event EventHandler CanExecuteChanged;
```

```

public DerivedDoubleTappedCommand( )
{
}
public bool CanExecute(object parameter)
{
    return true;
}
public void Execute(object parameter)
{
    // your logics here.
}
}

```

You can download the sample demo [here](#).

#### *GridLongPressed command*

The `SfDataGrid.GridLongPressedCommand` will be executed when long pressing the SfDataGrid. You can directly assign any `ICommand` type property to the `SfDataGrid.GridTappedCommand` or write a class derived from `ICommand` interface, and assign your CustomClass type property to the `SfDataGrid.GridTappedCommand` property. By configuring this, the data grid comes handy if you have your own logics determining whether to execute the command or not.

#### **XML**

```

<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
GridLongPressedCommand="{Binding LongPressedCommandAction}">

```

#### **C#**

```

// viewModel.cs
public ViewModel()
{
    // assigning command action to ICommand type property
    LongPressedCommandAction = new Command(add);
    // assigning command action to your custom command type property
    LongPressedCommandAction1 = new DerivedLongPressedCommand();
}
// ICommand type property for binding with DataGrid.GridLongPressedCommand
public ICommand LongPressedCommandAction
{
    get;
    set;
}
// Custom command type property for binding with
DataGrid.GridLongPressedCommand
public DerivedLongPressedCommand LongPressedCommandAction1
{
    get;
    set;
}
public void add()

```

```

{
    // your logics here
}
// below codes for writing custom command derived from ICommand.
public class DerivedLongPressedCommand : ICommand
{
    public event EventHandler CanExecuteChanged;
    public DerivedLongPressedCommand( )
    {
    }
    public bool CanExecute(object parameter)
    {
        return true;
    }
    public void Execute(object parameter)
    {
        // your logics here.
    }
}

```

### XML

```

<sfgrid:SfDataGrid x:Name="dataGrid"
    ItemsSource="{Binding OrdersInfo}"
    GridLongPressedCommand="{Binding LongPressedCommandAction}">

```

You can download the sample demo [here](#).

### Sorting

The data grid sorts the data by setting the `SfDataGrid.AllowSorting` property to `true`. It allows sorting the data against one or more columns. When sorting is applied, the control automatically rearranges the data to match with the current sort criteria. When `SfDataGrid.AllowSorting` is `true`, you can sort the data simply by tapping the column header. Once sorting is applied, the control shows a sort icon in the respective column header indicating the direction of sorting.

**Note:** To update sorting for the newly added row or column, set the `SfDataGrid.View.LiveDataUpdateMode` to `LiveDataUpdateMode.AllowDataShaping`.

#### Programmatic sorting

The data grid also sorts from the code. This requires to manually define the `SortColumnDescription` objects, and add it in the `SfDataGrid.SortColumnDescriptions` collection. The control sorts the data based on the `SortColumnDescription` objects added to this collection.

The `SortColumnDescription` object holds the following two properties:

- **ColumnName:** Name of the sorted column.
- **SortDirection:** An object of type `ListSortDirection` defines the sorting direction.

The following code example illustrates this:

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"

```

```
ItemsSource="{Binding OrdersInfo}">
<syncfusion:SfDataGrid.SortColumnDescriptions>
<syncfusion:SortColumnDescription ColumnName="OrderID"
SortDirection="Descending" />
</syncfusion:SfDataGrid.SortColumnDescriptions>
</syncfusion:SfDataGrid>
```

**C#**

```
dataGrid.AllowSorting = true;
dataGrid.SortColumnDescriptions.Add (new SortColumnDescription () {
ColumnName = "OrderID",
SortDirection = ListSortDirection.Descending
});
```

The following screenshot shows the sorting functionality in the data grid:

**Tri-State sorting**

In addition to sort the data in ascending/descending order, the SfDataGrid unsort the data in the original order by clicking the header again after sorting to descending order by setting the [SfDataGrid.AllowTriStateSorting](#) property to true. When this property is set, sorting in each column iterates through three sort states: ascending, descending, and unsort.

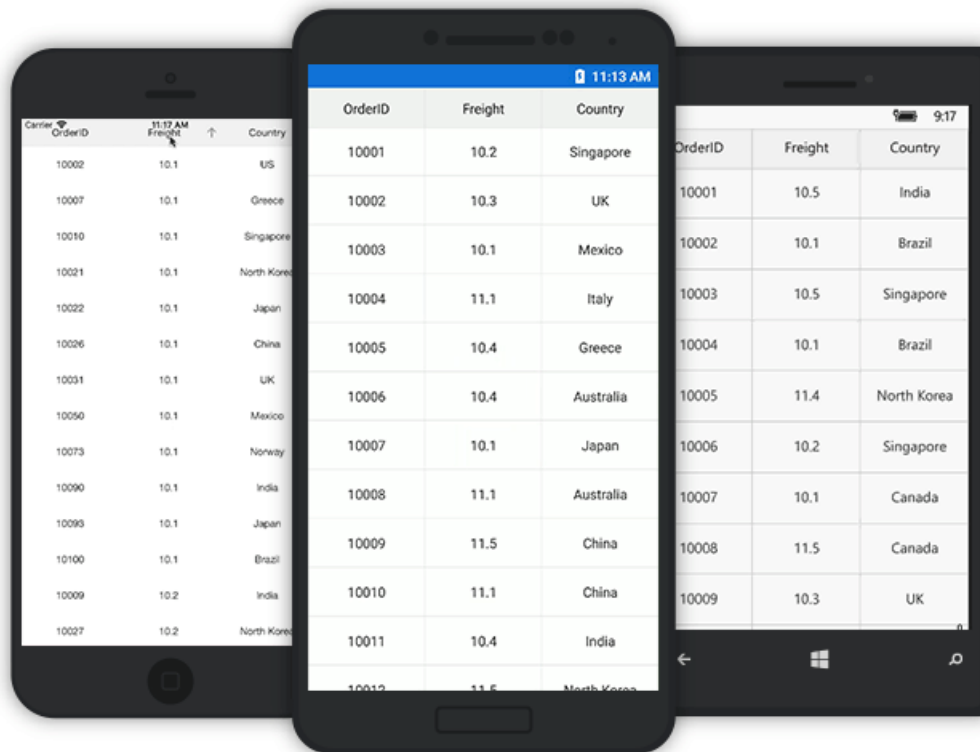
To enable tri-State sorting in the SfDataGrid, follow the code example:

**XML**

```
<syncfusion:SfDataGrid AllowTriStateSorting="True" />
```

**C#**

```
dataGrid.AllowTriStateSorting = true;
```

**Multi-column sorting**

The data grid sorts the data against more than one columns by setting the [SfDataGrid.AllowMultiSorting](#) property to `true`. The number of columns by which the data can be sorted is unlimited. To apply sorting for multiple columns, tap the desired column headers after setting the [SfDataGrid.AllowMultiSorting](#) property.

To enable multi-sorting, follow the code example:

**XML**

```
<syncfusion:SfDataGrid AllowMultiSorting="True" />
```

**C#**

```
dataGrid.AllowMultiSorting = true;
```



### Sort column in double click

By default, column gets sorted when column header clicked. This behavior can be changed to sort the column in double click action by setting [SfDataGrid.SortTapAction](#) property to `DoubleTap`.

To set `SortTapAction` as `DoubleTap`, follow the code example:

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    SortTapAction="DoubleTap"
    ItemsSource="{Binding OrdersInfo}">
</syncfusion:SfDataGrid>
```

#### C#

```
dataGrid.SortTapAction=SortTapAction.DoubleTap;
```

### Sorting events

The data grid provides the following events for the sorting functionality:

- [SortColumnsChanging](#): This event is raised while sorting the column at execution time before the column gets sorted. It helps to cancel the sorting action by setting the `Cancel` property of [DataGridSortColumnsChangingEventArgs](#).
- [SortColumnsChanged](#): This event is raised after the column is sorted.



These two events are triggered with `DataGridSortColumnsChangingEventArgs` and `DataGridSortColumnsChangedEventArgs` that contains the following properties:

- `AddedItems`: Gets the collection of `SortColumnDescription` objects that are added to `SortColumnDescriptions` collection for sorting.
- `RemovedItems`: Gets the collection of `SortColumnDescription` objects that are removed from `SortColumnDescriptions` collection.

To hook the `SortColumnsChanging` event, and cancel the sorting of a column, follow the code example:

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowSorting="True"
    SortColumnsChanging="DataGrid_SortColumnsChanging"
    ItemsSource="{Binding OrdersInfo}">
</syncfusion:SfDataGrid>
```

#### C#

```
dataGrid.SortColumnsChanging += DataGrid_SortColumnsChanging;
```

#### C#

```
void DataGrid_SortColumnsChanging (object sender, DataGridSortColumnsChangingEventArgs e)
{
    if (e.AddedItems[0].ColumnName == "OrderID")
    {
        e.Cancel = true;
    }
}
```

#### Custom sorting

The data grid sort columns based on custom logic, when the standard sorting techniques do not meet the requirements. For each column, you can apply different sorting criteria by adding `SortComparer` objects to `SfDataGrid.SortComparers` collection.

A `SortComparer` object has the following properties:

- `PropertyName`: [MappingName](#) of the column that applies custom sorting.
- `Comparer`: Gets or sets the custom comparer that implements the `IComparer` and `ISortDirection` interfaces.

To perform custom sorting for the `FirstName` column based on the string length of the names, follow the code example:

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
```

```

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:data="clr-namespace:Syncfusion.Data;assembly=Syncfusion.Data.Portable"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
"
xmlns:local ="clr-namespace:DataGridSample;assembly=DataGridSample"
x:Class="DataGridSample.Sample">
<ContentPage.Resources>
<ResourceDictionary>
<local:CustomComparer x:Key="Comparer" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowSorting="True"
ItemsSource="{Binding OrdersInfo}">
<syncfusion:SfDataGrid.SortComparers>
<data:SortComparer Comparer="{StaticResource Comparer}"
PropertyName="FirstName" />
</syncfusion:SfDataGrid.SortComparers>
<syncfusion:SfDataGrid.SortColumnDescriptions>
<syncfusion:SortColumnDescription ColumnName="FirstName"
SortDirection="Descending" />
</syncfusion:SfDataGrid.SortColumnDescriptions>
</syncfusion:SfDataGrid>
</ContentPage>

```

**C#**

```

dataGrid.SortComparers.Add (new SortComparer () {
PropertyName = "FirstName",
Comparer = new CustomComparer()
});
dataGrid.SortColumnDescriptions.Add (new SortColumnDescription () {
ColumnName = "FirstName",
SortDirection = ListSortDirection.Descending
});

```

To write custom comparer, follow the code example:

**C#**

```

public class CustomComparer : IComparer<Object>, ISortDirection
{
public int Compare(object x, object y)
{
int nameX;
int nameY;
//For OrderInfo type data
if (x.GetType () == typeof(OrderInfo)) {
//Calculating the length of FirstName in OrderInfo objects
nameX = ((OrderInfo)x).FirstName.Length;
nameY = ((OrderInfo)y).FirstName.Length;
}
}

```

```

//For Group type data
else if (x.GetType () == typeof(Group)) {
    //Calculating the group key length
    nameX = ((Group)x).Key.ToString ().Length;
    nameY = ((Group)y).Key.ToString ().Length;
} else {
    nameX = x.ToString ().Length;
    nameY = y.ToString ().Length;
}
// Objects are compared and return the SortDirection
if (nameX.CompareTo (nameY) > 0)
return SortDirection == ListSortDirection.Ascending ? 1 : -1;
else if (nameX.CompareTo (nameY) == -1)
return SortDirection == ListSortDirection.Ascending ? -1 : 1;
else
return 0;
}
//Gets or sets the SortDirection value
public ListSortDirection SortDirection { get; set; }
}

```

### Animating sorting icon

The data grid loads two different icons for denoting the **Ascending** and **Descending** sort direction states. However, rotate the [DataGridStyle.GetHeaderSortIndicatorUp](#) icon animatedly for denoting the descending state by overriding the [DataGridStyle.GetHeaderSortIndicatorDown](#) method, and returning null.

To animate the sorting icon, follow the code example:

### XML

```

<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
AllowSorting ="true"
GridStyle="{local:CustomStyle}">
</sfgrid:SfDataGrid>

```

### C#

```

dataGrid.AllowSorting = true;
dataGrid.GridStyle = new CustomStyle();

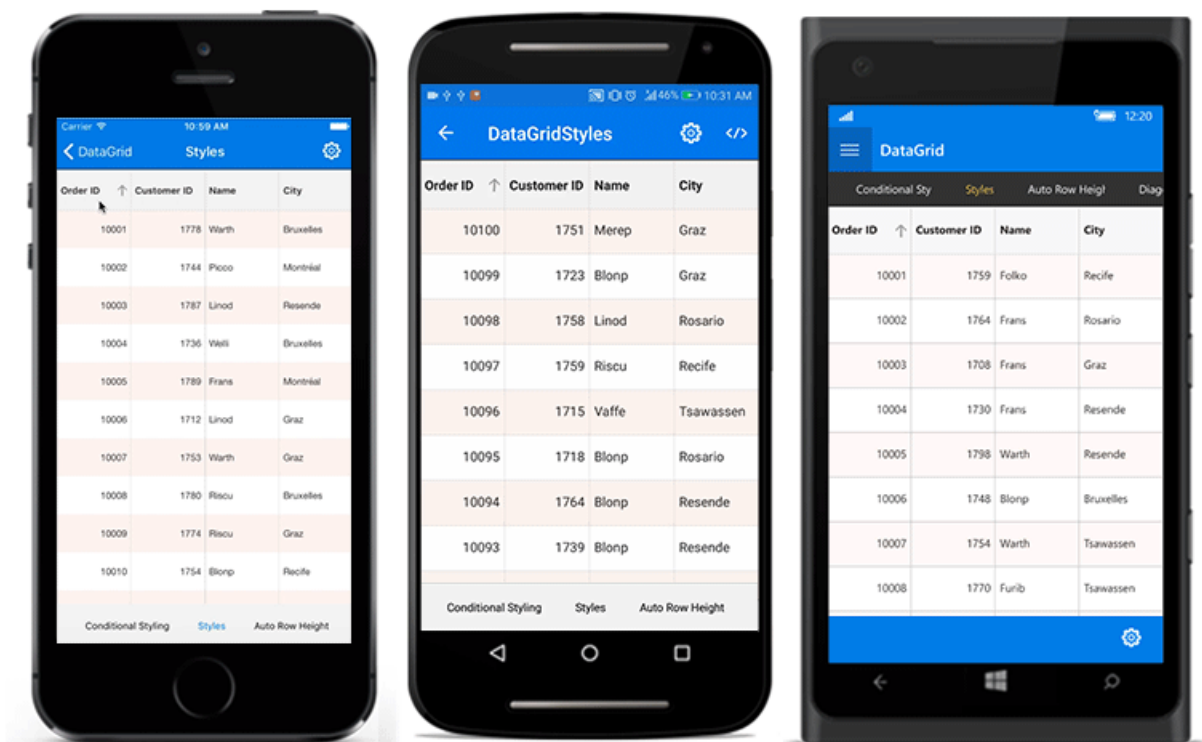
```

### C#

```

//Custom style class
public class CustomStyle :DataGridStyle
{
    public CustomStyle()
    {
    }
    public override ImageSource GetHeaderSortIndicatorDown()
    {
        return null;
    }
}

```



## How to disable sorting

### for an individual column

The data grid disables sorting for an individual column using the [GridColumn.AllowSorting](#) property. The default value of this property is true. So all the columns in the [SfDataGrid.Columns](#) collection can be sorted when [SfDataGrid.AllowSorting](#) is set to true.

To disable sorting for an individual column, follow the code example:

### for auto generated column

#### C#

```
public MainPage()
{
    InitializeComponent();
    viewModel = new ViewModel();
    dataGrid = new SfDataGrid();
    dataGrid.ItemsSource = viewModel.OrdersInfo;
    dataGrid.AllowSorting = true;
    dataGrid.AutoGeneratingColumn += DataGrid_AutoGeneratingColumn;
    this.Content = dataGrid;
}

private void DataGrid_AutoGeneratingColumn(object sender,
AutoGeneratingColumnEventArgs e)
{
    // Sorting will not be done for the Freight column
    if (e.Column.MappingName == "Freight")
        e.Column.AllowSorting = false;
}
```

for manually defined column

### C#

```
public MainPage()
{
    InitializeComponent();
    viewModel = new ViewModel();
    dataGrid = new SfDataGrid();
    dataGrid.ItemsSource = viewModel.OrdersInfo;
    dataGrid.AutoGenerateColumns = false;
    dataGrid.AllowSorting = true;
    dataGrid.Columns.Add(new GridTextColumn() { MappingName = "OrderID" });
    // Sorting will not be done for the Freight column
    dataGrid.Columns.Add(new GridTextColumn() { MappingName = "Freight",
    AllowSorting = false });
    dataGrid.Columns.Add(new GridTextColumn() { MappingName = "Country" });
    this.Content = dataGrid;
}
```

## Grouping

A group represents a collection of records that belongs to a particular category. When grouping, the data is organized into hierarchical structure based on matching field values. The records having identical values in the grouped column are combined to form a group. Each group is identified by its [CaptionSummaryRow](#) to get the underlying records in view.

**Note:** To update grouping for the newly added row or column, set the `SfDataGrid.View.LiveDataUpdateMode` to `LiveDataUpdateMode.AllowDataShaping`.

**Note:** When `BeginInit` method is called, it suspends all the updates until `EndInit` method is called.

### Programmatic grouping

The `SfDataGrid` also allows to perform grouping from the code by defining the [GroupColumnDescription](#) object and adding it in the `SfDataGrid.GroupColumnDescriptions` collection. It groups the data based on the `GroupColumnDescription` object added to this collection.

`GroupColumnDescription` object holds following two properties:

- `ColumnName`: Name of the grouped column.
- `Converter`: Get the `IValueConverter` as input that helps to apply the custom grouping.

To apply column grouping, follow the code example:

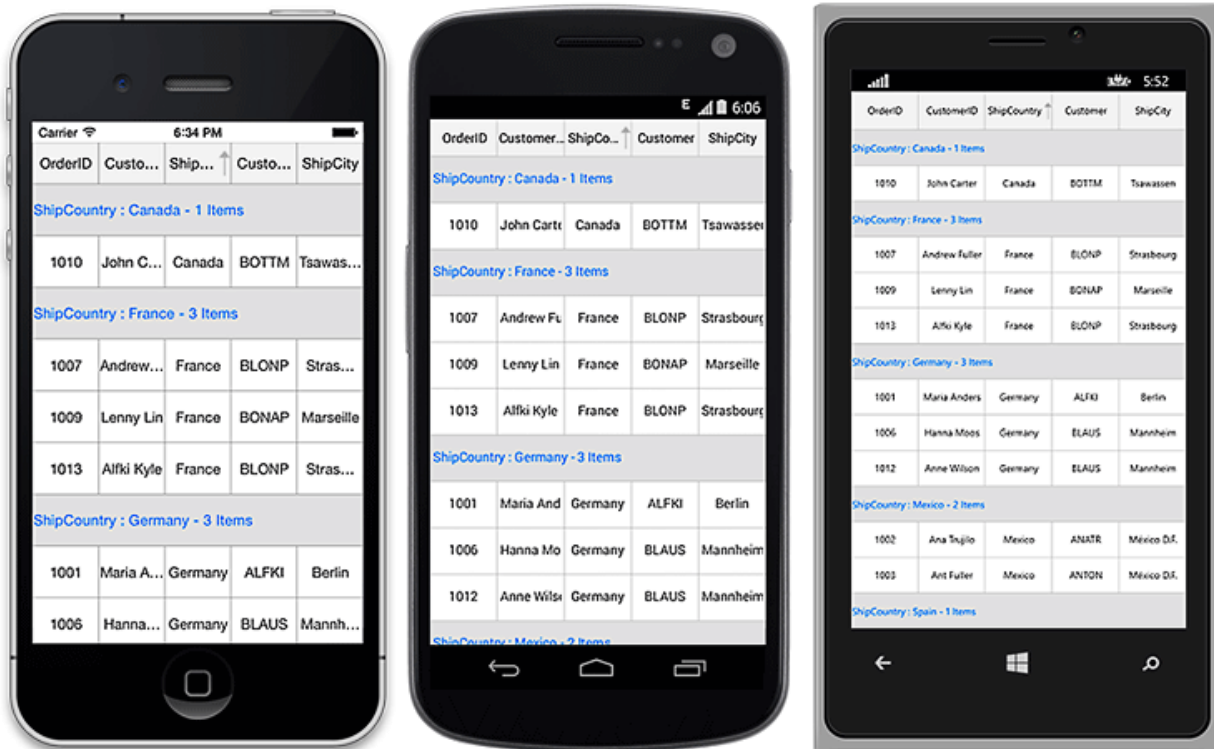
### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    ItemsSource="{Binding OrdersInfo}">
    <syncfusion:SfDataGrid.GroupColumnDescriptions>
    <syncfusion:GroupColumnDescription ColumnName="CustomerID" />
    </syncfusion:SfDataGrid.GroupColumnDescriptions>
</syncfusion:SfDataGrid>
```

### C#

```
dataGrid.GroupColumnDescriptions.Add (new GroupColumnDescription () {
    ColumnName = "CustomerID",
});
```

The following screenshot shows the output rendered when grouping is applied:



### MultiGrouping

The SfDataGrid also allows to group the data against one or more columns using the [SfDataGrid.GroupingMode](#) property. When `GroupingMode` is set as `GroupingMode.Multiple`, the data is organized into hierarchical tree structure based on identical values of that column. MultiGrouping feature works similarly as MultiSorting feature. Initially the data is grouped according to the first column added in the `GroupColumnDescriptions` collection. When more columns are added to the `GroupColumnDescriptions`, the newly added column will be grouped in consideration to the previous group(s). This results in a tree like hierarchy. Refer to the following code snippet to enable MultiGrouping:

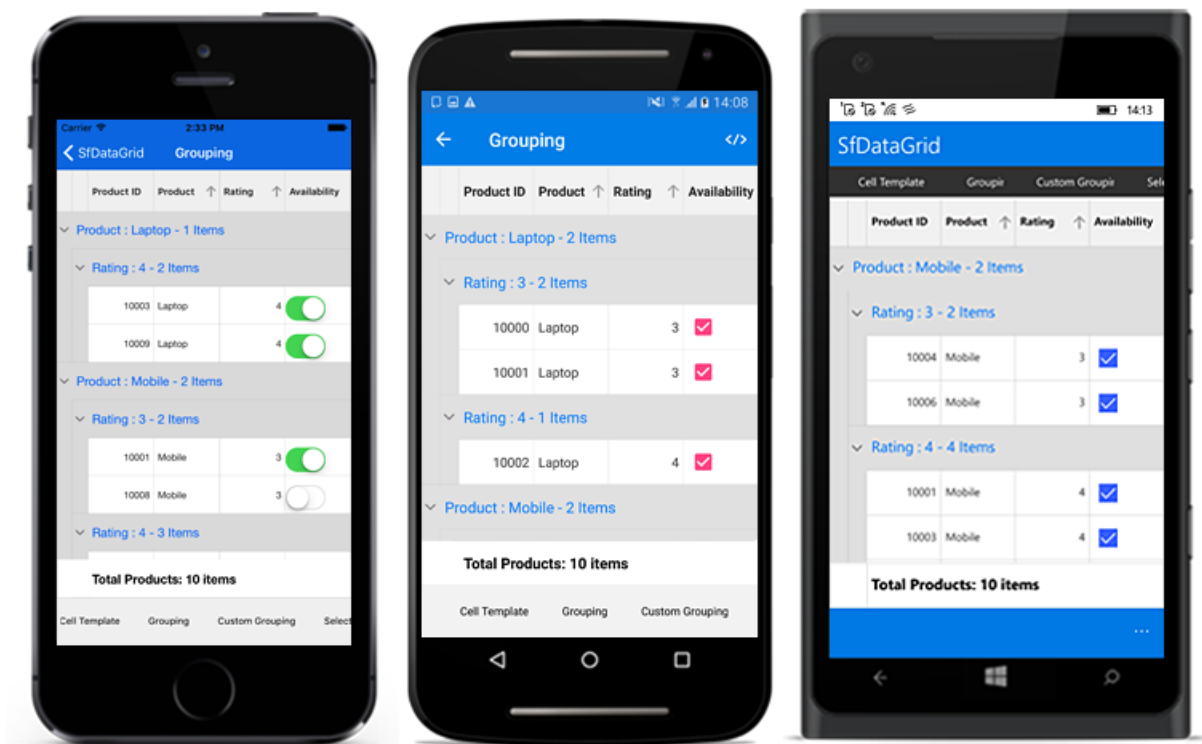
#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    GroupingMode = "Multiple"/>
```

#### C#

```
this.dataGrid.GroupingMode = GroupingMode.Multiple;
```

The following screenshot shows the output rendered when above code is executed:



### Indent column customizations

Indent columns are the columns present to the left of the `CaptionSummaryRows` when `GroupingMode` is set as multiple. The number of indent cells in each `CaptionSummaryRow` will be determined by the level of that `Group`. For example, the first group will have only one indent cell and the next immediate group will have an extra indent cell. It keeps on adding by one for each lower level groups to maintain the tree structure. Each data row will have indent cells count equal to the level of the last sub group in view. The following customizations can be done for indent cells:

#### Customize indent column width

By default, the width of the indent column is 20. To customize the width of indent column, use the `IndentColumnWidth` property as follows:

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    IndentColumnWidth="60"/>
```

#### C#

```
this.dataGrid.IndentColumnWidth = 60;
```

#### Customize indent column background color

Set background colors to indent cells based on the row where indent cells present. To set the desired background color, use the `GetIndentBackgroundColor()` override in the custom `DataGridStyle` class.




Refer to this [link](#) to know how to apply custom style to the SfDataGrid. Refer to the following code snippet to apply background color to indent cells based on the row type:

**C#**

```
this.dataGrid.GridStyle = new CustomStyle();  
public class CustomStyle : DataGridStyle  
{  
    public override Color GetIndentBackgroundColor(RowType rowType)  
    {  
        if (rowType == RowType.DefaultRow)  
            return Color.DarkOrange;  
        if (rowType == RowType.CaptionCoveredRow)  
            return Color.Navy;  
        else return Color.FromRgb(224, 224, 224);  
    }  
}
```



IsClosed	OrderID ↑	ShippingDate	EmployeeID
OrderID : 1 - 1 Items			
EmployeeID : 15 - 1 Items			
<input type="checkbox"/>	1	11/20/2017 6:56:40 PM	15
OrderID : 2 - 1 Items			
EmployeeID : 4 - 1 Items			
<input checked="" type="checkbox"/>	2	11/20/2017 6:56:40 PM	4
OrderID : 3 - 1 Items			
EmployeeID : 1 - 1 Items			
<input type="checkbox"/>	3	11/20/2017 6:56:40 PM	1
OrderID : 4 - 1 Items			
EmployeeID : 18 - 1 Items			
<input checked="" type="checkbox"/>	4	11/20/2017 6:56:40 PM	18

-  **CaptionCoveredRow**
-  **DefaultRow**
-  **Header Row**

### Custom grouping

The SfDataGrid allows to group a column based on custom logic when the standard grouping techniques do not meet the requirements. To achieve this, write a converter that implements `IValueConverter` with custom grouping logic. Assign that converter to the `GroupColumnDescription.Converter` property.

To set custom grouping converter for the group description that is added to group the freight column, follow the code example:

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```

xmlns:syncfusion="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
"
xmlns:local ="clr-namespace:DataGridSample;assembly=DataGridSample"
x:Class="DataGridSample.Sample">
<ContentPage.Resources>
<ResourceDictionary>
<local:GroupConverter x:Key="groupConverter" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:GroupColumnDescription ColumnName="Freight"
Converter="{StaticResource groupConverter}" />
</syncfusion:SfDataGrid.GroupColumnDescriptions>
</syncfusion:SfDataGrid>
</ContentPage>

```

**C#**

```

dataGrid.GroupColumnDescriptions.Add (new GroupColumnDescription () {
ColumnName = "Freight",
Converter = new GroupConverter()
});

```

The following code example illustrates the converter used for applying custom grouping logic.

**C#**

```

public class GroupConverter : IValueConverter
{
    public GroupConverter()
    {
    }
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var orderInfo = value as OrderInfo;
        if (orderInfo.Freight > 0 && orderInfo.Freight <= 250)
            return "<=250";
        else if (orderInfo.Freight > 250 && orderInfo.Freight <= 500)
            return ">250 & <=500";
        else if (orderInfo.Freight > 500 && orderInfo.Freight <= 750)
            return ">500 & <=750";
        else
            return ">1000";
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return null;
    }
}

```

```
}
```

### Expand groups while grouping

To expand all groups while grouping, set the [SfDataGrid.AutoExpandGroups](#) to `true`. While grouping any column, all groups will be in expanded state.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoExpandGroups="True"
    AllowGroupExpandCollapse="True"
    ItemsSource="{Binding Orders}"/>
```

#### C#

```
this.dataGrid.AutoExpandGroups = true;
this.dataGrid.AllowGroupExpandCollapse = true;
```

### Expand or collapse the groups

By default, the groups will be in expanded state. However expand or collapse a group in runtime by setting the [SfDataGrid.AllowGroupExpandCollapse](#) to `true`.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowGroupExpandCollapse="True"
    ItemsSource="{Binding Orders}"/>
```

#### C#

```
this.dataGrid.AllowGroupExpandCollapse = true;
```

### Expand or collapse all the groups

Programmatically expand or collapse all the groups at runtime using the [SfDataGrid.ExpandAllGroup](#) and [SfDataGrid.CollapseAllGroup](#) methods.

#### C#

```
this.dataGrid.ExpandAllGroup();
this.dataGrid.CollapseAllGroup();
```

### Expand or collapse a specific group

Expand or collapse specific group by using the [SfDataGrid.ExpandGroup](#) and [SfDataGrid.CollapseGroup](#) methods.

#### C#

```
var group = (dataGrid.View.Groups[0] as Group);
this.dataGrid.ExpandGroup(group);
this.dataGrid.CollapseGroup(group);
```

ID	Name	Remarks	Date
Name : Arun - 41 Items			
Name : Bala - 32 Items			
7	Bala	9	13/06/2016 C
8	Bala	10	13/06/2016 C
10	Bala	12	13/06/2016 C
22	Bala	24	13/06/2016 C
23	Bala	25	13/06/2016 C
24	Bala	26	13/06/2016 C
31	Bala	33	13/06/2016 C
37	Bala	39	13/06/2016 C
38	Bala	41	13/06/2016 C

### Display based grouping using GroupMode property

By default, column grouping occurs based on the value in the underlying collection thereby creating a new group for each new value of that column. However, a column can also be grouped based on the Display value by setting the [GridColumn.GroupMode](#) property as **Display**. In the following code example, set the [GridColumn.Format](#) property as "#" which displays only the rounded off value in the [GridCell](#):

#### XML

```
<syncfusion:GridColumn HeaderText="Shipment Weight"
MappingName="ShipmentWeight"
GroupMode="Display"
Format="#" />
```

#### C#

```
GridNumericColumn cargoWeight = new GridTextColumn();
cargoWeight.MappingName = "ShipmentWeight";
cargoWeight.GroupMode = Syncfusion.Data.DataReflectionMode.Display;
cargoWeight.Format = "#";
```

Following screenshot shows the comparison between two Group modes. GroupMode.Value on the left and GroupMode.Display on the right:



GroupMode.Value

GroupMode.Display

### Clearing or removing a group

To clear applied grouping, remove the items from the `SfDataGrid.GroupColumnDescriptions` collection or clear the collection.

Refer to the following code snippets to remove grouping:

#### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GridInForms"
x:Class="GridInForms.MainPage"
xmlns:sfgrid="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
" >
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<StackLayout VerticalOptions="FillAndExpand">
<Button Text="Remove Grouping"
TextColor="Black"
BackgroundColor="White"
Clicked="ClearGroupingButton_Click"/>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
>
<syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:GroupColumnDescription ColumnName="Freight" />
</syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:GroupColumnDescription ColumnName="CustomerID" />
</syncfusion:SfDataGrid.GroupColumnDescriptions>
</sfgrid:SfDataGrid>
</StackLayout>
</ContentPage>

```

**C#**

```

public partial class MainPage : ContentPage
{
    StackLayout stackLayout;
    SfDataGrid dataGrid;
    ViewModel viewModel;
    Button clearGroupingButton;
    public MainPage()
    {
        InitializeComponent();
        viewModel = new ViewModel();
        dataGrid = new SfDataGrid();
        clearGroupingButton = new Button();
        stackLayout = new StackLayout();
        dataGrid.ItemsSource = viewModel.OrdersInfo;
        dataGrid.GroupingMode = GroupingMode.Multiple;
        dataGrid.GroupColumnDescriptions.Add(new GroupColumnDescription()
        {
            ColumnName = "Freight",
        });
        dataGrid.GroupColumnDescriptions.Add(new GroupColumnDescription()
        {
            ColumnName = "CustomerID",
        });
        clearGroupingButton.Text = "Remove Grouping";
        clearGroupingButton.TextColor = Color.Black;
        clearGroupingButton.BackgroundColor = Color.White;
    }
}

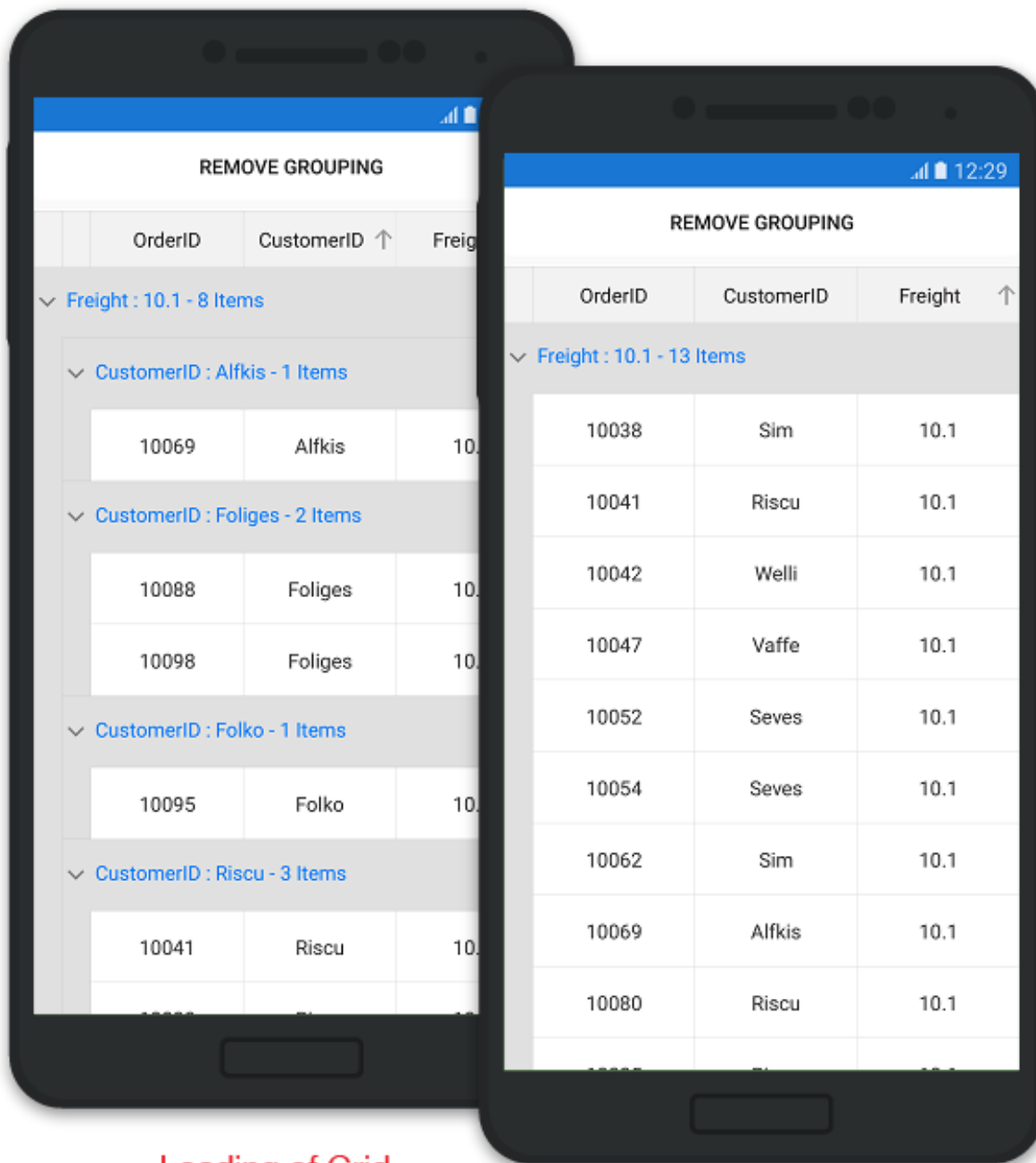
```

```
clearGroupingButton.Clicked += ClearGroupingButton_Click;
stackLayout.Children.Add(clearGroupingButton);
stackLayout.Children.Add(dataGrid);
this.Content = stackLayout;
}
```

### C#

```
private void ClearGroupingButton_Click(object sender, System.EventArgs e)
{
    //Clearing the Group
    dataGrid.GroupColumnDescriptions.Clear();
    //Removing the Group based on group item
    //var groupColumn = dataGrid.GroupColumnDescriptions[0];
    //dataGrid.GroupColumnDescriptions.Remove(groupColumn);
    //Removing the Group based on group index
    //dataGrid.GroupColumnDescriptions.RemoveAt(0);
}
```

Run the application to render the following output:



Loading of Grid

After clicking the remove grouping button

**Note:** Clear or remove grouping on [GridTapped event](#), [GridDoubleTapped event](#), or [GridLongPressed event](#).

#### Events

##### *GroupExpanding event*

The [SfDataGrid.GroupExpanding](#) event occurs when group is being expanded.



The [GroupChangingEventArgs](#) of the **GroupExpanding** event provides the information about the expanding group and it has the following members:

**Syncfusion.Data.Group**: Gets expanded group.

**Cancel**: Decides to cancel group expansion.

Cancel group expansion by setting [GroupChangingEventArgs.Cancel](#) to **true**.

#### C#

```
this.dataGrid.GroupExpanding += dataGrid_GroupExpanding;
void dataGrid_GroupExpanding(object sender,
Syncfusion.SfDataGrid.XForms.GroupChangingEventArgs e)
{
    if (e.Group.Key.Equals(1001))
        e.Cancel = true;
}
```

#### *GroupExpanded event*

The [SfDataGrid.GroupExpanded](#) event occurs after group is expanded.

The [GroupChangedEventArgs](#) of the **GroupExpanded** event provides the information about the expanded group and it has the following member:

**Syncfusion.Data.Group**: Gets the expanded group.

#### *GroupCollapsing event*

The [SfDataGrid.GroupCollapsing](#) event occurs when group is being collapsed.

The [GroupChangingEventArgs](#) of the **GroupCollapsing** event provides the information about the collapsing group and it contains the following member:

**Syncfusion.Data.Group**: Gets collapsed group.

**Cancel**: Decides to cancel the group collapsing.

Cancel the group is being collapsed by using the [GroupChangingEventArgs.Cancel](#) of **GroupCollapsing** event.

#### C#

```
this.dataGrid.GroupCollapsing += dataGrid_GroupCollapsing;
void dataGrid_GroupCollapsing(object sender,
Syncfusion.SfDataGrid.XForms.GroupChangingEventArgs e)
{
    if (e.Group.Key.Equals(1001))
        e.Cancel = true;
}
```

#### *GroupCollapsed event*

The [SfDataGrid.GroupCollapsed](#) event occurs after group collapsed.

The [GroupChangedEventArgs](#) of the **GroupCollapsed** event provides the information about collapsed group and it contains the following member:

**Syncfusion.Data.Group**: Gets the collapsed group.

### Animate group expand/collapse icon

The SfDataGrid loads two different icons for denoting the group expanded and collapsed state. However rotate the expander icon animatedly for denoting the collapsed status by overriding the `DataGridStyle.GetGroupCollapseIcon` method and returning `null`.

To enable group expand/collapse icons animation by writing a custom style, follow the code example:

#### C#

```
//Apply custom style to SfDataGrid from code
dataGrid.GridStyle = new CustomStyle ();
//Custom Style class
public class CustomStyle : DataGridStyle
{
    public CustomStyle ()
    {
    }
    public override ImageSource GetGroupCollapseIcon ()
    {
        return null;
    }
}
```

### How to hide the grouped column in SfDataGrid

To hide/show a particular column gets grouped, set the [SfDataGrid.ShowColumnWhenGrouped](#) property to `false/true`. Any column(s) added in the `GroupColumnDescriptions` collection will be shown or hidden based on the value of the [SfDataGrid.ShowColumnWhenGrouped](#) property. Refer to the following code snippet:

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    ShowColumnWhenGrouped="False"/>
```

#### C#

```
this.dataGrid.ShowColumnWhenGrouped = false;
```

### Summary

The data grid supports to display the concise information about the bound data objects using summaries. The control provides the following summary types:

- **Caption Summary** - Used to display the summary information in the caption of the group.
- **Group Summary** - Used to display summary information of data objects in each group.
- **Table Summary** - Used to display the summary information at top and/or bottom in SfDataGrid.



Summary rows are represented by using the [GridSummaryRow](#) that hold summary information of columns in the [SummaryColumns](#) property. The [SummaryColumns](#) contains the collection of [GridSummaryColumn](#) which carries name, format, and summary aggregate type of the column.

Derive additional information from the data like sum, average, maximum, minimum, and count using summaries in the data grid. These summary values can be computed for groups or for the entire control using [GridSummaryRow](#) and [GridSummaryColumn](#) that implements [ISummaryRow](#) and [ISummaryColumn](#) interfaces.

**Note:** The Summary does not refresh with data. To update the summary for the newly added row, or for the modified summary column, set the [SfDataGrid.View.LiveDataUpdateMode](#) to [LiveDataUpdateMode.AllowDataShaping](#) or [LiveDataUpdateMode.AllowSummaryUpdate](#).

### Caption summaries

The data grid provides built-in support for caption summaries. The caption summary value calculated based on the records in a group. The summary information will be displayed in the caption of group.

The following screenshot shows the built-in caption summary of a group:

![DataGrid with caption summary](SfDataGrid\_images/Caption summaries.png)

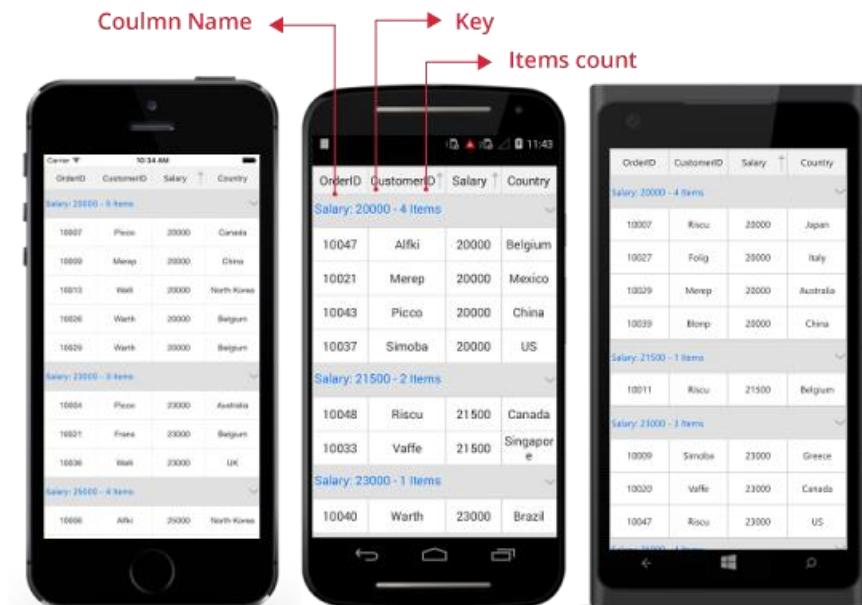
### Formatting built-in caption summary

By default, caption summary rows are displayed with the [SfDataGrid.GroupCaptionTextFormat](#) property.

Default group caption format is `{ColumnName}: {Key} - {ItemsCount} Items`.

- **ColumnName:** Displays the name of the currently grouped column.

- **Key:** Displays the key value of the group.
- **ItemCount:** Displays the number of items in a group.



The group caption text format can be customized by setting the `SfDataGrid.GroupCaptionTextFormat` property. The following code example illustrates how to customize group caption text in the data grid:

#### XML

```
<sfGrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ColumnSizer="Star"
GroupCaptionTextFormat="{0}{ColumnName}: {Key}">
```

#### C#

```
//Customized group caption text
dataGrid.GroupCaptionTextFormat = "{0}{ColumnName} : {Key}";
```

The following screenshot shows the outcome of the previous code:



### Displaying summary for a row

Display summary information in a row by setting the [GridGroupSummaryRow.ShowSummaryInRow](#) property to `true` and define summary columns. You have to define the [GridGroupSummaryRow.Title](#) based on the [GridSummaryColumn.Name](#) property to format summary columns value in a row.

### XML

```
<sfGrid:SfDataGrid.CaptionSummaryRow>
  <sfGrid:GridGroupSummaryRow Title="Total Salary :{TotalSalary} for
  {ProductCount} members"
  ShowSummaryInRow="True">
    <sfGrid:GridGroupSummaryRow.SummaryColumns>
      <sfGrid:GridSummaryColumn Name="TotalSalary"
      Format="{ }{Sum:c}"
      MappingName="Salary"
      Format="{ }{Count}"
      MappingName="Salary"
      SummaryType="CountAggregate" />
    </sfGrid:GridGroupSummaryRow.SummaryColumns>
  </sfGrid:GridGroupSummaryRow>
</sfGrid:SfDataGrid.CaptionSummaryRow>
```

### C#

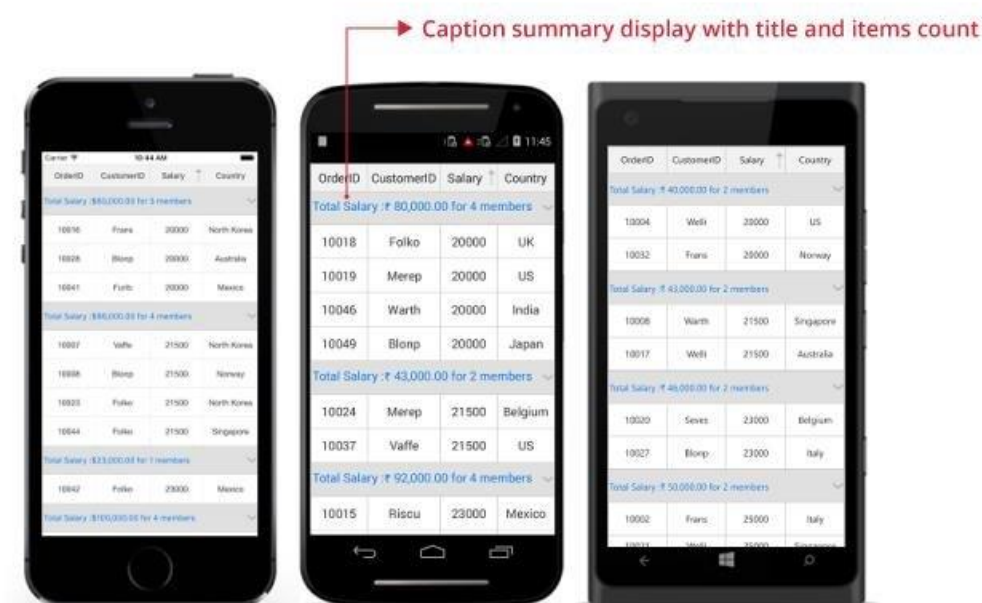
```
GridGroupSummaryRow summaryRow = new GridGroupSummaryRow();
summaryRow.Title = "Total Salary:{TotalSalary} for {ProductCount} members";
summaryRow.ShowSummaryInRow = true;
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "TotalSalary",
    MappingName = "Salary",
```

```

Format = "{Sum:c}",
SummaryType = SummaryType.DoubleAggregate
});
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "ProductCount",
    MappingName = "Salary",
    Format = "{Count}",
    SummaryType = SummaryType.CountAggregate
});
sfGrid.CaptionSummaryRow= summaryRow;

```

The following screenshot shows the outcome for both values of `ShowSummaryInRow` to `true`:



### Displaying summary for column

Display summary information in the column by setting the `GridSummaryRow.ShowSummaryInRow` to `false` and define summary columns. `SfDataGrid.GridSummaryColumn` is the object of [GridSummaryRow.SummaryColumns](#) collection that contains the following important properties:

- **Name:** Defines name of the `GridSummaryColumn` to denote the `GridSummaryColumn` in `GridSummaryRow` with title.
- **MappingName:** Defines the corresponding column name used for the summary calculation.
- **SummaryType:** Defines the `SummaryType` (enum) property to define the aggregate type for the summary calculation.

The `DataGrid` control provides the following predefined aggregates:

- `CountAggregate`

- Int32Aggregate
- DoubleAggregate

[CustomAggregate](#) defines the `CustomAggregate` class object when the summary type is set as `Custom` that calculates the custom summaries.

The [Format](#) defines the `string` property that formats the summary value and displays it. The `Format` property may contains two parts separated by a colon (:). First part denotes the aggregate function name, and second part denotes display format of the summary value.

Refer to the [Formatting Summary](#) section to know more about how to format summary, and [Aggregate Types](#) section to know about different summary types.

In the following code snippet, summary is defined for `Salary` column:

#### XML

```
<sfGrid:SfDataGrid.CaptionSummaryRow>
<sfGrid:GridSummaryRow Name="CaptionSummary" ShowSummaryInRow="False">
<sfGrid:GridSummaryRow.SummaryColumns>
<sfGrid:GridSummaryColumn Name="CaptionSummary"
Format="{Sum}"
MappingName="Salary"
SummaryType="DoubleAggregate" />
</sfGrid:GridSummaryRow.SummaryColumns>
</sfGrid:GridSummaryRow>
</sfGrid:SfDataGrid.CaptionSummaryRow>
```

#### C#

```
GridSummaryRow summaryRow = new GridSummaryRow();
summaryRow.ShowSummaryInRow = false;
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "CaptionSummary",
    MappingName = "Salary",
    Format = "{Sum:c}",
    SummaryType = SummaryType.DoubleAggregate
});
sfGrid.CaptionSummaryRow= summaryRow;
```



**Note:** The CaptionSummaryColumn text will be aligned based on the `GridColumn.TextAlignment`.

#### *Caption summary template*

The data grid hosts any view(s) inside a caption summary for the entire row or for individual columns by loading a template.

#### *Displaying template for a row*

The template for a caption summary row can be set by using [SfDataGrid.CaptionSummaryTemplate](#) to customize it based on requirement.

Refer the below code example in which a label is loaded in the caption summary template of caption summary row.

#### **XML**

```
<ContentPage.Resources>
<ResourceDictionary>
<local:DisplayBindingConverter x:Key="SummaryConverter" />
</ResourceDictionary>
</ContentPage.Resources>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<sfgrid:SfDataGrid.CaptionSummaryTemplate>
<DataTemplate>
<StackLayout Orientation="Horizontal" BackgroundColor="Gray">
<Image Source="{local:ImageResource UG_Sample.SalaryIcon.Png}"
Margin="0,5,0,5"
HorizontalOptions="Start"
VerticalOptions="Center"/>
<Label Text="{Binding Converter={StaticResource SummaryConverter},
ConverterParameter = {x:Reference dataGrid} }"
TextColor="White"
```



```

FontSize="Large"
VerticalTextAlignment="Center"
HorizontalTextAlignment="Start"
LineBreakMode="NoWrap"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
<Label.Style>
<Style TargetType="Label">
<Setter Property="FontAttributes" Value="Bold, Italic" />
</Style>
</Label.Style>
</Label>
</StackLayout>
</DataTemplate>
</sfgrid:SfDataGrid.CaptionSummaryTemplate>
<sfgrid:SfDataGrid.CaptionSummaryRow>
<sfgrid:GridSummaryRow Name="CaptionSummary" ShowSummaryInRow="True"
Title="Salary: {CaptionSummary}">
<sfgrid:GridSummaryRow.SummaryColumns>
<sfgrid:GridSummaryColumn Name="CaptionSummary"
Format="{ }{Sum}"
MappingName="Salary"
SummaryType="DoubleAggregate" />
</sfgrid:GridSummaryRow.SummaryColumns>
</sfgrid:GridSummaryRow>
</sfgrid:SfDataGrid.CaptionSummaryRow>
</sfgrid:SfDataGrid>

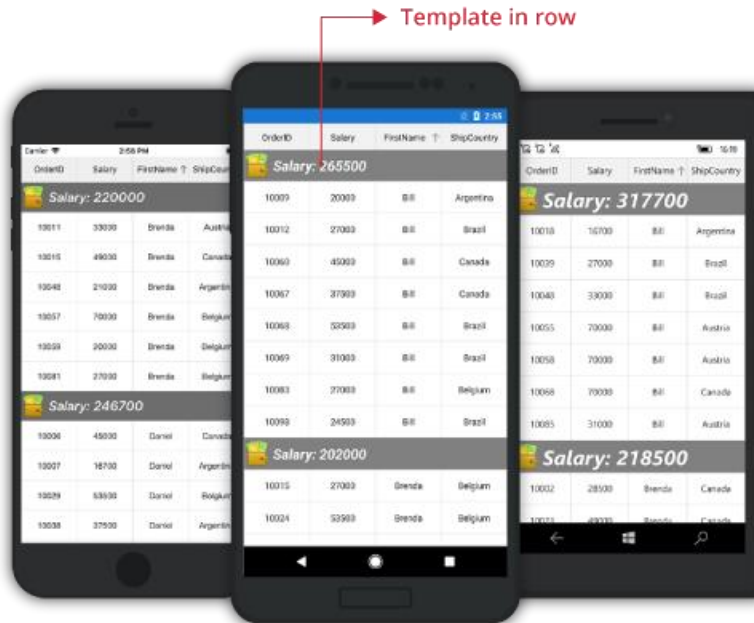
```

## C#

```

// To write a converter, follow the code example:
public class GroupCaptionConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        {
            var data = value != null ? value as Group : null;
            if (data != null)
            {
                SfDataGrid dataGrid = (SfDataGrid)parameter;
                var summaryText = SummaryCreator.GetSummaryDisplayTextForRow((value as
                    Group).SummaryDetails, dataGrid.View);
                return summaryText;
            }
            return null;
        }
        public object ConvertBack(object value, Type targetType, object parameter,
            CultureInfo culture)
        {
            return null;
        }
    }
}

```



**Note:** The `DataTemplateSelector` can also be directly assigned to the `CaptionSummaryTemplate`.

#### Displaying template for a column

The template for a Caption summary column can be set by using `GridSummaryColumn.Template` to customize it based on requirement.

Refer the below code example in which a label is loaded in the template of caption summary column.

#### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<local:GroupCaptionConverter x:Key="SummaryConverter" />
</ResourceDictionary>
</ContentPage.Resources>
<sfgrid:SfDataGrid x:Name="dataGrid" AllowResizingColumn="True"
AutoGenerateColumns="False" ColumnSizer="Star">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn MappingName="OrderID" />
<sfgrid:GridTextColumn MappingName="Salary" />
<sfgrid:GridTextColumn MappingName="CustomerID" />
</sfgrid:SfDataGrid.Columns>
<sfgrid:SfDataGrid.CaptionSummaryRow>
<sfgrid:GridSummaryRow Name="CaptionSummary" ShowSummaryInRow="False"
Title="Salary: {CaptionSummary}">
<sfgrid:GridSummaryRow.SummaryColumns>
<sfgrid:GridSummaryColumn Name="CaptionSummary" Format="{{Sum}}"
MappingName="Salary"
SummaryType="DoubleAggregate" >
<sfgrid:GridSummaryColumn.Template>
<DataTemplate>
<StackLayout Orientation="Horizontal" BackgroundColor="DarkCyan">
```

```

<Image PropertyChanged="Image_PropertyChanged" Margin="0,5,0,5"
HorizontalOptions="Start"
VerticalOptions="Center"/>
<Label Text="{Binding Converter={StaticResource SummaryConverter},
ConverterParameter = {x:Reference dataGrid} }"
TextColor="White" FontSize="Small"
VerticalTextAlignment="Center"
HorizontalTextAlignment="Start"
LineBreakMode="NoWrap"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
<Label.Style>
<Style TargetType="Label">
<Setter Property="FontAttributes" Value="Bold, Italic" />
</Style>
</Label.Style>
</Label>
</StackLayout>
</DataTemplate>
</sfgrid:GridSummaryColumn.Template>
</sfgrid:GridSummaryColumn>
</sfgrid:GridSummaryRow.SummaryColumns>
</sfgrid:GridSummaryRow>
</sfgrid:SfDataGrid.CaptionSummaryRow>
</sfgrid:SfDataGrid>

```

## C#

```

// To write a converter, follow the code example:
public class GroupCaptionConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var data = value != null ? value as Group : null;
        if (data != null)
        {
            SfDataGrid dataGrid = (SfDataGrid)parameter;
            var summaryText = SummaryCreator.GetSummaryDisplayTextForRow((value as
                Group).SummaryDetails, dataGrid.View);
            return summaryText;
        }
        return null;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return null;
    }
}

```



### Group summary

Group summary values are calculated based on records in the group. The summary information will be displayed at the bottom of each group. You can view the group summary row by expanding the corresponding group header. The data grid adds any number of group summary row.

Add group summary rows in the data grid by adding the [GridGroupSummaryRow](#) to [SfDataGrid.GroupSummaryRows](#) collection.

### Displaying summary in the row

The summary information can be displayed in the row by setting the [GridGroupSummaryRow.ShowSummaryInRow](#) to true and by defining summary columns. You have to define the [GridGroupSummaryRow.Title](#) based on the [GridGroupSummaryRow.Name](#) property to format summary columns value in a row.

Refer to [Formatting Summary](#) section to know more about how to format summary.

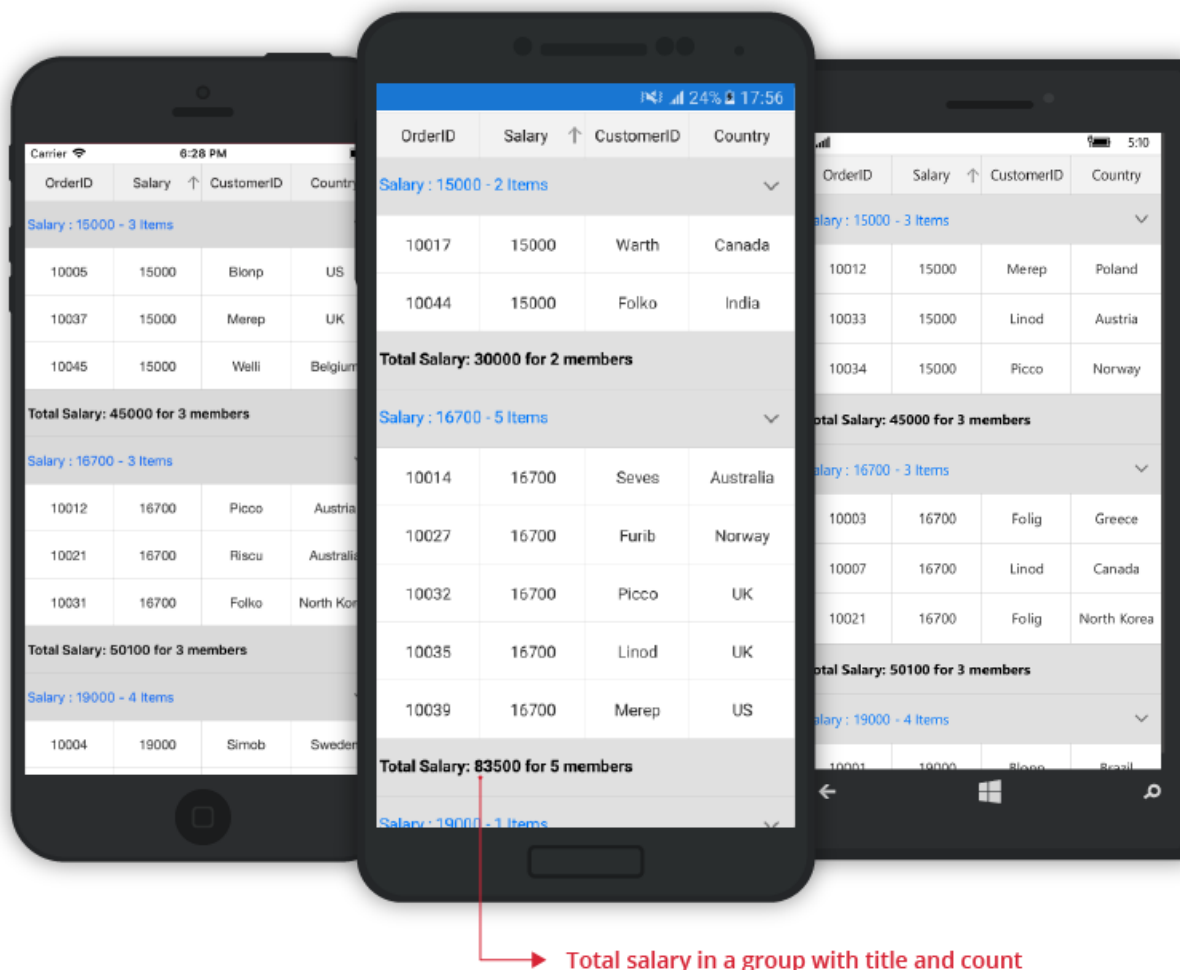
### XML

```
<ContentPage x:Class="SummaryDemo.Summary"
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SummaryDemo"
xmlns:sfgrid="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
">
<sfgrid:SfDataGrid x:Name="dataGrid" AllowResizingColumn="True"
AllowGroupExpandCollapse="True"
AutoGenerateColumns="True"
ColumnSizer="Star">
<sfgrid:SfDataGrid.GroupColumnDescriptions>
<sfgrid:GroupColumnDescription ColumnName="Salary" />
</sfgrid:SfDataGrid.GroupColumnDescriptions>
```

```
<sfgrid:SfDataGrid.GroupSummaryRows>
<sfgrid:GridGroupSummaryRow ShowSummaryInRow="True" Title="Total Salary:
{Salary} for {customerID} members">
<sfgrid:GridGroupSummaryRow.SummaryColumns>
<sfgrid:GridSummaryColumn Name="Salary"
MappingName="Salary"
Format="{0}{Sum}"
SummaryType="DoubleAggregate">
</sfgrid:GridSummaryColumn>
<sfgrid:GridSummaryColumn Name="customerID"
MappingName="CustomerID"
Format="{0}{Count}"
SummaryType="CountAggregate">
</sfgrid:GridSummaryColumn>
</sfgrid:GridGroupSummaryRow.SummaryColumns>
</sfgrid:GridGroupSummaryRow>
</sfgrid:SfDataGrid.GroupSummaryRows>
</sfgrid:SfDataGrid>
</ContentPage>
```

## C#

```
this.dataGrid.GroupSummaryRows.Add(new GridGroupSummaryRow()
{
    ShowSummaryInRow = true,
    Title = "Total Salary: {Salary} for {customerID} members",
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="Salary",
            MappingName="Salary",
            SummaryType=SummaryType.DoubleAggregate,
            Format="{0}{Sum}"
        },
        new GridSummaryColumn()
        {
            Name="customerID",
            MappingName="customerID",
            Format="{0}{Count}",
            SummaryType=SummaryType.CountAggregate
        }
    }
});
```



### Displaying summary in the column

The summary information can be displayed in the column by setting the [GridGroupSummaryRow.ShowSummaryInRow](#) to false and by defining summary columns. To calculate summary based on the column, specify the following properties:

1. [GridSummaryColumn.MappingName](#): Provides MappingName of the column (Property name of data object) that you want to calculate summary.
2. [GridSummaryColumn.SummaryType](#): Provides different built-in summary calculation functions for various types.
3. [GridSummaryColumn.Format](#): Provides format string for the summary based on support function name in the specified SummaryType.

Refer to [Formatting Summary](#) section to know more about how to format summary and [Aggregate Types](#) section to know about different Summary Types.

In the following code snippet, summary is defined for `Salary` and `CustomerID` columns:

### XML

```
<ContentPage x:Class="SummaryDemo.Summary"
```

```

xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SummaryDemo"
xmlns:sfgrid="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
">
<sfgrid:SfDataGrid x:Name="dataGrid" AllowResizingColumn="True"
AutoGenerateColumns="True"
ColumnSizer="Star">
<sfgrid:SfDataGrid.GroupSummaryRows>
<sfgrid:GridGroupSummaryRow ShowSummaryInRow="False" >
<sfgrid:GridGroupSummaryRow.SummaryColumns>
<sfgrid:GridSummaryColumn Name="Salary"
MappingName="Salary"
Format="{Sum}"
SummaryType="DoubleAggregate">
</sfgrid:GridSummaryColumn>
<sfgrid:GridSummaryColumn Name="customerID"
MappingName="CustomerID"
Format="Total members - {Count}"
SummaryType="CountAggregate">
</sfgrid:GridSummaryColumn>
</sfgrid:GridGroupSummaryRow.SummaryColumns>
</sfgrid:GridGroupSummaryRow>
</sfgrid:SfDataGrid.GroupSummaryRows>
</sfgrid:SfDataGrid>
</ContentPage>

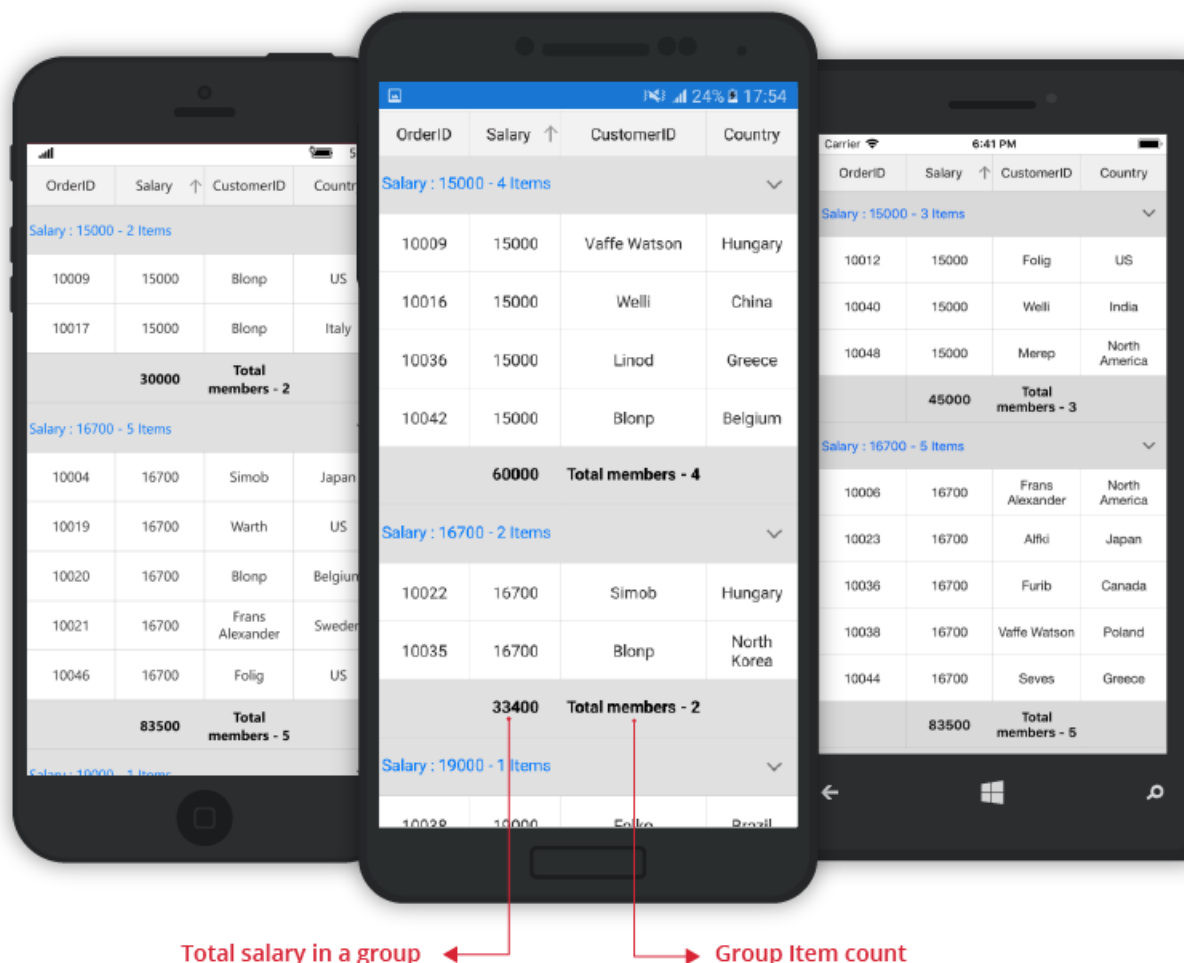
```

## C#

```

this.dataGrid.GroupSummaryRows.Add(new GridGroupSummaryRow()
{
    ShowSummaryInRow = false,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="Salary",
            MappingName="Salary",
            SummaryType=SummaryType.DoubleAggregate,
            Format="{Sum}"
        },
        new GridSummaryColumn()
        {
            Name="customerID",
            MappingName="CustomerID",
            Format="Total members - {Count:d}",
            SummaryType=SummaryType.CountAggregate
        }
    }
});

```



### Group summary template

The data grid hosts any view(s) inside a group summary for the entire row or for individual columns by loading a template.

### Displaying template for a row

The template for a group summary row can be set by using `SfDataGrid.GroupSummaryTemplate` to customize it based on requirement.

Refer the below code example in which a label is loaded in the group summary template of group summary row.

### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<local:GroupSummaryConverter x:Key="SummaryConverter" />
</ResourceDictionary>
</ContentPage.Resources>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<sfgrid:SfDataGrid.GroupSummaryTemplate>
<DataTemplate>
```



```

<StackLayout Orientation="Horizontal" BackgroundColor="Gray">
<Label Text="{Binding Converter={StaticResource SummaryConverter},
ConverterParameter = {x:Reference dataGrid} }"
TextColor="White"
FontSize="Large"
VerticalTextAlignment="Center"
HorizontalTextAlignment="Start"
LineBreakMode="NoWrap"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
<Label.Style>
<Style TargetType="Label">
<Setter Property="FontAttributes" Value="Bold, Italic" />
</Style>
</Label.Style>
</Label>
</StackLayout>
</DataTemplate>
</sfgrid:SfDataGrid.GroupSummaryTemplate>
<sfgrid:SfDataGrid.GroupSummaryRows>
<sfgrid:GridGroupSummaryRow ShowSummaryInRow="True">
<sfgrid:GridGroupSummaryRow.SummaryColumns>
<sfgrid:GridSummaryColumn Name="Salary"
MappingName="Salary"
Format="{ }{Sum}"
SummaryType="DoubleAggregate">
</sfgrid:GridSummaryColumn>
<sfgrid:GridSummaryColumn Name="customerID"
MappingName="CustomerID"
Format="{ }{Count}"
SummaryType="CountAggregate">
</sfgrid:GridSummaryColumn>
</sfgrid:GridGroupSummaryRow.SummaryColumns>
</sfgrid:GridGroupSummaryRow>
</sfgrid:SfDataGrid.GroupSummaryRows>
</sfgrid:SfDataGrid>

```

## C#

```

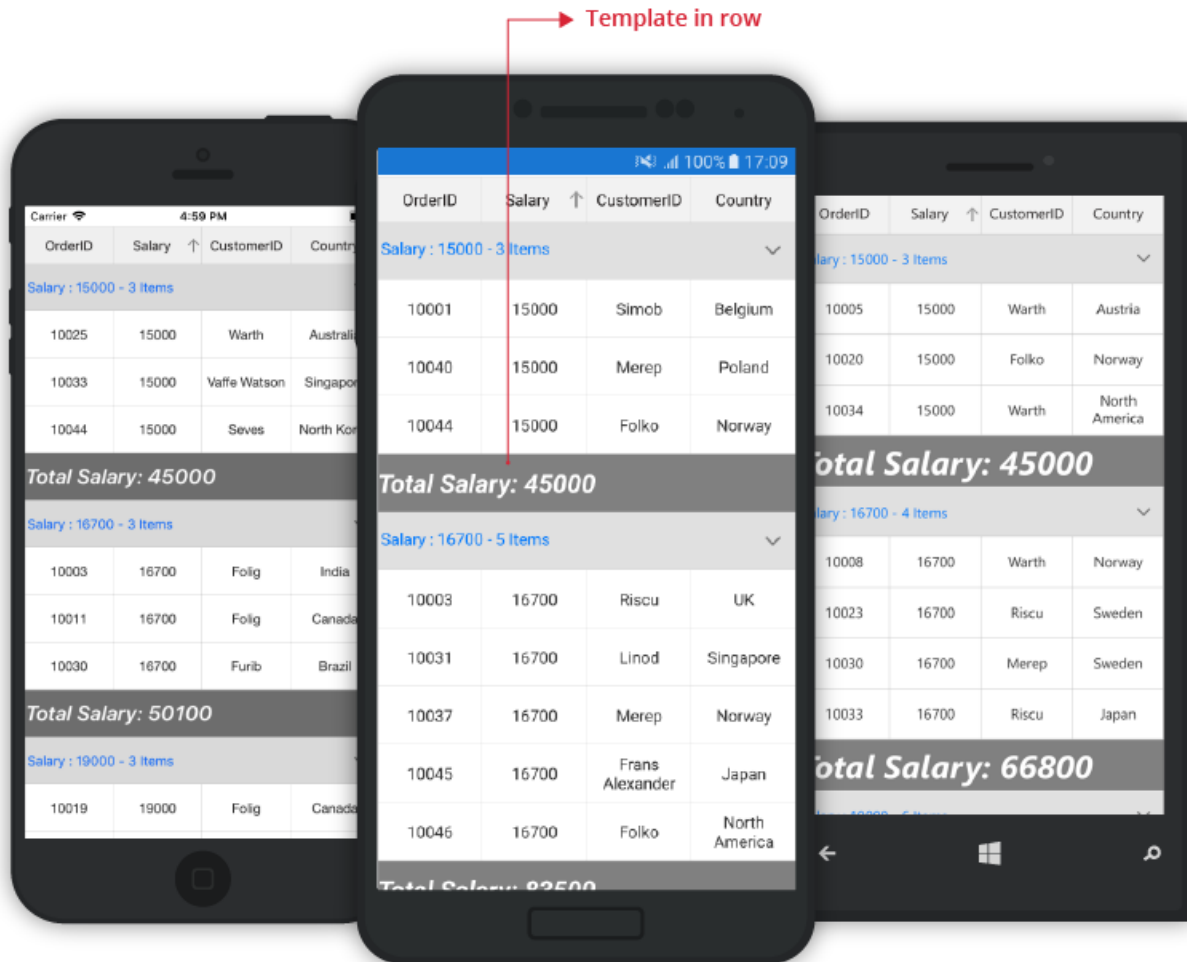
// To write a converter, follow the code example:
public class GroupSummaryConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
{
var data = value != null ? value as SummaryRecordEntry : null;
if (data != null)
{
SfDataGrid dataGrid = (SfDataGrid)parameter;
var summaryText =
SummaryCreator.GetSummaryDisplayText(data, "Salary", dataGrid.View);
return "Total Salary:" + " " + summaryText.ToString();
}
return null;
}
}

```

```

public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
{
    return null;
}
}

```



**Note:** The `DataTemplateSelector` can also be directly assigned to the `SfDataGrid.GroupSummaryTemplate`.

#### Displaying template for a column

The template for a group summary column can be set by using `GridSummaryColumn.Template` to customize it based on requirement.

Refer the below code example in which a label is loaded in the template of group summary column.

#### XML

```

<ContentPage.Resources>
<ResourceDictionary>
<local:GroupSummaryConverter x:Key="SummaryConverter" />

```

```

</ResourceDictionary>
</ContentPage.Resources>
<sfgrid:SfDataGrid x:Name="dataGrid" AllowResizingColumn="True"
AutoGenerateColumns="False" ColumnSizer="Star">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn MappingName="OrderID" />
<sfgrid:GridTextColumn MappingName="Salary" />
<sfgrid:GridTextColumn MappingName="CustomerID" />
<sfgrid:GridTextColumn MappingName="Country" />
</sfgrid:SfDataGrid.Columns>
<sfgrid:SfDataGrid.GroupSummaryRows>
<sfgrid:GridGroupSummaryRow ShowSummaryInRow="False" >
<sfgrid:GridGroupSummaryRow.SummaryColumns>
<sfgrid:GridSummaryColumn Name="Salary"
MappingName="Salary"
Format="{ }{Sum}"
SummaryType="DoubleAggregate">
<sfgrid:GridSummaryColumn.Template>
<DataTemplate>
<StackLayout Orientation="Horizontal" BackgroundColor="Gray">
<Label Text="{Binding Converter={StaticResource SummaryConverter},
ConverterParameter = {x:Reference dataGrid} }"
TextColor="White"
FontSize="Large"
VerticalTextAlignment="Center"
HorizontalTextAlignment="Center"
LineBreakMode="NoWrap"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
<Label.Style>
<Style TargetType="Label">
<Setter Property="FontAttributes" Value="Bold, Italic" />
</Style>
</Label.Style>
</Label>
</StackLayout>
</DataTemplate>
</sfgrid:GridSummaryColumn.Template>
</sfgrid:GridSummaryColumn>
</sfgrid:GridGroupSummaryRow.SummaryColumns>
</sfgrid:GridGroupSummaryRow>
</sfgrid:SfDataGrid.GroupSummaryRows>
</sfgrid:SfDataGrid>

```

## C#

```

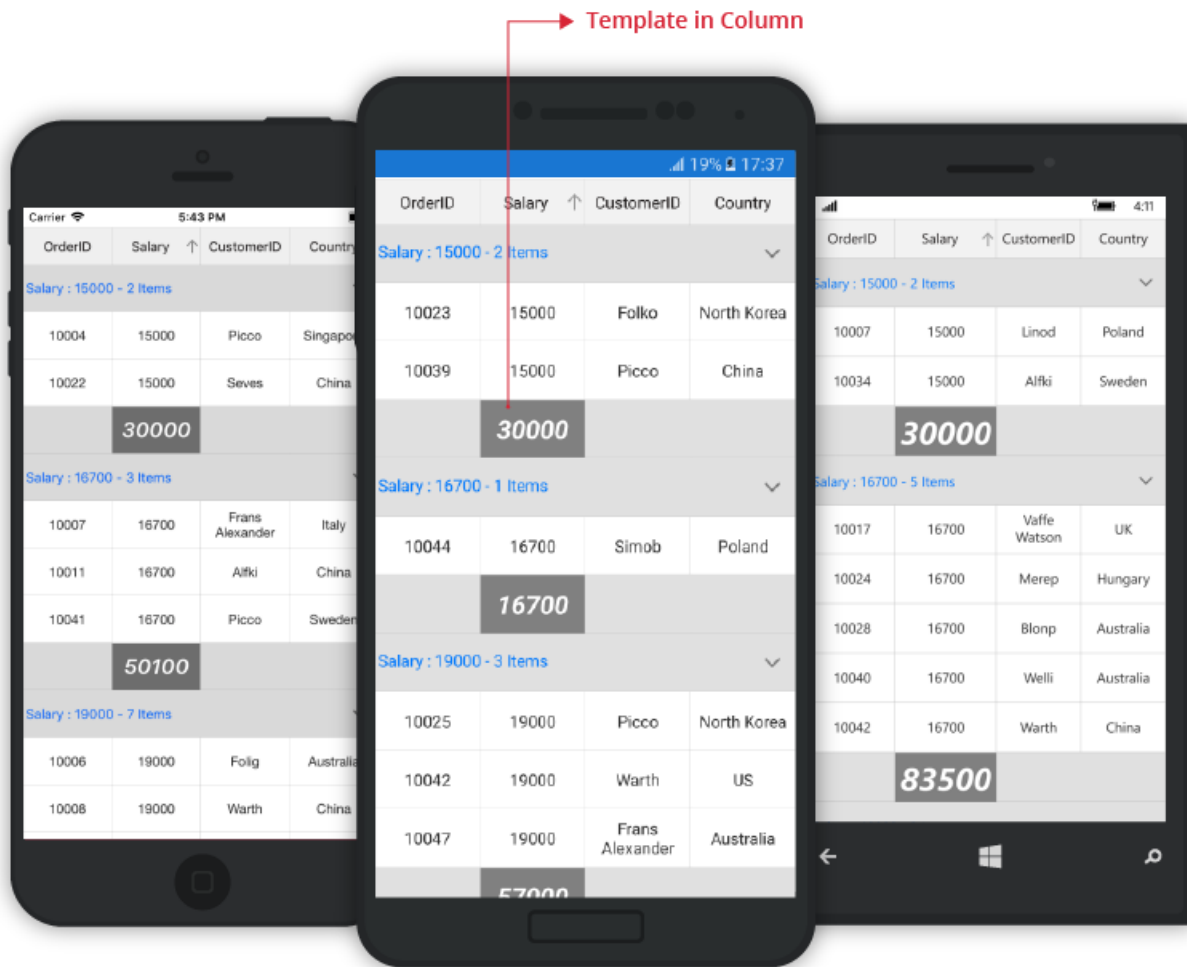
// To write a converter, follow the code example:
public class GroupSummaryConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var data = value != null ? value as SummaryRecordEntry : null;
        if (data != null)
        {
            SfDataGrid dataGrid = (SfDataGrid)parameter;

```

```

var summaryText =
SummaryCreator.GetSummaryDisplayText(data, "Salary", dataGrid.View);
return summaryText.ToString();
}
return null;
}
public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
{
return null;
}
}

```

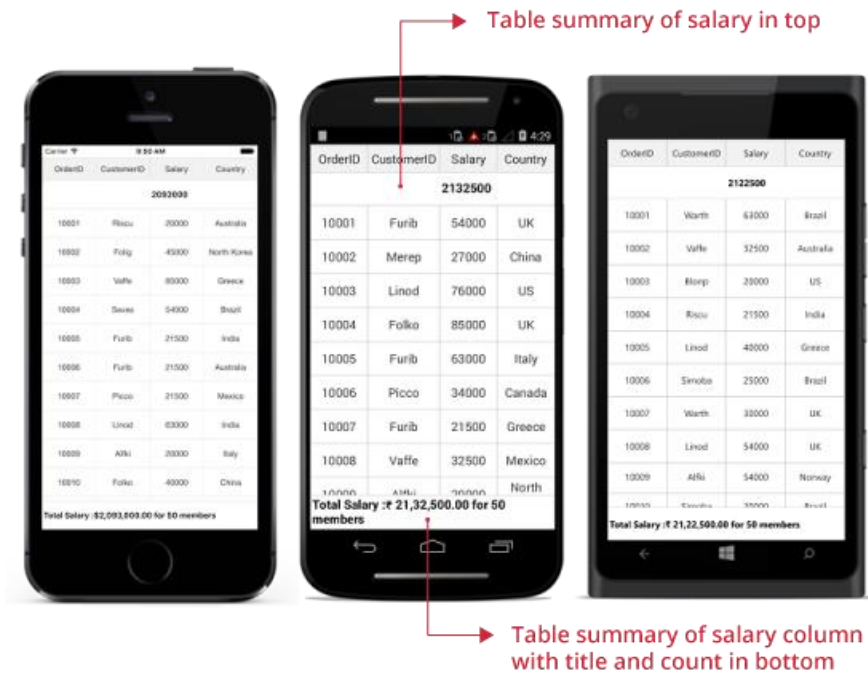


### Table summaries

The data grid provides built-in support for table summaries. The table summary value is calculated based on all records in the control.

You can add table summary row in the data grid by adding the [GridTableSummaryRow](#) to the [SfDataGrid.TableSummaryRows](#) collection.

The following screenshot illustrates table summary rows in the data grid:



### XML

```
<sfGrid:SfDataGrid.TableSummaryRows>
  <sfGrid:GridTableSummaryRow Title="Total Salary :{TotalSalary} for
  {ProductCount} members"
  Position="Top"
  ShowSummaryInRow="True">
    <sfGrid:GridTableSummaryRow.SummaryColumns>
      <sfGrid:GridSummaryColumn Name="TotalSalary"
      Format="{ }{Sum:c}"
      MappingName="Salary"
      SummaryType="DoubleAggregate" />
      <sfGrid:GridSummaryColumn Name="ProductCount"
      Format="{ }{Count}"
      MappingName="Salary"
      SummaryType="CountAggregate" />
    </sfGrid:GridTableSummaryRow.SummaryColumns>
  </sfGrid:GridTableSummaryRow>
  <sfGrid:GridTableSummaryRow Position="Top" ShowSummaryInRow="True">
    <sfGrid:GridTableSummaryRow.SummaryColumns>
      <sfGrid:GridSummaryColumn Name="TotalSalary"
      Format="{ }{Sum}"
      MappingName="Salary"
      SummaryType="DoubleAggregate" />
    </sfGrid:GridTableSummaryRow.SummaryColumns>
  </sfGrid:GridTableSummaryRow>
</sfGrid:SfDataGrid.TableSummaryRows>
```

### C#

```
GridTableSummaryRow summaryRow1 = new GridTableSummaryRow();
summaryRow1.Title = "Total Salary:{TotalSalary} for {ProductCount} members";
```

```

summaryRow1.ShowSummaryInRow = true;
summaryRow1.Position = Position.Top;
summaryRow1.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "TotalSalary",
    MappingName = "Salary",
    Format = "{Sum:c}",
    SummaryType = SummaryType.DoubleAggregate
});
summaryRow1.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "ProductCount",
    MappingName = "Salary",
    Format = "{Count}",
    SummaryType = SummaryType.CountAggregate
});
sfGrid.TableSummaryRows.Add(summaryRow1);
GridTableSummaryRow summaryRow2 = new GridTableSummaryRow();
summaryRow2.ShowSummaryInRow = false;
summaryRow2.Position = Position.Top;
summaryRow2.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "TotalSalary",
    MappingName = "Salary",
    Format = "{Sum}",
    SummaryType = SummaryType.DoubleAggregate
});
sfGrid.TableSummaryRows.Add(summaryRow2);

```

Table summary of salary column  
with title and count in top

Table summary of  
salary in top



### Displaying summary in a row

Display summary information in a row by setting the [GridTableSummaryRow.ShowSummaryInRow](#) to `true` and define summary columns. You have to define the [GridTableSummaryRow.Title](#) based on the [GridSummaryColumn.Name](#) property to format summary columns values in a row.

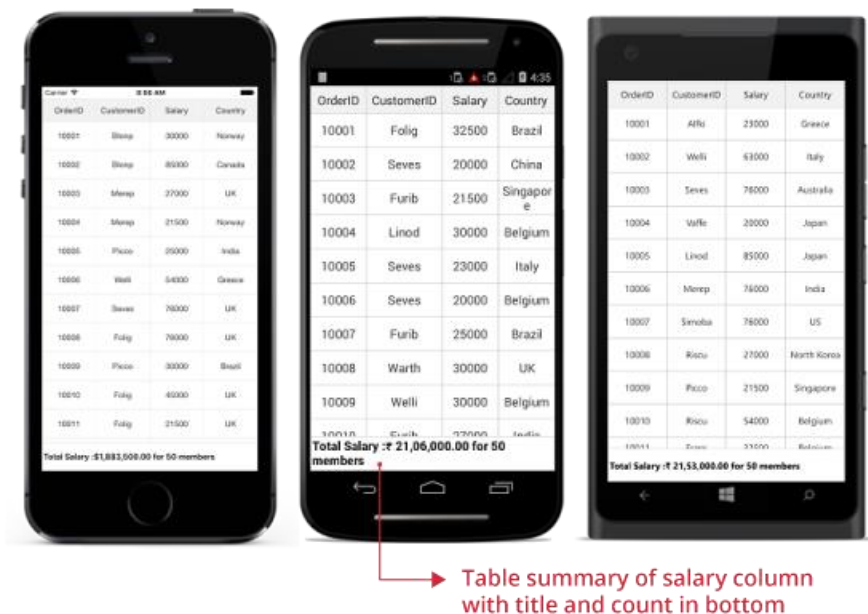
### XML

```
<sfGrid:SfDataGrid.TableSummaryRows>
  <sfGrid:GridTableSummaryRow Title="Total Salary :{TotalSalary} for
    {ProductCount} members"
    ShowSummaryInRow="True">
    <sfGrid:GridTableSummaryRow.SummaryColumns>
      <sfGrid:GridSummaryColumn Name="TotalSalary"
        Format="{Sum:c}"
        MappingName="Salary"
        SummaryType="DoubleAggregate" />
      <sfGrid:GridSummaryColumn Name="ProductCount"
        Format="{Count}"
        MappingName="Salary"
        SummaryType="CountAggregate" />
    </sfGrid:GridTableSummaryRow.SummaryColumns>
  </sfGrid:GridTableSummaryRow>
</sfGrid:SfDataGrid.TableSummaryRows>
```

### C#

```
GridTableSummaryRow summaryRow = new GridTableSummaryRow();
summaryRow.Title = "Total Salary:{TotalSalary} for {ProductCount} members";
summaryRow.ShowSummaryInRow = true;
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "TotalSalary",
    MappingName = "Salary",
    Format = "{Sum:c}",
    SummaryType = SummaryType.DoubleAggregate
});
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "ProductCount",
    MappingName = "Salary",
    Format = "{Count}",
    SummaryType = SummaryType.CountAggregate
});
sfGrid.TableSummaryRows.Add(summaryRow);
```

The following screenshot shows the table summary row if `ShowSummaryInRow` is `true`:



### Displaying summary in a column

Display summary information in a column by setting `GridTableSummaryRow.ShowSummaryInRow` to false and defining summary columns. `GridSummaryColumn` is the object of [GridTableSummaryRow.SummaryColumns](#) collection that contains the following important properties:

- **Name:** Defines name of the `GridSummaryColumn` to denote the `GridSummaryColumn` in `GridTableSummaryRow` with title.
- **MappingName:** Defines the corresponding column name for the summary calculation.
- **SummaryType:** Defines the `SummaryType` (enum) property to define the aggregate type for the summary calculation.

The data grid control provides the following predefined aggregates:

- `CountAggregate`
- `Int32Aggregate`
- `DoubleAggregate`

[CustomAggregate](#) defines the `CustomAggregate` class object when the summary type is set as `Custom` that calculates custom summaries.

The **Format** defines the string property that formats the summary value and displays it. The `Format` property may contains two parts that are separated by a colon (:). First part denotes the aggregate function name, and second part denotes display format of the summary value.

Refer to the [Formatting Summary](#) section to know more about how to format summary and [Aggregate Types](#) section to know about different summary types.

In the following code snippet, summary is defined for `Salary` column:



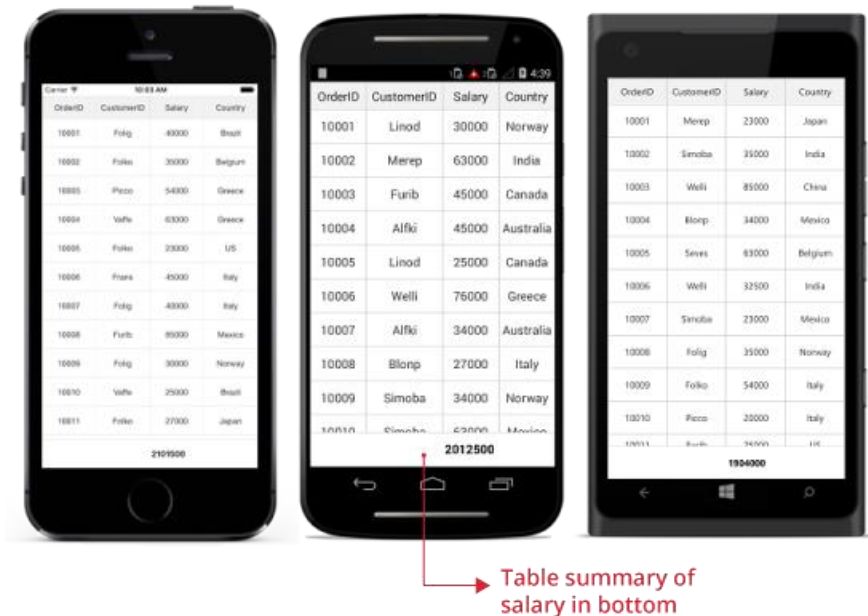
**XML**

```
<sfGrid:SfDataGrid.TableSummaryRows>
<sfGrid:GridTableSummaryRow Name="TableSummary" ShowSummaryInRow="False">
<sfGrid:GridTableSummaryRow.SummaryColumns>
<sfGrid:GridSummaryColumn Name="TableSummary"
Format="{ }{Sum}"
MappingName="Salary"
SummaryType="DoubleAggregate" />
</sfGrid:GridTableSummaryRow.SummaryColumns>
</sfGrid:GridTableSummaryRow>
</sfGrid:SfDataGrid.TableSummaryRows>
```

**C#**

```
GridTableSummaryRow summaryRow = new GridTableSummaryRow();
summaryRow.ShowSummaryInRow = false;
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "TableSummary",
    MappingName = "Salary",
    Format = "{Sum}",
    SummaryType = SummaryType.DoubleAggregate
});
sfGrid.TableSummaryRows.Add(summaryRow);
```

The following screenshot shows the table summary row if `ShowSummaryInRow` is `false`.

*Positioning TableSummaryRows*

The data grid add table summary rows either at top or bottom positions using the [GridTableSummaryRow.Position](#) property.

**XML**

```

<sfGrid:SfDataGrid.TableSummaryRows>
  <sfGrid:GridTableSummaryRow Position="Top"
    ShowSummaryInRow="False">
    <sfGrid:GridTableSummaryRow.SummaryColumns>
      <sfGrid:GridSummaryColumn Name="TotalSalary"
        Format="{Sum}"
        MappingName="Salary"
        SummaryType="DoubleAggregate" />
    </sfGrid:GridTableSummaryRow.SummaryColumns>
  </sfGrid:GridTableSummaryRow>
  <sfGrid:GridTableSummaryRow Position="Bottom"
    ShowSummaryInRow="True"
    Title="Total Salary :{TotalSalary} for {ProductCount} members">
    <sfGrid:GridTableSummaryRow.SummaryColumns>
      <sfGrid:GridSummaryColumn Name="TotalSalary"
        Format="{Sum:c}"
        MappingName="Salary"
        SummaryType="DoubleAggregate" />
      <sfGrid:GridSummaryColumn Name="ProductCount"
        Format="{Count}"
        MappingName="Salary"
        SummaryType="CountAggregate" />
    </sfGrid:GridTableSummaryRow.SummaryColumns>
  </sfGrid:GridTableSummaryRow>
</sfGrid:SfDataGrid.TableSummaryRows>

```

**C#**

```

GridTableSummaryRow topSummaryRow = new GridTableSummaryRow();
topSummaryRow.Position = Position.Top;
topSummaryRow.ShowSummaryInRow = false;
topSummaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "TotalSalary",
    MappingName = "Salary",
    Format = "{Sum}",
    SummaryType = SummaryType.DoubleAggregate
});
sfGrid.TableSummaryRows.Add(topSummaryRow);
GridTableSummaryRow bottomSummaryRow = new GridTableSummaryRow();
bottomSummaryRow.Position = Position.Bottom;
bottomSummaryRow.Title = "Total Salary:{TotalSalary} for {ProductCount}
members";
bottomSummaryRow.ShowSummaryInRow = true;
bottomSummaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "TotalSalary",
    MappingName = "Salary",
    Format = "{Sum:c}",
    SummaryType = SummaryType.DoubleAggregate
});
bottomSummaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "ProductCount",

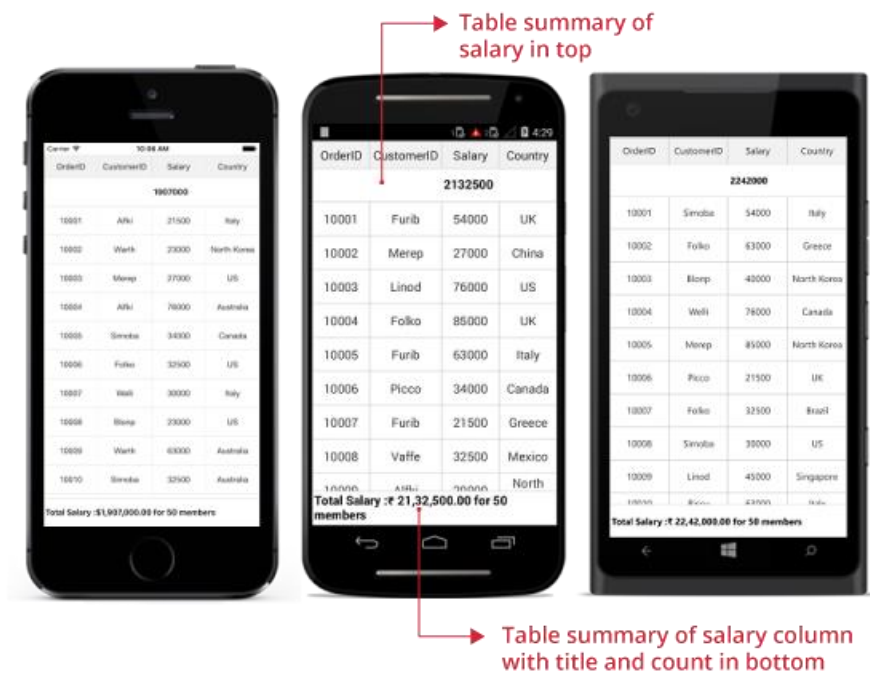
```

```

MappingName = "Salary",
Format = "{Count}",
SummaryType = SummaryType.CountAggregate
});
sfGrid.TableSummaryRows.Add(bottomSummaryRow);

```

The below screenshot illustrates the positioning of table summary rows in SfDataGrid.



### Table summary template

The data grid hosts any view(s) inside a table summary for the entire row or for individual columns by loading a template.

#### Displaying template for a row

The template for a table summary row can be set by using [SfDataGrid.TableSummaryTemplate](#) and it can be customized based on the requirement.

Refer the below code example in which a label is loaded in the table summary template of table summary row.

### XML

```

<ContentPage.Resources>
<ResourceDictionary>
<local:TableSummaryConverter x:Key="SummaryConverter" />
</ResourceDictionary>
</ContentPage.Resources>
<StackLayout>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
AutoGenerateColumns="False"
AllowEditing="True"
NavigationMode="Cell"

```

```

SelectionMode="Single"
ColumnSizer="Star">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridNumericColumn MappingName="OrderID" />
<sfgrid:GridTextColumn MappingName="EmployeeID" />
<sfgrid:GridTextColumn MappingName="FirstName" />
<sfgrid:GridTextColumn MappingName="LastName" />
</sfgrid:SfDataGrid.Columns>
<sfgrid:SfDataGrid.TableSummaryTemplate>
<DataTemplate>
<ViewCell>
<StackLayout Orientation="Horizontal" BackgroundColor="Gray">
<Label Text="{Binding Converter={StaticResource SummaryConverter},
ConverterParameter = {x:Reference dataGrid} }"
TextColor="White"
FontSize="Large"
VerticalTextAlignment="Center"
HorizontalTextAlignment="Start"
LineBreakMode="NoWrap"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
<Label.Style>
<Style TargetType="Label">
<Setter Property="FontAttributes" Value="Italic" />
</Style>
</Label.Style>
</Label>
</StackLayout>
</ViewCell>
</DataTemplate>
</sfgrid:SfDataGrid.TableSummaryTemplate>
<sfgrid:SfDataGrid.TableSummaryRows>
<sfgrid:GridTableSummaryRow Title="Total Salary :{TotalSalary} for
{ProductCount} members"
Position="Bottom"
ShowSummaryInRow="True">
<sfgrid:GridTableSummaryRow.SummaryColumns>
<sfgrid:GridSummaryColumn Name="TotalSalary"
Format="{ }{Sum:c}"
MappingName="OrderID"
SummaryType="DoubleAggregate" />
</sfgrid:GridTableSummaryRow.SummaryColumns>
</sfgrid:GridTableSummaryRow>
</sfgrid:SfDataGrid.TableSummaryRows>
</sfgrid:SfDataGrid>
</StackLayout>

```

## C#

```

public class TableSummaryConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var data = value != null ? value as SummaryRecordEntry : null;
        if (data != null)

```

```
{
    SfDataGrid dataGrid = (SfDataGrid)parameter;
    var summaryText = SummaryCreator.GetSummaryDisplayText(data, "OrderID",
dataGrid.View);
    return "Total Value:" + " " + summaryText.ToString();
}
return null;
}
public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
{
    return null;
}
}
```

OrderID	EmployeeID	FirstName
10001.00	1703	Oscar
10002.00	1704	Michael
10003.00	1702	Danielle
10004.00	1700	Torrey
10005.00	1716	Irene
10006.00	1706	Danielle
10007.00	1719	Torrey
10008.00	1712	Bill
10009.00	1715	Kyle
10010.00	1709	Daniel
10011.00	1702	Brenda
10012.00	1716	Torrey
Total Value: ₹ 5,01,275.00		

*Displaying template for a column*

The template for a table summary column can be set by using `GridSummaryColumn.Template` and it can be customized based on the requirement.

Refer the below code example in which a label is loaded in the template of table summary column.

### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<local:TableSummaryConverter x:Key="SummaryConverter" />
</ResourceDictionary>
</ContentPage.Resources>
<StackLayout>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
AutoGenerateColumns="False"
AllowEditing="True"
NavigationMode="Cell"
SelectionMode="Single"
ColumnSizer="Star">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridNumericColumn MappingName="OrderID" />
<sfgrid:GridTextColumn MappingName="EmployeeID" />
<sfgrid:GridTextColumn MappingName="FirstName" />
<sfgrid:GridTextColumn MappingName="LastName" />
</sfgrid:SfDataGrid.Columns>
<sfgrid:SfDataGrid.TableSummaryRows>
<sfgrid:GridTableSummaryRow Title="Total Salary :{TotalSalary} for
{ProductCount} members"
Position="Bottom"
ShowSummaryInRow="False">
<sfgrid:GridTableSummaryRow.SummaryColumns>
<sfgrid:GridSummaryColumn Name="TotalSalary"
Format="{ }{Sum:c}"
MappingName="OrderID"
SummaryType="DoubleAggregate" >
<sfgrid:GridSummaryColumn.Template>
<DataTemplate>
<StackLayout Orientation="Horizontal" BackgroundColor="Crimson">
<Label Text="{Binding Converter={StaticResource SummaryConverter},
ConverterParameter = {x:Reference dataGrid} }"
TextColor="White"
FontSize="Large"
VerticalTextAlignment="Center"
HorizontalTextAlignment="Start"
LineBreakMode="NoWrap"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
<Label.Style>
<Style TargetType="Label">
<Setter Property="FontAttributes" Value="Italic" />
</Style>
</Label.Style>
</Label>
</StackLayout>
</DataTemplate>
</sfgrid:GridSummaryColumn.Template>
</sfgrid:GridSummaryColumn>
</sfgrid:GridTableSummaryRow.SummaryColumns>
</sfgrid:GridTableSummaryRow>
```

```
</sfgrid:SfDataGrid.TableSummaryRows>  
</sfgrid:SfDataGrid>  
</StackLayout>
```

## C#

```
// To write a converter, follow the code example:  
public class TableSummaryConverter : IValueConverter  
{  
    public object Convert(object value, Type targetType, object parameter,  
        CultureInfo culture)  
    {  
        var data = value != null ? value as SummaryRecordEntry : null;  
        if (data != null)  
        {  
            SfDataGrid dataGrid = (SfDataGrid)parameter;  
            var summaryText = SummaryCreator.GetSummaryDisplayText(data, "OrderID",  
                dataGrid.View);  
            return summaryText.ToString();  
        }  
        return null;  
    }  
    public object ConvertBack(object value, Type targetType, object parameter,  
        CultureInfo culture)  
    {  
        return null;  
    }  
}
```



OrderID	EmployeeID	FirstName
10001.00	1703	Oscar
10002.00	1704	Michael
10003.00	1702	Danielle
10004.00	1700	Torrey
10005.00	1716	Irene
10006.00	1706	Danielle
10007.00	1719	Torrey
10008.00	1712	Bill
10009.00	1715	Kyle
10010.00	1709	Daniel
10011.00	1702	Brenda
10012.00	1716	Torrey
₹ 5,01,275.00		

**Note:** The `DataTemplateSelector` can also be directly assigned to the `SfDataGrid.TableSummaryTemplate`.

### Formatting summary

In the following sections, the formatting is explained using the `CaptionSummary`. However, the formatting can also be applied for `TableSummaries`.

#### Defining summary function

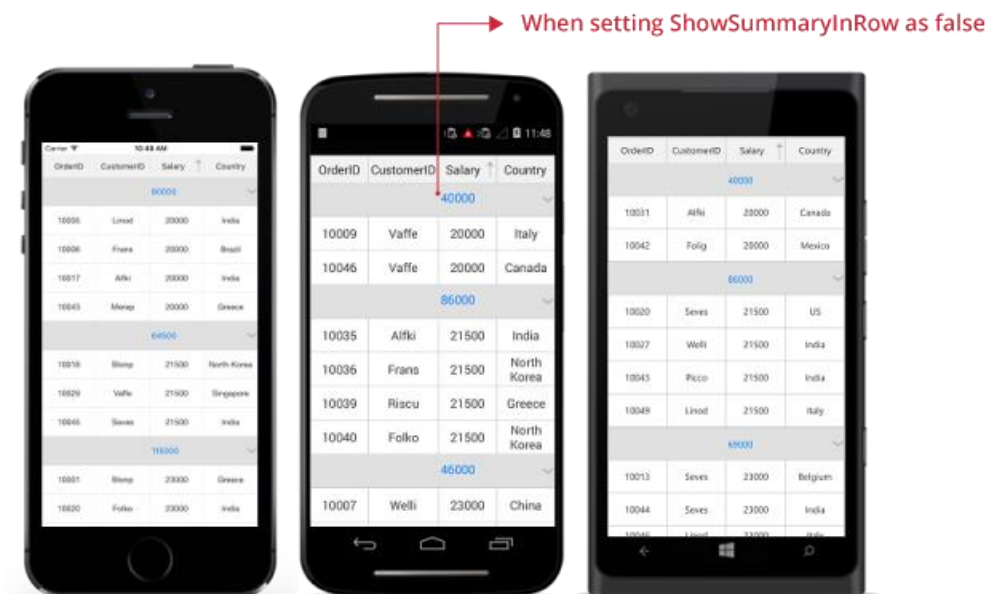
In the following code snippet, the `Format` property is defined to display sum of `Salary` by specifying the function name inside curly braces:

#### XML

```
<sfGrid:SfDataGrid.CaptionSummaryRow>
<sfGrid:GridSummaryRow ShowSummaryInRow="False">
<sfGrid:GridSummaryRow.SummaryColumns>
<sfGrid:GridSummaryColumn Name="CaptionSummary"
Format="{Sum}"
MappingName="Salary"
SummaryType="DoubleAggregate" />
</sfGrid:GridSummaryRow.SummaryColumns>
</sfGrid:GridSummaryRow>
</sfGrid:SfDataGrid.CaptionSummaryRow>
```

#### C#

```
GridSummaryRow summaryRow = new GridSummaryRow();
summaryRow.ShowSummaryInRow = false;
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "CaptionSummary",
    MappingName = "Salary",
    Format = "{Sum}",
    SummaryType = SummaryType.DoubleAggregate
});
sfGrid.CaptionSummaryRow= summaryRow;
```



### Formatting summary value

Format the summary value by setting the appropriate format after the aggregate function followed by a colon(:) in the `GridSummaryColumn.Format` property.

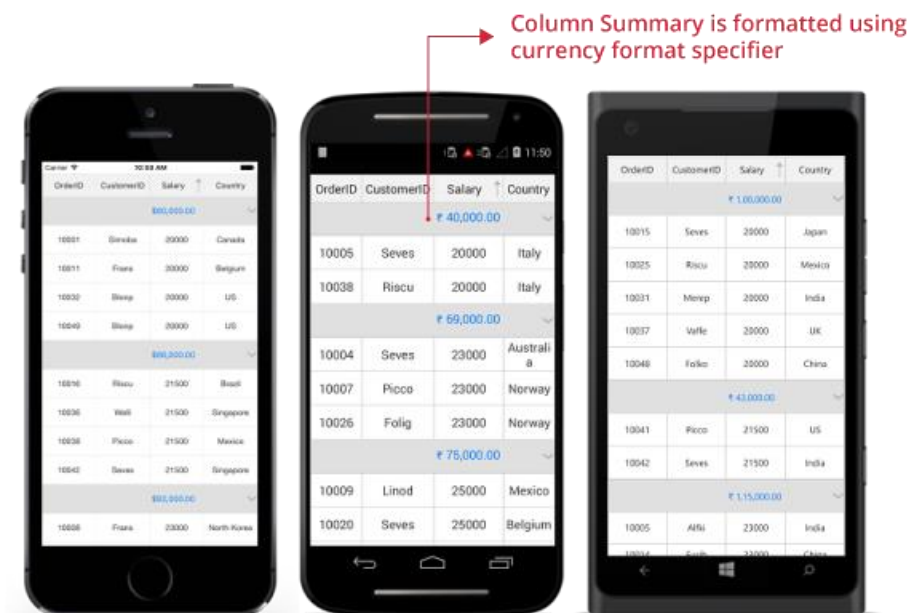
In the following code snippet Salary column summary is formatted using `c` format specifier. Refer to [here](#) to know about how to set different formats.

### XML

```
<sfGrid:SfDataGrid.CaptionSummaryRow>
  <sfGrid:GridSummaryRow ShowSummaryInRow="False">
    <sfGrid:GridSummaryRow.SummaryColumns>
      <sfGrid:GridSummaryColumn Name="CaptionSummary"
        Format="{Sum:c}"
        MappingName="Salary"
        SummaryType="DoubleAggregate" />
    </sfGrid:GridSummaryRow.SummaryColumns>
  </sfGrid:GridSummaryRow>
</sfGrid:SfDataGrid.CaptionSummaryRow>
```

### C#

```
GridSummaryRow summaryRow = new GridSummaryRow();
summaryRow.ShowSummaryInRow = false;
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "CaptionSummary",
    MappingName = "Salary",
    Format = "{Sum:c}",
    SummaryType = SummaryType.DoubleAggregate
});
sfGrid.CaptionSummaryRow= summaryRow;
```



### Displaying additional content in summary

Append additional content with summary value using the `GridSummaryColumn.Format` property.

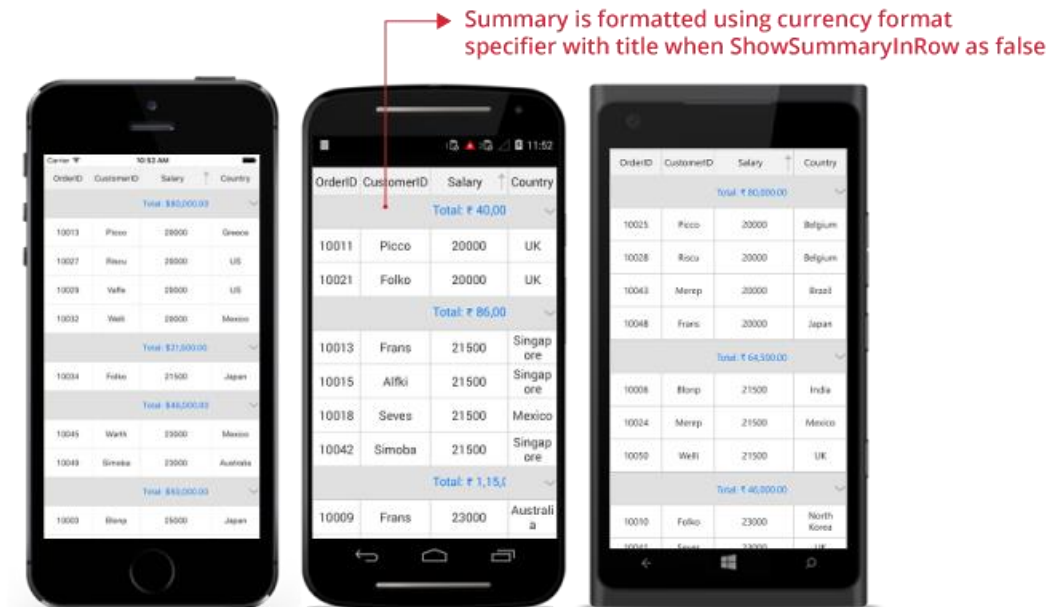
In the following code snippet `Total:` text is appended before summary value:

#### XML

```
<sfGrid:SfDataGrid.CaptionSummaryRow>
<sfGrid:GridSummaryRow ShowSummaryInRow="False">
<sfGrid:GridSummaryRow.SummaryColumns>
<sfGrid:GridSummaryColumn Name="CaptionSummary"
Format="Total: {Sum:c}"
MappingName="Salary"
SummaryType="DoubleAggregate" />
</sfGrid:GridSummaryRow.SummaryColumns>
</sfGrid:GridSummaryRow>
</sfGrid:SfDataGrid.CaptionSummaryRow>
```

#### C#

```
GridSummaryRow summaryRow = new GridSummaryRow();
summaryRow.ShowSummaryInRow = false;
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "CaptionSummary",
    MappingName = "Salary",
    Format = "Total:{Sum:c}",
    SummaryType = SummaryType.DoubleAggregate
});
sfGrid.CaptionSummaryRow= summaryRow;
```



### Formatting summary for a row using Title property

Format the summary value for a row using the [GridSummaryRow.Title](#) when `ShowSummaryInRow` set to `true`.

#### XML

```
<sfGrid:SfDataGrid.CaptionSummaryRow>
<sfGrid:GridSummaryRow Title="Total Salary:{TotalSalary} for {ProductCount}
members" ShowSummaryInRow="True">
<sfGrid:GridSummaryRow.SummaryColumns>
<sfGrid:GridSummaryColumn Name="TotalSalary"
Format="{0} {Sum:c}"
MappingName="Salary"
SummaryType="DoubleAggregate" />
<sfGrid:GridSummaryColumn Name="ProductCount"
Format="{0} {Count}"
MappingName="Salary"
SummaryType="CountAggregate" />
</sfGrid:GridSummaryRow.SummaryColumns>
</sfGrid:GridSummaryRow>
</sfGrid:SfDataGrid.CaptionSummaryRow>
```

#### C#

```
GridSummaryRow summaryRow = new GridSummaryRow();
summaryRow.Title = "Total Salary:{TotalSalary} for {ProductCount} members";
summaryRow.ShowSummaryInRow = true;
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "TotalSalary",
```

```

MappingName = "Salary",
Format = "{{Sum:c}}",
SummaryType = SummaryType.DoubleAggregate
});
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "ProductCount",
    MappingName = "Salary",
    Format = "{Count}",
    SummaryType = SummaryType.DoubleAggregate
});
sfGrid.CaptionSummaryRow= summaryRow;

```

Summary is formatted using currency format specifier with title when ShowSummaryInRow as true



**Note:** Setting the `SummaryColumn.Format` property to `{Sum:c}` throws an exception since the compiler treats it like how we set binding to a string, since the syntax is the same. But here we are trying to set the culture format for the string. Hence set the format as `Format = "{{Sum:c}}"` when setting the format in XAML.

### Aggregate types

Specify different summary aggregate types by using the [GridSummaryColumn.SummaryType](#) property, and use the built-in function in [GridSummaryColumn.Format](#).

List of predefined aggregate types and its built-in functions are as follows:

Aggregate Type	Built-in function
CountAggregate	Count
Int32Aggregate	Count, max, min, average, and sum.

DoubleAggregate	Count, max, min, average, and sum.
Custom	Used for custom summaries

**Note:** The above aggregate types can be applied for both [CaptionSummaries](#) and [TableSummaries](#).

#### Custom summaries

The data grid implements your own aggregate functions when the built-in aggregate functions do not meet your requirement.

Summary values can be calculated based on custom logic using the [GridSummaryColumn.CustomAggregate](#) property.

#### Implementing custom aggregate

1. Create a custom aggregate class by deriving from [ISummaryAggregate](#) interface.
2. In the `CalculateAggregateFunc()` method, you have to calculate the summary and assign it to the property.

In the following code snippet, **Standard Deviation** is calculated for quantity of products:

#### C#

```
public class CustomAggregate : ISummaryAggregate
{
    public CustomAggregate()
    {
    }
    public double StdDev { get; set; }
    public Action<System.Collections.IEnumerable, string,
    System.ComponentModel.PropertyDescriptor> CalculateAggregateFunc()
    {
        return (items, property, pd) =>
        {
            var enumerableItems = items as IEnumerable<OrderInfo>;
            if (pd.Name == "StdDev")
            {
                this.StdDev = enumerableItems.StdDev<OrderInfo>(q => q.OrderID);
            }
        };
    }
}

public static class LinqExtensions
{
    public static double StdDev<T>(this IEnumerable<T> values, Func<T, double?>
    selector)
    {
        double value = 0;
        var count = values.Count();
        if (count > 0)
        {
            double? avg = values.Average(selector);
            double sum = values.Select(selector).Sum(d =>
            {
```

```

if (d.HasValue)
{
    return Math.Pow(d.Value - avg.Value, 2);
}
return 0.0;
});
value = Math.Sqrt((sum) / (count - 1));
}
return value;
}
}

```

Assign the custom aggregate to `GridSummaryColumn.CustomAggregate` property and set the `SummaryType` as `Custom`. `GridSummaryColumn.Format` property is defined based on property name in custom aggregate `StdDev`.

### XML

```

<sfGrid:SfDataGrid.CaptionSummaryRow>
<sfGrid:GridSummaryRow Title="Standard Deviation:{CaptionSummary}"
ShowSummaryInRow = "True">
<sfGrid:GridSummaryRow.SummaryColumns>
<sfGrid:GridSummaryColumn Name="CaptionSummary"
CustomAggregate="{StaticResource customAggregate}"
Format="{}{StdDev}"
MappingName="OrderID"
SummaryType="Custom" />
</sfGrid:GridSummaryRow.SummaryColumns>
</sfGrid:GridSummaryRow>
</sfGrid:SfDataGrid.CaptionSummaryRow>

```

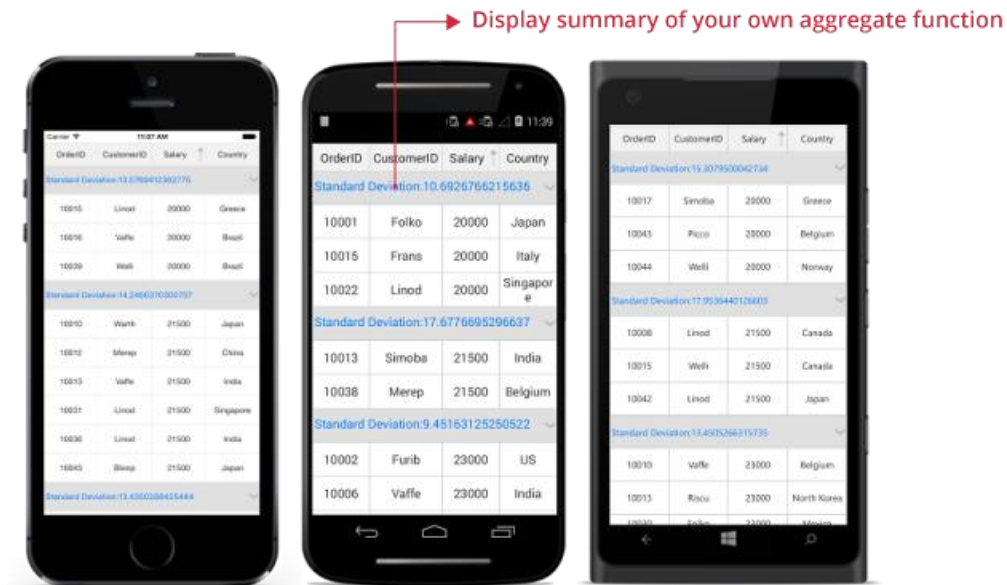
### C#

```

GridSummaryRow summaryRow = new GridSummaryRow();
summaryRow.Title = "Standard Deviation:{CaptionSummary}";
summaryRow.ShowSummaryInRow = true;
summaryRow.SummaryColumns.Add(new GridSummaryColumn
{
    Name = "CaptionSummary",
    CustomAggregate = new CustomAggregate(),
    MappingName = "OrderID",
    Format = "{StdDev}",
    SummaryType = Syncfusion.Data.SummaryType.Custom
});
dataGrid.CaptionSummaryRow = summaryRow;

```





**Note:** The above custom summaries section is explained using `CaptionSummary` but the custom summaries can be also applied for `TableSummaries`.

### Overriding summary renderer

Each summary cell in the data grid is associated with its own cell renderer. The data grid allows to extend this renderer to customize the grid cells based on your requirement. Customization can be applied by overriding the available virtual methods in the each cell renderer.

Each summary has a specific key using which the custom summary renderer can be registered to the [SfDataGrid.CellRenderers](#) collection. Remove the key from collection and add a new entry with the same key along with the instance of custom renderer to register.

Types of summary	Renderer	Key
Table summary	<a href="#">GridTableSummaryCellRenderer</a>	TableSummary
Caption summary	<a href="#">GridCaptionSummaryCellRenderer</a>	CaptionSummary
Group summary	<a href="#">GridSummaryCellRenderer</a>	GroupSummary

### Customizing table summary

The data grid allows customizing the table summary by extending the [GridTableSummaryCellRenderer](#) class.

**Note:** By default, `LoadUIView` property of `GridColumn` is `false` for android. Hence, `OnInitializeDisplayView()` will not be called.

To customize the table summary, follow the code example:

**C#**

```
// To remove default summary and Add custom summary.
public class Summary : ContentPage
{
    public Summary()
    {
        InitializeComponent();
        dataGrid.CellRenderers.Remove("TableSummary");
        dataGrid.CellRenderers.Add("TableSummary", new
        GridTableSummaryCellRendererExt());
    }
}

public class GridTableSummaryCellRendererExt : GridTableSummaryCellRenderer
{
    public GridTableSummaryCellRendererExt()
    {
    }

    public override void OnInitializeDisplayView(DataColumnBase dataColumn,
    SfLabel view)
    {
        base.OnInitializeDisplayView(dataColumn, view);
        view.HorizontalTextAlignment = TextAlignment.Center;
        view.BackgroundColor = Color.DarkCyan;
        view.FontSize = 16;
        view.TextColor = Color.White;
    }
}
```

The following screenshot shows the final outcome upon execution of the above code.



#### Customizing caption summary

The data grid customizes the caption summary by overriding the [GridCaptionSummaryCellRenderer](#).

---

**Note:** By default, `LoadUIView` property of `GridColumn` is `false` for android. Hence, `OnInitializeDisplayView()` will not be called.

---

### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AllowResizingColumn="True"
AutoGenerateColumns="False"
ColumnSizer="Star">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridColumn MappingName="OrderID" LoadUIView="True" />
<sfgrid:GridColumn MappingName="Salary" />
<sfgrid:GridColumn MappingName="CustomerID" />
<sfgrid:GridColumn MappingName="Freight" />
<sfgrid:GridColumn MappingName="Country" />
</sfgrid:SfDataGrid.Columns>
<sfgrid:SfDataGrid.GroupColumnDescriptions>
<sfgrid:GroupColumnDescription ColumnName="CustomerID" />
</sfgrid:SfDataGrid.GroupColumnDescriptions>
</sfgrid:SfDataGrid>
```

### C#

```
// To remove default summary and Add custom summary.
public class Summary : ContentPage
{
    public Summary()
    {
        InitializeComponent();
        dataGrid.CellRenderers.Remove("CaptionSummary");
        dataGrid.CellRenderers.Add("CaptionSummary", new
        GridCaptionSummaryCellRendererExt());
        dataGrid.GroupCaptionTextFormat = "{ColumnName} is Grouped by {Key} with
        Count- {ItemsCount}";
    }
}

public class GridCaptionSummaryCellRendererExt :
GridCaptionSummaryCellRenderer
{
    public GridCaptionSummaryCellRendererExt()
    {
    }

    public override void OnInitializeDisplayView(DataColumnBase dataColumn,
    SfLabel view)
    {
        base.OnInitializeDisplayView(dataColumn, view);
        view.HorizontalTextAlignment = TextAlignment.Center;
        view.BackgroundColor = Color.DarkCyan;
        view.FontAttributes = FontAttributes.Bold;
        view.FontSize = 16;
        view.TextColor = Color.White;
    }
}
```



### Customizing group summary

The data grid customizes the group summary by overriding the [GridGroupSummaryCellRenderer](#).

**Note:** By default, `LoadUIView` property of `GridColumn` is `false` for android. Hence, `OnInitializeDisplayView()` will not be called.

### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrdersInfo}">
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn MappingName="OrderID" LoadUIView="True" Width="85" />
<sfgrid:GridTextColumn MappingName="Salary" LoadUIView="True" Width="90"/>
<sfgrid:GridTextColumn MappingName="CustomerID" LoadUIView="True"
Width="150"/>
<sfgrid:GridTextColumn MappingName="Country" LoadUIView="True" Width="90"/>
</sfgrid:SfDataGrid.Columns>
<sfgrid:SfDataGrid.GroupColumnDescriptions>
<sfgrid:GroupColumnDescription ColumnName="Salary" />
</sfgrid:SfDataGrid.GroupColumnDescriptions>
</sfgrid:SfDataGrid>
```

### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        // To remove default summary and Add custom summary.
        dataGrid.CellRenderers.Remove("GroupSummary");
    }
}
```

```
dataGrid.CellRenderers.Add("GroupSummary", new
GridGroupSummaryCellRendererExt());
dataGrid.GroupSummaryRows.Add(new GridGroupSummaryRow()
{
    ShowSummaryInRow = true,
    Title = "Total Salary: {Salary} for {customerID} members",
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="Salary",
            MappingName="Salary",
            SummaryType=SummaryType.DoubleAggregate,
            Format="{Sum}"
        },
        new GridSummaryColumn()
        {
            Name="customerID",
            MappingName="customerID",
            Format="{Count}",
            SummaryType=SummaryType.CountAggregate
        }
    }
});
// Custom CellRenderer
public class GridGroupSummaryCellRendererExt : GridGroupSummaryCellRenderer
{
    public GridGroupSummaryCellRendererExt()
    {
    }
    public override void OnInitializeDisplayView(DataColumnBase dataColumn,
    SfLabel view)
    {
        base.OnInitializeDisplayView(dataColumn, view);
        base.OnInitializeDisplayView(dataColumn, view);
        view.HorizontalTextAlignment = TextAlignment.Center;
        view.BackgroundColor = Color.Gray;
        view.FontAttributes = FontAttributes.Italic;
        view.FontSize = 20;
        view.TextColor = Color.White;
    }
}
```

OrderID   Salary   ↑   CustomerID   Country			
<b>Salary : 10000 - 9 Items</b>			
10010	10000	Folko	China
10013	10000	Alfkis	Belgium
10017	10000	Folko	Norway
10021	10000	Foliges	Brazil
10030	10000	Vaffe	Netherland
10032	10000	Furib	Italy
10056	10000	Picco	US
10071	10000	Foliges	Singapore
10092	10000	Sim	Japan
<i>Total Salary: 90000 for 9 members</i>			
<b>Salary : 12000 - 8 Items</b>			
10018	12000	Seves	China
10040	12000	Riscu	Mexico
10060	12000	Alfkis	Greece
10067	12000	Furib	Brazil

You can download the sample demo [here](#) .

## Filtering

The SfDataGrid supports to view filtering.

### View Filtering

The SfDataGrid supports to filter the records in view by setting `SfDataGrid.View.Filter` property where `Filter` is a predicate.

In order to filter the records, assign the filtered strings to the `ViewModel.FilterText` property which will be later applied in `FilterPredicate` that is assigned to `SfDataGrid.View.Filter` in `OnFilterChanged()` method.

---

**Note:** To update filtering for newly added row or column, set the `SfDataGrid.View.LiveDataUpdateMode` to `LiveDataUpdateMode.AllowDataShaping`.

---

The following code example illustrates the delegate, properties, and methods used in the `ViewModel` class in order to perform filtering operation:

### C#

```
// ViewModel.cs
#region Filtering
#region Fields
private string filterText = "";
private string selectedColumn = "All Columns";
private string selectedCondition = "Equals";
internal delegate void FilterChanged();
internal FilterChanged filterTextChanged;
#endregion
#region Property
public string FilterText
{
    get { return filterText; }
    set
    {
        filterText = value;
        OnFilterTextChanged();
        RaisePropertyChanged("FilterText");
    }
}
public string SelectedCondition
{
    get { return selectedCondition; }
    set { selectedCondition = value; }
}
public string SelectedColumn
{
    get { return selectedColumn; }
    set { selectedColumn = value; }
}
#endregion
#region Private Methods
private void OnFilterTextChanged()
{
    filterTextChanged();
}
private bool MakeStringFilter(OrderInfo o, string option, string condition)
```

```
{
var value = o.GetType().GetProperty(option);
var exactValue = value.GetValue(o, null);
exactValue = exactValue.ToString().ToLower();
string text = FilterText.ToLower();
var methods = typeof(string).GetMethods();
if (methods.Count() != 0)
{
if (condition == "Contains")
{
var methodInfo = methods.FirstOrDefault(method => method.Name == condition);
bool result1 = (bool)methodInfo.Invoke(exactValue, new object[] { text });
return result1;
}
else if (exactValue.ToString() == text.ToString())
{
bool result1 = String.Equals(exactValue.ToString(), text.ToString());
if (condition == "Equals")
return result1;
else if (condition == "NotEquals")
return false;
}
else if (condition == "NotEquals")
{
return true;
}
return false;
}
else
return false;
}

private bool MakeNumericFilter(OrderInfo o, string option, string condition)
{
var value = o.GetType().GetProperty(option);
var exactValue = value.GetValue(o, null);
double res;
bool checkNumeric = double.TryParse(exactValue.ToString(), out res);
if (checkNumeric)
{
switch (condition)
{
case "Equals":
try
{
if (exactValue.ToString() == FilterText)
{
if (Convert.ToDouble(exactValue) == (Convert.ToDouble(FilterText)))
return true;
}
}
catch (Exception e)
{
Console.WriteLine(e);
}
break;
case "NotEquals":
try
```



```
{
    if (Convert.ToDouble(FilterText) != Convert.ToDouble(exactValue))
        return true;
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return true;
    }
    break;
}
return false;
}
#endregion
#region Public Methods
public bool FilterRecords(object o)
{
    double res;
    bool checkNumeric = double.TryParse(FilterText, out res);
    var item = o as OrderInfo;
    if (item != null && FilterText.Equals(""))
    {
        return true;
    }
    else
    {
        if (item != null)
        {
            if (checkNumeric && !SelectedColumn.Equals("All Columns"))
            {
                bool result = MakeNumericFilter(item, SelectedColumn, SelectedCondition);
                return result;
            }
            else if (SelectedColumn.Equals("All Columns"))
            {
                if (item.CustomerID.ToLower().Contains(FilterText.ToLower()) ||
                    item.Country.ToLower().Contains(FilterText.ToLower()) ||
                    item.Freight.ToString().ToLower().Contains(FilterText.ToLower()) ||
                    item.OrderID.ToString().ToLower().Contains(FilterText.ToLower()))
                {
                    return true;
                }
                return false;
            }
            else
            {
                bool result = MakeStringFilter(item, SelectedColumn, SelectedCondition);
                return result;
            }
        }
        return false;
    }
}
#endregion
#endregion
```

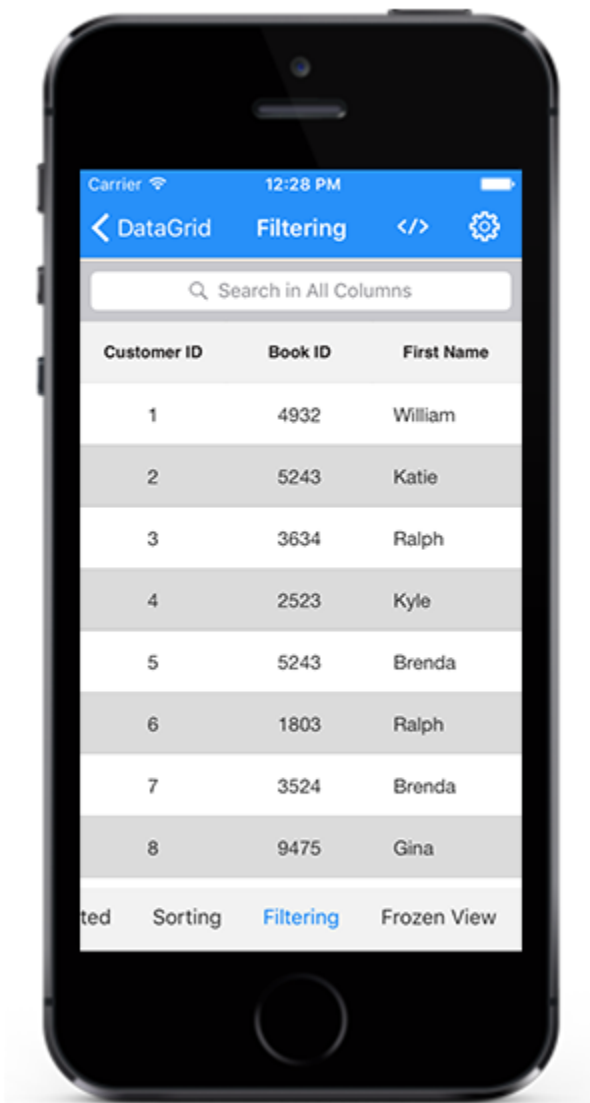
To create a `SearchBar` and apply the filtered records to `ViewModel.FilterText` property in `SearchBar.TextChanged` event, follow the code example:

#### XML

```
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<SearchBar x:Name="filterText"
Grid.Row="0"
Grid.Column="0"
IsVisible="true"
Placeholder="Search here to Filter"
TextChanged="OnFilterTextChanged" />
```

#### C#

```
private void OnFilterTextChanged(object sender, TextChangedEventArgs e)
{
    if (e.NewTextValue == null)
        viewModel.FilterText = "";
    else
        viewModel.FilterText = e.NewTextValue;
}
```



Once you create a `SearchBar` and a view model, filtering can be performed by setting `SfDataGrid.View.Filter` property. Call the `SfDataGrid.View.RefreshFilter()` method after setting the filtered records to the `SfDataGrid.View.Filter` property as in the following code example:

#### C#

```
// Code-Behind
viewModel.filterTextChanged = OnFilterChanged; //where 'filterTextChanged'
is a delegate declared in ViewModel class.
private void OnFilterChanged()
{
    if (dataGrid.View != null)
    {
        this.dataGrid.View.Filter = viewModel.FilerRecords;
        this.dataGrid.View.RefreshFilter();
    }
}
```

```
}

```

### DataTable filtering

To filter the rows in SfDataGrid using [DataView.RowFilter](#) expression, set the value of [SfDataGrid.CanUseViewFilter](#) property to **true**. The default filter which created in DataView can be applied or canceled through this property.

### Filter individual columns

To filter records in all the columns or in a particular column, use codes in OnColumnSelected() method.

For example, the records can be filtered in **OrderID** or any other particular column alone. The following code example illustrates how to create a **Picker** for columns and how the records will be filtered based on the column selected:

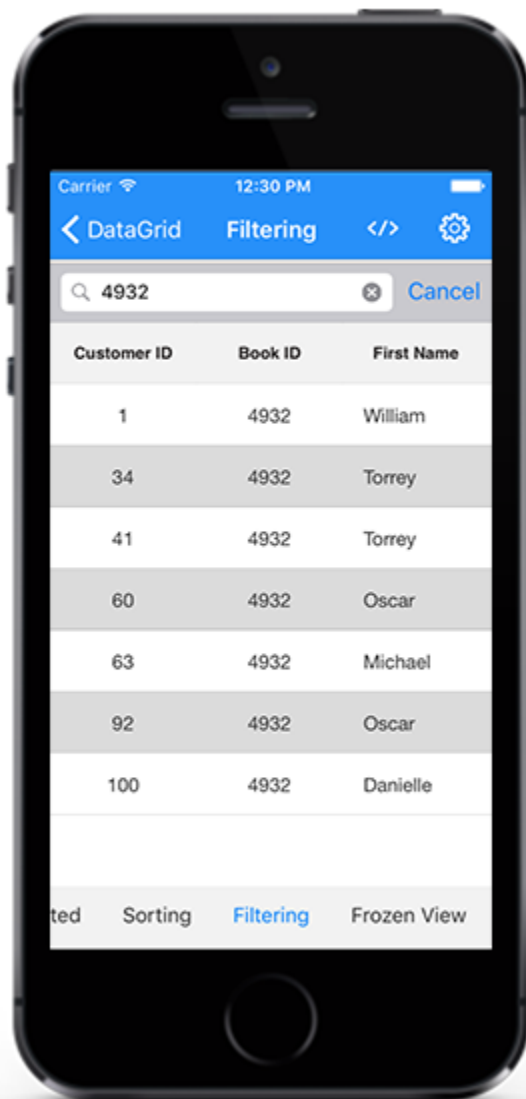
### XML

```
<Picker x:Name="ColumnsList"
HorizontalOptions="Start"
SelectedIndexChanged="OnColumnsSelectionChanged"
WidthRequest="200">
  <Picker.Items>
    <x:String>All Columns</x:String>
    <x:String>CustomerID</x:String>
    <x:String>BookID</x:String>
    <x:String>FirstName</x:String>
    <x:String>LastName</x:String>
    <x:String>BookName</x:String>
  </Picker.Items>
</Picker>
```

### C#

```
private void OnColumnsSelectionChanged(object sender, EventArgs e)
{
    Picker newPicker = (Picker)sender;
    viewModel.SelectedColumn = newPicker.Items[newPicker.SelectedIndex];
    if (viewModel.SelectedColumn == "All Columns")
    {
        viewModel.SelectedCondition = "Contains";
        OptionsList.IsVisible = false;
        this.OnFilterChanged();
    }
    else
    {
        OptionsList.IsVisible = true;
        foreach (var prop in typeof(OrderInfo).GetProperties())
        {
            // Records will be filtered based on selected column
            if (prop.Name == viewModel.SelectedColumn)
            {
                if (prop.PropertyType == typeof(string))
                {
                    OptionsList.Items.Clear();
                    OptionsList.Items.Add("Contains");
                    OptionsList.Items.Add("Equals");
                }
            }
        }
    }
}
```

```
OptionsList.Items.Add("NotEquals");
if (this.viewModel.SelectedCondition == "Equals")
OptionsList.SelectedIndex = 1;
else if (this.viewModel.SelectedCondition == "NotEquals")
OptionsList.SelectedIndex = 2;
else
OptionsList.SelectedIndex = 0;
}
else
{
OptionsList.Items.Clear();
OptionsList.Items.Add("Equals");
OptionsList.Items.Add("NotEquals");
if (this.viewModel.SelectedCondition == "Equals")
OptionsList.SelectedIndex = 0;
else
OptionsList.SelectedIndex = 1;
}
}
}
}
```



### Filter based on conditions

In addition to column based filtering, the records can be filtered based on some conditions. For example, the records can be filtered based on the given input or the records can be filtered contrast to the input. The condition based filtering can be achieved for all the columns or any particular column.

The records can be filtered in view based on any of the following conditions:

- Equals
- NotEquals
- Contains

The above conditions are the mostly used conditions. However any other conditions can be added based on the requirement and alter the following code example based on the condition.

**C#**

```
// ViewModel.cs
public bool FilterRecords(object o)
{
    double res;
    bool checkNumeric = double.TryParse(FilterText, out res);
    var item = o as OrderInfo;
    if (item != null && FilterText.Equals(""))
    {
        return true;
    }
    else
    {
        if (item != null)
        {
            if (checkNumeric && !SelectedColumn.Equals("All Columns"))
            {
                bool result = MakeNumericFilter(item, SelectedColumn, SelectedCondition);
                return result;
            }
            else if (SelectedColumn.Equals("All Columns"))
            {
                if (item.CustomerID.ToLower().Contains(FilterText.ToLower()) ||
                    item.Country.ToLower().Contains(FilterText.ToLower()) ||
                    item.Freight.ToString().ToLower().Contains(FilterText.ToLower()) ||
                    item.OrderID.ToString().ToLower().Contains(FilterText.ToLower()))
                {
                    return true;
                }
                return false;
            }
            else
            {
                bool result = MakeStringFilter(item, SelectedColumn, SelectedCondition);
                return result;
            }
        }
        return false;
    }
}

private bool MakeStringFilter(OrderInfo o, string option, string condition)
{
    var value = o.GetType().GetProperty(option);
    var exactValue = value.GetValue(o, null);
    exactValue = exactValue.ToString().ToLower();
    string text = FilterText.ToLower();
    var methods = typeof(string).GetMethods();
    if (methods.Count() != 0)
    {
        if (condition == "Contains")
        {
            var methodInfo = methods.FirstOrDefault(method => method.Name == condition);
            bool result1 = (bool)methodInfo.Invoke(exactValue, new object[] { text });
            return result1;
        }
        else if (exactValue.ToString() == text.ToString())
        {
            bool result1 = String.Equals(exactValue.ToString(), text.ToString());
            return result1;
        }
    }
}
```

```
if (condition == "Equals")
return result1;
else if (condition == "NotEquals")
return false;
}
else if (condition == "NotEquals")
{
return true;
}
return false;
}
else
return false;
}
private bool MakeNumericFilter(OrderInfo o, string option, string condition)
{
var value = o.GetType().GetProperty(option);
var exactValue = value.GetValue(o, null);
double res;
bool checkNumeric = double.TryParse(exactValue.ToString(), out res);
if (checkNumeric)
{
switch (condition)
{
case "Equals":
try
{
if (exactValue.ToString() == FilterText)
{
if (Convert.ToDouble(exactValue) == (Convert.ToDouble(FilterText)))
return true;
}
}
catch (Exception e)
{
Console.WriteLine(e);
}
break;
case "NotEquals":
try
{
if (Convert.ToDouble(FilterText) != Convert.ToDouble(exactValue))
return true;
}
catch (Exception e)
{
Console.WriteLine(e);
return true;
}
break;
}
}
return false;
}
```



The following code example illustrates how to create a **Picker** for conditions and add appropriate strings to that **Picker** and how the records will be filtered based on selected conditions:

#### XML

```
<Picker x:Name="OptionsList"
HorizontalOptions="Start"
IsVisible="False"
SelectedIndexChanged="OnFilterOptionsChanged"
WidthRequest="200">
  <Picker.Items>
    <x:String>Equals</x:String>
    <x:String>NotEquals</x:String>
    <x:String>Contains</x:String>
  </Picker.Items>
</Picker>
```

#### C#

```
private void OnFilterOptionsChanged(object sender, EventArgs e)
{
    Picker newPicker = (Picker)sender;
    if (newPicker.SelectedIndex >= 0)
    {
        viewModel.SelectedCondition = newPicker.Items[newPicker.SelectedIndex];
        if (filterText.Text != null)
            this.OnFilterChanged();
    }
}
```

Screenshot//

The following code example illustrates the complete **FilteringUI** and its operations:

#### XML

```
<StackLayout HorizontalOptions="FillAndExpand"
Orientation="Vertical"
VerticalOptions="FillAndExpand">
  <Grid HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <SearchBar x:Name="filterText"
Grid.Row="0"
Grid.Column="0"
IsVisible="true"
Placeholder="Search here to Filter"
TextChanged="OnFilterTextChanged" />
    <syncfusion:SfDataGrid x:Name="dataGrid"
Grid.Row="1"
Grid.Column="0"
```

```

AutoGenerateColumns="false"
ColumnSizer="Star"
HorizontalOptions="FillAndExpand"
SelectionMode="Single"
VerticalOptions="FillAndExpand">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID" />
<syncfusion:GridTextColumn MappingName="CustomerID" />
<syncfusion:GridTextColumn MappingName="Freight" />
<syncfusion:GridTextColumn MappingName="Country" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
</Grid>
<Grid ColumnSpacing="10" Padding="20">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Label x:Name="filterlabel"
Grid.Row="0"
Grid.Column="0"
Grid.ColumnSpan="2"
FontSize="15"
HorizontalOptions="Start"
Text="Filter Options"
VerticalOptions="Center" />
<Picker x:Name="ColumnsList"
Grid.Row="1"
Grid.Column="0"
HorizontalOptions="Start"
SelectedIndexChanged="OnColumnsSelectionChanged"
WidthRequest="200">
<Picker.Items>
<x:String>All Columns</x:String>
<x:String>CustomerID</x:String>
<x:String>BookID</x:String>
<x:String>FirstName</x:String>
<x:String>LastName</x:String>
<x:String>BookName</x:String>
</Picker.Items>
</Picker>
<Picker x:Name="OptionsList"
Grid.Row="1"
Grid.Column="1"
HorizontalOptions="Start"
IsVisible="False"
SelectedIndexChanged="OnFilterOptionsChanged"
WidthRequest="200">
<Picker.Items>
<x:String>Equals</x:String>
<x:String>NotEquals</x:String>
<x:String>Contains</x:String>
</Picker.Items>

```

```
</Picker>
</Grid>
</StackLayout>
```

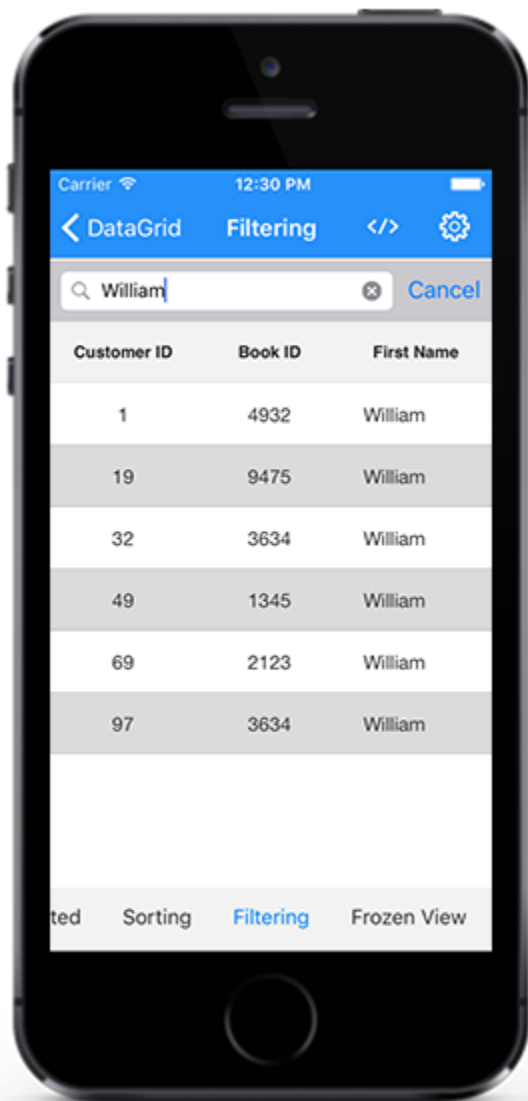
## C#

```
this.dataGrid.ItemsSource = viewModel.OrdersInfo;
viewModel.filterTextChanged = OnFilterChanged;
ColumnsList.SelectedIndex = 0;
private void OnColumnsSelectionChanged(object sender, EventArgs e)
{
    Picker newPicker = (Picker)sender;
    viewModel.SelectedColumn = newPicker.Items[newPicker.SelectedIndex];
    if (viewModel.SelectedColumn == "All Columns")
    {
        viewModel.SelectedCondition = "Contains";
        OptionsList.IsVisible = false;
        this.OnFilterChanged();
    }
    else
    {
        OptionsList.IsVisible = true;
        foreach (var prop in typeof(OrderInfo).GetProperties())
        {
            if (prop.Name == viewModel.SelectedColumn)
            {
                if (prop.PropertyType == typeof(string))
                {
                    OptionsList.Items.Clear();
                    OptionsList.Items.Add("Contains");
                    OptionsList.Items.Add("Equals");
                    OptionsList.Items.Add("NotEquals");
                    if (this.viewModel.SelectedCondition == "Equals")
                        OptionsList.SelectedIndex = 1;
                    else if (this.viewModel.SelectedCondition == "NotEquals")
                        OptionsList.SelectedIndex = 2;
                    else
                        OptionsList.SelectedIndex = 0;
                }
                else
                {
                    OptionsList.Items.Clear();
                    OptionsList.Items.Add("Equals");
                    OptionsList.Items.Add("NotEquals");
                    if (this.viewModel.SelectedCondition == "Equals")
                        OptionsList.SelectedIndex = 0;
                    else
                        OptionsList.SelectedIndex = 1;
                }
            }
        }
    }
}
private void OnFilterOptionsChanged(object sender, EventArgs e)
{
    Picker newPicker = (Picker)sender;
```

```
if (newPicker.SelectedIndex >= 0)
{
    viewModel.SelectedCondition = newPicker.Items[newPicker.SelectedIndex];
    if (filterText.Text != null)
    this.OnFilterChanged();
}
}

private void OnFilterTextChanged(object sender, TextChangedEventArgs e)
{
    if (e.NewTextValue == null)
        viewModel.FilterText = "";
    else
        viewModel.FilterText = e.NewTextValue;
}

private void OnFilterChanged()
{
    if (dataGrid.View != null)
    {
        this.dataGrid.View.Filter = viewModel.FilerRecords;
        this.dataGrid.View.RefreshFilter();
    }
}
```



### Clear filtering

The SfDataGrid allows to clear the applied filtering by setting the `SfDataGrid.View.Filter` property to `null`.

To clear the applied filtering, follow the code example:

#### C#

```
// Code-Behind
private void OnFilterChanged()
{
    if (dataGrid.View != null)
    {
        this.dataGrid.View.Filter = null;
        this.dataGrid.View.RefreshFilter();
    }
}
```

```
}
```

You can download the filtering demo [here](#).

## Editing

The SfDataGrid supports for editing the cell values by setting the [SfDataGrid.AllowEditing](#) property, [SfDataGrid.NavigationMode](#) as Cell and setting the [SfDataGrid.SelectionMode](#) as any other than None.

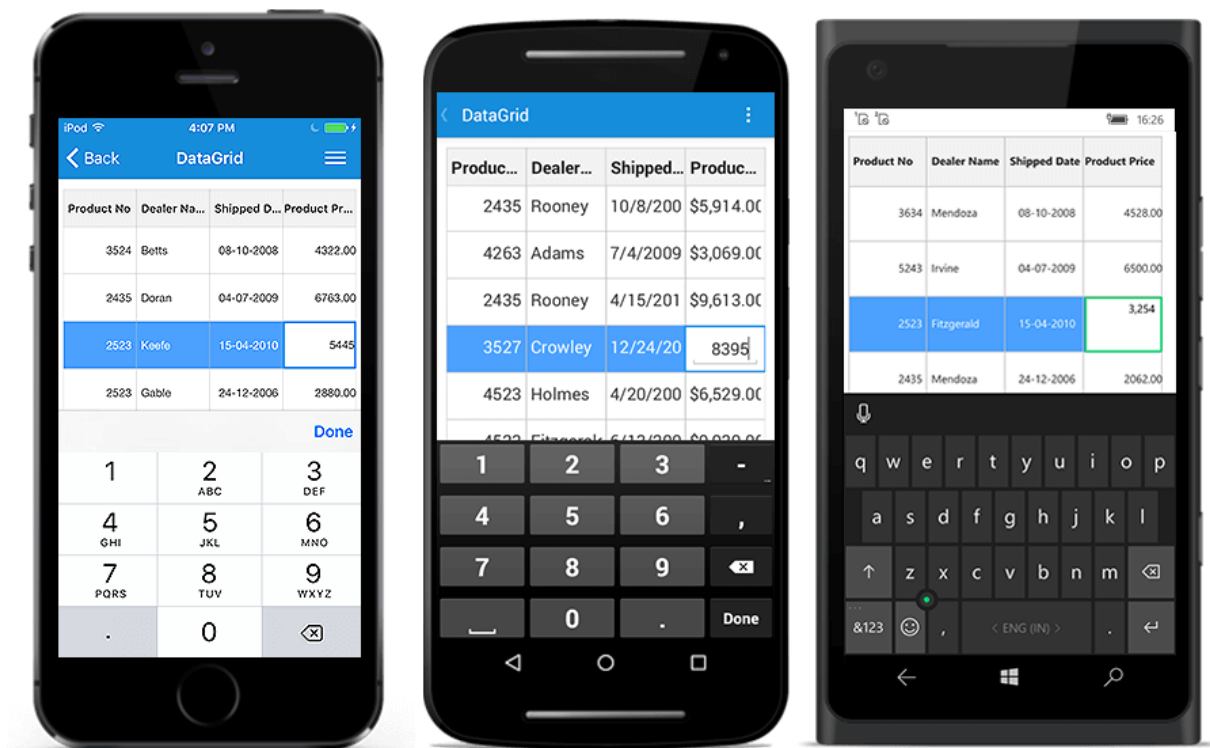
To enable editing, follow the code example:

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowEditing="True"
    SelectionMode="Multiple"
    NavigationMode="Cell"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}" />
```

### C#

```
dataGrid.AllowEditing = true;
dataGrid.SelectionMode = SelectionMode.Multiple;
dataGrid.NavigationMode = NavigationMode.Cell;
```



## Column editing

To enable or disable editing for a particular column, set the [GridColumn.AllowEditing](#) property.

## XML

```
<syncfusion:GridTextColumn AllowEditing="True" MappingName="OrderID" />
```

## C#

```
GridTextColumn column = new GridTextColumn();  
column.MappingName="OrderID";  
column.AllowEditing = false;
```

**Note:** The `GridColumn.AllowEditing` takes higher priority than the `SfDataGrid.AllowEditing`.

### Entering into edit mode

To enter into edit mode by just tapping or double tapping the grid cells, set the [SfDataGrid.EditTapAction](#) property.

## XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"  
AllowEditing="True"  
AutoGenerateColumns="True"  
EditTapAction="OnTap">
```

## C#

```
//Enter edit mode in single tap  
this.dataGrid.EditTapAction = TapAction.OnTap;  
//Enter edit mode in double tap  
this.dataGrid.EditTapAction = TapAction.OnDoubleTap;
```

**Note:** The keyboard will be collapsed when editing grid cell gets unfocused.

### Cursor behavior

When the cell enters into edit mode, cursor is placed based on the [SfDataGrid.EditorSelectionBehavior](#) property.

- **SelectAll:** Selects the text of edit element loaded inside cell.
- **MoveLast:** Places the cursor to the end of edit element loaded inside cell.

## XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"  
AllowEditing="True"  
AutoGenerateColumns="True"  
EditTapAction="OnTap"  
EditorSelectionBehavior="SelectAll">
```

## C#

```
//Selects all the text in the edit mode  
this.dataGrid.EditorSelectionBehavior = EditorSelectionBehavior.SelectAll;  
//Places the cursor at the last  
this.dataGrid.EditorSelectionBehavior = EditorSelectionBehavior.MoveLast;
```

---

**Note:** Editing supports for GridTemplateColumn and GridUnboundColumn are not provided yet.

### Support for IEditableObject

The SfDataGrid supports to commit and roll back the changes in row level when underlying data object implements the [IEditableObject](#) interface.

The editing changes in a row will be committed only when tapping on next row.

The [IEditableObject](#) has the following methods to capture editing:

- [BeginEdit](#): Gets called to begin edit on underlying data object when cells in a row enters into edit mode.
- [CancelEdit](#): Gets called when you cancel editing to discard the changes in a row since last BeginEdit call.
- [EndEdit](#): Gets called when you move to the next row by tapping to commit changes in underlying data object since last BeginEdit call.

The following code snippet explains the simple implementation of [IEditableObject](#):

### C#

```
public class OrderInfo : INotifyPropertyChanged, IEditableObject
{
    public OrderInfo()
    {
    }
    #region private variables
    private int _orderID;
    private int _employeeID;
    private int _customerID;
    private bool _isClosed;
    private string _firstName;
    private string _lastName;
    private string _gender;
    private string _shipCity;
    private string _shipCountry;
    private string _freight;
    private DateTime _shippingDate;
    #endregion
    #region Public Properties
    public int OrderID
    {
        get { return _orderID; }
        set
        {
            this._orderID = value;
            RaisePropertyChanged("OrderID");
        }
    }
    public int EmployeeID
    {
        get { return _employeeID; }
        set
        {

```



```
this._employeeID = value;
RaisePropertyChanged("EmployeeID");
}
}
public int CustomerID
{
    get { return _customerID; }
    set
    {
        this._customerID = value;
        RaisePropertyChanged("CustomerID");
    }
}
public bool IsClosed
{
    get { return _isClosed; }
    set
    {
        this._isClosed = value;
        RaisePropertyChanged("IsClosed");
    }
}
public string FirstName
{
    get { return _firstName; }
    set
    {
        this._firstName = value;
        RaisePropertyChanged("FirstName");
    }
}
public string LastName
{
    get { return _lastName; }
    set
    {
        this._lastName = value;
        RaisePropertyChanged("LastName");
    }
}
public string Gender
{
    get { return _gender; }
    set
    {
        this._gender = value;
        RaisePropertyChanged("Gender");
    }
}
public string ShipCity
{
    get { return _shipCity; }
    set
    {
        this._shipCity = value;
        RaisePropertyChanged("ShipCity");
    }
}
```

```

}
public string ShipCountry
{
    get { return _shipCountry; }
    set
    {
        this._shipCountry = value;
        RaisePropertyChanged("ShipCountry");
    }
}
public string Freight
{
    get { return _freight; }
    set
    {
        this._freight = value;
        RaisePropertyChanged("Freight");
    }
}
public DateTime ShippingDate
{
    get { return _shippingDate; }
    set
    {
        this._shippingDate = value;
        RaisePropertyChanged("ShippingDate");
    }
}
#endregion
#region INotifyPropertyChanged implementation
public event PropertyChangedEventHandler PropertyChanged;
private void RaisePropertyChanged (String Name)
{
    if (PropertyChanged != null)
        this.PropertyChanged (this, new PropertyChangedEventArgs (Name));
}
private Dictionary<string, object> storedValues;
public void BeginEdit()
{
    this.storedValues = this.BackUp();
}
public void CancelEdit()
{
    if (this.storedValues == null)
        return;
    foreach (var item in this.storedValues)
    {
        var itemProperties = this.GetType().GetTypeInfo().DeclaredProperties;
        var pDesc = itemProperties.FirstOrDefault(p => p.Name == item.Key);
        if (pDesc != null)
            pDesc.SetValue(this, item.Value);
    }
}
public void EndEdit()
{
    if (this.storedValues != null)
    {

```

```

this.storedValues.Clear();
this.storedValues = null;
}
Debug.WriteLine("End Edit Called");
}
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;
}
#endregion
}

```

### Editing events

The SfDataGrid triggers the following events while editing:

#### *CurrentCellBeginEdit*

The [SfDataGrid.CurrentCellBeginEdit](#) event occurs when the CurrentCell enters into edit mode. The [GridCurrentCellBeginEditEventArgs](#) has the following members which provides information for SfDataGrid.CurrentCellBeginEdit event:

- [Cancel](#): When this member set to 'true', the event is canceled and the CurrentCell does not enter into the edit mode.
- [RowColumnIndex](#): Gets the current row and column index of the DataGrid.
- [Column](#): Gets the Grid Column of the SfDataGrid.

To hook the [SfDataGrid.CurrentCellBeginEdit](#) event, follow the code example:

#### **C#**

```

this.dataGrid.CurrentCellBeginEdit += DataGrid_CurrentCellBeginEdit;
private void DataGrid_CurrentCellBeginEdit(object sender,
GridCurrentCellBeginEditEventArgs args)
{
// Editing prevented for the cell at RowColumnIndex(2,2).
if (args.RowColumnIndex == new
Syncfusion.GridCommon.ScrollAxis.RowColumnIndex(2, 2))
args.Cancel = true;
}

```

#### *CurrentCellEndEdit*

The [CurrentCellEndEdit](#) event occurs when the CurrentCell exits the edit mode. The [GridCurrentCellEndEditEventArgs](#) has following members which provides information for SfDataGrid.CurrentCellEndEdit event:

- [RowColumnIndex](#): Gets the current row and column index of the DataGrid.

- [Cancel](#): When this member set to 'true', the event is canceled and the edited value is not committed in the underlying collection.

To hook the `SfDataGrid.CurrentCellEndEdit` event, follow the code example:

#### C#

```
this.dataGrid.CurrentCellEndEdit += DataGrid_CurrentCellEndEdit;
private void DataGrid_CurrentCellEndEdit(object sender,
GridCurrentCellEndEditEventArgs args)
{
    // Editing prevented for the cell at RowColumnIndex(1,3).
    if (args.RowColumnIndex == new
Syncfusion.GridCommon.ScrollAxis.RowColumnIndex(1, 3))
    args.Cancel = true;
}
```

Programmatically edit a cell

#### Begin editing

The SfDataGrid allows to edit the cell programmatically by calling the [SfDataGrid.BeginEdit](#) method. By calling this method, the particular cell enters into edit mode. It edits the data manually or programmatically. To edit a cell programmatically, follow the code example:

#### C#

```
this.dataGrid.Loaded += dataGrid_Loaded;
void dataGrid_Loaded(object sender, RoutedEventArgs e)
{
    //Edit the cell at 2nd row,2nd column programmatically
    this.dataGrid.BeginEdit(2, 2);
}
```

#### End editing

The [SfDataGrid.EndEdit](#) method can be called to programmatically end editing. A cell that is currently in edit mode commits the edited value to the underlying collection and exits the edit mode when this method is called. To end the editing programmatically, follow the code example:

#### C#

```
this.dataGrid.EndEdit();
```

#### Cancel editing

The [SfDataGrid.CancelEdit](#) method can be called to programmatically cancel editing. A cell that is currently in edit mode exits the edit mode without committing the edited value in the underlying collection when this method is called. To cancel the editing programmatically, follow the code example:

#### C#

```
this.dataGrid.CancelEdit();
```

How to

### Cancel editing

The [SfDataGrid.CurrentCellBeginEdit](#) event can be used to cancel the editing operation for the corresponding cell. To cancel the editing operation using the [SfDataGrid.CurrentCellBeginEdit](#) event, follow the code example:

#### C#

```
this.dataGrid.CurrentCellBeginEdit += DataGrid_CurrentCellBeginEdit;
private void DataGrid_CurrentCellBeginEdit(object sender,
GridCurrentCellBeginEditEventArgs args)
{
    if (args.Column.MappingName == "OrderID" || args.RowColumnIndex.RowIndex == 2)
        args.Cancel = true;
}
```

### Cancel edited value from getting committed

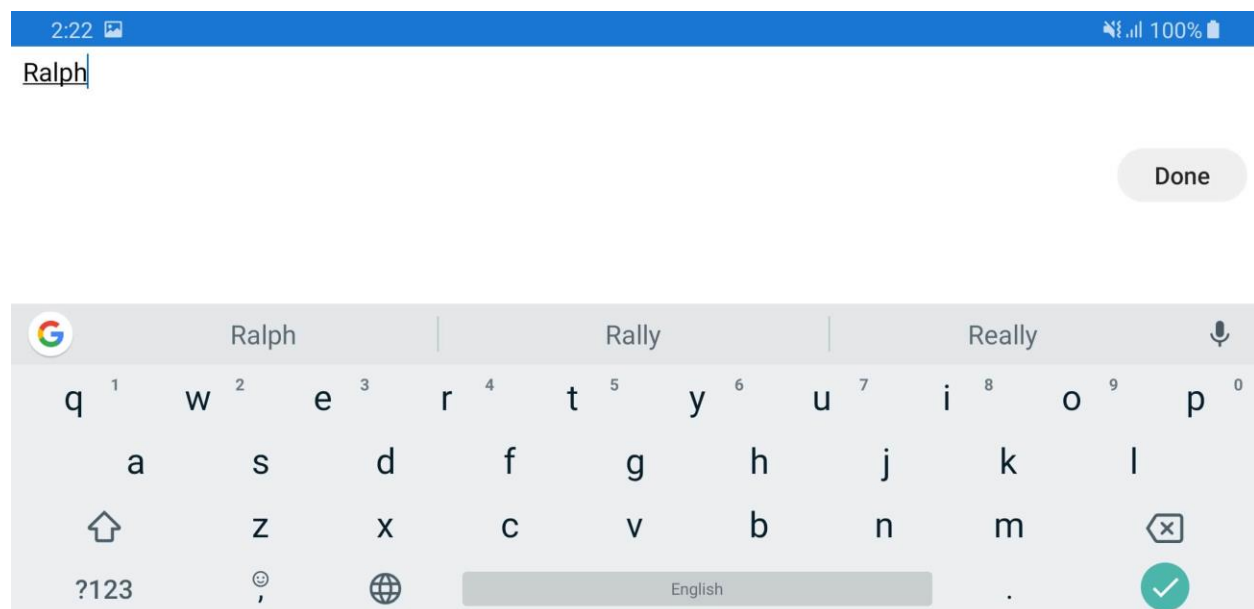
To prevent the edited value from getting committed, use the [CurrentCellEndEdit](#) event. To prevent the edited values from getting committed in the underlying collection, follow the code example:

#### C#

```
this.dataGrid.CurrentCellEndEdit += DataGrid_CurrentCellEndEdit;
private void DataGrid_CurrentCellEndEdit(object sender,
GridCurrentCellEndEditEventArgs args)
{
    if (args.RowColumnIndex.RowIndex == 2)
        args.Cancel == true;
}
```

### Customize editor UI in landscape mode

By default in Android platform, when entering edit mode for a text column in landscape orientation, the editor view loads in full width and height of the screen like below.



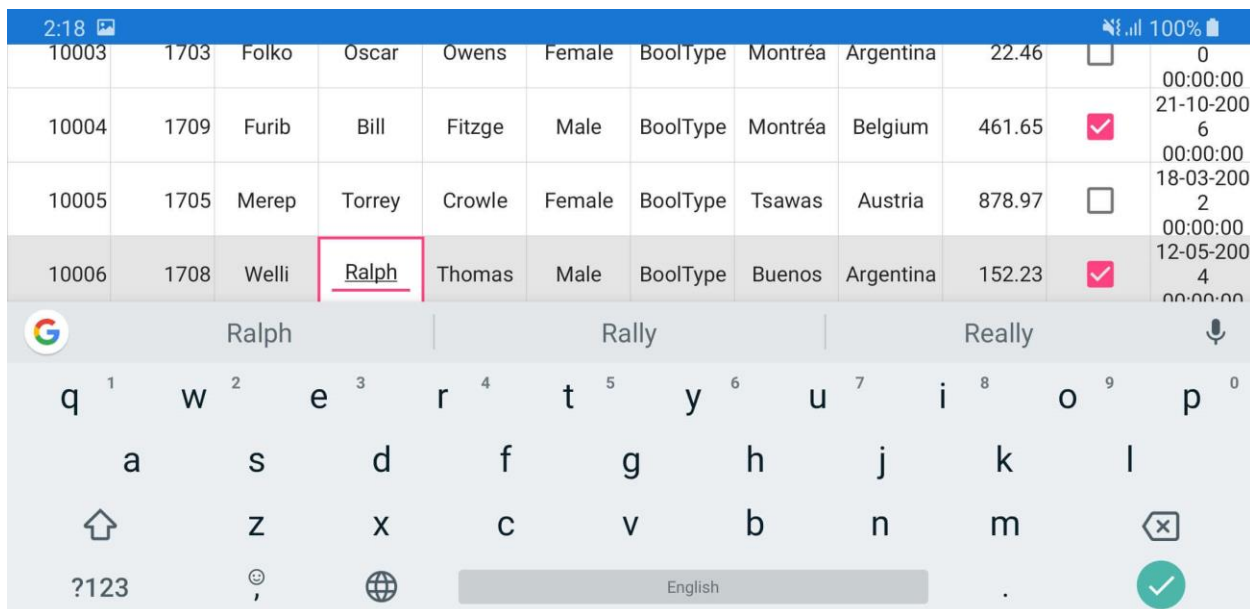
If in case when entering edit mode you want to load the editor just like in portrait orientation, where the editor is loaded within the cells and the grid rows are visible in the background, set the [SfDataGrid.ImeOptions](#) property as [GridImeOptions.NoExtractUi](#). The default value of [SfDataGrid.ImeOptions](#) is [GridImeOptions.Done](#).

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo,Mode=TwoWay}"
AllowEditing="True"
NavigationMode="Cell"
SelectionMode="Single"
ImeOptions="NoExtractUi">
</syncfusion:SfDataGrid>
```

### C#

```
this.dataGrid.ImeOptions = GridImeOptions.NoExtractUi;
```



### Selection

This section explains how to enable selection in the data grid; modes, properties, and events involved in selection; and customizations available for selection.

The data grid allows you to select a specific row or group of rows either programmatically or by touch interactions. To enable selection, set the [SfDataGrid.SelectionMode](#) property to a value other than [None](#). This control has different selection modes to perform the selection operation as follows:

#### Selection modes

Modes	Description
<a href="#">None</a>	Disables selection, and no rows can be selected. This is the default value.

<a href="#">Single</a>	Allows selection of a single row only. Upon selecting the next row, the selection in the previous row is cleared.
<a href="#">Multiple</a>	Allows selection of more than one row. Selection is not cleared when selecting more than one record. When you click on a selected row for the second time, the selection is cleared.
<a href="#">SingleDeselect</a>	Allows selection of only a single row. However, upon tapping the row again, the selection is cleared. Similar to single mode, upon selecting the next row, the selection in the previous row is cleared.

To set the selection mode, follow the code example:

#### **XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ColumnSizer="Star"
AutoGenerateColumns="True"
SelectionMode="Multiple"
ItemsSource="{Binding OrdersInfo}"/>
```

#### **C#**

```
dataGrid.SelectionMode = SelectionMode.Multiple;
```

OrderID	Genders	EmployeeID	CustomerID
10001	Male	10501	Linod
10002	Female	10502	Furib
10003	Male	10503	Folig
10004	Female	10504	Frans
10005	Male	10505	Picco
10006	Male	10506	Warth
10007	Female	10507	Riscu
10008	Female	10508	Seves
10009	Male	10509	Linod
10010	Male	10510	Folko
10011	Female	10511	Vaffe
10012	Male	10512	Welli
10013	Female	10513	Vaffe
10014	Female	10514	Alfki

#### Getting selected rows

The SfDataGrid provides `SelectedIndex`, `SelectedItem`, and `CurrentItem` properties to get details of the selected rows when the selection mode is `Single`, `Multiple`, and `SingleDeselect`.



- [SfDataGrid.SelectedItem](#): Provides the underlying data object of the selected row. Denotes the first selected row in multiple selection.
- [SfDataGrid.SelectedIndex](#): Provides the index of [SfDataGrid.SelectedItem](#).
- [SfDataGrid.CurrentItem](#): Provides the underlying data object of the currently selected row in the data grid. Denotes the first selected row in multiple selection.

#### Row selection

When multiple rows are selected, the [SelectedItems](#) and [SelectionController.SelectedRows](#) properties provide information of all the selected rows.

- [SfDataGrid.SelectedItems](#): Provides all the selected records of the selected items when multiple selection is enabled.
- [SfDataGrid.SelectionController.SelectedRows](#): Provides collection of the underlying model object(row data) of all the selected items.

#### CurrentItem vs SelectedItem

Both the [SelectedItem](#) and [CurrentItem](#) returns the same data object when the selection mode is single. When multiple selection is enabled, the initially selected row will be maintained in the [SelectedItem](#) and the currently selected row will be maintained in the [CurrentItem](#).

#### Programmatic selection

When [SfDataGrid.SelectionMode](#) is set a value other than [None](#), select row/rows from the code by setting the [SfDataGrid.SelectedIndex](#), [SfDataGrid.SelectedItem](#), or [SfDataGrid.SelectedItems](#) property based on the selection mode. To enable selection from code, follow the code example:

When the selection mode is [Single](#), programmatically select a row in two ways either by setting the row index to the [SfDataGrid.SelectedIndex](#) property, or by setting the underlying object to be selected to the [SfDataGrid.SelectedItem](#) property.

To programmatically select a row from the code, follow the code example:

#### C#

```
//Perform selection using selected index
dataGrid.SelectedIndex = 3;
//Perform selection using selected item
dataGrid.SelectedItem = viewModel.OrdersInfo [5];
```

When the selection mode is multiple, programmatically select more than one row by adding the underlying object to be selected to the [SfDataGrid.SelectedItems](#) property.

To programmatically select more than one row from the code, follow the code example:

#### C#

```
//Perform multiple selection using selected item
dataGrid.SelectedItems.Add (viewModel.OrdersInfo [4]);
dataGrid.SelectedItems.Add (viewModel.OrdersInfo [6]);
dataGrid.SelectedItems.Add (viewModel.OrdersInfo [9]);
dataGrid.SelectedItems.Add (viewModel.OrdersInfo [11]);
```

The following screenshot shows the selection functionality in the data grid:

OrderID	Genders	EmployeeID	CustomerID
10001	Male	10501	Linod
10002	Female	10502	Furib
10003	Male	10503	Folig
10004	Female	10504	Frans
10005	Male	10505	Picco
10006	Male	10506	Warth
10007	Female	10507	Riscu
10008	Female	10508	Seves
10009	Male	10509	Linod
10010	Male	10510	Folko
10011	Female	10511	Vaffe
10012	Male	10512	Welli
10013	Female	10513	Vaffe
10014	Female	10514	Alfki

### Keyboard behavior

**SfDataGrid** supports selection via keyboard interaction for the Xamarin.Forms.macOS and Xamarin.Forms.UWP platforms. Keyboard interaction will not have any effect when the **SfDataGrid.SelectionMode** is set as **SelectionMode.None**.

Key or key combinations	Description
DownArrow	Moves selection to the next row directly below the currently selected row when the SelectionMode is Single. Upon reaching the bottom of screen, pressing down arrow scrolls and applies selection to the next row if grid has scrollable content. If the selected row is the last row of grid, pressing down arrow does nothing. In <b>Multiple</b> and <b>SingleDeselect</b> selection modes, if the next row has already been selected, pressing down arrow key will deselect the row.
UpArrow	Moves selection to the previous row directly above the currently selected row when the SelectionMode is Single. Upon reaching the top of screen, pressing up arrow scrolls and applies selection to the previous row if grid has scrollable content. If the selected row is the first row of grid, pressing up arrow does nothing. In <b>Multiple</b> and <b>SingleDeselect</b> selection modes, if the previous row has already been selected, pressing up arrow key will deselect the row.
PageDown	SfDataGrid will be scrolled to the next set of rows that are not displayed in view including the partially displayed row. If selection is applied to any row, pressing PageDown arrow will move selection to the last row of the next set of rows when the SelectionMode is Single.
PageUp	SfDataGrid will be scrolled to the previous set of rows that are not displayed in view including the partially displayed row. If the selection is applied to any row, pressing PageUp will move selection to the first row of the next set of rows when the SelectionMode is Single.
Tab	Moves selection to the next row directly below the currently selected row when the SelectionMode is Single. Upon reaching the bottom of screen, pressing Tab key scrolls and applies selection to the next row if grid has scrollable content. If the selected row is the last row of grid, pressing Tab key does nothing. In <b>Multiple</b> and <b>SingleDeselect</b> selection modes, if the next row has already been selected, pressing Tab key will deselect the row.
Ctrl + Home or Ctrl + UpArrow or Home	Scrolls grid to the first row of the collection.
Ctrl + End or Ctrl + DownArrow or End	Scrolls grid to the last row of the collection.

Enter or Ctrl + Enter	If the active current cell is in edit mode, changes will be committed and selection will be moved to the row below the active current cell. If the active current cell is in last row, commits changes and only editing is ended.
Esc	If the current cell is in edit mode, reverts the changes done in the current cell. If the underlying source implements the <a href="#">IEditableObject</a> , pressing Esc key for the second time will cancel the edit mode for the entire row.
Delete	Deletes all the rows that are currently in selection. To prevent rows from being deleted set the <a href="#">SfDataGrid.AllowDeleting</a> as <code>False</code> .

### Shift key combinations

Key combinations	Description
Shift + DownArrow	If the Selection Mode is multiple, the next multiple rows will be continuously selected below the currently selected row. If the Selection Mode is Single or SingleDeselect, selection will be moved to the next row.
Shift + UpArrow	If the Selection Mode is multiple, the previous multiple rows will be continuously selected above the currently selected row. If the Selection Mode is Single or SingleDeselect, selection will be moved to the previous row.
Shift + Tab	Moves selection to the previous row from the currently selected row when the SelectionMode is Single or SingleDeselect. If the selected row is in first row, pressing Shift +Tab does nothing.

### Customize key functionalities

To perform custom actions apart from the functionalities mentioned in the above tables for key press actions of the keyboard, implement your custom actions in the [ProcessKeyDown\(\)](#) override of the custom written selection controller class derived from [GridSelectionController](#) and assign it to the [SfDataGrid.GridSelectionController](#) property.

### XML

```
<ContentPage.Resources>
<local:CustomSelectionController x:Key="CustomSelectionController" />
</ContentPage.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo,Mode=TwoWay}"
AllowEditing="True"
NavigationMode="Cell"
SelectionMode="Single"
SelectionController="{x:StaticResource CustomSelectionController}">
</syncfusion:SfDataGrid>
```

### C#

```
dataGrid.SelectionController = new CustomSelectionController();
```

### C#

```
public class CustomSelectionController : GridSelectionController
{
    public CustomSelectionController()
    {
        this.SelectedRows = new GridSelectedRowsCollection();
    }
    protected override void ProcessKeyDown(string keyCode, bool
isCtrlKeyPressed, bool isShiftKeyPressed)
    {
        if(keyCode == "Down")
        {
            // your logics here
        }
        else
        {
            // default key action
            base.ProcessKeyDown(keyCode, isCtrlKeyPressed, isShiftKeyPressed);
        }
    }
}
```

#### Scroll to selected item

You can scroll programmatically to the selected item by passing the `SelectedIndex` to the [SfDataGrid.ScrollToRowIndex](#) method.

### C#

```
dataGrid.ScrollToRowIndex((int) dataGrid.SelectedIndex);
```

#### Clear selection

Data grid allows you to clear the selection applied in the grid rows either by setting the `SfDataGrid.SelectionMode` to `None` or by calling the [SfDataGrid.SelectionController.ClearSelection \(\)](#) method.

### C#

```
//Clear selection using selection mode
dataGrid.SelectionMode = SelectionMode.None;
//Clear selection using selection controller
dataGrid.SelectionController.ClearSelection ();
```

**Note:** Selected items and selections will be cleared whenever the `ItemsSource` is changed at runtime.

#### Row header selection

Data grid allows you to select the grid row(s) upon tapping them over the grid cells. It also allows you to select the grid rows when you tap the row header cells. To enable selection in the data grid, set the [SfDataGrid.SelectionMode](#) property to a value other than `None`.

*Select records in the data grid when tapping only on the row header cells*

The data grid allows you to select a specific row or group of rows by touching the grid cells. However, to select the record only when tapping the row header cells, use the `SfDataGrid.SelectionChanging` event.

**C#**

```
dataGrid.SelectionMode = SelectionMode.Single;
private void DataGrid_SelectionChanging(object sender,
GridSelectionChangingEventArgs e)
{
    e.Cancel = true;
}
private void DataGrid_GridTapped(object sender, GridTappedEventArgs e)
{
    if(e.RowColumnIndex.ColumnIndex == 0)
    {
        dataGrid.SelectedIndex = e.RowColumnIndex.RowIndex;
    }
}
```

**Note:** To enable the row header in the data grid, set the `SfDataGrid.ShowRowHeader` to true.

*Selection animation*

The data grid supports selecting one or more rows programmatically or by touch interactions. In addition, the control also provides extensibility to animate the selected rows.

It can be done by extending the [GridSelectionController](#).

Refer to the following example in which a CustomSelectionController is derived from `GridSelectionController` and an instance of it is assigned to the `SfDataGrid.SelectionController` property to perform selection animation.

**XML**

```
<ContentPage.Resources>
<ResourceDictionary>
<local:CustomSelectionController x:Key="CustomSelectionController"
DataGrid="{x:Reference dataGrid}">
</local:CustomSelectionController>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<sfgrid:SfDataGrid x:Name="dataGrid"
ColumnSizer="Star"
SelectedIndex="1"
SelectionMode="Multiple"
VerticalOverScrollMode="None"
SelectionController="{StaticResource CustomSelectionController}" />
</ContentPage.Content>
```

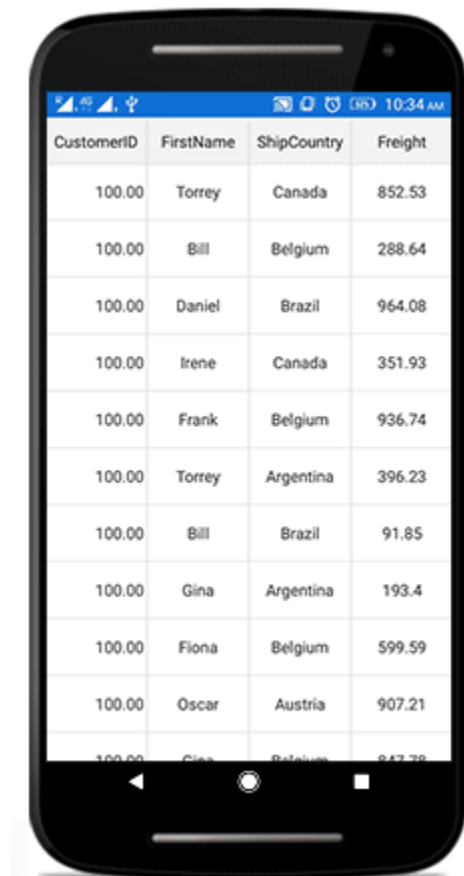
**C#**

```
this.dataGrid.ItemsSource = viewModel.OrdersInfo;
this.dataGrid.SelectionController = new CustomSelectionController();
```

```
this.dataGrid.SelectionMode = SelectionMode.Multiple;  
SelectionPicker.SelectedIndex = 1;
```

## C#

```
public class CustomSelectionController : GridSelectionController  
{  
    public CustomSelectionController()  
    {  
        this.SelectedRows = new GridSelectedRowsCollection();  
    }  
    protected override async void SetSelectionAnimation(VirtualizingCellsControl  
rowElement)  
    {  
        rowElement.Opacity = 0.50;  
        await rowElement.FadeTo(1, 400, Easing.CubicInOut);  
    }  
}
```



## Events in selection

The data grid provides the following events for selection:

- [SelectionChanging](#): This event is raised while selecting a row at the execution time before the row is selected. So it allows canceling the selection action by setting the Cancel property of `GridSelectionChangingEventArgs`.
- [SelectionChanged](#): This event is raised after the column is selected.

These two events are triggered with `GridSelectionChangingEventArgs` and [GridSelectionChangedEventArgs](#) that contain the following properties:

- `AddedItems`: Gets collection of the underlying data objects added for selection.
- `RemovedItems`: Gets collection of the underlying data objects removed from selection.

To hook the `SelectionChanging` event and cancel the selection of a column, follow the code example:

#### C#

```
dataGrid.SelectionChanging += DataGrid_SelectionChanging;
void DataGrid_SelectionChanging (object sender, GridSelectionChangingEventArgs e)
{
    e.Cancel = true;
}
```

To get the selected item in code-behind using the `SelectionChanged` event, follow the code example:

#### C#

```
dataGrid.SelectionChanged += DataGrid_SelectionChanged;
private void DataGrid_SelectionChanged (object sender,
GridSelectionChangedEventArgs e)
{
    // Gets the selected item
    var selectedItem = e.AddedItems[0];
}
```

#### CurrentItem

The [SfDataGrid.CurrentItem](#) property holds the underlying data of the last selected row in data grid.

Get the current item in the `SfDataGrid.SelectionChanged` event by setting the `SfDataGrid.SelectionMode` as `Multiple` or `SingleDeselect`. If the `SelectionMode` is `Single`, the current item and selected item are same.

#### C#

```
dataGrid.SelectionMode = SelectionMode.Multiple;
dataGrid.SelectionChanged += DataGrid_SelectionChanged;
void DataGrid_SelectionChanged (object sender, GridSelectionChangedEventArgs e)
{
    var currentItem = dataGrid.CurrentItem;
    //your codes
}
```



## Customizing Selection Appearance

### *Adding multiple selection color*

The data grid supports selecting one or more rows either programmatically or by touch interactions. By default, the control applies a common background color for the selected rows based on the current theme. However, it also provides extensibility to have multiple selection colors when touching the rows by writing a custom SelectionController derived from [GridSelectionController](#), and assigning it to the [SfDataGrid.SelectionController](#) property. Override the GetSelectionColor() method to apply different colors for selection at runtime.

To set different colors for the selected rows in the data grid, follow the code example:

#### **C#**

```
sfGrid.SelectionController = new CustomSelectionController(sfGrid);  
sfGrid.SelectionMode = SelectionMode.Multiple;
```

#### **C#**

```
public class CustomSelectionController : GridSelectionController  
{  
    public Color[] SelectionColors { get; set; }  
    public CustomSelectionController(SfDataGrid datagrid) : base(datagrid)  
    {  
        this.DataGrid = datagrid;  
        SelectionColors = new Color[11]  
        {  
            Color.DarkSalmon,  
            Color.DarkSlateGray,  
            Color.Red,  
            Color.Blue,  
            Color.DarkOliveGreen,  
            Color.Black,  
            Color.Gray,  
            Color.MediumPurple,  
            Color.BurlyWood,  
            Color.DarkCyan,  
            Color.DarkGoldenrod  
        };  
    }  
    //Code to set multiple selection colors  
    public override Color GetSelectionColor(int rowIndex, object rowData)  
    {  
        if (SelectionColors != null)  
            return SelectionColors[rowIndex % 11];  
        else  
            return Color.Blue;  
    }  
}
```

The following screenshot shows the final outcome upon execution of the above code:

OrderID	Genders	EmployeeID	CustomerID
10001	Male	10501	Frans
10002	Female	10502	Warth
10003	Female	10503	Warth
10004	Female	10504	Warth
10005	Male	10505	Welli
10006	Male	10506	Riscu
10007	Male	10507	Vaffe
10008	Male	10508	Blonp
10009	Male	10509	Vaffe
10010	Male	10510	Folig
10011	Male	10511	Seves
10012	Female	10512	Warth
10013	Male	10513	Linod
10014	Female	10514	Folko
10015	Male	10515	Warth

### *Changing selection background and foreground color*

The SfDataGrid allows you to change the selection background and foreground colors by returning the required color in the `GetSelectionBackgroundColor` and `GetSelectionForegroundColor` properties overrides in the custom style class overriding from [DataGridStyle](#) and assign it to the [SfDataGrid.GridStyle](#) property.

#### **XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:sfgrid="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
"
xmlns:local="clr-namespace:SummaryDemo"
x:Class="SummaryDemo.Summary">
<ContentPage.Resources>
<ResourceDictionary>
<local:SelectionStyle x:Key="selectionStyle"/>
</ResourceDictionary>
</ContentPage.Resources>
<sfgrid:SfDataGrid x:Name="dataGrid"
GridStyle="{StaticResource selectionStyle}"
ItemsSource="{Binding OrdersInfo}" />
</ContentPage>
```

#### **C#**

```
//Apply selection background and foreground color to SfDataGrid from code.
SfDataGrid dataGrid = new SfDataGrid();
dataGrid.GridStyle = new SelectionStyle();
```

#### **C#**

```
//Custom style class
public class SelectionStyle : DataGridStyle
{
    public SelectionStyle()
    {
    }
    public override Color GetSelectionBackgroundColor()
    {
        return Color.Blue;
    }
    public override Color GetSelectionForegroundColor()
    {
        return Color.White;
    }
}
```

OrderID	Genders	EmployeeID	CustomerID
10001	Female	10501	Simob
10002	Female	10502	Folko
10003	Female	10503	Furib
10004	Female	10504	Simob
10005	Male	10505	Riscu
10006	Male	10506	Riscu
10007	Female	10507	Riscu
10008	Female	10508	Seves
10009	Male	10509	Blonp
10010	Female	10510	Warth
10011	Male	10511	Riscu
10012	Male	10512	Furib
10013	Male	10513	Welli

#### *Changing current cell border color*

The SfDataGrid allows you to change the current cell border color applied to the grid cells when entering the edit mode by returning the required color in the `GetCurrentCellBorderColor` override of your custom style class derived from [DataGridStyle](#) and assign it to the [SfDataGrid.GridStyle](#) property.

#### **XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms">
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:sfgrid="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
"
xmlns:local="clr-namespace:SummaryDemo"
x:Class="SummaryDemo.Summary">
<ContentPage.Resources>
<ResourceDictionary>
<local:SelectionStyle x:Key="selectionStyle"/>
</ResourceDictionary>
</ContentPage.Resources>
<sfgrid:SfDataGrid x:Name="dataGrid"
GridStyle="{StaticResource selectionStyle}"
ItemsSource="{Binding OrdersInfo}" />
</ContentPage>
```

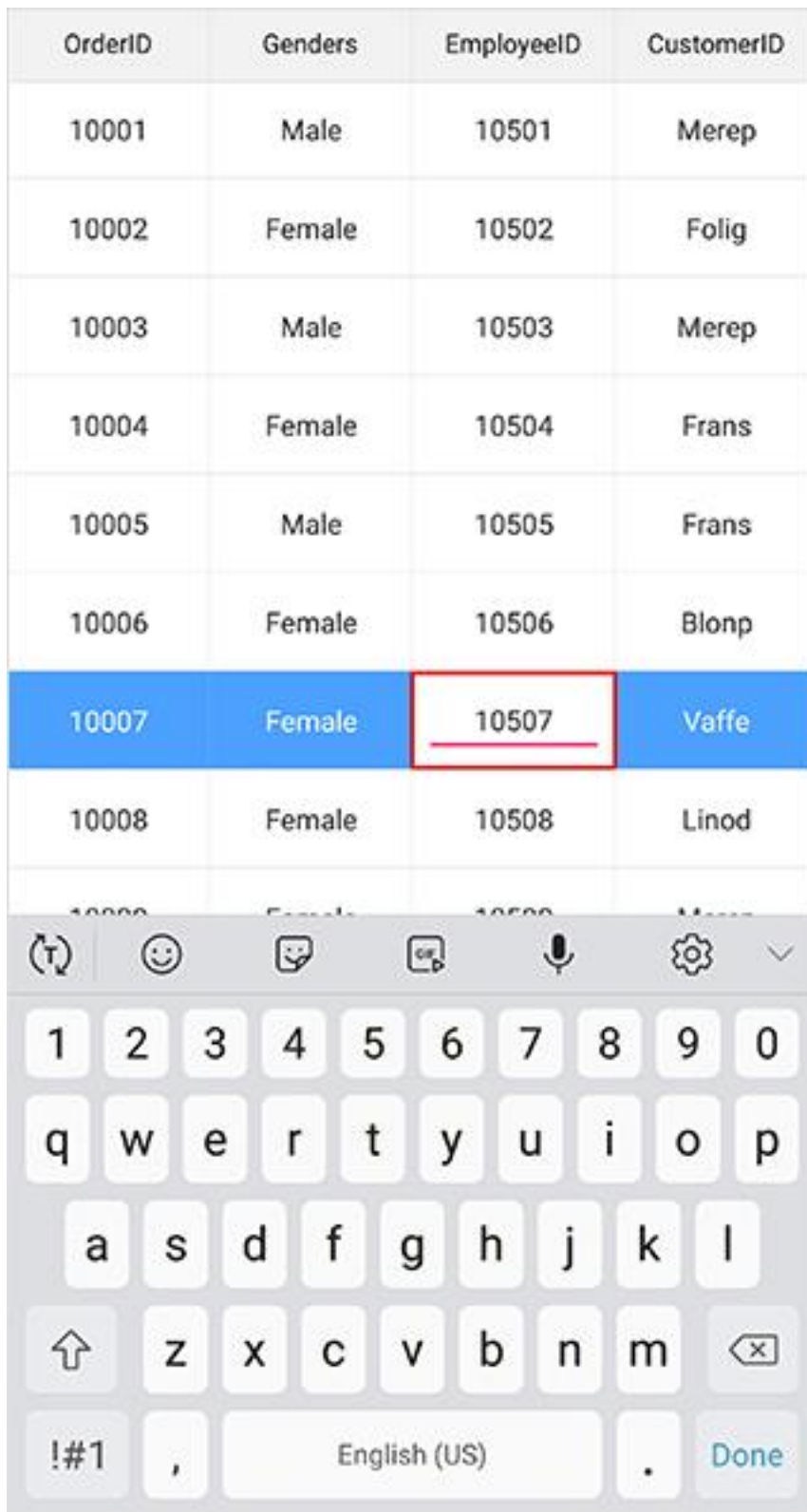
### C#

```
//Apply selection background and foreground color to SfDataGrid from code.
SfDataGrid dataGrid = new SfDataGrid();
dataGrid.GridStyle = new SelectionStyle();
```

### C#

```
//Custom style class
public class SelectionStyle : DataGridStyle
{
    public SelectionStyle()
    {
    }
    public override Color GetCurrentCellBorderColor()
    {
        return Color.Pink;
    }
}
```

OrderID	Genders	EmployeeID	CustomerID
10001	Male	10501	Merep
10002	Female	10502	Folig
10003	Male	10503	Merep
10004	Female	10504	Frans
10005	Male	10505	Frans
10006	Female	10506	Blonp
10007	Female	10507	Vaffe
10008	Female	10508	Linod
10009	Female	10509	Merep



#### Binding selection properties

The SfDataGrid allows you to bind the selection properties such as [SelectedIndex](#) and [SelectedItem](#) to the properties in the ViewModel directly.

## XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:sfgrid="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
"
xmlns:local="clr-namespace:SummaryDemo"
x:Class="SummaryDemo.Summary">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<sfgrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
SelectionMode="Multiple"
SelectedIndex="{Binding DataGridSelectedIndex,Mode=TwoWay}"
SelectedItem="{Binding DataGridSelectedItem,Mode=TwoWay}"/>
```

## C#

```
// In ViewModel.cs
private int dataGridSelectedIndex;
private object dataGridSelectedItem;
public int DataGridSelectedIndex
{
    get
    {
        return dataGridSelectedIndex;
    }
    set
    {
        this.dataGridSelectedIndex = value;
        RaisePropertyChanged("DataGridSelectedIndex");
    }
}
public object DataGridSelectedItem
{
    get
    {
        return dataGridSelectedItem;
    }
    set
    {
        this.dataGridSelectedItem = value;
        RaisePropertyChanged("DataGridSelectedItem");
    }
}
public ViewModel()
{
    DataGridSelectedIndex = 2;
    DataGridSelectedItem = OrdersInfo[5];
}
```

## Stacked Headers

The SfDataGrid supports displaying additional unbound, multiple/multilevel header rows known as **StackedHeaderRows** that spans across the DataGrid columns. You can group one or more columns under each stacked header.

Each **StackedHeaderRow** contains **StackedColumns**, which contains a number of child columns. The **StackedColumn.ChildColumns** property contains the columns grouped under the stacked header row. The **StackedColumn.MappingName** is a unique name used for mapping a specific child column grouped under the same stacked header row, whereas the **StackedColumn.Text** contains the text displayed in the stacked header row.

### Adding stacked header

The stacked headers can be added using the following steps:

1. Create an object of **StackedHeaderRow** for adding stacked columns.
2. Add the columns using the **ChildColumns** property of **StackedColumn**.
3. Add the **StackedColumn** to **StackedColumns** collection.
4. Finally, add the **StackedHeaderRow** to **StackedHeaderRows** collection of the SfDataGrid.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}">
  <syncfusion:SfDataGrid.StackedHeaderRows>
    <syncfusion:StackedHeaderRow>
      <syncfusion:StackedHeaderRow.StackedColumns>
        <syncfusion:StackedColumn
ChildColumns="OrderID, OrderDate, CustomerID, ContactName"
Text="Order Shipment Details"
MappingName="SalesDetails"
FontAttribute="Bold"
TextAlignment="Center"
/>
      </syncfusion:StackedHeaderRow.StackedColumns>
    </syncfusion:StackedHeaderRow>
    <syncfusion:StackedHeaderRow>
      <syncfusion:StackedHeaderRow.StackedColumns>
        <syncfusion:StackedColumn
ChildColumns="OrderID, OrderDate"
Text="Order Details"
MappingName="OrderDetails"
FontAttribute="Bold"
TextAlignment="Center"
/>
        <syncfusion:StackedColumn
ChildColumns="CustomerID, ContactName"
Text="Customer Details"
MappingName="CustomerDetails"
FontAttribute="Bold"
TextAlignment="Center"
/>
      </syncfusion:StackedHeaderRow.StackedColumns>
    </syncfusion:StackedHeaderRow>
  </syncfusion:SfDataGrid.StackedHeaderRows>
</syncfusion:SfDataGrid>
```



```
</syncfusion:SfDataGrid.StackedHeaderRows>  
</syncfusion:SfDataGrid>
```

## C#

```
var stackedHeaderRow = new StackedHeaderRow();  
stackedHeaderRow.StackedColumns.Add(new StackedColumn()  
{  
    ChildColumns = "OrderID" + "," + "OrderDate" + "," + "CustomerID" + "," +  
    "ContactName",  
    Text = "Order Shipment Details",  
    MappingName = "SalesDetails",  
    FontAttribute = FontAttributes.Bold,  
    TextAlignment = TextAlignment.Center,  
});  
dataGrid.StackedHeaderRows.Add(stackedHeaderRow);  
var stackedHeaderRow1 = new StackedHeaderRow();  
stackedHeaderRow1.StackedColumns.Add(new StackedColumn()  
{  
    ChildColumns = "OrderID" + "," + "OrderDate",  
    Text = "Order Details",  
    MappingName = "OrderDetails",  
    FontAttribute = FontAttributes.Bold,  
    TextAlignment = TextAlignment.Center  
});  
stackedHeaderRow1.StackedColumns.Add(new StackedColumn()  
{  
    ChildColumns = "CustomerID" + "," + "ContactName",  
    Text = "Customer Details",  
    MappingName = "CustomerDetails",  
    FontAttribute = FontAttributes.Bold,  
    TextAlignment = TextAlignment.Center  
});  
this.dataGrid.StackedHeaderRows.Add(stackedHeaderRow1);
```



5:21 100%

Order Shipment Details			
Order Details		Customer Details	
Order ID	Order Date	Customer ID	Contact Name
10001	08-10-2008	1761	Ralph
10002	04-07-2009	1790	Torrey
10003	15-04-2010	1795	Bill
10004	24-12-2006	1740	Oscar
10005	20-04-2002	1773	Oscar
10006	13-06-2004	1701	William
10007	11-11-2000	1781	Katie
10008	29-07-2011	1720	Michael
10009	04-05-2005	1791	Michael
10010	20-03-2010	1779	Oscar
10011	01-10-2010	1736	Bill
10012	08-03-2012	1714	Michael
10013	10-03-2007	1749	Oscar
10014	30-08-2013	1729	Torrey
10015	21-06-2012	1708	William
10016	20-05-2001	1712	Brenda
10017	09-10-2008	1711	William

### Adding child columns

You can add the child columns to a particular stacked header row directly.

#### C#

```
var childColumn =  
dataGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns;  
dataGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns = childColumn +  
", " + "OrderDate";
```

### Removing child columns

Similarly, you can remove the child columns from a particular stacked header row directly.

#### C#

```
var removingColumns =  
this.dataGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns.Split(',')  
.ToList<string>();  
string childColumns = string.Empty;  
foreach(var stackedColumnName in removingColumns.ToList())  
{  
    if (stackedColumnName.Equals("OrderID"))  
    {  
        removingColumns.Remove(stackedColumnName);  
    }  
    else  
    {  
        childColumns = childColumns + stackedColumnName + ", ";  
    }  
}  
dataGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns = childColumns;
```

### Changing stacked header row height

You can change the height of StackedHeaderRows using the `SfDataGrid.HeaderRowHeight` property.

#### C#

```
dataGrid.HeaderRowHeight = 50;
```

You can also change the height of stacked header rows using the `SfDataGrid.QueryRowHeight` event.

#### C#

```
dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;  
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)  
{  
    if(e.RowIndex < this.dataGrid.GetHeaderIndex())  
    {  
        // Using the following code, you can set a desired height based on the row  
        // index.  
        e.Height = 50;  
        // Uncomment the below line of code to apply auto fit height based on the  
        // contents of the stacked header row.  
        //e.Height = dataGrid.GetRowHeight(e.RowIndex);  
        e.Handled = true;  
    }  
}
```

```
}
}
```

## Appearance

### Font customization

Customize the font's size, family and attribute of the text displayed in stacked header column using the `StackedColumn.TextSize`, `StackedColumn.Font`, and `StackedColumn.FontAttribute` properties, respectively. The default font size and font attribute are 14 and normal, respectively.

### XML

```
<syncfusion:StackedHeaderRow>
  <syncfusion:StackedHeaderRow.StackedColumns>
    <syncfusion:StackedColumn
      ChildColumns="OrderID,OrderDate,CustomerID,ContactName"
      Text="Order Shipment Details"
      MappingName="SalesDetails"
      TextSize = 16
      Font="Helvetica Neue"
      FontAttribute="Bold"
    />
  </syncfusion:StackedHeaderRow.StackedColumns>
</syncfusion:StackedHeaderRow>
```

### C#

```
var stackedHeaderRow = new StackedHeaderRow();
stackedHeaderRow.StackedColumns.Add(new StackedColumn()
{
    ChildColumns = "OrderID" + "," + "OrderDate" + "," + "CustomerID" + "," +
    "ContactName",
    Text = "Order Shipment Details",
    MappingName = "SalesDetails",
    TextSize = 14,
    Font = "Helvetica Neue",
    FontAttribute = FontAttributes.Bold
});
dataGrid.StackedHeaderRows.Add(stackedHeaderRow);
```

### Foreground and background customization

The appearance of stacked header row can be customized by returning a desired color in the `GetStackedHeaderBackgroundColor()` and `GetStackedHeaderForegroundColor()` overrides of the custom written style class derived from `DataGridStyle` and assigning it to the `SfDataGrid.GridStyle` property.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:syncfusion="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
  xmlns:local ="clr-namespace:DataGridSample;assembly=DataGridSample"
  x:Class="DataGridSample.Sample">
```

```
<ContentPage.Resources>
<ResourceDictionary>
<local:Dark x:Key="dark" />
</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
GridStyle="{StaticResource dark}"
ItemsSource="{Binding OrdersInfo}" />
</ContentPage>
```

### C#

```
//Apply custom style to SfDataGrid from code
SfDataGrid dataGrid = new SfDataGrid();
dataGrid.GridStyle = new Dark ();
```

### C#

```
//Custom style class
public class Dark : DataGridStyle
{
    public Dark ()
    {
    }
    public override Color GetStackedHeaderBackgroundColor(int rowIndex)
    {
        return Color.DarkGoldenrod;
    }
    public override Color GetStackedHeaderForegroundColor(int rowIndex)
    {
        return Color.FromRgb(255, 255, 255);
    }
}
```



Order Shipment Details			
Order Details		Customer Details	
Order ID	Order Date	Customer ID	Contact Name
10001	08-10-2008	1764	Fiona
10002	04-07-2009	1738	Fiona
10003	15-04-2010	1778	Oscar
10004	24-12-2006	1725	Frank
10005	20-04-2002	1725	Oscar
10006	13-06-2004	1706	Torrey
10007	11-11-2000	1721	Katie
10008	29-07-2011	1763	Michael
10009	04-05-2005	1768	Ralph
10010	20-03-2010	1772	Brenda
10011	01-10-2010	1736	Danielle
10012	08-03-2012	1756	Kyle
10013	10-03-2007	1794	Bill
10014	30-08-2013	1780	Danielle
10015	21-06-2012	1791	Michael
10016	20-05-2001	1764	Michael
10017	09-10-2008	1724	Danielle

### Conditional styling

The SfDataGrid also allows to customize the appearance of stacked header rows conditionally based on its row index.

### C#

```
//Custom style class
public class Dark : DataGridStyle
{
    public Dark ()
    {
    }
    public override Color GetStackedHeaderBackgroundColor(int rowIndex)
    {
        if (rowIndex == 0)
```

```
{
    return Color.Black;
}
else if (rowIndex == 1)
{
    return Color.DarkOliveGreen;
}
}
public override Color GetStackedHeaderForegroundColor(int rowIndex)
{
    if (rowIndex == 0 || rowIndex == 1)
    {
        return Color.FromRgb(255, 255, 255);
    }
    else
    {
        return Color.FromRgb(43, 43, 43);
    }
}
}
```



Order Shipment Details			
Order Details		Customer Details	
Order ID	Order Date	Customer ID	Contact Name
10001	08-10-2008	1724	Kyle
10002	04-07-2009	1765	Irene
10003	15-04-2010	1759	Irene
10004	24-12-2006	1743	Michael
10005	20-04-2002	1770	Torrey
10006	13-06-2004	1747	Fiona
10007	11-11-2000	1778	William
10008	29-07-2011	1746	Bill
10009	04-05-2005	1716	Michael
10010	20-03-2010	1751	Danielle
10011	01-10-2010	1747	William
10012	08-03-2012	1793	Fiona
10013	10-03-2007	1756	Fiona
10014	30-08-2013	1771	Bill
10015	21-06-2012	1784	Danielle
10016	20-05-2001	1764	Gina
10017	09-10-2008	1762	Danielle

Loading template in stacked column

The SfDataGrid allows you to load any desired view inside a `StackedColumn` using the `StackedColumn.Template` property.

#### XML

```
<syncfusion:SfDataGrid.StackedHeaderRows>
<syncfusion:StackedHeaderRow>
<syncfusion:StackedHeaderRow.StackedColumns>
<syncfusion:StackedColumn
ChildColumns="OrderID,OrderDate"
Text="Order Details"
MappingName="OrderDetails"
FontAttribute="Bold"
TextAlignment="Center"
```



```

/>
<syncfusion:StackedColumn
ChildColumns="CustomerID,ContactName"
MappingName="CustomerDetails"
>
<syncfusion:StackedColumn.Template>
<DataTemplate>
<Grid BackgroundColor="MediumPurple">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="50" />
</Grid.ColumnDefinitions>
<Label Text="Customer Details" TextColor="White"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
Grid.Column="0" />
<Image Source="customer_details.png" VerticalOptions="Center"
HorizontalOptions="Start" Aspect="AspectFit" Grid.Column="1" />
</Grid>
</DataTemplate>
</syncfusion:StackedColumn.Template>
</syncfusion:StackedColumn>
</syncfusion:StackedHeaderRow.StackedColumns>
</syncfusion:StackedHeaderRow>
</syncfusion:SfDataGrid.StackedHeaderRows>

```

**C#**

```

var stackedHeaderRow1 = new StackedHeaderRow();
stackedHeaderRow1.StackedColumns.Add(new StackedColumn()
{
    ChildColumns = "OrderID" + "," + "OrderDate",
    Text = "Order Details",
    MappingName = "OrderDetails",
    FontAttribute = FontAttributes.Bold,
    TextAlignment = TextAlignment.Center
});
stackedHeaderRow1.StackedColumns.Add(new StackedColumn()
{
    ChildColumns = "CustomerID" + "," + "ContactName",
    MappingName = "CustomerDetails",
    Template = new DataTemplate(() =>
    {
        var gridView = new Grid()
        {
            BackgroundColor = Color.MediumPurple,
            ColumnDefinitions = new ColumnDefinitionCollection()
            {
                new ColumnDefinition{Width = new GridLength(1, GridUnitType.Star) },
                new ColumnDefinition{ Width = 50}
            }
        };
        var imageView = new Image()
        {
            Source = ImageSource.FromFile("customer_details.png"),
            Aspect = Aspect.AspectFit,
            VerticalOptions = LayoutOptions.Center,

```

```
HorizontalOptions = LayoutOptions.Start
};
var label = new Label()
{
    Text = "Customer Details",
    TextColor = Color.White,
    VerticalTextAlignment = TextAlignment.Center,
    HorizontalTextAlignment = TextAlignment.Center
};
gridView.Children.Add(label, 0, 0);
gridView.Children.Add(imageView, 1, 0);
return gridView;
})
});
this.dataGrid.StackedHeaderRows.Add(stackedHeaderRow1);
```



## Load More

The data grid enables **LoadMore** option when the [SfDataGrid.AllowLoadMore](#) property is set to **true**, and also by setting the [SfDataGrid.LoadMoreCommand](#) property. When **LoadMore** is enabled, the control loads a subset of data to its data source at runtime using [LoadMoreView](#).

When the grid reaches maximum offset while scrolling down, an interactive load more view is displayed. Tapping the load more view triggers a command to add more data to the data source of the grid at runtime.

## Load more command

The data grid load records to its data source at runtime by triggering an **ICommand** bound to the [SfDataGrid.LoadMoreCommand](#) property. When the load more view is tapped, the **CanExecute** of the **ICommand** returns **true**, and this command is triggered to load the records at runtime.

Set the [SfDataGrid.IsBusy](#) property to `true` before loading items to notify the grid that more items are to be loaded. Set the property to `false` after loading items to the grid. When loading items, alter the time for the [LoadMore](#) animation from the sample by setting a delay based on the requirement.

To enable and load items at runtime, follow the code example:

### C#

```
//Enable load more in SfDataGrid
dataGrid.AllowLoadMore = true;
dataGrid.LoadMoreCommand = new Command(ExecuteLoadMoreCommand);
private async void ExecuteLoadMoreCommand()
{
    this.dataGrid.IsBusy = true;
    await Task.Delay(new TimeSpan(0, 0, 5));
    viewModel.LoadMoreItems ();
    this.dataGrid.IsBusy = false;
}
//ViewModel.cs
private OrderInfoRepository order;
public ViewModel()
{
    SetRowsToGenerate(50);
}
//ItemsSource
private ObservableCollection<OrderInfo> ordersInfo;
public ObservableCollection<OrderInfo> OrdersInfo
{
    get { return ordersInfo; }
    set { this.ordersInfo = value; }
}
//ItemsSource Generator
public void SetRowsToGenerate(int count)
{
    order = new OrderInfoRepository();
    ordersInfo = order.GetOrderDetails(count);
}
public void LoadMoreItems()
{
    for (int i = 0; i < 20; i++)
    {
        this.OrdersInfo.Add(order.GenerateOrder(OrdersInfo.Count + 1));
    }
}
//OrderInfoRepository.cs
public ObservableCollection<OrderInfo> GetOrderDetails (int count)
{
    ObservableCollection<OrderInfo> orderDetails = new
    ObservableCollection<OrderInfo> ();
    for (int i = 10001; i <= count + 10000; i++) {
        var order = new OrderInfo () {
            OrderID = i,
            CustomerID = CustomerID [random.Next (7)],
            EmployeeID = random.Next (1700, 1800).ToString (),
            FirstName = FirstNames [random.Next (7)],
            LastName = LastNames [random.Next (7)]
        };
        orderDetails.Add (order);
    }
}
```

```

return orderDetails;
}
public OrderInfo GenerateOrder(int id)
{
var order = new OrderInfo(){
OrderID = (id + 10000),
CustomerID = CustomerID [random.Next (15)],
EmployeeID = random.Next (1700, 1800).ToString (),
FirstName = FirstNames [random.Next (15)],
LastName = LastNames [random.Next (15)],
};
return order;
}
//Main DataSources
string[] FirstNames = new string[] {
"Kyle",
"Gina",
"Irene",
"Katie",
"Michael",
"Oscar",
"Ralph"
};
string[] LastNames = new string[] {
"Adams",
"Crowley",
"Ellis",
"Gable",
"Irvine",
"Keefe",
"Mendoza"
};
string[] CustomerID = new string[] {
"Adams",
"Frans",
"Katie",
"Laura",
"Wilson",
"Andrew",
"Lenny"
};
};

```

### Customize load more display text

Customize the text displayed in the `LoadMoreView` by setting the [SfDataGrid.LoadMoreText](#) property as follows:

#### C#

```

//setting load more text in SfDataGrid
dataGrid.LoadMoreText = "Load More Items";

```

### Customize load more view position

Customize the displayed `LoadMoreView` to either `top` or `bottom` based on the requirement.

#### C#

```
//Enable load more in SfDataGrid  
dataGrid.LoadMorePosition = LoadMoreViewPosition.Bottom;
```

### Customize LoadMoreView

The data grid also customizes **LoadMoreView** based on the requirements. To do this, write custom **LoadMoreView** class inheriting from **LoadMoreView**, and perform the **LoadMoreOperation**.

To customize **LoadMoreView**, follow the code example:

#### C#

```
public class CustomLoadMoreView : LoadMoreView  
{  
    private Button loadMoreView;  
    public CustomLoadMoreView()  
    {  
        this.BackgroundColor = Color.Red;  
        loadMoreView = new Button ();  
        loadMoreView.Text = "LoadItems";  
        this.Children.Add(loadMoreView);  
        loadMoreView.Clicked += loadMoreView_Tapped;  
    }  
    void loadMoreView_Tapped (object sender, EventArgs e)  
    {  
        if (this.LoadMoreCommand != null)  
        {  
            this.LoadMoreCommand.Execute(null);  
        }  
    }  
    protected override void LayoutChildren(double x, double y, double width, double height)  
    {  
        loadMoreView.Layout(new Rectangle(x, y, width, height));  
    }  
}
```

Running application renders the following output:



You can download a readily runnable sample that illustrates enabling and using the load more functionality, [here](#).

### Pull To Refresh

The data grid enables the `PullToRefresh` option by setting the `SfDataGrid.AllowPullToRefresh` property to `true` and by setting the `SfDataGrid.PullToRefreshCommand` property. When the `PullToRefresh` is enabled, the control supports for refreshing the data source at runtime while doing the pull to refresh action.

### Pull to refresh command

The data grid refreshes the data in view at runtime by triggering an `ICommand` bound to the `SfDataGrid.PullToRefreshCommand` property. While you perform pull to refresh action, if the progress bar meets 100 %, then this command is triggered to refresh the records in view.

Set the `SfDataGrid.IsBusy` property to `true` before refreshing the records to notify the grid that pull to refresh action is being performed and set the property to `false` after the view is refreshed. You can also alter the pull to refresh animation time from the sample by setting a delay.

To enable and perform pull to refresh operation, follow the code example:

### C#

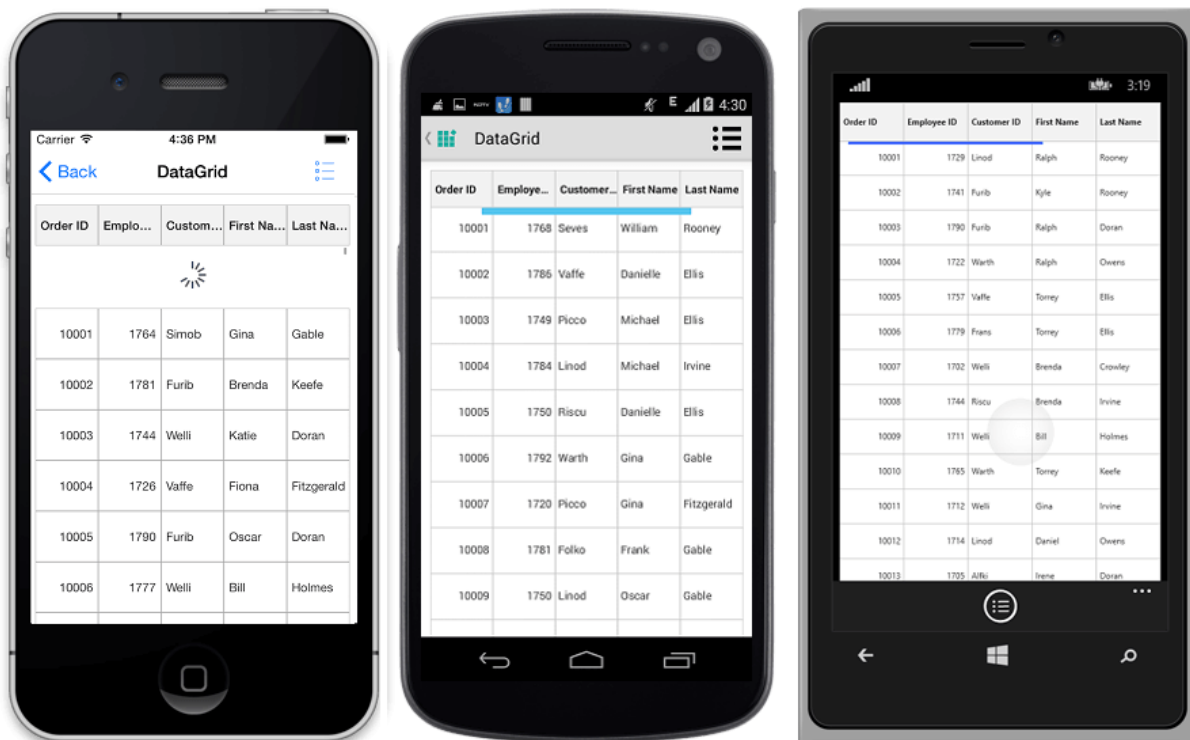
```
//Enable PullToRefresh in SfDataGrid
dataGrid.AllowPullToRefresh = true;
dataGrid.PullToRefreshCommand = new Command(ExecutePullToRefreshCommand);
private async void ExecutePullToRefreshCommand()
{
    this.dataGrid.IsBusy = true;
```

```
await Task.Delay(new TimeSpan(0, 0, 5));
viewModel.ItemsSourceRefresh ();
this.dataGrid.IsBusy = false;
}
//ViewModel.cs
private OrderInfoRepository order;
public ViewModel()
{
    SetRowsToGenerate(50);
}
//ItemsSource
private ObservableCollection<OrderInfo> ordersInfo;
public ObservableCollection<OrderInfo> OrdersInfo
{
    get { return ordersInfo; }
    set { this.ordersInfo = value; }
}
//ItemsSource Generator
public void SetRowsToGenerate(int count)
{
    order = new OrderInfoRepository();
    ordersInfo = order.GetOrderDetails(count);
}
public void ItemsSourceRefresh()
{
    int count = random.Next (1, 6);
    for (int i = 11000; i < 11000 + count; i++)
    {
        int value = i + random.Next (100, 150);
        this.OrdersInfo.Insert (0, order.RefreshItemsSource (value));
    }
}
//OrderInfoRepository.cs
public ObservableCollection<OrderInfo> GetOrderDetails (int count)
{
    ObservableCollection<OrderInfo> orderDetails = new
    ObservableCollection<OrderInfo> ();
    for (int i = 10001; i <= count + 10000; i++) {
        var order = new OrderInfo () {
            OrderID = i,
            CustomerID = CustomerID [random.Next (7)],
            EmployeeID = random.Next (1700, 1800).ToString (),
            FirstName = FirstNames [random.Next (7)],
            LastName = LastNames [random.Next (7)]
        };
        orderDetails.Add (order);
    }
    return orderDetails;
}
public OrderInfo RefreshItemsource(int i)
{
    var order = new OrderInfo(){
        OrderID = i,
        CustomerID = CustomerID[random.Next(7)],
        EmployeeID = random.Next(1700, 1800).ToString(),
        FirstName = FirstNames[random.Next(7)],
        LastName = LastNames[random.Next(7)]
    }
```



```
};  
return order;  
}  
//Main DataSources  
string[] FirstNames = new string[] {  
    "Kyle",  
    "Gina",  
    "Irene",  
    "Katie",  
    "Michael",  
    "Oscar",  
    "Ralph"  
};  
string[] LastNames = new string[] {  
    "Adams",  
    "Crowley",  
    "Ellis",  
    "Gable",  
    "Irvine",  
    "Keefe",  
    "Mendoza"  
};  
string[] CustomerID = new string[] {  
    "Hanna",  
    "Frans",  
    "Maria",  
    "John",  
    "Andrew",  
    "Fuller",  
    "Carter"  
};
```

Running application renders the following output:

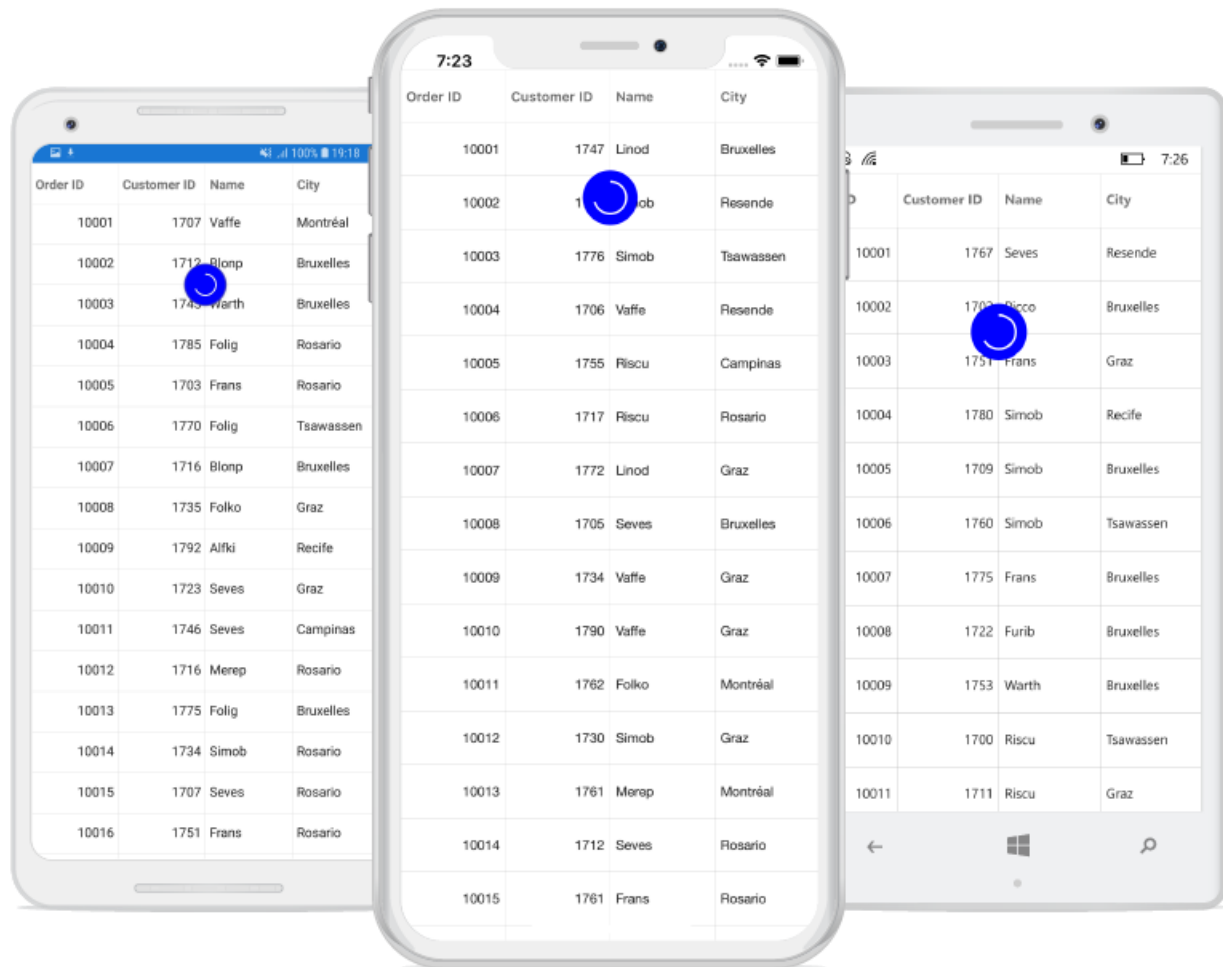


### Host the data grid inside pull-to-refresh

The pull-to-refresh is a refreshing control allows you to interact and refresh the view loaded in it. When the control is hosted inside the pull-to-refresh, it is used to refresh the item while performing the pull to refresh action.

For more details and code example for hosting the SfDataGrid inside SfPullToRefresh, refer [here](#).

The output will look like on iOS, Android, and Windows Phone devices as follows:



## Column Drag And Drop

The SfDataGrid allows dragging and dropping a column header by setting the [SfDataGrid.AllowDraggingColumn](#) property to `true`. The Drag view is displayed while dragging a column header. Based on the requirements, drag and drop operations can be handled by using the [SfDataGrid.QueryColumnDragging](#) event.

To enable column drag and drop, follow the code example:

### XML

```
<syncfusion:SfDataGrid AllowDraggingColumn="True" />
```

### C#

```
dataGrid.AllowDraggingColumn = true;
```

OrderID	Gender	EmployeeID	CustomerID
10001	Male	10501	Frans
10002	Male	10502	Alfki
10003	Female	10503	Frans
10004	Female	10504	Simob
10005	Male	10505	Linod
10006	Female	10506	Alfki
10007	Male	10507	Furib
10008	Female	10508	Simob
10009	Male	10509	Vaffe
10010	Female	10510	Blonp

### Column drag and drop events

The `QueryColumnDragging` event is fired while dragging a column. It will be continuously fired till the dragging ends. By handling the `SfDataGrid.QueryColumnDragging` event, dragging of a particular column header can be canceled.

`QueryColumnDragging` event provides the following properties in the [QueryColumnDraggingEventArgs](#):

- [From](#): Returns index of the currently dragging column.
- [To](#): Returns dragging index where you try to drop the column.
- [Reason](#): Returns the column dragging details as the [QueryColumnDraggingReason](#).
- [DraggingPosition](#): Returns positions of the drag view during column drag and drop operations.
- [CanAutoScroll](#): Returns whether auto-scrolling should happen when column drag view reaches the left or right ends of the `SfDataGrid`.
- [Cancel](#): Returns the boolean property to cancel the event.

### Cancel dragging of a particular column

Dragging of a particular column can be canceled using `QueryColumnDraggingReason` argument of the `QueryColumnDragging` event handler.

### C#

```
this.SfGrid.QueryColumnDragging += SfGrid_QueryColumnDragging;
private void SfGrid_QueryColumnDragging(object sender,
QueryColumnDraggingEventArgs e)
```

```
{  
    //e.From returns the index of the dragged column.  
    //e.Reason returns the dragging status of the column.  
    if (e.From == 1 && e.Reason == QueryColumnDraggingReason.DragStarted)  
        e.Cancel = true;  
}
```

Cancel dropping when dragging for a particular column

Dropping when dragging a particular column can be canceled using `QueryColumnDraggingReason` argument of the `QueryColumnDragging` event handler.

**C#**

```
this.SfGrid.QueryColumnDragging += SfGrid_QueryColumnDragging;  
private void SfGrid_QueryColumnDragging(object sender,  
    QueryColumnDraggingEventArgs e)  
{  
    //e.To returns the index of the current column.  
    //e.Reason returns the dragging status of the column.  
    if ((e.To > 5 || e.To < 10) &&  
        (e.Reason == QueryColumnDraggingReason.DragEnded || e.Reason ==  
        QueryColumnDraggingReason.Dragging))  
        e.Cancel = true;  
}
```

Cancel dropping for a particular column

Dropping of a particular column can be canceled using `QueryColumnDraggingReason` argument of the `QueryColumnDragging` event handler.

**C#**

```
this.SfGrid.QueryColumnDragging += SfGrid_QueryColumnDragging;  
private void SfGrid_QueryColumnDragging(object sender,  
    QueryColumnDraggingEventArgs e)  
{  
    //e.From returns the index of the dragged column.  
    //e.Reason returns the dragging status of the column.  
    if (e.From == 1 && e.Reason == QueryColumnDraggingReason.DragEnded)  
        e.Cancel = true;  
}
```

Cancel dropping at a particular position

Dropping at a particular position can be canceled using `QueryColumnDraggingReason` argument of the `QueryColumnDragging` event handler.

**C#**

```
this.SfGrid.QueryColumnDragging += SfGrid_QueryColumnDragging;  
private void SfGrid_QueryColumnDragging(object sender,  
    QueryColumnDraggingEventArgs e)  
{  
    //e.To returns the index of the current column.  
    //e.Reason returns the dragging status of the column.
```

```
if ((e.To == 5 || e.To == 7) && e.Reason ==  
QueryColumnDraggingReason.DragEnded)  
e.Cancel = true;  
}
```

Cancel dropping of a particular column in a position

Dropping of a particular column in a position can be canceled using `QueryColumnDraggingReason` and `Position` arguments of the `QueryColumnDragging` event handler.

#### C#

```
this.SfGrid.QueryColumnDragging += SfGrid_QueryColumnDragging;  
private void SfGrid_QueryColumnDragging(object sender,  
QueryColumnDraggingEventArgs e)  
{  
    //e.To returns the index of the current column.  
    //e.DraggingPosition returns the x and y position of the current column  
    if (e.DraggingPosition == new Point(100,100) && e.Reason ==  
QueryColumnDraggingReason.DragEnded)  
e.Cancel = true;  
}
```

Cancel drag and drop between frozen and non-frozen columns

*Cancel dragging between frozen and non-frozen columns*

Dragging between frozen and non-frozen columns can be canceled using `QueryColumnDraggingReason` and `From` arguments of the `QueryColumnDragging` event handler by checking whether the value of `From` argument is a frozen column index.

#### C#

```
SfGrid.FrozenColumnsCount = 2;  
this.SfGrid.QueryColumnDragging += SfGrid_QueryColumnDragging;  
private void SfGrid_QueryColumnDragging(object sender,  
QueryColumnDraggingEventArgs e)  
{  
    //e.From returns the index of the dragged column.  
    //e.To returns the index of the current column.  
    if (e.From < 2 && e.Reason == QueryColumnDraggingReason.DragStarted)  
e.Cancel = true;  
}
```

*Cancel dropping between frozen and non-frozen columns*

Dropping between frozen and non-frozen columns can be canceled using `QueryColumnDraggingReason` and `From` argument of the `QueryColumnDragging` event handler by checking whether the `e.From` value is a frozen column index.

#### C#

```
SfGrid.FrozenColumnsCount = 2;  
this.SfGrid.QueryColumnDragging += SfGrid_QueryColumnDragging;  
private void SfGrid_QueryColumnDragging(object sender,  
QueryColumnDraggingEventArgs e)  
{
```

```
//e.From returns the index of the dragged column.  
//e.To returns the index of the current column.  
if (e.From < 2 && e.Reason == QueryColumnDraggingReason.DragEnded)  
    e.Cancel = true;  
}
```

### Customize column drag and drop indicators

The SfDataGrid allows customizing the column drag and drop indicators by writing the custom grid style, deriving from the [DataGridStyle](#) and assigning it to the [SfDataGrid.GridStyle](#) property.

#### C#

```
dataGrid.GridStyle = new CustomGridStyle();
```

#### C#

```
// Custom style class  
public class CustomGridStyle : DataGridStyle  
{  
    public CustomGridStyle()  
    {  
    }  
    public override ImageSource GetColumnDragUpIndicator()  
    {  
        return ImageSource.FromResource("DataGridDemo.icons.GreenUp.png");  
    }  
    public override ImageSource GetColumnDragDownIndicator()  
    {  
        return ImageSource.FromResource("DataGridDemo.icons.GreenDown.png");  
    }  
}
```

OrderID	 Gender	Country	CustomerID
10001	Female	America	Riscu
10002	Male	India	Picco
10003	Male	Poland	Vaffe
10004	Male	Italy	Vaffe
10005	Male	India	Blonp
10006	Female	US	Linod
10007	Female	Belgium	Vaffe
10008	Male	US	Merep
10009	Female	Australia	Welli
10010	Male	China	Folig
10011	Male	India	Simob

### Cancel auto scrolling

Horizontal auto-scrolling of the **SfDataGrid** during column drag and drop can be canceled using **CanAutoScroll** argument of the **QueryColumnDragging** event handler.

### C#

```
this.SfGrid.QueryColumnDragging += SfGrid_QueryColumnDragging;
private void SfGrid_QueryColumnDragging(object sender,
QueryColumnDraggingEventArgs e)
{
    // Disable scroll while dragging the columns.
    e.CanAutoScroll = false;
}
```



## Row Drag and Drop

The data grid allows you to drag and drop a row by setting the [SfDataGrid.AllowDraggingRow](#) property to `true`. A customizable row drag-and-drop template is displayed while dragging a row. Drag-and-drop operations can be handled using the [SfDataGrid.QueryRowDragging](#) event.

To enable row drag and drop, follow the code example:

### XML

```
<syncfusion:SfDataGrid AllowDraggingRow="True" />
```

### C#

```
dataGrid.AllowDraggingRow = true;
```

OrderID	Gender	EmployeeID	CustomerID
10001	Female	10501	Alfki
10002	Male	10502	Folig
10003	Male	10503	Folko
10004	Male	10504	Warth
10005	Male	10505	Linod
10006	Female	10506	Folig
10007	Male	10507	Folig
10008	Male	10508	Welli
10009	Male	10509	Alfki
10010	Female	10510	Merep

## Dragging scenarios

The data grid performs drag-and-drop operations with both data rows and groups as in the following scenarios:

- Records can be reordered to any position with auto scrolling.
- Group position can be reordered using drag and drop. But no groups can be added inside other groups.
- Data rows can be reordered within the same group or into other groups, as well.

**Note:** Reordering changes are made only in the [SfDataGrid.View](#), and not in the underlying data. Thus, the changes will be reverted when performing sorting, grouping, or any other operation refreshing the view. Reordering changes in the underlying data can be achieved by handling a `QueryRowDragging` event in the sample side as explained below in [Reordering underlying data](# Reordering underlying data ).

**Note:** Now you can drag and drop the grid rows using your mouse in the UWP (Desktop) platform.

#### Row drag-and-drop template

The data grid allows you to load desired content when performing row drag-and-drop operations using the [SfDataGrid.RowDragDropTemplate](#). The template can be defined either in code or XAML.

#### Default template

The default template will be loaded if another template is not explicitly assigned for row drag-and-drop operations.

OrderID	Gender	EmployeeID	CustomerID
10001	Female	10501	Furib
10002	Female	10502	Alfki
10003	Male	10503	Picco
10005	Female	10505	Alfki
10004	Female	10504	Folko
10006	Male	10506	Linod
10007	Male	10507	Simob
Drop below			
10008	Female	10508	Furib
10009	Male	10509	Simob
10010	Female	10510	Picco
10011	Male	10511	Welli

## Customizing row drag-and-drop template

Any type of custom view can be loaded inside the [SfDataGrid.RowDragDropTemplate](#).

Refer to the following code example that shows how to load a view in a template:

**XML**

```
<sfgrid:SfDataGrid.RowDragDropTemplate>
  <DataTemplate>
    <sfgrid:BorderView BackgroundColor="White" BorderColor="Black">
      <sfgrid:BorderView.Content>
        <local:RowTemplate />
      </sfgrid:BorderView.Content>
    </sfgrid:BorderView>
  </DataTemplate>
</sfgrid:SfDataGrid.RowDragDropTemplate>
```

**C#**

```
//Row template with a custom view representing rows.
public class RowTemplate : Grid
{
    #region Constructor
    public RowTemplate()
    {
        this.BackgroundColor = Color.White;
        this.Children.Add(CreateLabel("OrderID"));
        this.Children.Add(new BoxView() { Color = Color.Gray, WidthRequest = 1 });
        this.Children.Add(CreateLabel("EmployeeID"));
        this.Children.Add(new BoxView() { Color = Color.Gray, WidthRequest = 1 });
        this.Children.Add(CreateLabel("CustomerID"));
        this.Children.Add(new BoxView() { Color = Color.Gray, WidthRequest = 1 });
        this.Children.Add(CreateLabel("FirstName"));
        this.Children.Add(new BoxView() { Color = Color.Gray, WidthRequest = 1 });
        this.Children.Add(CreateLabel("LastName"));
    }
    #endregion
    #region Private Method
    private ContentView CreateLabel(string Property)
    {
        var label = new Label();
        label.TextColor = Color.Black;
        label.LineBreakMode = LineBreakMode.NoWrap;
        label.FontSize = 12;
        label.HorizontalTextAlignment = TextAlignment.Center;
        label.VerticalTextAlignment = TextAlignment.Center;
        label.SetBinding(Label.TextProperty, Property);
        return new ContentView() { Content = label };
    }
    #endregion
    #region Override Method
    protected override void LayoutChildren(double x, double y, double width,
        double height)
    {
        foreach (var child in Children)
        {
            if (Device.OS == TargetPlatform.Android || Device.OS == TargetPlatform.iOS)

```

```
{
    if (child is ContentView)
        child.Layout(new Rectangle(x, y, (width / ((Children.Count + 1) / 2)) - 0.5,
            height));
    else
        child.Layout(new Rectangle(x, y, 0.5, height));
    }
    else
    {
        if (child is ContentView)
            child.Layout(new Rectangle(x, y, (width / ((Children.Count + 1) / 2)) - 1,
                height));
        else
            child.Layout(new Rectangle(x, y, 1, height));
        }
        x += child.Width;
    }
}
#endregion
}
```

OrderID	Gender	EmployeeID	CustomerID
10001	Male	10501	Simob
10002	Male	10502	Welli
10003	Female	10503	Simob
10004	Male	10504	Simob
10005	Female	10505	Welli
10006	Male	10506	Welli
10007	Female	10507	Welli
10008	Male	10508	Alfki
10009	Male	10509	Merep
10010	Female	10510	Merep
10011	Male	10511	Simob

**Note:** Currently, the row drag-and-drop features cannot be used if different rows sets with different heights using the [QueryRowHeight](#) event.

You can download the customizing row drag-and-drop template sample [here](#).

#### Events in row drag-and-drop

The `QueryRowDragging` event is fired upon starting to drag a row and will be continuously fired until the dragging ends. By handling the [SfDataGrid.QueryRowDragging](#) event, you can also cancel the dragging of a particular row.

The `QueryRowDragging` event provides the following properties in [QueryRowDraggingEventArgs](#):

- [From](#): Returns index of the currently being dragged row.
- [To](#): Returns the dragging index where you try to drop the row.
- [Position](#): Returns current x and y coordinates of the RowDragView.

- [Reason](#): Returns row dragging details as [QueryRowDraggingReason](#).
- [RowData](#): Returns the underlying data associated with the dragged row.
- [CurrentRowData](#): Returns the corresponding row data over which the row drag view is currently placed.
- [CanAutoScroll](#): Returns whether auto-scrolling should happen when row drag view reaches the top or bottom of the [SfDataGrid](#).
- [Cancel](#): Returns the Boolean property to cancel the event.

#### Cancel dragging of a particular row

Dragging of a particular row can be canceled using [QueryRowDraggingReason](#) argument of the [QueryRowDragging](#) event handler.

#### C#

```
this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;
private void SfGrid_QueryRowDragging(object sender,
QueryRowDraggingEventArgs e)
{
    //e.From returns the index of the dragged row.
    //e.Reason returns the dragging status of the row.
    if (e.From == 1 && e.Reason == QueryRowDraggingReason.DragStarted)
        e.Cancel = true;
}
```

#### Cancel dropping when dragging over particular rows

Dropping when dragging over particular rows can be canceled using [QueryRowDraggingReason](#) argument of the [QueryRowDragging](#) event handler.

#### C#

```
this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;
private void SfGrid_QueryRowDragging(object sender,
QueryRowDraggingEventArgs e)
{
    //e.To returns the index of the current row.
    //e.Reason returns the dragging status of the row.
    if ((e.To > 5 || e.To < 10) &&
        (e.Reason == QueryRowDraggingReason.DragEnded || e.Reason ==
        QueryRowDraggingReason.Dragging))
        e.Cancel = true;
}
```

#### Cancel dropping of a particular row

Dropping of a particular row can be canceled using [QueryRowDraggingReason](#) argument of the [QueryRowDragging](#) event handler.

#### C#

```
this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;
private void SfGrid_QueryRowDragging(object sender,
QueryRowDraggingEventArgs e)
{
    //e.From returns the index of the dragged row.
}
```

```
//e.Reason returns the dragging status of the row.  
if (e.From == 1 && e.Reason == QueryRowDraggingReason.DragEnded)  
e.Cancel = true;  
}
```

Cancel dropping at a particular position

Dropping at a particular position can be canceled using `QueryRowDraggingReason` argument of the `QueryRowDragging` event handler.

**C#**

```
this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;  
private void SfGrid_QueryRowDragging(object sender,  
QueryRowDraggingEventArgs e)  
{  
//e.To returns the index of the current row.  
//e.Reason returns the dragging status of the row.  
if ((e.To == 5 || e.To == 7) && e.Reason ==  
QueryRowDraggingReason.DragEnded)  
e.Cancel = true;  
}
```

Cancel dropping of a particular row in a position

Dropping of a particular row in a position can be canceled using `QueryRowDraggingReason` and `Position` arguments of the `QueryRowDragging` event handler.

**C#**

```
this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;  
private void SfGrid_QueryRowDragging(object sender,  
QueryRowDraggingEventArgs e)  
{  
//e.To returns the index of the current row.  
//e.Position returns the x and y position of the current row  
if ((e.To == 3) && e.Position == new Point(927,1167) && e.Reason ==  
QueryRowDraggingReason.DragEnded)  
e.Cancel = true;  
}
```

Cancel drag and drop between frozen and non-frozen rows

*Cancel dragging between frozen and non-frozen rows*

Dragging between frozen and non-frozen rows can be canceled using `QueryRowDraggingReason` and `From` arguments of the `QueryRowDragging` event handler by checking whether the value of `From` argument is a frozen row index.

**C#**

```
SfGrid.FrozenRowsCount = 4;  
this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;  
private void SfGrid_QueryRowDragging(object sender,  
QueryRowDraggingEventArgs e)  
{  
//e.From returns the index of the dragged frozen row.
```

```
//e.To returns the index of the current row.
if (e.From > sfGrid.GetHeaderIndex() && e.From <= sfGrid.FrozenRowCount
&& e.Reason == QueryRowDraggingReason.DragStarted)
e.Cancel = true;
}
```

#### Cancel dropping between frozen and non-frozen rows

Dropping between frozen and non-frozen rows can be canceled using `QueryRowDraggingReason` and `From` arguments of the `QueryRowDragging` event handler by checking whether the value of `From` argument is a frozen row index.

#### C#

```
SfGrid.FrozenRowCount = 4;
this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;
private void SfGrid_QueryRowDragging(object sender,
QueryRowDraggingEventArgs e)
{
//e.From returns the index of the dragged frozen row.
//e.To returns the index of the current row.
if (e.From > sfGrid.GetHeaderIndex() && e.From <= sfGrid.FrozenRowCount &&
e.Reason == QueryRowDraggingReason.DragEnded)
e.Cancel = true;
}
```

---

**Note:** `FrozenRowCount` must be less than rows in view.

---

#### Reorder the underlying data

Reordering changes directly on the underlying data can be done using `QueryRowDraggingReason` argument of the `QueryRowDragging` event handler. Refer to the following code sample to make permanent reordering changes:

#### C#

```
this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;
private void SfGrid_QueryRowDragging(object sender,
QueryRowDraggingEventArgs e)
{
//e.To returns the index of the current row.
//e.From returns the index of the dragged row.
if (e.Reason == QueryRowDraggingReason.DragEnded)
{
var collection = (sender as SfDataGrid).ItemsSource as IList;
collection.RemoveAt(e.From - 1);
collection.Insert(e.To - 1, e.RowData);
//To skip default collection change inside the SfDataGrid source for a
successful drag and drop operation.
e.Cancel = true;
}
}
```



### Drop a grid row in the last position

The **To** property of the **QueryRowDraggingEventArgs** denotes the current drop index of the dragged row when dragging over the grid rows. It returns the same index when dragging a row over the rows in last position or last but one. In order to programmatically track whether the dragged row is dropped at the last position or last but one, the data grid provides the **Position** property in **QueryRowDraggingEventArgs**, which denotes the position of the RowDragView.

Refer to the following code example in which the **Position** property is used to determine whether the row is dropped in the last position or not:

#### C#

```
this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;
private void SfGrid_QueryRowDragging(object sender,
QueryRowDraggingEventArgs e)
{
    var totalHeight = dataGrid.RowColumnIndexToPoint(new
RowColumnIndex(viewModel.OrdersInfo.Count, 0)).Y + this.dataGrid.RowHeight;
    if (e.Reason == QueryRowDraggingReason.DragEnded)
    {
        if (Math.Ceiling(e.Position.Y + (dataGrid.RowHeight / 2)) > totalHeight &&
e.To == viewModel.OrdersInfo.Count)
        {
            //Will hit if the row is dropped at the last position
            DisplayAlert("RowDragAndDrop info", "The row is dropped at the last
position", "OK");
        }
    }
}
```

### Customizing row drag-and-drop indicators

Data grid allows you to customize the row drag-and-drop indicators by writing a custom grid style, deriving from the [DataGridStyle](#) and assigning it to the [SfDataGrid.GridStyle](#) property.

#### C#

```
dataGrid.GridStyle = new CustomGridStyle();
```

#### C#

```
// Custom style class
public class CustomGridStyle : DataGridStyle
{
    public CustomGridStyle()
    {
    }
    public override ImageSource GetRowDragUpIndicator()
    {
        return ImageSource.FromResource("DataGridDemo.icons.RedUp.png");
    }
    public override ImageSource GetRowDragDownIndicator()
    {
        return ImageSource.FromResource("DataGridDemo.icons.RedDown.png");
    }
}
```

OrderID	Gender	CustomerID	Country
10001	Female	Furib	Austria
10002	Female	Merep	India
10003	Female	Blonp	America
10004	Male	Warth	Belgium
10005	Male	Warth	Korea
10006	Female	Alfki	China
10007	Female	Picco	Singapore
10008	Female	Furib	Australia
10009	Male	Alfki	Korea
10010	Male	Merep	China
10011	Male	Folko	US

#### Updating summaries when dragging and dropping a row between groups

Grouping and summarization of items in the data grid are manipulated based on group key. When dragging an item from one group to another group, the group key of the dragged item will differ from the group key of the items in the dropped group. Thus, by default, the summaries will not be updated. This is the actual behavior of this control.

Hence, in order to update the summaries when a row is dragged and dropped between groups, call the `UpdateCaptionSummaries` and the `Refresh` methods in the `QueryRowDragging` event.

#### C#

```
public partial class MainPage : ContentPage
{
    private SfDataGrid dataGrid;
```

```

private ViewModel viewModel;
public MainPage()
{
    InitializeComponent();
    dataGrid = new SfDataGrid();
    viewModel = new ViewModel();
    dataGrid.ItemsSource = viewModel.OrdersInfo;
    dataGrid.QueryRowDragging += DataGrid_QueryRowDragging;
    this.Content = dataGrid;
}
private async void DataGrid_QueryRowDragging(object sender,
QueryRowDraggingEventArgs e)
{
    if (e.Reason == QueryRowDraggingReason.DragEnded)
    {
        // Delay is given for refreshing the view.
        await Task.Delay(100);
        this.dataGrid.View.TopLevelGroup.UpdateCaptionSummaries();
        this.dataGrid.View.Refresh();
    }
}
}

```

The following screenshot shows the output rendered when executing the above code example.

### Before DragAndDrop

OrderID	Gender	EmployeeID	CustomerID ↑
Total Salary:2'			
10010	Male	10510	Alfki
10013	Female	10513	Alfki
Total Salary:1'			
10019	Male	10519	Folko
Total Salary:4'			
10003	Female	10503	Frans
10011	Female	10511	Frans
10017	Male	10517	Frans
10018	Male	10518	Frans
Total Salary:4'			
10002	Male	10502	Furib
10007	Male	10507	Furib
10014	Female	10514	Furib

Drag and Drop  
to reorder the  
values

### After DragAndDrop

OrderID	Gender	EmployeeID	CustomerID ↑
Total Salary:1'			
10010	Male	10510	Alfki
Total Salary:2'			
10013	Female	10513	Alfki
10019	Male	10519	Folko
Total Salary:4'			
10003	Female	10503	Frans
10011	Female	10511	Frans
10017	Male	10517	Frans
10018	Male	10518	Frans
Total Salary:4'			
10002	Male	10502	Furib
10007	Male	10507	Furib
10014	Female	10514	Furib

### Cancel auto scrolling

Vertical auto-scrolling of the `SfDataGrid` during row drag and drop can be canceled using `CanAutoScroll` argument of the `QueryRowDragging` event handler.

**C#**

```

this.SfGrid.QueryRowDragging += SfGrid_QueryRowDragging;
private void SfGrid_QueryRowDragging(object sender,
QueryRowDraggingEventArgs e)
{
    // Disable scroll while dragging the Rows.
    e.CanAutoScroll = false;
}

```

**Swiping**

The

[SfDataGrid](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid.html# "") allows enabling the swiping option by setting the [SfDataGrid.AllowSwiping](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~AllowSwiping.html# "") property to true. Swipe views are displayed when swiping from 'left to right' or 'right to left' on a data row. The control provides customizable swipe templates for swiping on the left and right side. The swipe gesture can be restricted to a certain point on the row by setting the [SfDataGrid.MaxSwipeOffset](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~MaxSwipeOffset.html# "") property.

**Swipe template**

The data grid enables loading a desired content using the

[SfDataGrid.LeftSwipeTemplate](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~LeftSwipeTemplate.html# "") when swiping towards right. The template can be defined either in code or XAML. The content inside the swipe template is arranged based on the offset values when swiping a data row.

**XAML**

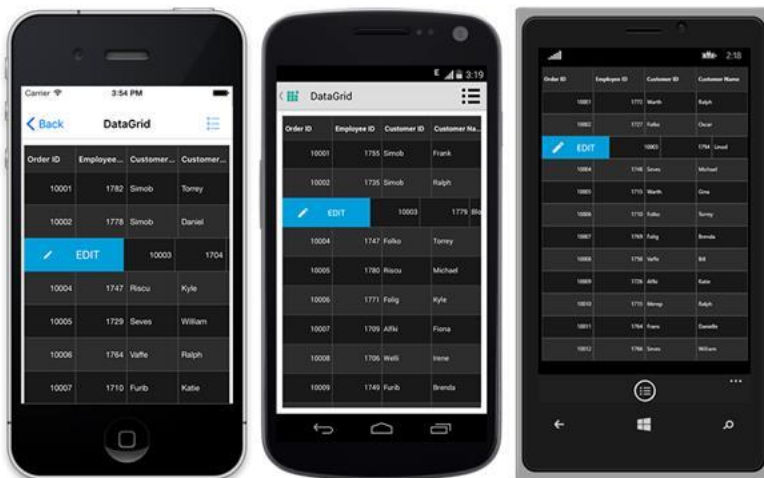
```

//Defining left swipe template
<sfgrid:SfDataGrid.LeftSwipeTemplate>
<DataTemplate>
<Grid BackgroundColor="#009EDA" Padding="9">
<Grid.ColumnDefinitions>
<ColumnDefinition/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<Image Grid.Column="0"
BackgroundColor="Transparent"
HorizontalOptions="CenterAndExpand"
Source="EditIcon.png"
BindingContextChanged="rightImage_BindingContextChanged" />
<Label Grid.Column="1"
Text ="EDIT"
HorizontalTextAlignment="Start"
VerticalTextAlignment="Center"
LineBreakMode ="NoWrap"
BackgroundColor="Transparent"
TextColor ="White" />
</Grid>
</DataTemplate>
</sfgrid:SfDataGrid.LeftSwipeTemplate>

```

**C#**

```
//Defining left swipe template
dataGrid.LeftSwipeTemplate = new DataTemplate(() =>
{
    Grid myGrid = new Grid();
    myGrid.HorizontalOptions = LayoutOptions.FillAndExpand;
    myGrid.BackgroundColor = Color.FromHex("#009EDA");
    myGrid.Padding = 9;
    myGrid.ColumnDefinitions = new ColumnDefinitionCollection
    {
        new ColumnDefinition {},
        new ColumnDefinition {}
    };
    var image = new Image();
    image.BackgroundColor = Color.Transparent;
    image.BindingContextChanged += MainPage_BindingContextChanged;
    image.HorizontalOptions = LayoutOptions.FillAndExpand;
    image.Source = "EditIcon.png";
    var label = new Label();
    label.Text = "EDIT";
    label.HorizontalTextAlignment = TextAlignment.Start;
    label.VerticalTextAlignment = TextAlignment.Center;
    label.LineBreakMode = LineBreakMode.NoWrap;
    label.BackgroundColor = Color.Transparent;
    label.TextColor = Color.White;
    myGrid.Children.Add(image, 0, 0);
    myGrid.Children.Add(label, 1, 0);
    return myGrid;
});
```



**Note:** Similarly, desired content can be loaded using the [SfDataGrid.RightSwipeTemplate](http://help.syncfusion.com/cr/cref\_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~RightSwipeTemplate.html# "") when swiping towards left.

---

**Note:** The `DataTemplateSelector` can also be directly assigned to the `SfDataGrid.RightSwipeTemplate` and `SfDataGrid.LeftSwipeTemplate`. You can load the desired template using the `RowData` and the row element passed in the arguments.

---

### Swipe events

`[SwipeStarted]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~SwipeStartedEV.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~SwipeStartedEV.html#)): Fired when the swipe offset changes from its initial value. The swipe action can be canceled by setting the

`[Cancel]`([https://msdn.microsoft.com/query/dev10.query?appId=Dev10IDEF1&l=EN-US&k=k\(System.ComponentModel.CancelEventArgs.Cancel\)&rd=true#""](https://msdn.microsoft.com/query/dev10.query?appId=Dev10IDEF1&l=EN-US&k=k(System.ComponentModel.CancelEventArgs.Cancel)&rd=true#)) property of the `[SwipeStartedEventArgs]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipeStartedEventArgs.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipeStartedEventArgs.html#)) to `true`.

`[SwipeEnded]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~SwipeEndedEV.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~SwipeEndedEV.html#)): Fired when the swipe offset value reaches the `SfDataGrid.MaxSwipeOffset` indicating that the swipe action is completed. This event is triggered with

`[SwipeEndedEventArgs]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipeEndedEventArgs.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipeEndedEventArgs.html#)).

`[Swiping]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~SwipingEV.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~SwipingEV.html#)): Raised while swiping a row is in progress. This event is triggered with

`[SwipingEventArgs]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs.html#)).

The swipe events provides the following properties in their arguments:

- `[RowIndex]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs~RowIndex.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs~RowIndex.html#)): Defines the swiping row index.
- `[RowData]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs~RowData.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs~RowData.html#)): Defines the underlying data associated with the swiped row as its arguments.
- `[SwipeDirection]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs~SwipeDirection.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs~SwipeDirection.html#)): Defines the swipe direction of the swiped row.
- `[SwipeOffset]`([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs~SwipeOffset.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SwipingEventArgs~SwipeOffset.html#)): Defines the current swipe offset of the row being swiped.

By handling the swipe events, you can use these properties value from the arguments to perform any desired action such as deleting the row, editing the data, etc.

### Loading multiple views as swipe template

The swipe templates can be customized by loading any view in the templates, and assigning custom actions to them such as deleting a row, adding a row, editing the underlying data associated, etc. Multiple views can also be displayed in a template like in the following example, where two views are loaded for editing the cell values in the row and deleting the row respectively.

**Swiping.xaml****XML**

```
//Defining left swipe template
<sfgrid:SfDataGrid.LeftSwipeTemplate>
<DataTemplate>
<Grid BackgroundColor="#009EDA">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="50"/>
<ColumnDefinition Width="100"/>
<ColumnDefinition Width="130"/>
</Grid.ColumnDefinitions>
<Image Grid.Column="0"
BackgroundColor="Transparent"
BindingContextChanged="leftImage_BindingContextChanged"
HorizontalOptions="CenterAndExpand"
Source="EditIcon.png" />
<Label Grid.Column="1"
BackgroundColor="Transparent"
LineBreakMode="NoWrap"
Text="EDIT"
TextColor="White"
HorizontalTextAlignment ="Start"
VerticalTextAlignment="Center" />
<Grid Grid.Column="2" BackgroundColor="#DC595F">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="50" />
<ColumnDefinition Width="80" />
</Grid.ColumnDefinitions>
<Image Grid.Column="0"
BackgroundColor="Transparent"
BindingContextChanged="rightImage_BindingContextChanged"
HorizontalOptions="CenterAndExpand"
Source="TrashImage.png" Margin="9"/>
<Label Grid.Column="1"
BackgroundColor="Transparent"
LineBreakMode="NoWrap"
Text="DELETE"
TextColor="White"
HorizontalTextAlignment ="Start"
VerticalTextAlignment ="Center" />
</Grid>
</Grid>
</DataTemplate>
</sfgrid:SfDataGrid.LeftSwipeTemplate>
```

**Swiping.xaml.cs****C#**

```
public partial class Swiping : SamplePage
{
private Image leftImage;
private Image rightImage;
private int swipedRowIndex;
private FormsView formView;
```

```
public Swiping()
{
    InitializeComponent();
    this.PropertyChanged += Swiping_PropertyChanged;
    formView = new FormsView(dataGrid);
    customLayout.Children.Add(formView);
}

private void Swiping_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
    if (e.PropertyName == "Width" && this.formView != null)
    {
        formView.IsVisible = false;
        dataGrid.Opacity = 1.0;
    }
}

private void dataGrid_GridTapped(object sender, GridTappedEventArgs e)
{
    this.formView.IsVisible = false;
    dataGrid.Opacity = 1.0;
    dataGrid.IsEnabled = true;
}

//Gesture listener to perform edit
private void leftImage_BindingContextChanged(object sender, EventArgs e)
{
    if (leftImage == null)
    {
        leftImage = sender as Image;
        (leftImage.Parent as View).GestureRecognizers.Add(new TapGestureRecognizer()
        { Command = new Command(Edit)});
        leftImage.Source =
        ImageSource.FromResource("SfDataGridSample.EditIcon.png");
    }
}

//Code to edit row
private void Edit()
{
    if (Device.RuntimePlatform != Device.UWP || Device.Idiom ==
    TargetIdiom.Phone)
    formView.LayoutTo(new Rectangle(10, (this.Height / 2) - (350 / 2),
    this.dataGrid.Width - 20, 370), 250, null);
    else
    formView.LayoutTo(new Rectangle(10, (this.dataGrid.Height / 2) - (350 / 2),
    this.dataGrid.Width - 20, 350), 450, null);
    formView.IsVisible = true;
}

//Code to delete row
private void Delete()
{
    this.viewModel.OrdersInfo.RemoveAt(swipedRowIndex - 1);
}

//Gesture listener to perform delete
private void rightImage_BindingContextChanged(object sender, EventArgs e)
{
    if (rightImage == null)
    {
        rightImage = sender as Image;
```



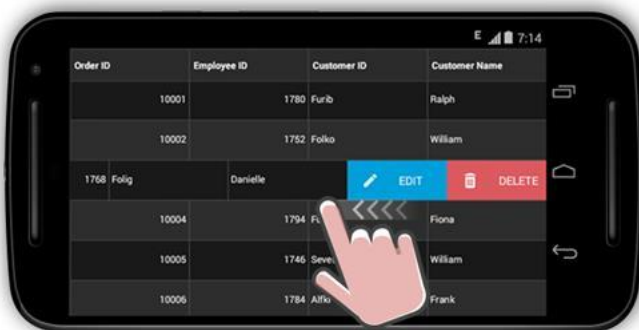
```

(rightImage.Parent as View).GestureRecognizers.Add(new
TapGestureRecognizer() { Command = new Command(Delete) });
rightImage.Source =
ImageSource.FromResource("SfDataGridSample.TrashImage.png");
}
}
private void dataGrid_SwipeEnded(object sender, SwipeEndedEventArgs e)
{
formView.BindingContext = e.RowData;
swipedRowIndex = e.RowIndex;
}
}

```



**Note:** Similarly, you can load two views using the `SfDataGrid.RightSwipeTemplate` when swiping towards left will result in the following outcome:



### Customized swipe delete functionality

Operations such as deleting a row when swiping a data row from one extent to other in the view can be performed.

### Swiping.Xaml

**XML**

```
//Creating view for left swipe
<sfgrid:SfDataGrid.LeftSwipeTemplate>
<DataTemplate>
<ContentView BindingContextChanged="leftTemplate_BindingContextChanged"
BackgroundColor="#1AAA87">
<Label FontSize="15"
Text ="Deleted"
TextColor ="White"
HorizontalTextAlignment ="Start"
VerticalTextAlignment ="Center"
LineBreakMode ="NoWrap" />
</ContentView>
</DataTemplate>
</sfgrid:SfDataGrid.LeftSwipeTemplate>
//Creating view for right swipe
<sfgrid:SfDataGrid.RightSwipeTemplate>
<DataTemplate>
<ContentView BindingContextChanged="rightTemplate_BindingContextChanged"
BackgroundColor="#1AAA87">
<Label FontSize="15"
HorizontalTextAlignment ="Center"
Text ="Deleted"
TextColor ="White"
VerticalTextAlignment ="Center"
LineBreakMode ="NoWrap" />
</ContentView>
</DataTemplate>
</sfgrid:SfDataGrid.RightSwipeTemplate>
```

Swiping.Xaml.cs

**C#**

```
//Call to delete() when swipe is ended
private void dataGrid_SwipeEnded(object sender, SwipeEndedEventArgs e)
{
    swipedRowIndex = e.RowIndex;
    if (Math.Abs(e.SwipeOffset) >= dataGrid.Width)
    {
        if (e.SwipeOffset > 0)
            leftTemplateView.Content.IsVisible = true;
        else
            rightTemplateView.Content.IsVisible = true;
        doDeleting();
    }
}
//Code to delete row
private async void doDeleting()
{
    await Task.Delay(2000);
    if (leftTemplateView.Content.IsVisible)
        leftTemplateView.Content.IsVisible = false;
    else if (rightTemplateView.Content.IsVisible)
        rightTemplateView.Content.IsVisible = false;
    if (!IsUndoClicked)
```

```
viewModel.OrdersInfo.RemoveAt (swipedRowIndex - 1);
else
{
    var removedData = viewModel.OrdersInfo[swipedRowIndex - 1];
    var isSelected = dataGrid.SelectedItems.Contains(removedData);
    viewModel.OrdersInfo.Remove(removedData);
    viewModel.OrdersInfo.Insert(swipedRowIndex - 1, removedData);
    if (isSelected)
    {
        dataGrid.SelectedItems.Add(removedData);
        IsUndoClicked = false;
    }
}
```



### Loading complex template for swiping

When a complex layout is loaded in the `GridTemplateColumn`, swiping may not occur or may not be smooth at times, since the touches will be intercepted by the views loaded in the template. As a result, the data grid may not recognize the touches that might restrict the swiping operation. To resolve this issue, set the `InputTransparent` property of the views loaded in the `DataTemplate` of the `GridTemplateColumn` to `True`.

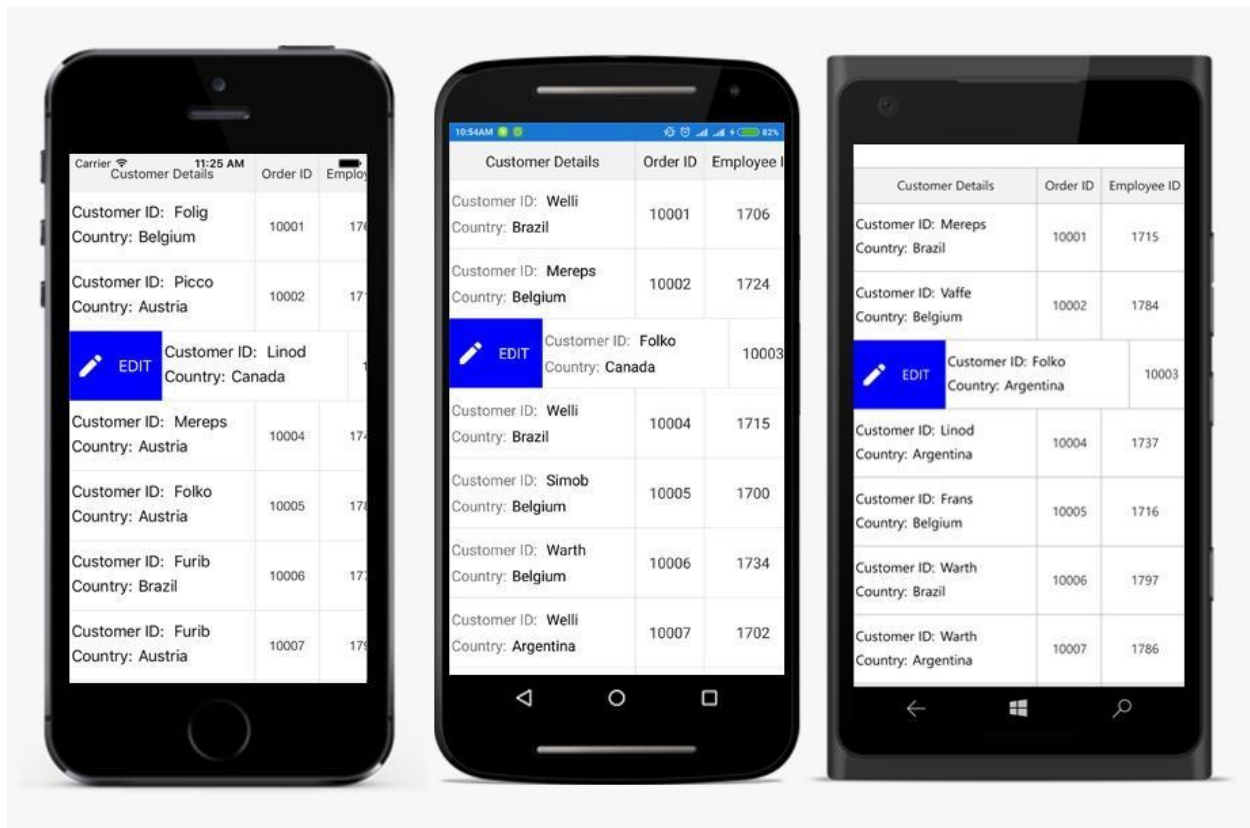
### XML

```
<sfgrid:GridTemplateColumn MappingName="CustomerID" HeaderText="Customer
Details" Width="200">
<sfgrid:GridTemplateColumn.CellTemplate>
<DataTemplate>
<Grid InputTransparent="True"
ColumnSpacing="0"
RowSpacing="0"
HorizontalOptions="FillAndExpand"
Margin="0" >
<Grid.ColumnDefinitions>
<ColumnDefinition />
<ColumnDefinition />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="10" />
```

```

<RowDefinition Height="*" />
<RowDefinition Height="10" />
</Grid.RowDefinitions>
<StackLayout InputTransparent="True"
Margin="2"
BackgroundColor="White"
Orientation="Vertical"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand"
Grid.Column="0"
Grid.ColumnSpan="3"
Grid.Row="1">
<StackLayout Orientation="Horizontal" HorizontalOptions="FillAndExpand" >
<Label Text="Customer ID: " HorizontalTextAlignment="Start"/>
<Label Text="{Binding CustomerID}" TextColor="Black"
HorizontalTextAlignment="Start"/>
</StackLayout>
<StackLayout Orientation="Horizontal" HorizontalOptions="FillAndExpand">
<Label Text="Country:" HorizontalTextAlignment="Start"/>
<Label Text="{Binding ShipCountry}" TextColor="Black"
HorizontalTextAlignment="Start"/>
</StackLayout>
</StackLayout>
</Grid>
</DataTemplate>
</sfgrid:GridTemplateColumn.CellTemplate>
</sfgrid:GridTemplateColumn>

```



How to cancel the swipe programmatically

The data grid allows canceling the swipe programmatically by calling the [SfDataGrid.ResetSwipeOffset](#) method in [SfDataGrid.SwipeEnded](#) event.

#### XML

```
<syncfusion:SfDataGrid x:Name="datagrid"
    ColumnSizer="Star"
    AutoGenerateColumns="True"
    AllowSwiping="True"
    ItemsSource="{Binding OrdersInfo}"
    SwipeEnded="Datagrid_SwipeEnded">
    <syncfusion:SfDataGrid.LeftSwipeTemplate>
    <DataTemplate>
    <Grid BackgroundColor="Blue" Padding="9">
    <Label Text ="EDIT"
        HorizontalTextAlignment="Start"
        VerticalTextAlignment="Center"
        LineBreakMode ="NoWrap"
        BackgroundColor="Transparent"
        TextColor ="White" />
    </Grid>
    </DataTemplate>
    </syncfusion:SfDataGrid.LeftSwipeTemplate>
    <syncfusion:SfDataGrid.RightSwipeTemplate>
    <DataTemplate>
    <Grid BackgroundColor="Red" Padding="9">
    <Label FontSize="15"
        HorizontalTextAlignment ="Center"
        Text ="Deleted"
        TextColor ="White"
        VerticalTextAlignment ="Center"
        LineBreakMode ="NoWrap" />
    </Grid>
    </DataTemplate>
    </syncfusion:SfDataGrid.RightSwipeTemplate>
</syncfusion:SfDataGrid>
```

#### C#

```
private void Datagrid_SwipeEnded(object sender, SwipeEndedEventArgs e)
{
    datagrid.ResetSwipeOffset();
}
```

You can download the source code of swiping sample [here](#).

#### Limitations

When data grid is loaded in MasterDetailPage with `AllowSwiping` as true, it behaves as follows:

- In iOS platform, when swiping a data row, touch and hold the row for some fraction of seconds (0.25 - 0.5 seconds) and then swipe.

## Styles

The data grid applies style for all of its elements by writing a Style class overriding from [DataGridStyle](#), and assigning it to the [SfDataGrid.GridStyle](#) property.

To apply custom style, follow the code example:

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
"
xmlns:local ="clr-namespace:DataGridSample;assembly=DataGridSample"
x:Class="DataGridSample.Sample">
<ContentPage.Resources>
<ResourceDictionary>
<local:Dark x:Key="dark" />
</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
GridStyle="{StaticResource dark}"
ItemsSource="{Binding OrdersInfo}" />
</ContentPage>
```

### C#

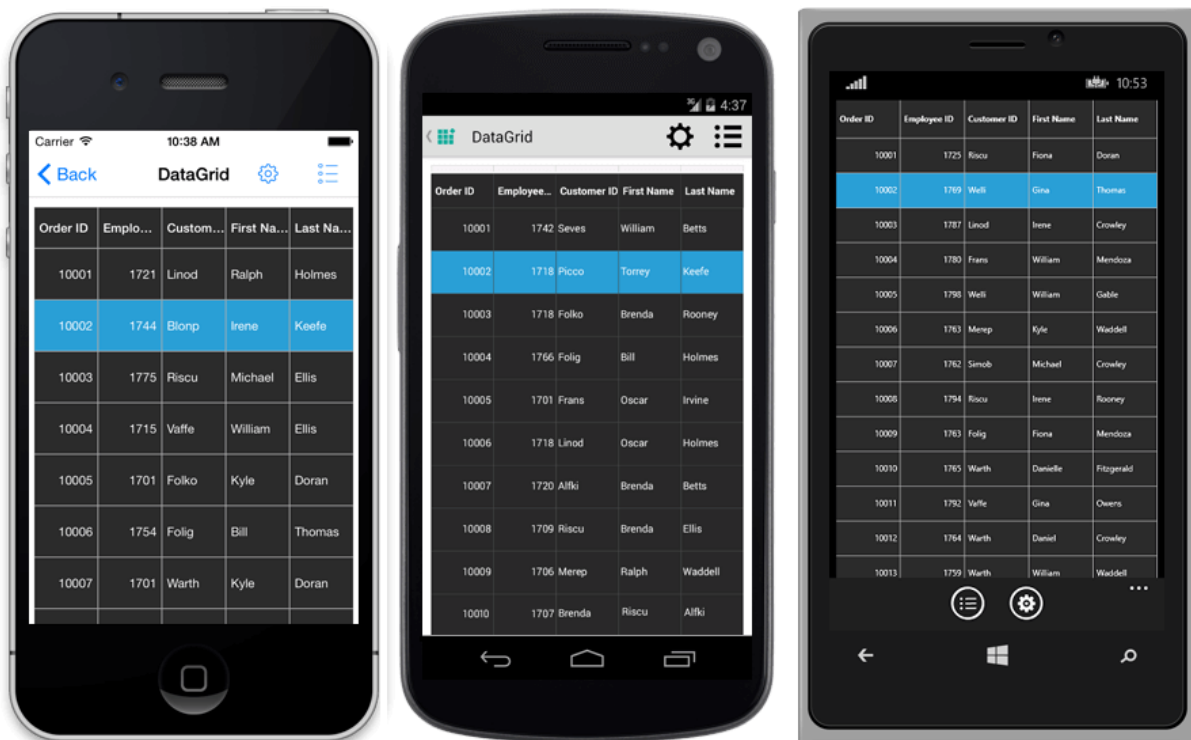
```
//Apply custom style to SfDataGrid from code
SfDataGrid dataGrid = new SfDataGrid();
dataGrid.GridStyle = new Dark ();
```

### C#

```
//Custom style class
public class Dark : DataGridStyle
{
    public Dark ()
    {
    }
    public override Color GetHeaderBackgroundColor()
    {
        return Color.FromRgb (15, 15, 15);
    }
    public override Color GetHeaderForegroundColor()
    {
        return Color.FromRgb (255, 255, 255);
    }
    public override Color GetRecordBackgroundColor ()
    {
        return Color.FromRgb (43, 43, 43);
    }
    public override Color GetRecordForegroundColor ()
    {
        return Color.FromRgb (255, 255, 255);
    }
    public override Color GetSelectionBackgroundColor ()
```

```
{  
    return Color.FromRgb (42, 159, 214);  
}  
public override Color GetSelectionForegroundColor ()  
{  
    return Color.FromRgb (255, 255, 255);  
}  
public override Color GetCaptionSummaryRowBackgroundColor ()  
{  
    return Color.FromRgb (02, 02, 02);  
}  
public override Color GetCaptionSummaryRowForegroundColor ()  
{  
    return Color.FromRgb (255, 255, 255);  
}  
public override Color GetBorderColor ()  
{  
    return Color.FromRgb (81, 83, 82);  
}  
public override Color GetLoadMoreViewBackgroundColor ()  
{  
    return Color.FromRgb(242, 242, 242);  
}  
public override Color GetLoadMoreViewForegroundColor ()  
{  
    return Color.FromRgb(34, 31, 31);  
}  
public override Color GetAlternatingRowBackgroundColor()  
{  
    return Color.Yellow;  
}  
}
```

The following picture shows the grid loaded in a different style:



**Note:** Xamarin.Forms.Style which has specified target type, that will not be applied to the internal components used in the SfDataGrid.

### Applying alternate row style

The SfDataGrid applies the alternative row style by writing a Style class deriving from `DataGridStyle`, and assigning it to the `SfDataGrid.GridStyle` property.

To apply alternate row style, follow the code example:

### C#

```
//Apply alternative row style
dataGrid.GridStyle = new CustomStyle ();
// Custom style class
public class CustomGridStyle : DataGridStyle
{
    public CustomGridStyle()
    {
    }
    public override Color GetAlternatingRowBackgroundColor()
    {
        return Color.Gray;
    }
}
```





OrderID	CustomerID	FirstName	ShipCountry	Freight
10001	10001	Brenda	Austria	748.59
10002	10002	Bill	Austria	655.22
10003	10003	Daniel	Austria	902.32
10004	10004	Daniel	Brazil	721.38
10005	10005	Daniel	Brazil	288.68
10006	10006	Bill	Canada	10.55
10007	10007	Frank	Belgium	343.44
10008	10008	William	Argentina	363.83
10009	10009	Ralph	Argentina	208.89
10010	10010	Gina	Austria	545.95

### Customizing the alternation count

The data grid customizes the alternate row count for applying the alternate row style using the [SfDataGrid.AlternationCount](#) property.

To set the alternate row count, follow the code example:

#### **C#**

```
//Apply alternative row count  
dataGrid.AlternationCount = 3;
```



OrderID	CustomerID	FirstName	ShipCountry	Freight
10001	10001	Irene	Argentina	169.86
10002	10002	Oscar	Belgium	492.24
10003	10003	Katie	Austria	630.33
10004	10004	Oscar	Austria	100.17
10005	10005	Michael	Belgium	263.67
10006	10006	Gina	Brazil	622.94
10007	10007	Daniel	Austria	501.2
10008	10008	Ralph	Argentina	89.81
10009	10009	Fiona	Argentina	690.84
10010	10010	Danielle	Austria	578.58

### Border customization

The data grid customizes the grid borders to vertical, horizontal, both, or none. Override the [DataGridStyle.GetGridLinesVisibility](#) method to customize borders in the data grid.

#### C#

```
//Apply custom style to SfDataGrid from code  
dataGrid.GridStyle = new CustomStyle ();
```

#### C#

```
//Custom Style class  
public class CustomStyle : DataGridStyle  
{  
    public CustomStyle ()  
    {  
    }  
    public override GridLinesVisibility GetGridLinesVisibility()  
    {  
        return base.GetGridLinesVisibility();  
    }  
}
```

Following are the list of options available to customize the grid borders:

- Both
- Horizontal
- Vertical
- None

#### *Both*

The [GridLinesVisibility.Both](#) displays the data grid with both horizontal and vertical borders.

#### **C#**

```
public override GridLinesVisibility GetGridLinesVisibility()  
{  
    return GridLinesVisibility.Both;  
}
```

The following screenshot shows the outcome upon execution of the above code:



#### *Horizontal*

The [GridLinesVisibility.Horizontal](#) allows displays the data grid with horizontal border only.

#### **C#**

```
public override GridLinesVisibility GetGridLinesVisibility()  
{  
    return GridLinesVisibility.Horizontal;  
}
```

The following screenshot shows the outcome upon execution of the above code:



### Vertical

The [GridLinesVisibility.Vertical](#) displays the data grid with vertical border only.

### C#

```
public override GridLinesVisibility GetGridLinesVisibility()
{
    return GridLinesVisibility.Vertical;
}
```

The following screenshot shows the outcome upon execution of the above code:



*None*

[GridLinesVisibility.None](#) allows you to display the data grid without borders.

**C#**

```
public override GridLinesVisibility GetGridLinesVisibility()
{
    return GridLinesVisibility.None;
}
```

The following screenshot shows the outcome upon execution of the above code:



### Header border color customization

The data grid customizes the header border color for different `DataGridStyle.GridLinesVisibility` by writing a custom style class deriving from `DataGridStyle`, and assigning it to the `SfDataGrid.GridStyle` property. Override the `GetHeaderBorderColor` method in the custom style to customize color of the column header and row header.

To customize the header border color by writing a custom style, follow the code example:

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
xmlns:local="clr-namespace:DataGridSample;assembly=DataGridSample"
x:Class="DataGridSample.Sample">
<ContentPage.Resources>
<ResourceDictionary>
<local:CustomStyle x:Key="customStyle" />
</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
GridStyle="{StaticResource customStyle}"
ItemsSource="{Binding OrdersInfo}" />
</ContentPage>
```

**C#**

```
//Applying custom style to SfDataGrid from code to customize header border color
SfDataGrid dataGrid = new SfDataGrid();
dataGrid.GridStyle = new customStyle ();
```

**C#**

```
//Custom style class
public class CustomStyle : DataGridStyle
{
    Public CustomStyle()
    {
    }
    Public override color GetHeaderBorderColor()
    {
        return Color.Red;
    }
}
```

The following screenshot shows the final outcome upon execution of the above code:

Order ID	Customer ID	Name	City
10001	1779	Alfki	Montréal
10002	1780	Picco	Bruxelles
10003	1769	Alfki	Tsawassen
10004	1775	Frans	Graz
10005	1766	Frans	Bruxelles
10006	1778	Folko	Campinas
10007	1768	Seves	Campinas
10008	1756	Furib	Bruxelles
10009	1754	Vaffe	Rosario
10010	1709	Vaffe	Graz

## Customizing sort icons in the header

Any desired image can be loaded as the sort indicator using the [GetHeaderSortIndicatorDown](#) and [GetHeaderSortIndicatorUp](#) overriding from the [DataGridStyle](#) class. To change the sort indicators, follow the code example:

**C#**





```
//Apply custom style to SfDataGrid from code
dataGrid.GridStyle = new Custom();
```

```

public class Custom : DataGridStyle
{
    public override ImageSource GetHeaderSortIndicatorDown()
    {
        return ImageSource.FromResource("SfDataGrid_Sample.SortDown.png");
    }
    public override ImageSource GetHeaderSortIndicatorUp()
    {
        return ImageSource.FromResource("SfDataGrid_Sample.SortUp.png");
    }
}

```

The following screenshots shows the final outcome of the above code:

Ascending Sort (SortUp)		Descending Sort (SortDown)	
Order ID	 Employee ID 	Order ID	 Employee ID 
1	6	99	104
10	15	98	103
100	105	97	102
11	16	96	101
12	17	95	100
13	18	94	99

**Note:** The BuildAction image must be set to EmbeddedResource in order to access the image as resource as shown in above code.



### Customizing resizing indicator

The color of the resizing indicator can be changed using the [GetResizingIndicatorColor](#) overriding from the [DataGridStyle](#) class. To change the color of the resizing indicator, follow the code example:

#### C#

```
//Apply custom style to SfDataGrid from code
dataGrid.GridStyle = new Custom();
public class Custom : DataGridStyle
{
    public override Color GetResizingIndicatorColor()
    {
        return Color.Blue;
    }
}
```

### Border width customization

SfDataGrid allows you to customize the border width of the grid cells and the header cells.

The default border width of the grid cell and the header cell for Forms.iOS and Forms.UWP is 0.5f and 1f for Forms.Android.

Refer the below code snippet to customize the width of the grid cells and header cells.

#### C#

```
//Apply custom style to SfDataGrid from code
dataGrid.GridStyle = new CustomStyle();
public class CustomStyle : DataGridStyle
{
    public CustomStyle()
    {
    }
    // Customize border width for grid cells
    public override float GetBorderWidth()
    {
        return 5;
    }
    // Customize border width for header cells
    public override float GetHeaderBorderWidth()
    {
        return 5;
    }
}
```

### Conditional Styles

The SfDataGrid allows to customize the style of the individual cells and rows based on the requirements. It can be customized in the following ways:

- Using Column CellStyle
- Using QueryCellStyle Event
- Using QueryRowStyle Event

### Styling cells using column CellStyle

The SfDataGrid allows to apply cell style for a [GridColumn](#) which is used to render the cells in that column. While applying cell style, the [GridColumn](#) appears in the custom style should be the default one. To apply cell style for a GridColumn using the [CellStyle](#), follow the code example:

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="Freight" Format="C">
<syncfusion:GridTextColumn.CellStyle>
<Style TargetType="syncfusion:GridColumn">
<Setter Property="Foreground" Value="Red" />
</Style>
</syncfusion:GridTextColumn.CellStyle>
</syncfusion:GridTextColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### Styling cells using converter

The SfDataGrid also allows to apply styles for the [GridColumn](#) in a column based on conditions by writing a converter for the property in a [GridColumn](#) for which conditional styles should be applied.

To apply conditional styling for a column by writing converter, follow the code example:

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
xmlns:local="clr-namespace:DataGridSample;assembly=DataGridSample"
x:Class="DataGridSample.Sample">
<ContentPage.Resources>
<ResourceDictionary>
<local:CellStyleConverter x:Key="cellStyleConverter" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="Freight" Format="C">
<syncfusion:GridTextColumn.CellStyle>
<Style TargetType="syncfusion:GridColumn">
<Setter Property="BackgroundColor"
Value="{Binding Freight,
Converter={StaticResource cellStyleConverter}}" />
<Setter Property="Foreground" Value="Red" />
</Style>
</syncfusion:GridTextColumn.CellStyle>
</syncfusion:GridTextColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

```

</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
</ContentPage>

```

## C#

```

public class CellStyleConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        if (System.Convert.ToDouble(value) < 300)
            return Color.White;
        return Color.Green;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return null;
    }
}

```



### Styling cells using QueryCellStyle event

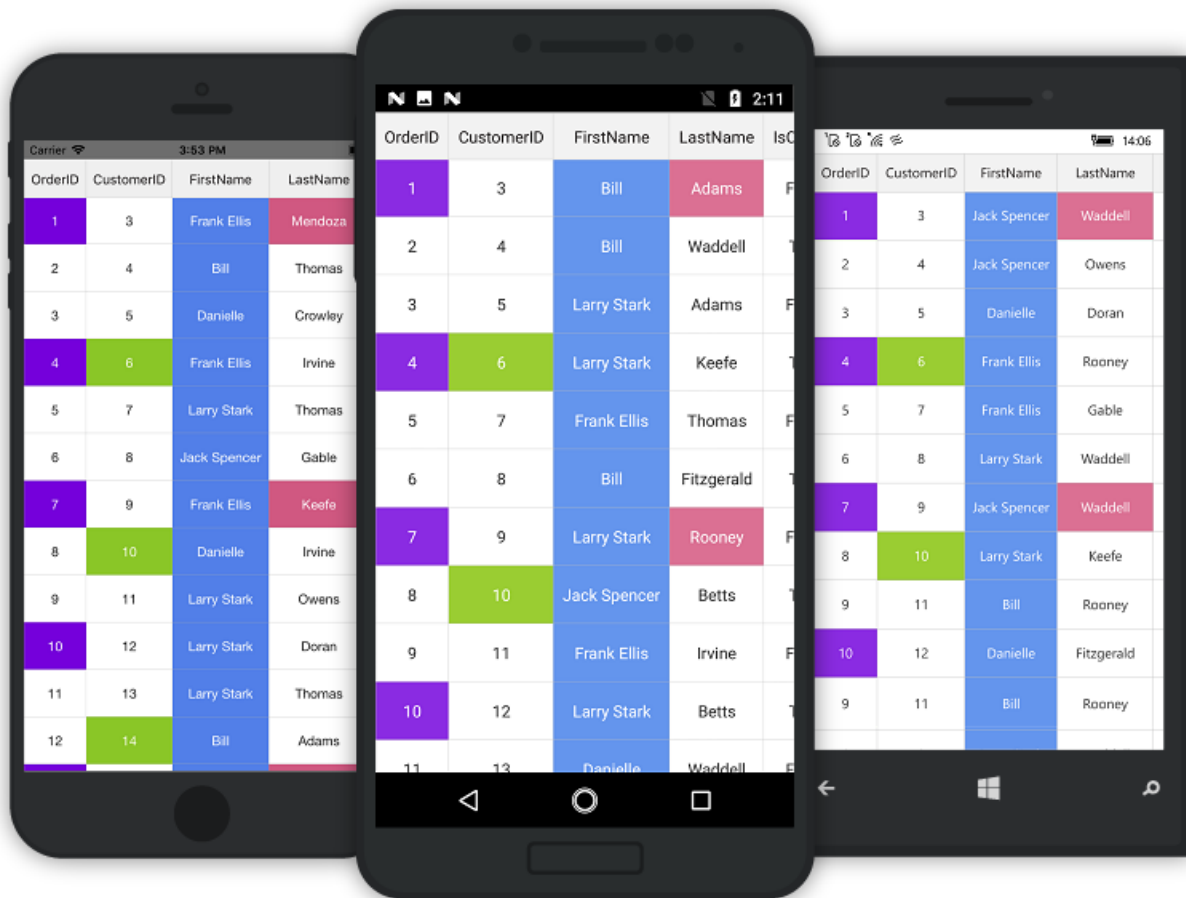
The conditional style can be applied for any cell using the [QueryCellStyle](#) event based on any condition. This event will be fired for each cell. It provides the following properties through the [QueryCellStyleEventArgs](#) in its `EventHandler`:

- [RowIndex](#): Provides the row index of current cell in iteration.

- [ColumnIndex](#): Provides the column index of current cell in iteration.
- [CellValue](#): Provides the cell value of current cell in iteration.
- [Column](#): Provides the [GridColumn](#) which belongs to current cell in iteration.
- [e.Handled](#): Should set to true to apply the changes.
- [Style](#): Sets style to the current cell in iteration.

**C#**

```
this.dataGrid.QueryCellStyle += DataGrid_QueryCellStyle;
private void DataGrid_QueryCellStyle(object sender, QueryCellStyleEventArgs e)
{
    if (e.ColumnIndex == 0 && e.RowIndex % 3 == 1)
    {
        e.Style.BackgroundColor = Color.BlueViolet;
        e.Style.ForegroundColor = Color.White;
    }
    else if (e.Column.MappingName == "FirstName")
    {
        e.Style.BackgroundColor = Color.CornflowerBlue;
        e.Style.ForegroundColor = Color.White;
    }
    else if (e.ColumnIndex == 1 && e.RowIndex % 4 == 0)
    {
        e.Style.BackgroundColor = Color.YellowGreen;
        e.Style.ForegroundColor = Color.White;
    }
    else if (e.ColumnIndex == 3 && e.RowIndex % 6 == 1)
    {
        e.Style.BackgroundColor = Color.PaleVioletRed;
        e.Style.ForegroundColor = Color.White;
    }
    e.Handled = true;
}
```

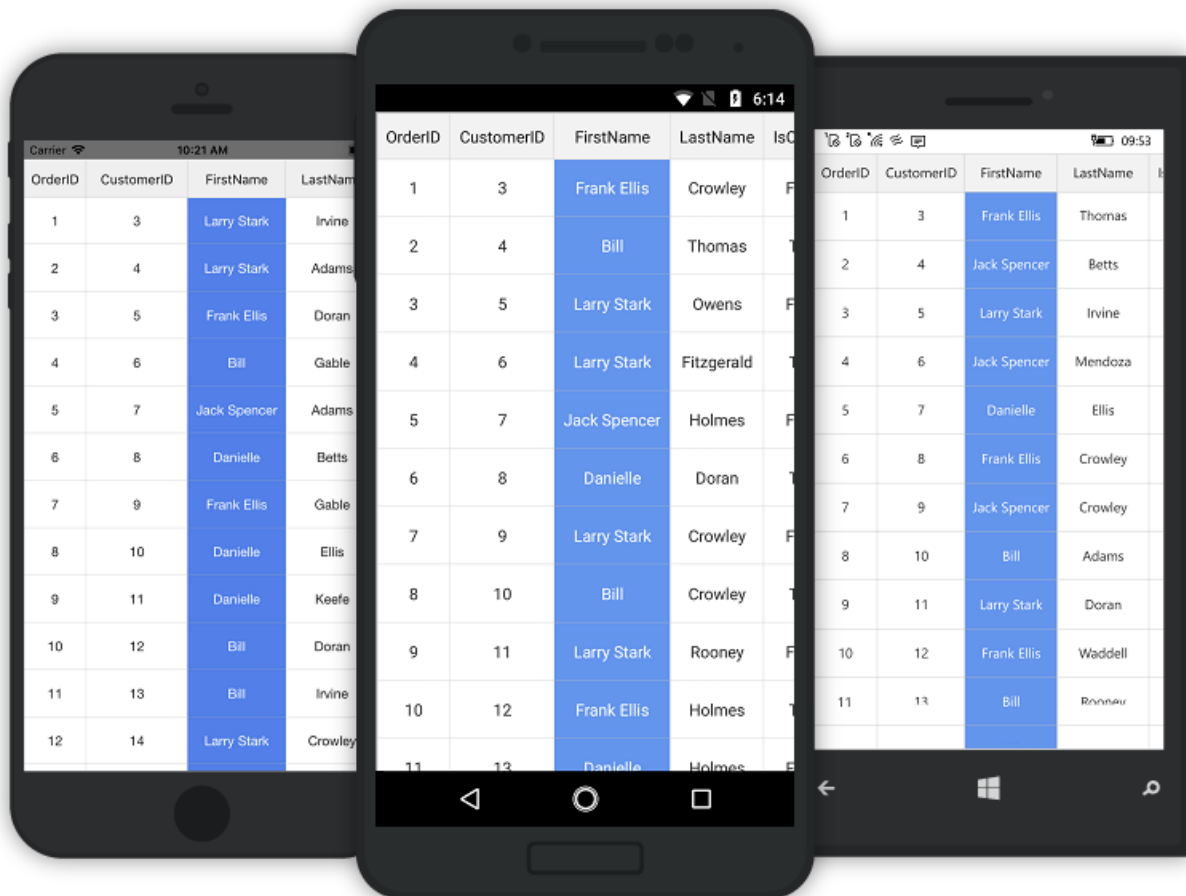


### How to style a particular column

Based on the properties of the `Column` provided in the `QueryCellStyleEventArgs` of the `QueryCellStyle` event, style can be applied to a particular column.

#### C#

```
private void DataGrid_QueryCellStyle(object sender, QueryCellStyleEventArgs e)
{
    if (e.Column.MappingName == "FirstName")
    {
        e.Style.BackgroundColor = Color.CornflowerBlue;
        e.Style.ForegroundColor = Color.White;
    }
    e.Handled = true;
}
```

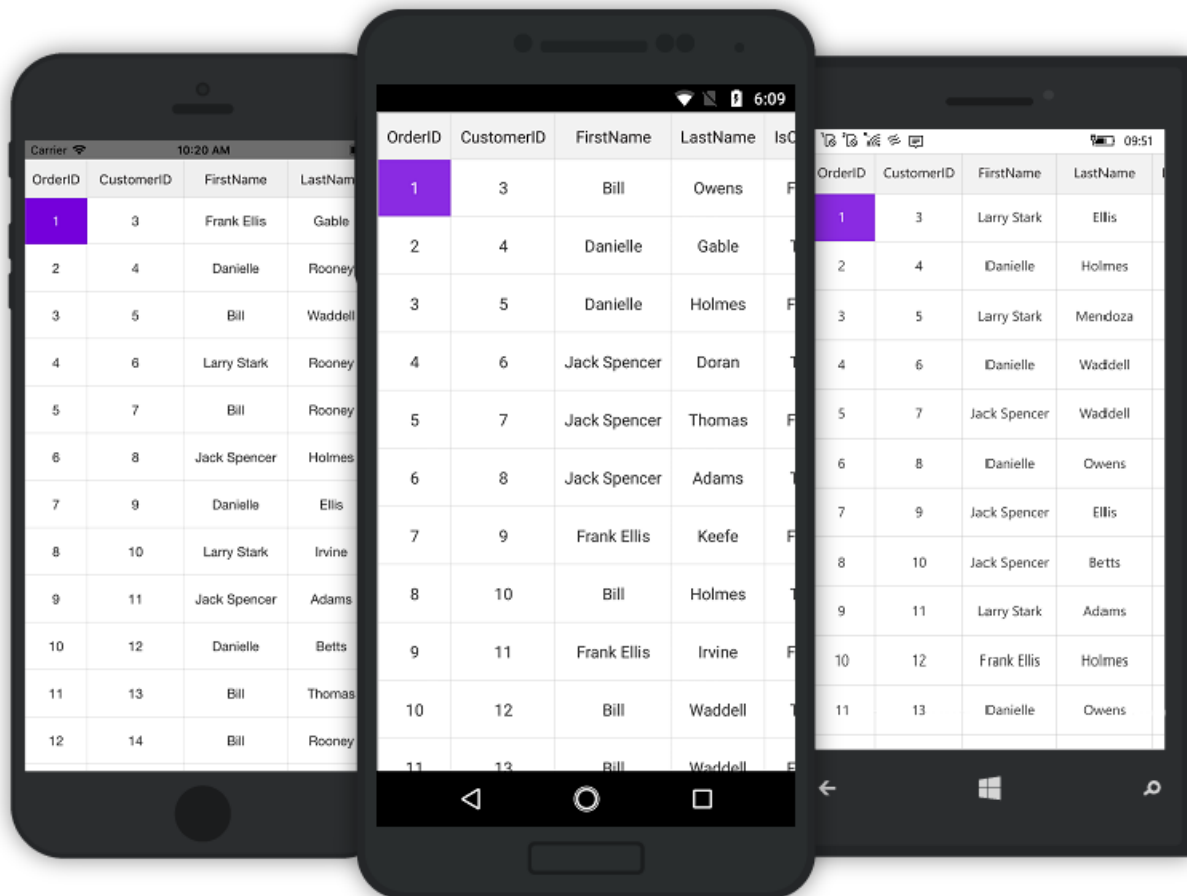


How to style a particular cell based on `RowIndex` and `ColumnIndex`

Styling can be applied to a particular cell based on the `RowIndex` and `ColumnIndex` properties in `QueryCellStyleEventArgs` of the `QueryCellStyle` event.

### C#

```
private void DataGrid_QueryCellStyle(object sender, QueryCellStyleEventArgs e)
{
    if (e.ColumnIndex == 0 && e.RowIndex == 1)
    {
        e.Style.BackgroundColor = Color.BlueViolet;
        e.Style.ForegroundColor = Color.White;
    }
    e.Handled = true;
}
```

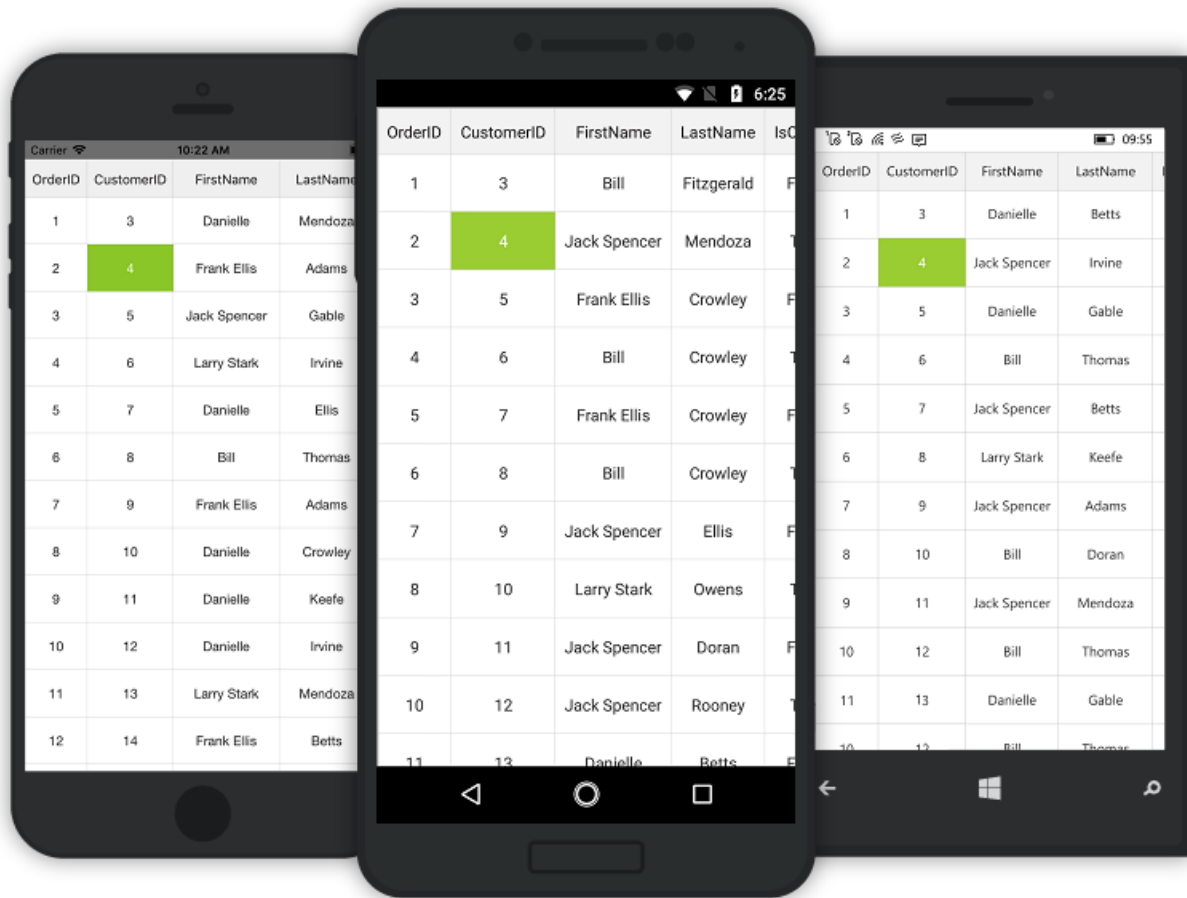


### How to style a particular cell based on CellValue

Styling can be applied to a particular cell based on `CellValue` property in `QueryCellStyleEventArgs` of the `QueryCellStyle` event.

#### C#

```
private void DataGrid_QueryCellStyle(object sender, QueryCellStyleEventArgs e)
{
    if (e.ColumnIndex == 1 && e.CellValue.ToString() == "4")
    {
        e.Style.BackgroundColor = Color.YellowGreen;
        e.Style.ForegroundColor = Color.White;
    }
    e.Handled = true;
}
```



## Styling cells using RowStyle event

The Conditional style can be applied for an entire row based on any condition using the [QueryRowStyle](#) event. This event will be fired for each row. It provides the following properties through the [QueryRowStyleEventArgs](#) in its `EventHandler`:

- [RowData](#): Provides the row data of current row in iteration.
- [RowIndex](#): Provides the row index of current row in iteration.
- [e.Handled](#): Should set to true to apply the changes.
- [Style](#): Sets style to the current row in iteration.

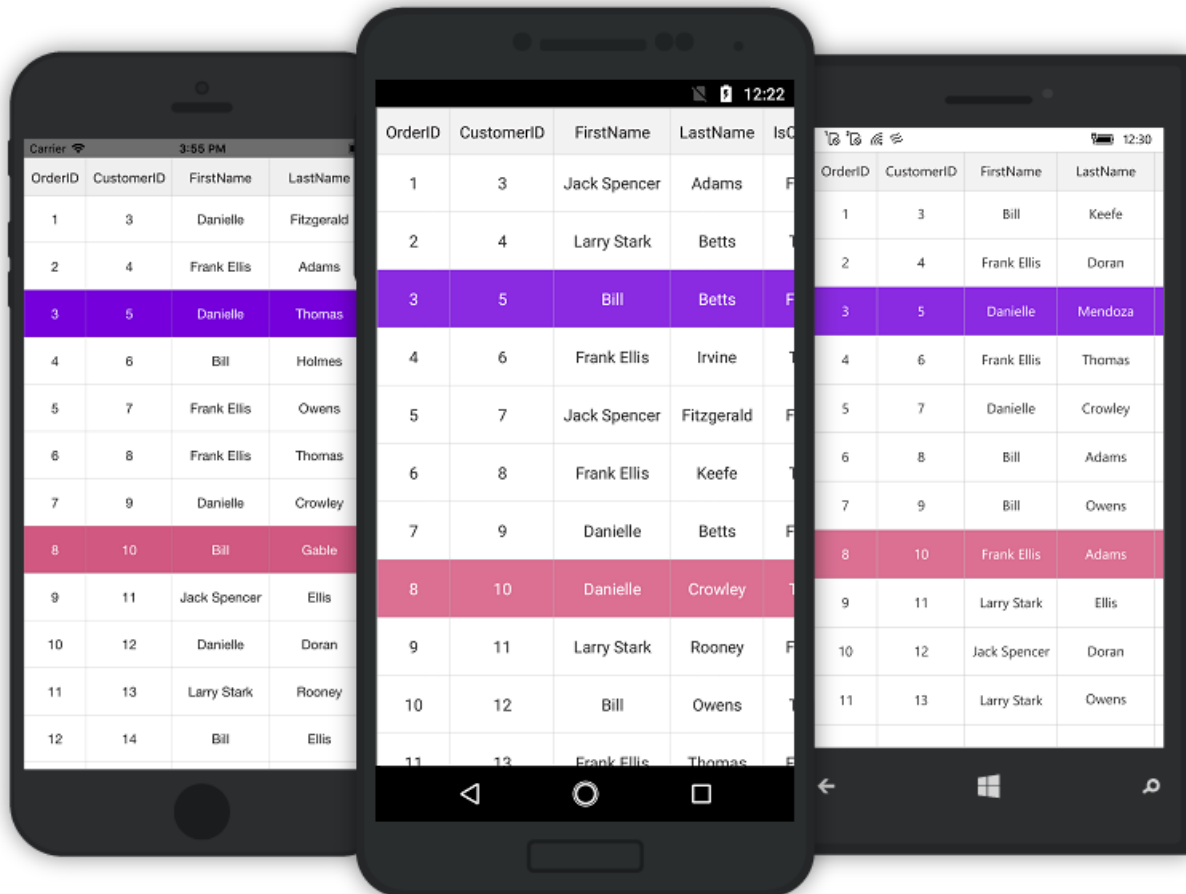
## C#

```
this.dataGrid.QueryRowStyle += DataGrid_QueryRowStyle;
private void DataGrid_QueryRowStyle(object sender, QueryRowStyleEventArgs e)
{
    if (e.RowIndex == 3)
    {
        e.Style.ForegroundColor = Color.White;
        e.Style.BackgroundColor = Color.BlueViolet;
    }
    else if (e.RowData == viewModel.OrdersInfo[7])
    {

```



```
e.Style.ForegroundColor = Color.White;
e.Style.BackgroundColor = Color.PaleVioletRed;
}
e.Handled = true;
}
```

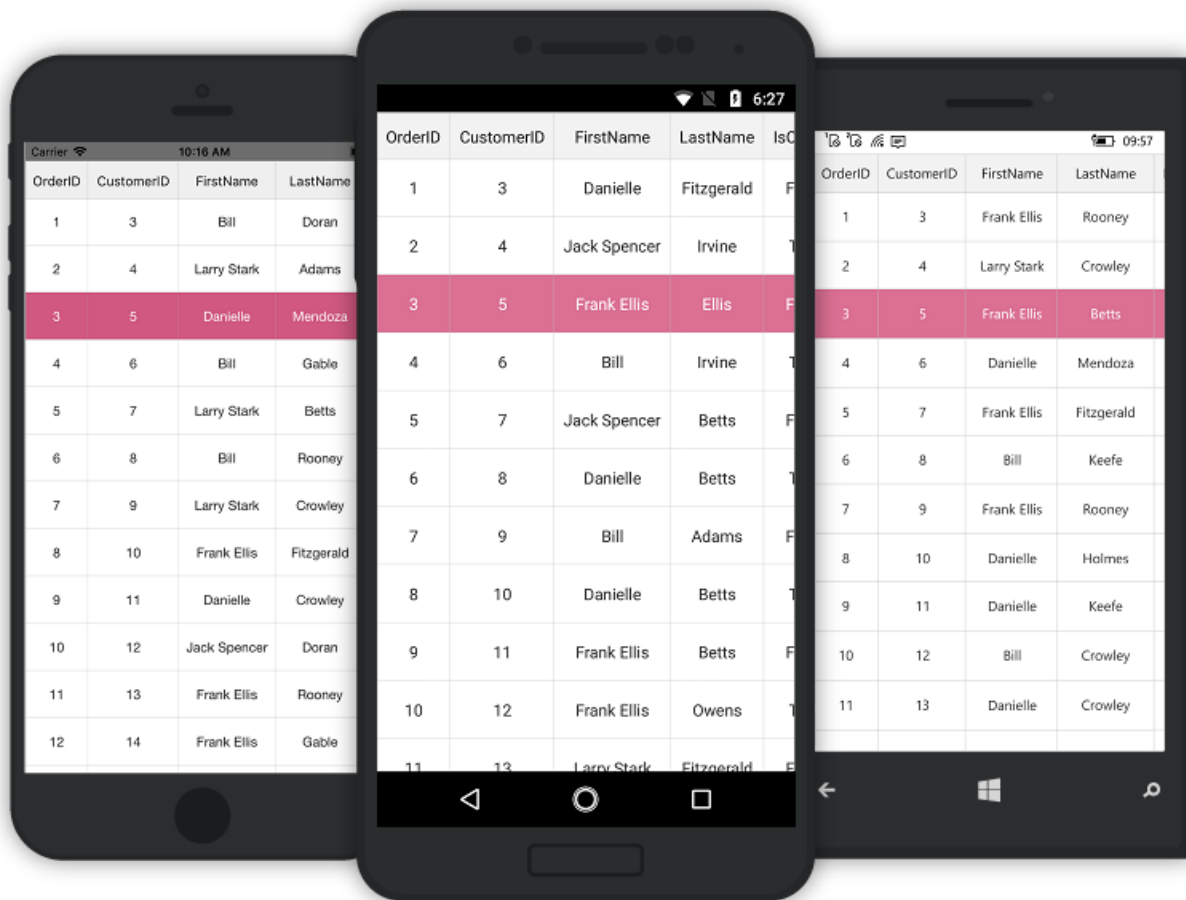


### How to style a particular row based on RowIndex

Styling can be applied to a particular row based on `RowIndex` property in `QueryRowStyleEventArgs` of the `QueryRowStyle` event.

#### C#

```
private void DataGrid_QueryRowStyle(object sender, QueryRowStyleEventArgs e)
{
    if (e.RowIndex == 3)
    {
        e.Style.ForegroundColor = Color.White;
        e.Style.BackgroundColor = Color.PaleVioletRed;
    }
    e.Handled = true;
}
```

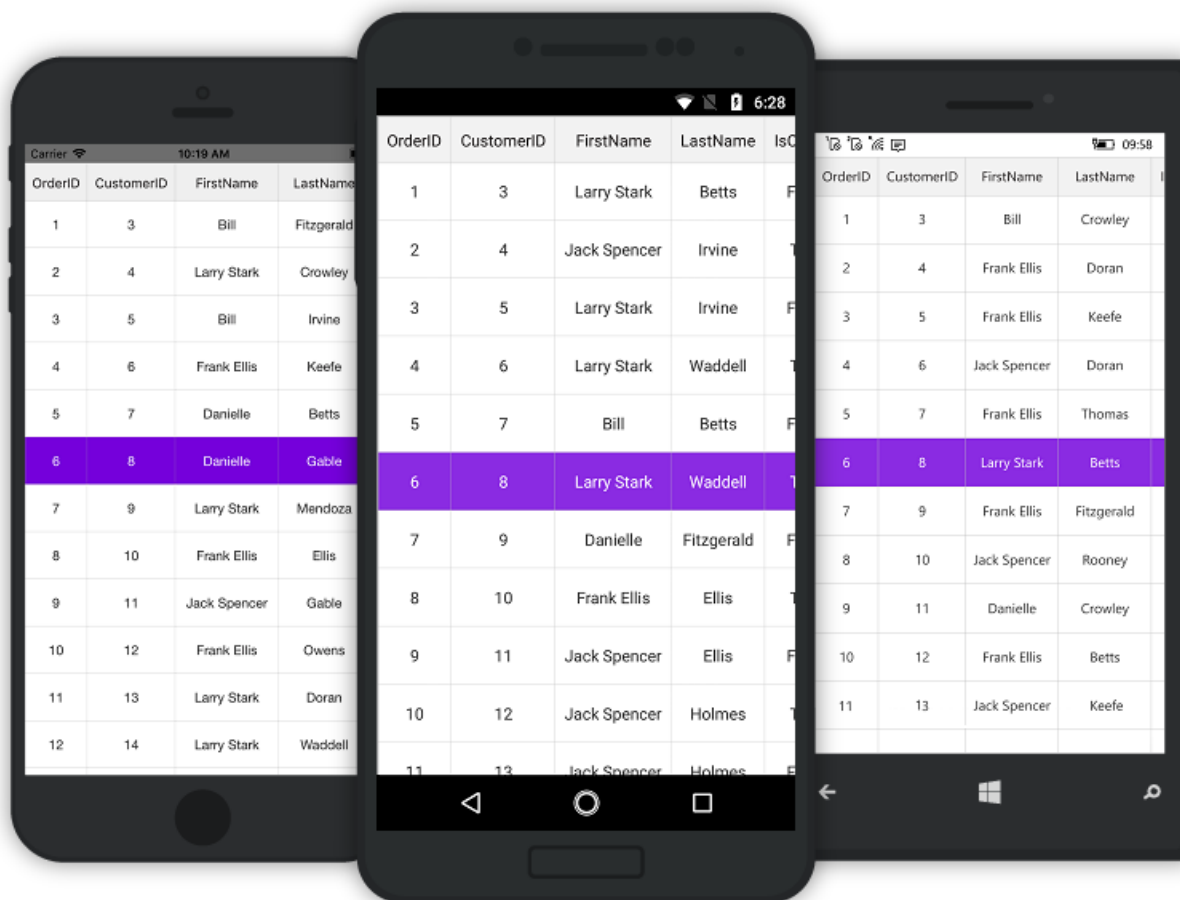


### How to style a particular row based on RowData

Styling can be applied to a particular row based on `RowData` property in `QueryRowStyleEventArgs` of the `QueryRowStyle` event.

#### C#

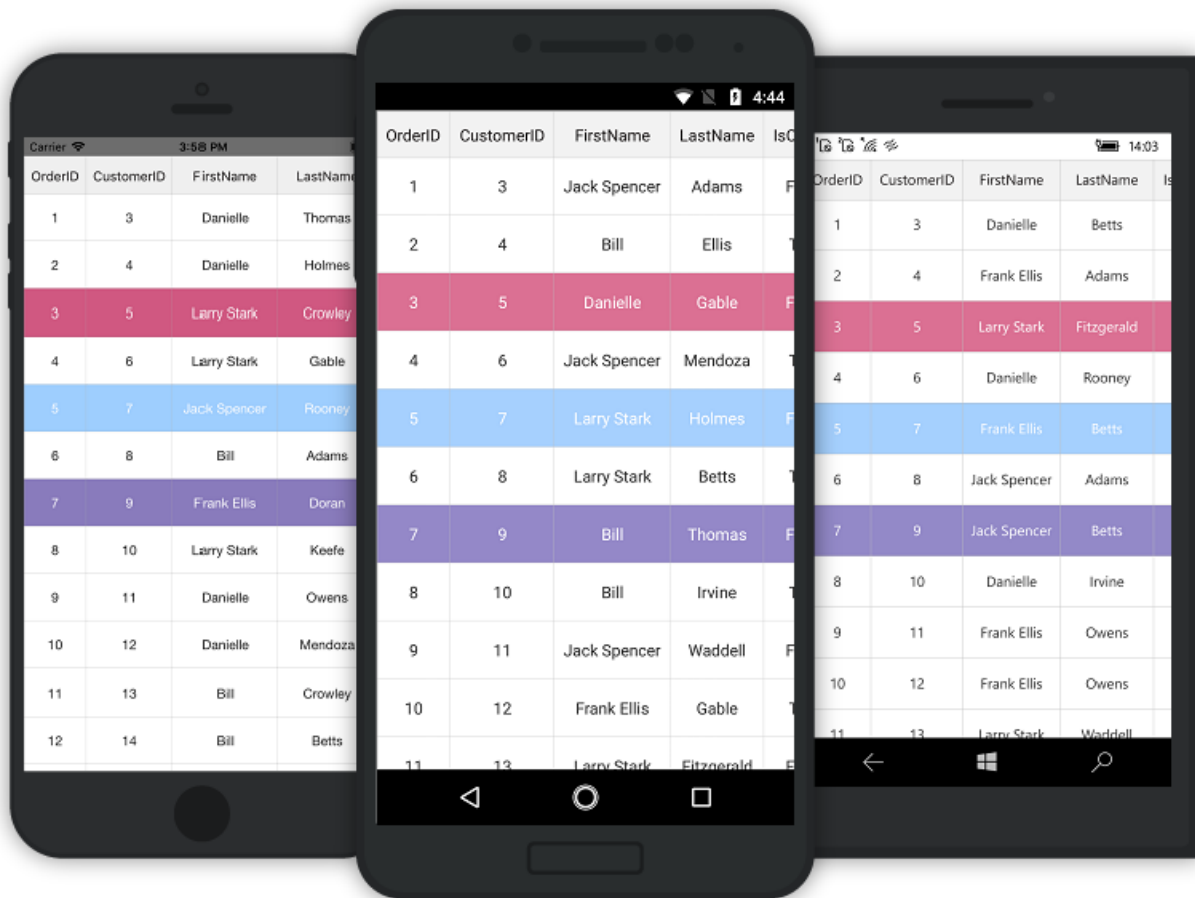
```
private void DataGrid_QueryRowStyle(object sender, QueryRowStyleEventArgs e)
{
    if (e.RowData == viewModel.OrdersInfo[5])
    {
        e.Style.ForegroundColor = Color.White;
        e.Style.BackgroundColor = Color.BlueViolet;
    }
    e.Handled = true;
}
```



**Note:** By default, only the selected background color will be applied for the selected row even if row style is applied for that row. If you want to apply selection color over while selecting row style, set the [ConditionalStylingPreference](#) property to [StylePreference.RowStyleAndSelection](#).

### C#

```
private void DataGrid_QueryRowStyle(object sender, QueryRowStyleEventArgs e)
{
    if (e.RowIndex == 3 || e.RowIndex == 7)
    {
        e.Style.ForegroundColor = Color.White;
        e.Style.BackgroundColor = Color.PaleVioletRed;
    }
    //Set the below code to display only selection
    //e.Style.ConditionalStylingPreference = StylePreference.Selection;
    e.Style.ConditionalStylingPreference = StylePreference.RowStyleAndSelection;
    e.Handled = true;
}
```



## Paging

The data grid interactively supports manipulation of data using [SfDataPager]([http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPagernamespace.html#""](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPagernamespace.html#)) control. This provides built-in options to page data on demand when dealing with large volumes of data. SfDataPager can be placed above or below based on the requirement to easily manage data paging.

To use paging functionality in the data grid, add the following namespace to the project:

[Syncfusion.SfDataGrid.XForms.DataPager]([http://help.syncfusion.com/cr/xamarin/sfdatagrid#""](http://help.syncfusion.com/cr/xamarin/sfdatagrid#))

There are two different modes in paging:

- **NormalPaging:** NormalPaging loads the entire data collection to the SfDataPager.
- **OnDemandPaging:** OnDemandPaging loads data to the current page dynamically in SfDataPager.

## Normal paging

The data grid performs paging of data using the SfDataPager. To enable paging, follow the procedure:

- Create a new SfDataPager instance, and bind the data collection to the [SfDataPager.Source](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~Source.html# "") property based on which [SfDataPager.PagedSource](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~PagedSource.html# "") is created internally.
- Bind the PagedSource property to the [ItemsSource](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.SfDataGrid~ItemsSource.html# "") of the data grid.
- Set the number of rows to be displayed on a page by setting the [SfDataPager.PageSize](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~PageSize.html# "") property.
- Set the number of buttons that should be displayed in view by setting the [SfDataPager.NumericButtonCount](http://help.syncfusion.com/cr/creffiles/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~NumericButtonCount.html# "") property.

---

**Note:** The SfDataPager.PageSize property should not be assigned with value 0.

---

The following code example illustrates using SfDataPager with the data grid control:

#### XML

```
<local:SamplePage x:Class="SampleBrowser.Paging"
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SampleBrowser;assembly=SampleBrowser"
xmlns:sfgrid="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
xmlns:sfPager="clr-namespace:Syncfusion.SfDataGrid.XForms.DataPager;assembly=Syncfusion.SfDataGrid.XForms">
<local:SamplePage.BindingContext>
<local:DataPagerViewModel x:Name="viewModel" />
</local:SamplePage.BindingContext>
<local:SamplePage.ContentView>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<sfPager:SfDataPager x:Name="dataPager"
Grid.Row="0"
PageSize="15"
HeightRequest="50"
NumericButtonCount="20"
Source="{Binding Info}">
<sfPager:SfDataPager.HeightRequest>
<OnPlatform x:TypeArguments="x:Double"
iOS="50"
Android="50"
WinPhone="70" />
</sfPager:SfDataPager.HeightRequest>
```

```
</sfPager:SfDataPager>
<sfgrid:SfDataGrid x:Name="dataGrid"
Grid.Row="1"
AutoGenerateColumns="true"
ColumnSizer="Star"
SelectionMode="Single"
ItemsSource="{Binding PagedSource, Source={x:Reference dataPager}}}"
>
</sfgrid:SfDataGrid>
</Grid>
</local:SamplePage.ContentView>
</local:SamplePage>
```

## C#

```
public partial class MainPage : ContentPage
{
    SfDataGrid sfGrid = new SfDataGrid();
    SfDataPager sfPager = new SfDataPager();
    ViewModel viewModel = new ViewModel();
    public MainPage()
    {
        InitializeComponent();
        sfPager.PageSize = 15; //Setting the number of rows in a page
        sfPager.Source = viewModel.Info; //Setting data source to SfDataPager
        sfGrid.ItemsSource = sfPager.PagedSource; //Setting ItemsSource to
        SfDataGrid
        sfGrid.ColumnSizer = ColumnSizer.Star;
        Grid myGrid = new Grid();
        myGrid.HorizontalOptions = LayoutOptions.FillAndExpand;
        myGrid.RowDefinitions = new RowDefinitionCollection
        {
            new RowDefinition { Height = 50 },
            new RowDefinition {},
        };
        myGrid.Children.Add(sfPager, 0, 0);
        myGrid.Children.Add(sfGrid, 0, 1);
        this.Content = myGrid;
    }
}
```

The following screenshot shows the outcome upon execution of the above code:



**Note:** The `SfDataPager` provides scrolling animation while tapping the [FirstPageButton](#) or [LastPageButton](#).

### OnDemandPaging

In normal Paging, data collection is entirely loaded initially to the `SfDataPager`. However, the control also allows loading the data for the current page dynamically by setting the `[SfDataPager.UseOnDemandPaging]` ([http://help.syncfusion.com/cr/cref\\_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~UseOnDemandPaging.html#""](http://help.syncfusion.com/cr/cref_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~UseOnDemandPaging.html#)) to `true`.

To load current page item dynamically, hook the `[OnDemandLoading]` ([http://help.syncfusion.com/cr/cref\\_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~OnDemandLoadingEV.html#""](http://help.syncfusion.com/cr/cref_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~OnDemandLoadingEV.html#)) event. In the `OnDemandLoading` event, use the `[LoadDynamicItems]` ([http://help.syncfusion.com/cr/cref\\_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~LoadDynamicItems.html#""](http://help.syncfusion.com/cr/cref_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.SfDataPager~LoadDynamicItems.html#)) method to load data for the corresponding page in the `SfDataPager`.

The `OnDemandLoading` event is triggered when the pager moves to the corresponding page. It contains the following event arguments:

- `[StartIndex]` ([http://help.syncfusion.com/cr/cref\\_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.OnDemandLoadingEventArgs~StartIndex.html#""](http://help.syncfusion.com/cr/cref_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.OnDemandLoadingEventArgs~StartIndex.html#)): Displays the corresponding page start index.
- `[PageSize]` ([http://help.syncfusion.com/cr/cref\\_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.OnDemandLoadingEventArgs~PageSize.html#""](http://help.syncfusion.com/cr/cref_files/xamarin/Syncfusion.SfDataGrid.XForms~Syncfusion.SfDataGrid.XForms.DataPager.OnDemandLoadingEventArgs~PageSize.html#)): Displays the number of items to be loaded for that page.

To load data for the `DataPager` control dynamically, follow the code example:

### C#

```
private void OnDemandPageLoading(object sender, OnDemandLoadingEventArgs args)
{
    sfDataPager.LoadDynamicItems(args.StartIndex,
    source.Skip(args.StartIndex).Take(args.PageSize));
}
```

**Note:** In on demand paging, you should not assign a value for the **Source** property. Also you have to define an integer value for **PageCount** property to generate the required numeric buttons in the view.

When using **OnDemandPaging**, **SfDataPager.PagedSource** loads only the current page data. Upon navigation to another page, **OnDemandLoading** event is fired which loads another set of data, but maintains the previous page data also. When you navigate to the previous page again, **OnDemandLoading** event is not fired, and the required data maintained in the cache is loaded. However, for further performance enhancement if you do not want to maintain the previous page data, call `[Syncfusion.Data.PagedCollectionView.ResetCache()]`([http://help.syncfusion.com/cr/cref\\_files/xamarin/Syncfusion.Data.Portable~Syncfusion.Data.PagedCollectionView~ResetCache.html#""](http://help.syncfusion.com/cr/cref_files/xamarin/Syncfusion.Data.Portable~Syncfusion.Data.PagedCollectionView~ResetCache.html#)) in **OnDemandLoading** event. **ResetCache** method call resets the cache except the current page.

To use **ResetCache** method, follow the code example:

#### C#

```
private void OnDemandPageLoading(object sender, OnDemandLoadingEventArgs args)
{
    sfDataPager.LoadDynamicItems(args.StartIndex,
    source.Skip(args.StartIndex).Take(args.PageSize));
    (sfDataPager.PagedSource as PagedCollectionView).ResetCache();
}
```

#### AppearanceManager

The data grid allows changing the appearance by writing a style class overriding from the **AppearanceManager**, and assigning it to the **SfDataPager.AppearanceManager** property.

To apply custom style, follow the code example:

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:sfgrid="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
xmlns:local="clr-namespace:SfDataGrid_Sample;assembly=SfDataGrid_Sample"
xmlns:sfDataPager="clr-namespace:Syncfusion.SfDataGrid.XForms.DataPager;assembly=Syncfusion.SfDataGrid.XForms"
x:Class="SfDataGrid_Sample.Page1">
<ContentPage.Resources>
<ResourceDictionary>
<local:CustomAppearance x:Key="customAppearance"/>
</ResourceDictionary>
</ContentPage.Resources>
```



```

<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<sfgrid:SfDataGrid x:Name="dataGrid" Grid.Row="1"
AutoGenerateColumns="True" />
<sfDataPager:SfDataPager x:Name="sfDataPager"
Grid.Row="0"
PageCount="10"
PageSize="10"
NumericButtonCount="10"
AppearanceManager="{StaticResource customAppearance}"
Source="{Binding OrdersInfo}">
</sfDataPager:SfDataPager>
</Grid>
</ContentPage>

```

**C#**

```

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        this.dataGrid.ItemsSource = sfDataPager.PagedSource;
        // Apply the custom appearance to SfDataPager
        this.sfDataPager.AppearanceManager = new CustomAppearance();
    }
}

```

**C#**

```

//Custom Appearance class
public class CustomAppearance: AppearanceManager
{
    public override Color GetNumericButtonSelectionBackgroundColor()
    {
        return Color.FromRgb(255, 0, 0);
    }
    public override Color GetNumericButtonSelectionForegroundColor()
    {
        return Color.FromRgb(0, 255, 0);
    }
    public override Color GetNumericButtonBackgroundColor()
    {
        return Color.FromRgb(0, 0, 255);
    }
    public override Color GetNumericButtonForegroundColor()
    {
        return Color.FromRgb(82, 82, 82);
    }
    public override Color GetNavigationButtonBackgroundColor()

```

```
{
    return Color.FromRgb(34, 34, 34);
}
public override Color GetPagerButtonBorderColor()
{
    return Color.Black;
}
}
```

The following picture shows the customize appearance of data pager:



ProductID	Product	UserRating	ProductM...
10001	Nokia	4	Lumia_800
10002	Geneva	3	Carrera
10003	Dell	3	XPS12
10004	ROLEX	3	Submariner
10005	Samsung	1	S3
10006	HTC	2	8x
10007	SonyMobile	2	Xperia_Tipo
10008	Dell	2	XPS15
10009	Dell	3	XPS12
10010	ROLEX	1	Sea_Dweller

#### *Numeric button border color*

Based on the requirement, customize the numeric button border color by overriding the [GetPagerButtonBorderColor](#) method.

To customize the numeric button border color, follow the code example:

#### **XML**

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<sfgrid:SfDataGrid x:Name="dataGrid" Grid.Row="1"
AutoGenerateColumns="True" />
```

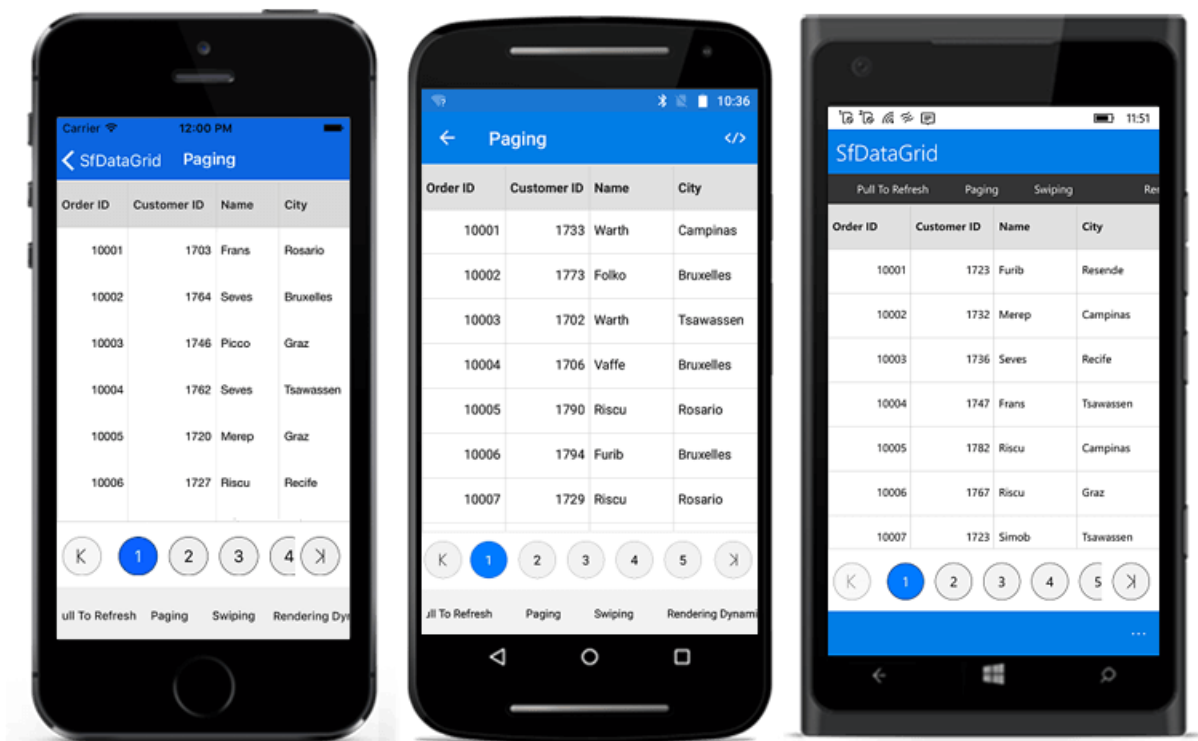
```
<sfDataPager:SfDataPager x:Name="sfDataPager"
Grid.Row="0"
PageCount="10"
PageSize="10"
NumericButtonCount="10"
AppearanceManager="{StaticResource customAppearance}"
Source="{Binding OrdersInfo}">
</sfDataPager:SfDataPager>
</Grid>
```

**C#**

```
this.dataGrid.ItemsSource = sfDataPager.PagedSource;
// Apply the custom appearance to SfDataPager
this.sfDataPager.AppearanceManager = new CustomAppearance();
```

**C#**

```
public class CustomAppearance : AppearanceManager
{
    public override Color GetPagerButtonBorderColor()
    {
        return Color.Black;
    }
}
```



## Row Height Customization

The data grid provides an option to customize the header row height and the row height of all the grid rows or a particular row. To achieve this customization, see the following sections:

### Customize HeaderRowHeight

The data grid allows you to customize the height of the header row by setting the [SfDataGrid.HeaderRowHeight](#) property. The default value of this property is 40. This property responds to runtime changes, so it can be customized. Setting [SfDataGrid.HeaderRowHeight](#) to zero will collapse the header row in the view.

To customize the header row height, follow the code example:

#### C#

```
//Customizing header row height in SfDataGrid
dataGrid.HeaderRowHeight = 50;
```

### Customize RowHeight for all rows

The data grid allows you to customize the height of grid rows in the scrolling region by setting the [SfDataGrid.RowHeight](#) property. The default value of this property is 50. This property responds to runtime changes, so it can be customized. Setting this property will change the height of all the rows in a body region with a common value. Setting [SfDataGrid.RowHeight](#) to zero will collapse all rows in the grid.

To customize header row height, follow the code example:

#### C#

```
//Customizing row height in SfDataGrid
dataGrid.RowHeight = 60;
```

### QueryRowHeight

The [SfDataGrid.QueryRowHeight](#) event returns row heights on demand. This event receives two arguments, namely the sender handles the data grid, and the [QueryRowHeightEventArgs](#). The [QueryRowHeightEventArgs](#) has the following properties:

- [RowIndex](#): This property helps to identify a particular row in the grid.
- [Height](#): This property sets and returns the height of a grid row on demand. Default line size of the rows is 50.
- [Handled](#): This property decides whether the specified height can be set to the row or not. The default value is `false`. When this property is not set, the decided height is not set to the row.

To hook the [SfDataGrid.QueryRowHeight](#) event, and customize height of a row, follow the code example:

#### C#

```
//Hooks QueryRowHeight event in SfDataGrid
dataGrid.QueryRowHeight += DataGrid_QueryRowHeight;
//Event to set the row height on demand
void DataGrid_QueryRowHeight (object sender, QueryRowHeightEventArgs e)
{
```

```
//Sets height of the fifth row
if (e.RowIndex == 5) {
    e.Height = 100;
    e.Handled = true;
}
```

#### QueryRowHeights customization

The data grid allows you to query a range of rows programmatically by using the `SfDataGrid.QueryingRowHeights` method.

`QueryRowHeights` has two arguments: start index and end index.

- Start index: Indicates from which row index the `SfDataGrid.QueryRowHeight` event has to fire.
- End index: Indicates the end row index for the `SfDataGrid.QueryRowHeight` event to fire.

To customize the row height for a range of rows, follow the code example:

#### C#

```
//Customizing the QueryingRowHeights in data grid
dataGrid.QueryingRowHeights(2,5);
//Its starts to query the rows from the second row to the fifth row.
```

#### GridRowSizingOptions

The data grid allows you to customize the height of grid rows with various customizing options while auto calculating the row height based on the content using the `GridRowSizingOptions`.

#### Calculate height based on certain columns

The data grid allows you to calculate the row height excluding certain columns using the `ExcludeColumns` property.

The following code example illustrates calculating the height of grid rows based on certain columns:

#### C#

```
private void DataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    GridRowSizingOptions options = new GridRowSizingOptions();
    options.ExcludeColumns.Add("Description");
    options.ExcludeColumns.Add("CustomerID");
    if (e.RowIndex == 0)
    {
        e.Height = 50;
    }
    else
    {
        e.Height = dataGrid.GetRowHeight(e.RowIndex, options);
    }
    e.Handled = true;
}
```

### Calculate height including hidden columns

Data grid allows you to calculate the row height based on content including or excluding hidden columns using the `CanIncludeHiddenColumns` property.

To calculate the height including hidden columns, follow the code example:

#### C#

```
private void DataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    GridRowSizingOptions options = new GridRowSizingOptions();
    options.CanIncludeHiddenColumns = true;
    if (e.RowIndex == 0)
    {
        e.Height = 50;
    }
    else
    {
        e.Height = dataGrid.GetRowHeight(e.RowIndex, options);
    }
    e.Handled = true;
}
```

### Reset row height at runtime

The data grid allows you to customize the height of a grid row on demand by handling the [SfDataGrid.QueryRowHeight](#) event. This event is raised for the grid rows whenever they come into view. So the height of a particular row can be customized on demand using the row index. Setting the height to zero will collapse all rows in the grid.

### Auto fit the grid rows based on content

The data grid supports the `AutoRowHeight` feature. The height of a row can be customized based on the content. This can be achieved using the `SfDataGrid.QueryRowHeight` event and the [SfDataGrid.GetRowHeight](#) method. The `SfDataGrid.QueryRowHeight` event returns the row height on demand. The `SfDataGrid.GetRowHeight` method returns the height of the row based on the content.

---

**Note:** The row drag and drop operation is not supported while customizing the row height based on content.

---

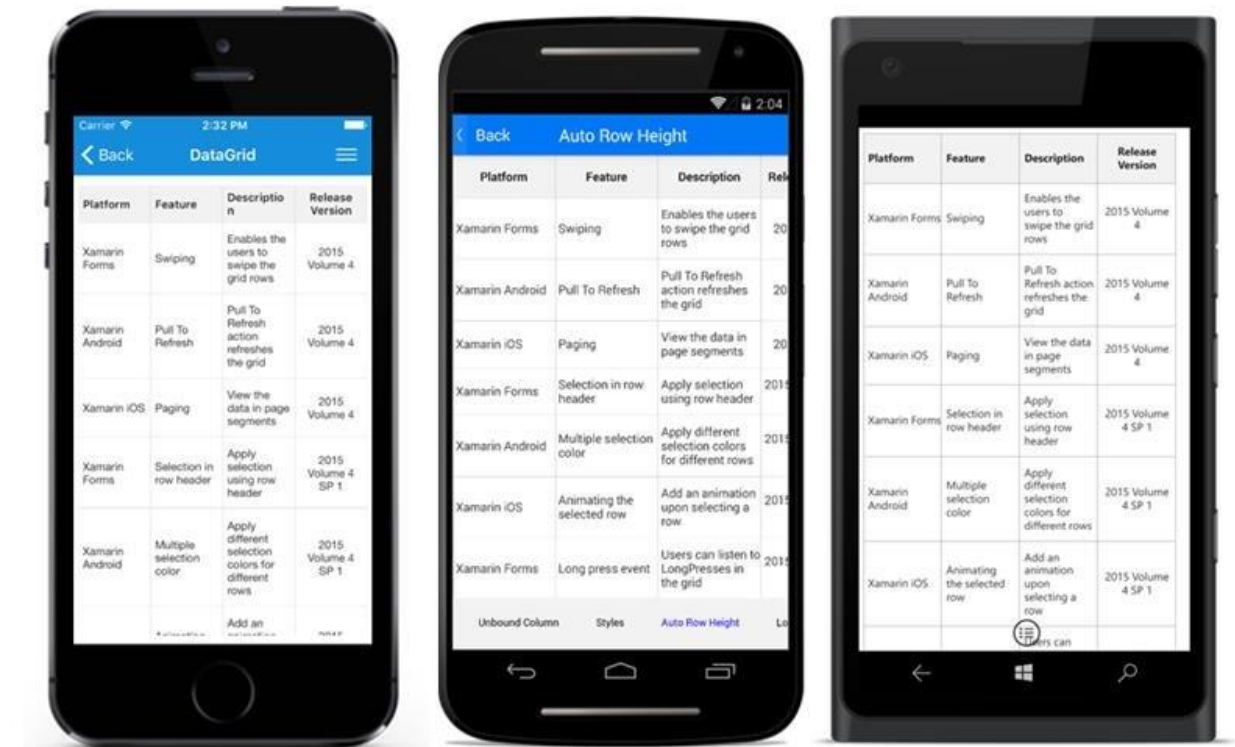
To hook the `SfDataGrid.QueryRowHeight` event and auto fit the height of a row based on content, follow the code example:

#### C#

```
//Hooks QueryRowHeight event in SfDataGrid to set the row height on demand
dataGrid.QueryRowHeight += DataGrid_QueryRowHeight;
private
void DataGrid_QueryRowHeight (object sender, QueryRowHeightEventArgs e)
{
    if (e.RowIndex != 0)
    {
        //Calculates and sets the height of the row based on its content.
        e.Height = dataGrid.GetRowHeight(e.RowIndex);
        e.Handled = true;
    }
}
```

```
}

```



### Customize header row height based on header content

The data grid allows you to customize the height of the header row based on its content using the `SfDataGrid.QueryRowHeight` event and [SfDataGrid.GetRowHeight](#) method.

To hook the `SfDataGrid.QueryRowHeight` event and change the header row height based on the content, follow the code example:

#### C#

```
//Hooks QueryRowHeight event in SfDataGrid to set the header row height on demand
dataGrid.QueryRowHeight += DataGrid_QueryRowHeight;
private void DataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (e.RowIndex == dataGrid.GetHeaderIndex())
    {
        e.Height = dataGrid.GetRowHeight(e.RowIndex);
        e.Handled = true;
    }
}
```



### Customize caption summary row height

The data grid allows you to customize the height of the `CaptionSummaryRow` by setting the height of the caption rows in the `SfDataGrid.QueryRowHeight` event. By default, the `CaptionSummaryRow` renders with the height of the `SfDataGrid.RowHeight`, which is 50.

To customize the `CaptionSummaryRow` height, follow the code example:

#### C#

```
//Hooks QueryRowHeight event in SfDataGrid to set the CaptionSummaryRow height on demand
dataGrid.QueryRowHeight += DataGrid_QueryRowHeight;
private void DataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (dataGrid.IsCaptionSummaryRow(e.RowIndex))
    {
        e.Height = 70;
        e.Handled = true;
    }
}
```





### Change TableSummaryRow height

The data grid allows you to customize the height of the **TableSummaryRow** by setting the height of the table summary rows in the **SfDataGrid.QueryRowHeight** event. By default, the **TableSummaryRow** renders with the height of the **SfDataGrid.RowHeight**, which is 50.

To customize header row height, follow the code example:

#### C#

```
//Hooks QueryRowHeight event in DataGrid to set the CaptionSummaryRow height on demand
dataGrid.QueryRowHeight += DataGrid_QueryRowHeight;
private void DataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (dataGrid.IsTableSummaryRow(e.RowIndex))
    {
        e.Height = 70;
        e.Handled = true;
    }
}
```



Order ID	Salary	First Name	Last Name	Shipping Country
10001	10001	Michael	Betts	Brazil
10002	10002	Gina	Irvine	Belgium
10003	10003	Daniel	Gable	Argentina
10004	10004	Danielle	Betts	Brazil
10005	10005	Gina	Gable	Belgium
10006	10006	Brenda	Owens	Brazil
10007	10007	William	Doran	Canada
10008	10008	Kyle	Irvine	Brazil
10009	10009	Danielle	Rooney	Argentina
Total Salary: ₹ 10,05,050.00 for 100 members				

### How to optimize performance when using QueryRowHeight event

By default, the `SfDataGrid.QueryRowHeight` event will be fired each time a row comes into view. If you want to prevent the same row from being queried again, you can check if the `Height` property in the `QueryRowHeightEventArgs` is not equal to the `SfDataGrid.RowHeight` property, which prevents the same row from being queried again. To enhance performance by preventing the same row from being queried again, follow the code example:

#### C#

```
//Hooks QueryRowHeight event in DataGrid to set the row height on demand
dataGrid.QueryRowHeight += DataGrid_QueryRowHeight;
private
void DataGrid_QueryRowHeight (object sender, QueryRowHeightEventArgs e)
{
    // Code to skip querying of a row if already queried
    if (e.Height != dataGrid.RowHeight)
        return;
    if (e.RowIndex != 0)
    {
        //Calculates and sets the height of the row based on its content.
        e.Height = dataGrid.GetRowHeight(e.RowIndex);
        e.Handled = true;
    }
}
```

### Limitations

When setting `SfDataGrid.ScrollingMode` to `ScrollingMode.Line`, the `SfDataGrid.QueryRowHeight` event is not supported.

### Scrolling

#### Scrolling mode

The data grid provides three types of scrolling mode that can be customized by using the `SfDataGrid.ScrollingMode` property. By default, the control will scroll the content based on pixel values. The scrolling modes are as follows:

- PixelLine
- Line
- Pixel

---

**Note:** The data grid supports for the vertical and horizontal scrollbars in UWP. In addition to that, mouse scrolling is also supporting in UWP desktop application.

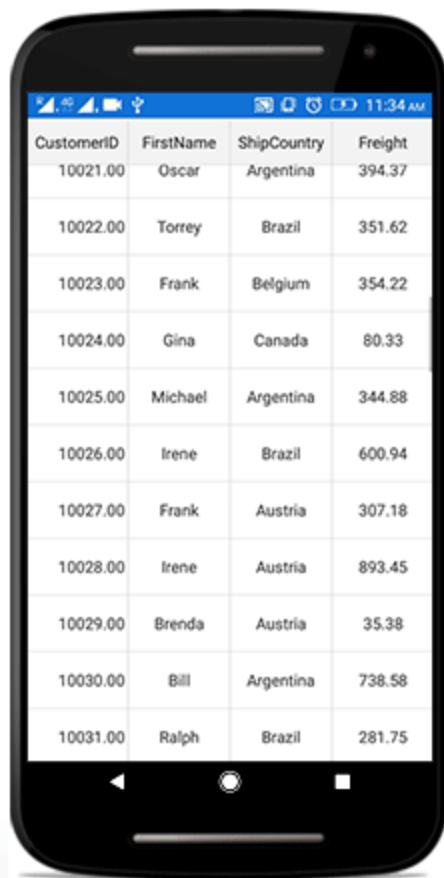
---

#### PixelLine

The `ScrollingMode.PixelLine` allows you to scroll its contents like an excel sheet i.e., whenever a row or a column is clipped on the top, the particular row or column will auto scroll to display fully in view.

#### C#

```
dataGrid.ScrollingMode = ScrollingMode.PixelLine;
```

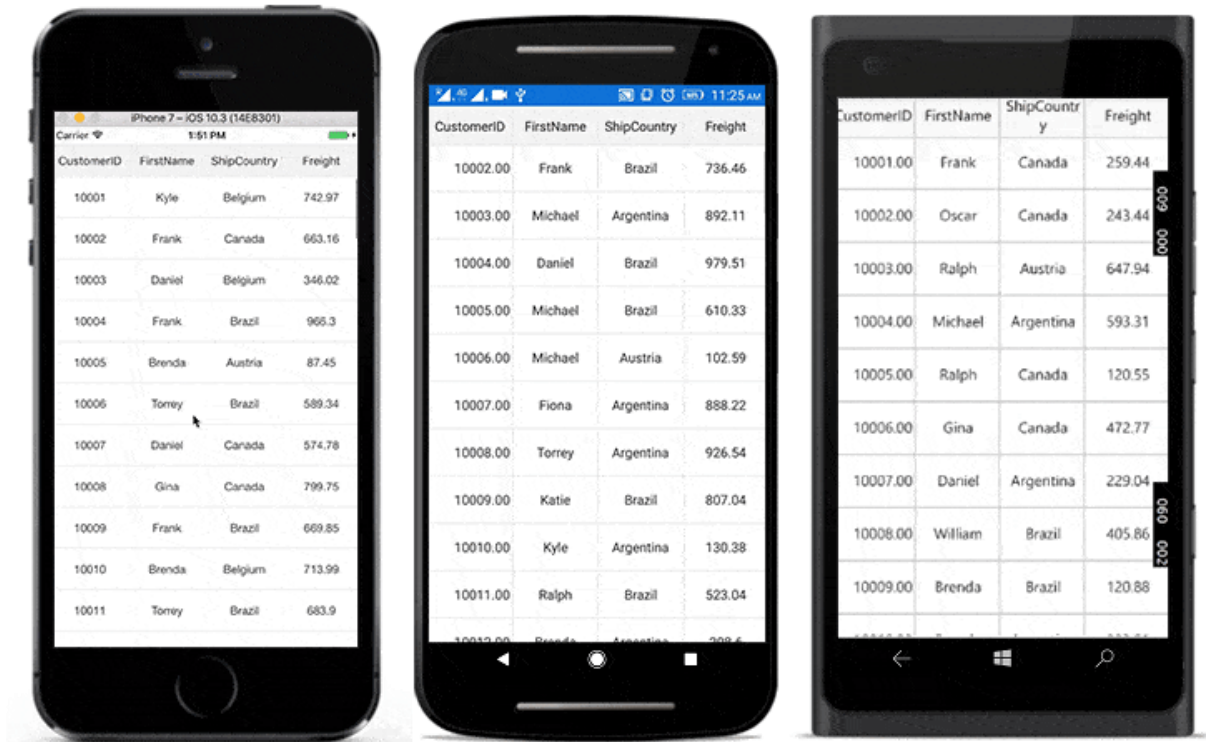


### Line

The `ScrollingMode.Line` allows you to scroll its contents based on lines i.e., the view will be updated only when the offset values reaches the origin of a row or column in the bound collection.

### C#

```
dataGrid.ScrollingMode = ScrollingMode.Line;
```

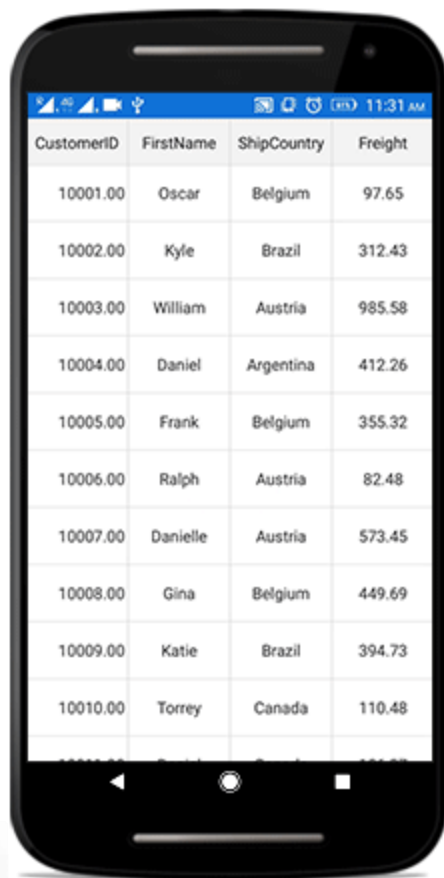


### Pixel

The `ScrollingMode.Pixel` allows you to scroll its contents based on pixel values i.e., the view will update each pixel change of the offsets, and row or column will appear clipped when offset exceeds the origin of the row or column.

### C#

```
dataGrid.ScrollingMode = ScrollingMode.Pixel;
```



CustomerID	FirstName	ShipCountry	Freight
10001.00	Oscar	Belgium	97.65
10002.00	Kyle	Brazil	312.43
10003.00	William	Austria	985.58
10004.00	Daniel	Argentina	412.26
10005.00	Frank	Belgium	355.32
10006.00	Ralph	Austria	82.48
10007.00	Danielle	Austria	573.45
10008.00	Gina	Belgium	449.69
10009.00	Katie	Brazil	394.73
10010.00	Torrey	Canada	110.48

### Programmatic scrolling

The data grid scrolls to a particular row and column index programmatically.

#### *Scroll to row and column index*

Scroll programmatically to a particular row and column using the [SfDataGrid.ScrollToRowColumnIndex](#) method by passing row and column indexes.

#### **C#**

```
dataGrid.ScrollToRowColumnIndex(int rowIndex, int columnIndex);  
//For example,  
dataGrid.ScrollToRowColumnIndex(20, 6);
```



### Scroll to row index

Scroll programmatically to a particular row using the [SfDataGrid.ScrollToRowIndex](#) method by passing the row index.

### C#

```
dataGrid.ScrollToRowIndex(int rowIndex);
//For example,
dataGrid.ScrollToRowIndex(20);
```



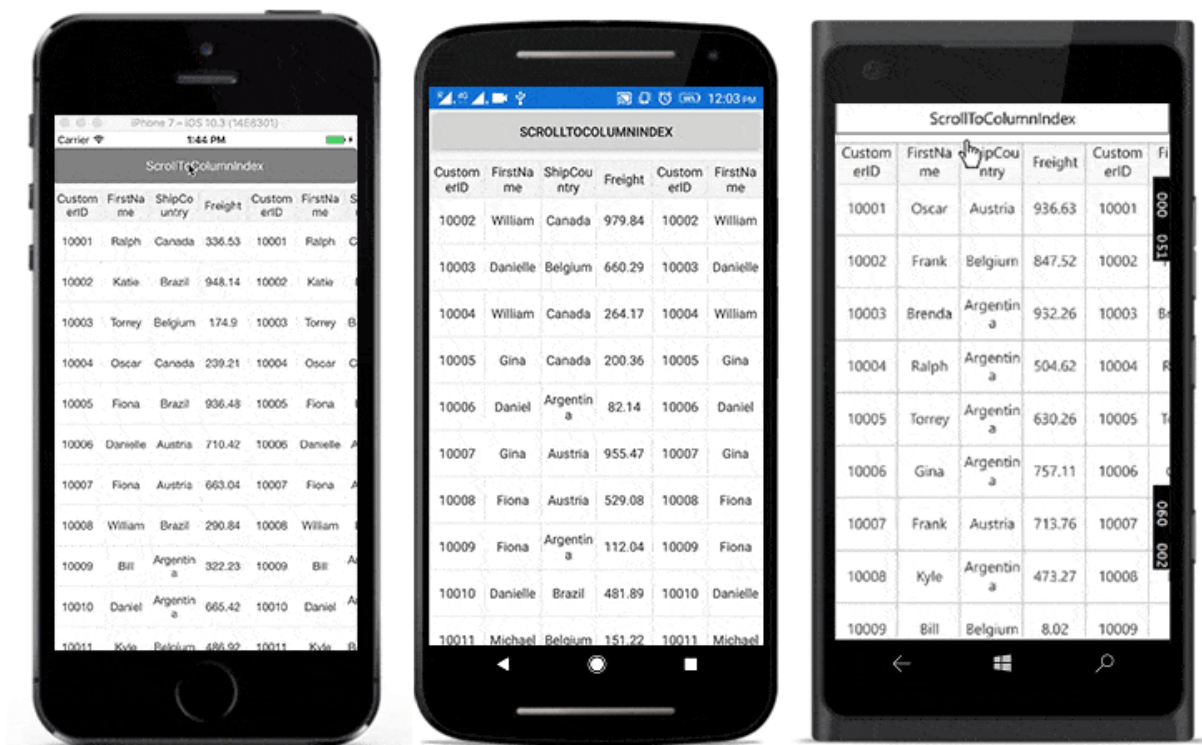
### *Scroll to column index*

Scroll programmatically to a particular column using the [SfDataGrid.ScrollToColumnIndex](#) method by passing the column index.

### **C#**

```
dataGrid.ScrollToColumnIndex(int columnIndex);
//For example,
dataGrid.ScrollToColumnIndex(7);
```





### Scroll a row/column to a specific position

The SfDataGrid allows to position the scrolled row/column in the datagrid by passing [ScrollToPosition](#) as parameter to the [ScrollToRowIndex](#), [ScrollToRowIndex](#), [ScrollToColumnIndex](#) methods. The scrolled row/column can take either of the four positions as explained below. The default position is [Start](#).

- **MakeVisible:** Scroll to make a specified row/column visible in datagrid. If the specified row/column is already in view, scrolling will not occur.
- **Start:** Scroll to make the row/column positioned at the start of the datagrid.
- **Center:** Scroll to make the row/column positioned at the center of the datagrid.
- **End:** Scroll to make the row/column positioned at the end of the datagrid.

Refer the below code snippet to scroll a column/row to a specific position.

### C#

```
// To scroll a column to a particular position,
datagrid.ScrollToColumnIndex(7,scrollToColumnPosition:
ScrollToPosition.Center);
// To scroll a row to a particular position,
datagrid.ScrollToRowIndex(7,scrollToColumnPosition:
ScrollToPosition.Center);
// To scroll a cell to a particular position,
datagrid.ScrollToRowColumnIndex(7, 7, scrollToColumnPosition:
ScrollToPosition.Center, scrollToRowPosition: ScrollToPosition.Center);
```

**Note:** Programmatic scrolling is not applicable for rows and columns that are frozen in view.

### Diagonal scrolling

By default, `SfDataGrid` supports diagonal scrolling(both vertical and horizontal scrolling simultaneously). Setting false to [SfDataGrid.AllowDiagonalScrolling](#) disables diagonal scrolling and scrolls the data grid in either horizontal or vertical direction but not simultaneously.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo,Mode=TwoWay}"
AllowDiagonalScrolling="False">
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.AllowDiagonalScrolling = false;
```

### Vertical Over Scroll Mode

The [SfDataGrid.VerticalOverScrollMode](#) property customizes the bouncing behavior of the data grid.

The `SfDataGrid.VerticalOverScrollMode` is of [VerticalScrollMode](#) type having the following two modes:

- Bounce
- None

#### Bounce

The Bounce mode allows the data grid to have bouncing effect. Default value of `SfDataGrid.VerticalOverScrollMode` is Bounce .

To customize the bouncing effect in the data grid, follow the code example:

#### C#

```
dataGrid.VerticalOverScrollMode = VerticalOverScrollMode.Bounce;
```

#### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
ColumnSizer="Star"
VerticalOverScrollMode="Bounce"
ItemsSource="{Binding OrdersInfo}">
</sfgrid:SfDataGrid>
```



### None

The **None** mode disables the bouncing effect in the data grid.

To customize the bouncing effect in the data grid, follow the code example:

### C#

```
dataGrid.VerticalOverScrollMode = VerticalOverScrollMode.None;
```

### XML

```
<sfgrid:SfDataGrid x:Name="dataGrid"
ColumnSizer="Star"
VerticalOverScrollMode="None"
ItemsSource="{Binding OrdersInfo}">
</sfgrid:SfDataGrid>
```



### Identifying scroll state changes

The **SfDataGrid** will notify the scrolling state changes via the [ScrollStateChanged](#) event.

Following states will be notified through the [ScrollState](#) property in the event argument.

- Dragging: Specifies that **SfDataGrid** is currently being dragged in the view.
- Fling: Specifies that fling action is performed on the **SfDataGrid**.
- Idle: Specifies that **SfDataGrid** is not scrolling currently.
- Programmatic: Specifies that scrolling is performed by using [ScrollToColumnIndex](#) or [ScrollToRowIndex](#) method.

### C#

```
dataGrid.ScrollStateChanged += DataGrid_ScrollStateChanged;
private void DataGrid_ScrollStateChanged(object sender,
ScrollStateChangedEventArgs e)
{
    if (e.ScrollState == ScrollState.Idle)
    {
        DisplayAlert("ScrollState", "Scrolling has stopped", "OK");
    }
}
```

### Scrolling customization using Slider

The data grid allows scrolling to a particular row by passing the row index to the `ScrollToRowIndex` method. To scroll the control when interacting with `Slider`, pass the `Slider.Value` as the row index to the `ScrollToRowIndex` method.

To customize the data grid scrolling programmatically using `Slider`, follow the code example:

#### C#

```
Slider slider = new Slider();
slider.ValueChanged += Slider_ValueChanged;
private void Slider_ValueChanged(object sender, ValueChangedEventArgs e)
{
    dataGrid.ScrollToRowIndex((int) (e.NewValue));
}
```



### Unbound Rows

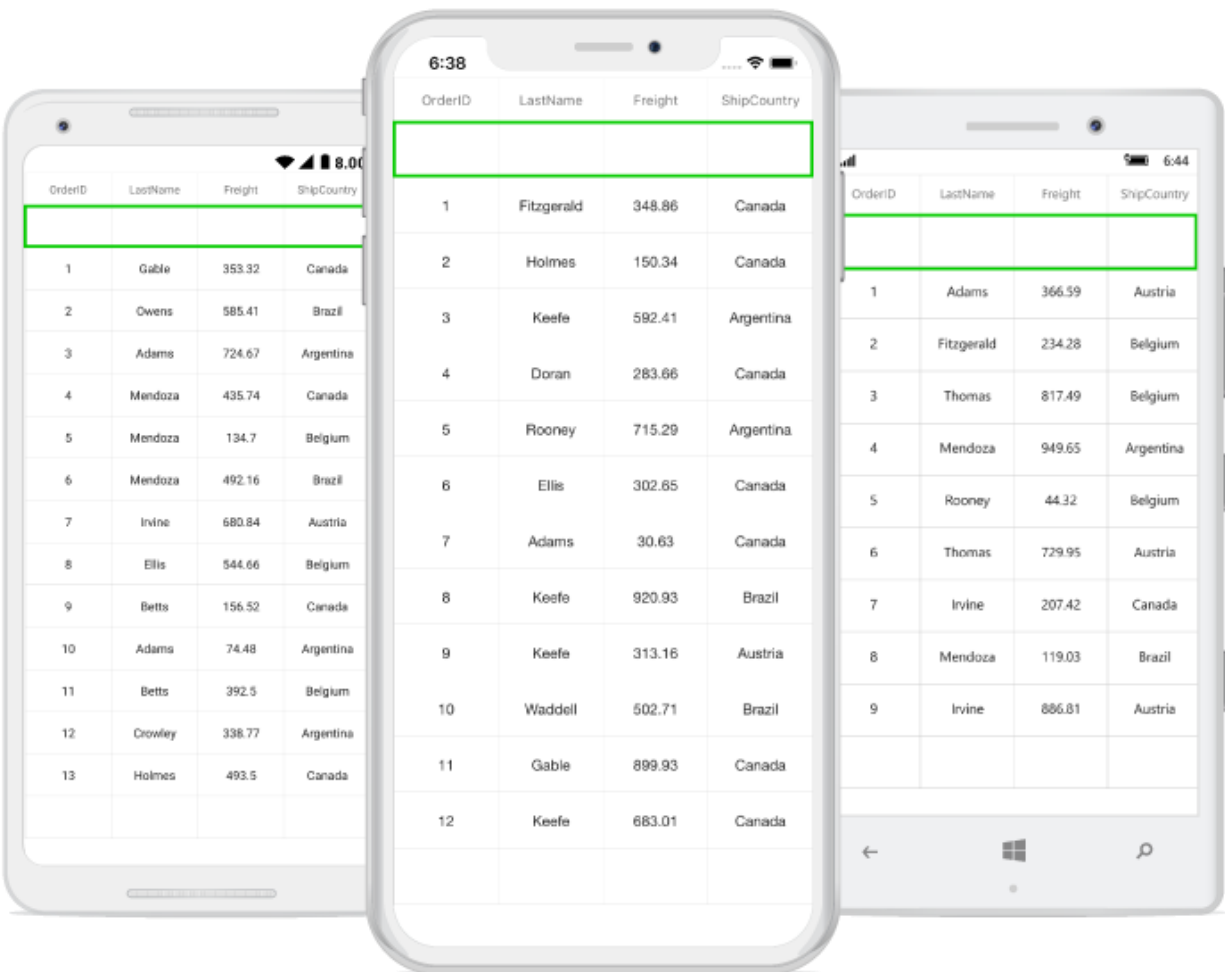
The Xamarin.Forms DataGrid allows you to add **additional rows** at top and also bottom of the DataGrid which are **not bound with data object** of underlying data source. You can add unbound rows using [SfDataGrid.UnboundRows](#) collection property. You can add any no of unbound rows to the DataGrid. Unbound rows can also be exported to pdf and excel documents.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid" ItemsSource="{Binding
OrdersInfo}">
<syncfusion:SfDataGrid.UnboundRows>
<syncfusion:GridUnboundRow Position="Top" />
</syncfusion:SfDataGrid.UnboundRows>
</syncfusion:SfDataGrid>
```

**C#**

```
this.dataGrid.UnboundRows.Add(new GridUnboundRow() { Position =
UnboundRowsPosition.Top });
```

**Positioning unbound rows**

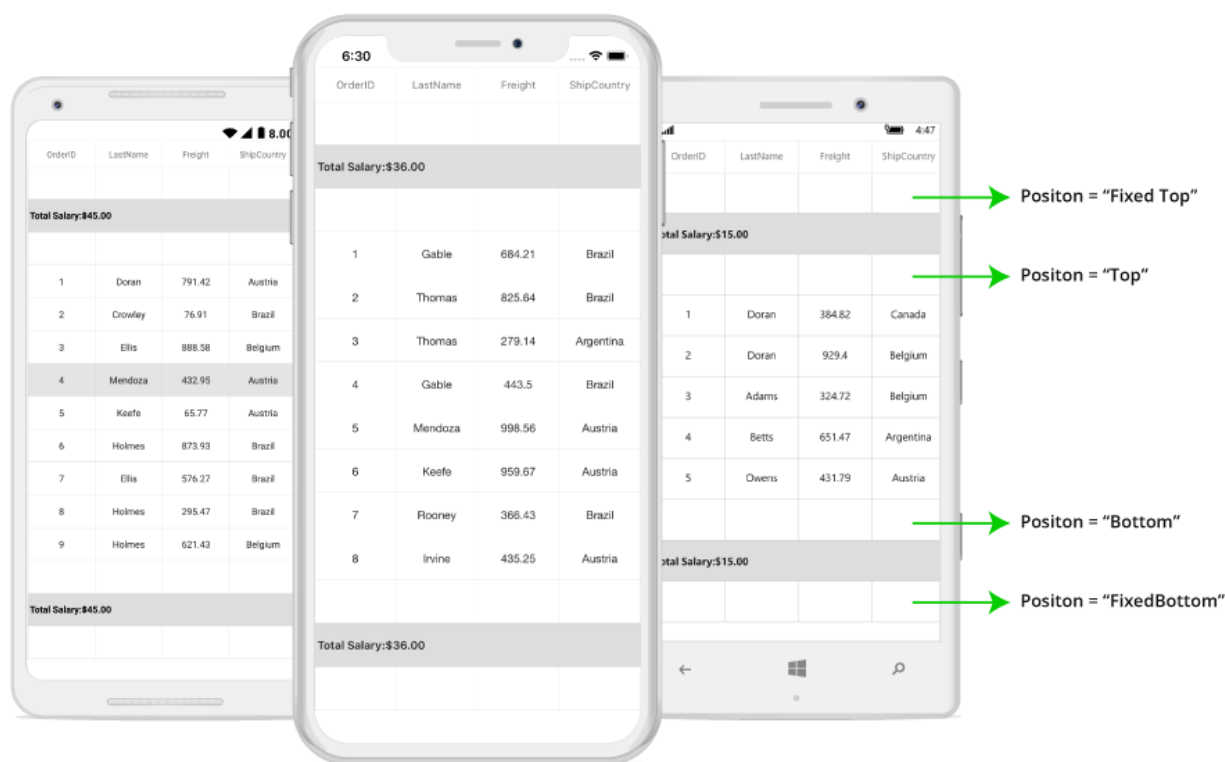
Unbound row can be placed in top or bottom of the DataGrid by setting the desired value to the [GridUnboundRow.Position](#) property.

Below table shows the available unbound row positions.

UnboundRowPosition	Position in DataGrid
--------------------	----------------------

FixedTop	Unbound row placed at top, right below the Header row. In this position, unbound row is not selectable, not editable and frozen when scrolling.
Top	Unbound row placed at top, right above the record rows. In this position, unbound row is selectable, editable and scrollable.
Bottom	Unbound row placed at bottom, right below record rows. In this position, unbound row is selectable, editable and scrollable.
FixedBottom	Unbound row placed at bottom of SfDataGrid. In this position, unbound row is not selectable, not editable and frozen when scrolling.

Below screenshot shows different unbound rows placed in all possible positions.



### Populating data for unbound rows

You can populate data for the unbound row by handling [QueryUnboundRow](#) event of Xamarin.Forms DataGrid. This event is fired for each cell of the unbound rows whenever the row gets refreshed or comes to view.

[GridUnboundRowEventArgs](#) of the [QueryUnboundRow](#) event provides information about the cell that triggered this event.

You can get or set the [GridUnboundRowEventArgs.Value](#) property based on the [UnboundAction](#). If [UnboundAction](#) is [QueryData](#) then you can set the value to be displayed. If the [UnboundAction](#) is [CommitData](#) then you can get the edited value.

### XML

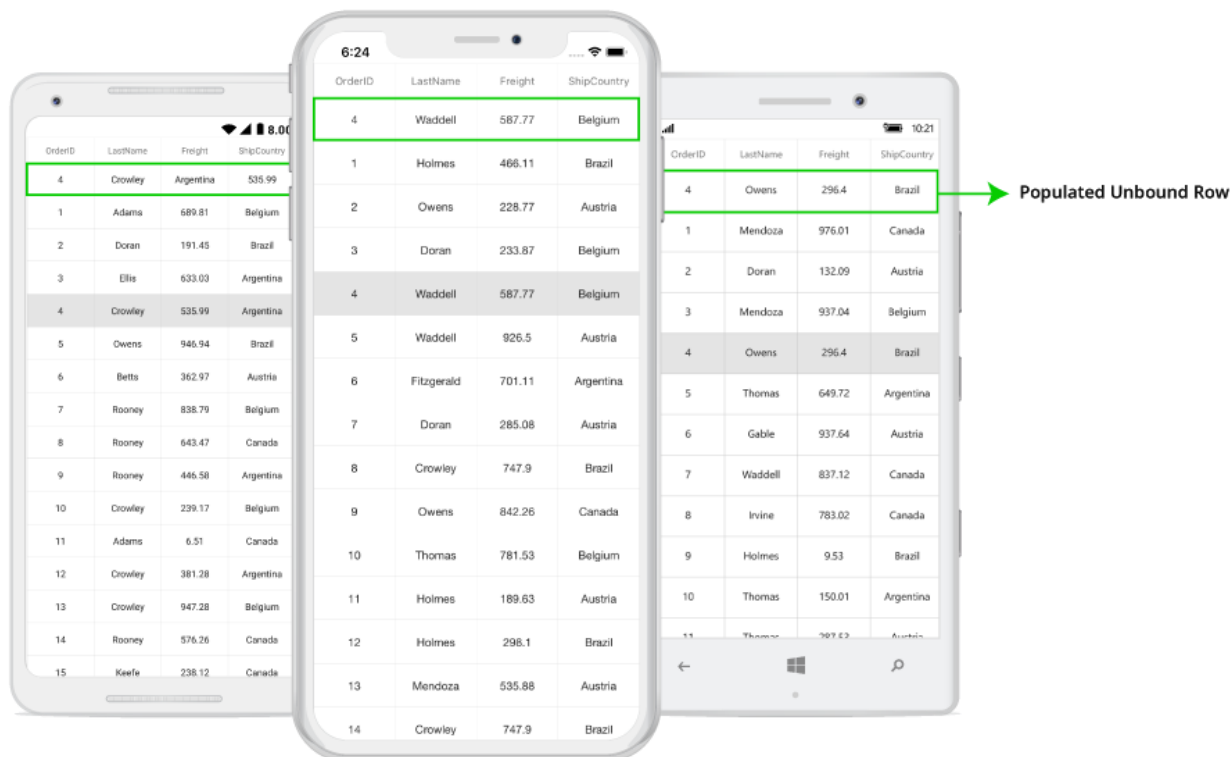
```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
SelectionMode="Multiple" >
<syncfusion:SfDataGrid.UnboundRows>
<syncfusion:GridUnboundRow Position="Top" />
</syncfusion:SfDataGrid.UnboundRows>
</syncfusion:SfDataGrid>
```

For example, now unbound row populated based on selected items in SfDataGrid.

### C#

```
dataGrid.GridViewCreated += DataGrid_GridViewCreated;
dataGrid.QueryUnboundRow += DataGrid_QueryUnboundRow;
private void DataGrid_GridViewCreated(object sender,
GridViewCreatedEventArgs e)
{
    dataGrid.SelectedItems.Add(viewModel.OrdersInfo[3]);
}
private void DataGrid_QueryUnboundRow(object sender, GridUnboundRowEventArgs
e)
{
    if (e.UnboundAction == UnboundActions.QueryData)
    {
        if (e.RowColumnIndex.ColumnIndex == 0)
        {
            e.Value = (dataGrid.CurrentItem as OrderInfo).OrderID;
            e.Handled = true;
        }
        else if (e.RowColumnIndex.ColumnIndex == 1)
        {
            e.Value = (dataGrid.CurrentItem as OrderInfo).LastName;
        }
        else if (e.RowColumnIndex.ColumnIndex == 2)
        {
            e.Value = (dataGrid.CurrentItem as OrderInfo).Freight;
            e.Handled = true;
        }
        else if (e.RowColumnIndex.ColumnIndex == 3)
        {
            e.Value = (dataGrid.CurrentItem as OrderInfo).ShipCountry;
            e.Handled = true;
        }
    }
}
```





## Refreshing the Unbound Rows at runtime

### *Add/Remove unbound rows*

You can add or remove unbound rows using [SfDataGrid.UnboundRows](#) property which reflects in UI immediately.

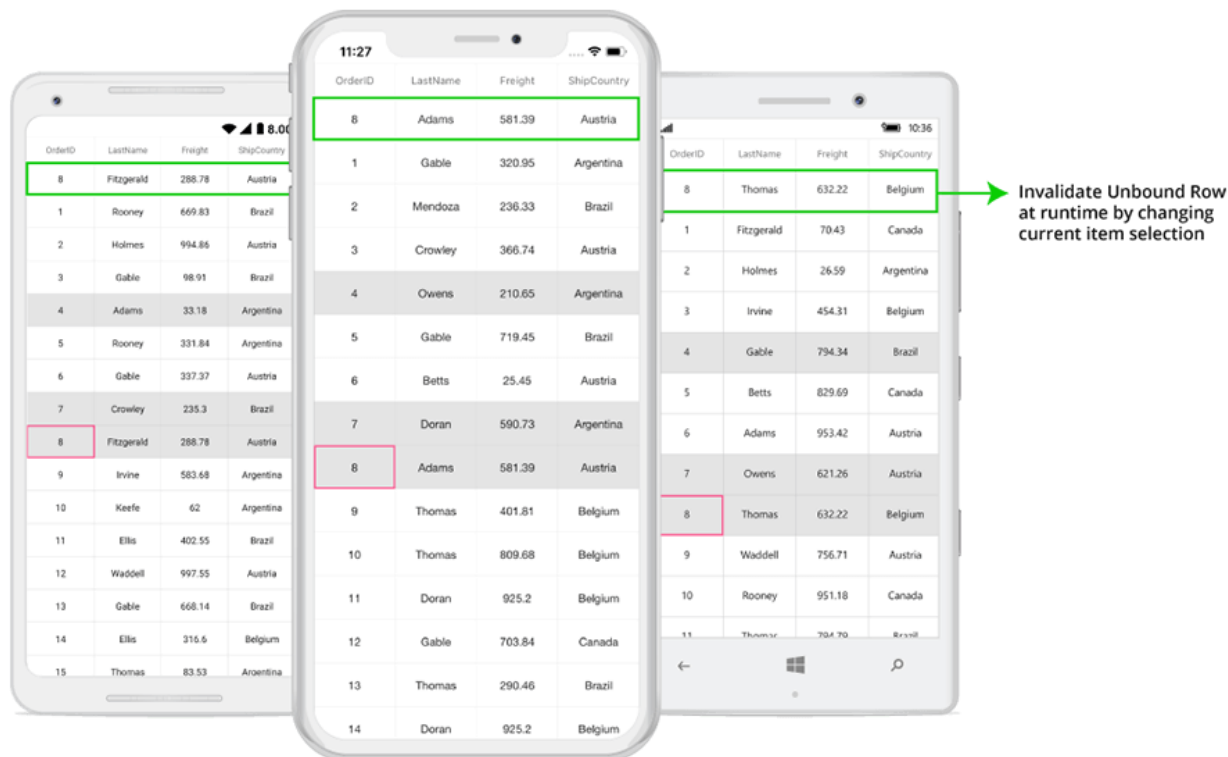
### *Trigger QueryUnboundRow event programmatically*

You can trigger the [QueryUnboundRow](#) event for the unbound row cells at runtime by calling [SfDataGrid.InValidateUnboundRow](#) method which invalidates the unbound row at the given index.

Here in the below code example, we have invalidated the unbound rows whenever selection is changed in the DataGrid.

### **C#**

```
this.dataGrid.SelectionChanged += dataGrid_SelectionChanged;
private void dataGrid_SelectionChanged(object sender,
GridSelectionChangedEventArgs e)
{
    this.dataGrid.InValidateUnboundRow(this.dataGrid.UnboundRows[0]);
}
```



## Editing in unbound rows

### *Cancel the editing for unbound row cell*

You can cancel the editing of unbound row cell in the event handler of [SfDataGrid.CurrentCellBeginEdit](#) event with the help of [SfDataGrid.GetUnboundRowAtRowIndex](#) method and row index.

### C#

```
using Syncfusion.SfDataGrid.XForms;
this.dataGrid.CurrentCellBeginEdit += DataGrid_CurrentCellBeginEdit;
private void DataGrid_CurrentCellBeginEdit(object sender,
GridCurrentCellBeginEditEventArgs e)
{
    var unboundRow =
dataGrid.GetUnboundRowAtRowIndex(e.RowColumnIndex.RowIndex);
    if (unboundRow == null)
        return;
    e.Cancel = true;
}
```

### *Saving edited unbound row cell value to external source*

You can get the edited value of unbound row cell from [GridUnboundRowEventArgs.Value](#) property of [QueryUnboundRow](#) event when [UnboundAction](#) is [CommitData](#).

### C#

```
private void DataGrid_QueryUnboundRow(object sender, GridUnboundRowEventArgs
e)
{
    if (e.UnboundAction == UnboundActions.CommitData)
```

```
{  
    var editedValue = e.Value;  
}
```

## Styling in Unbound rows

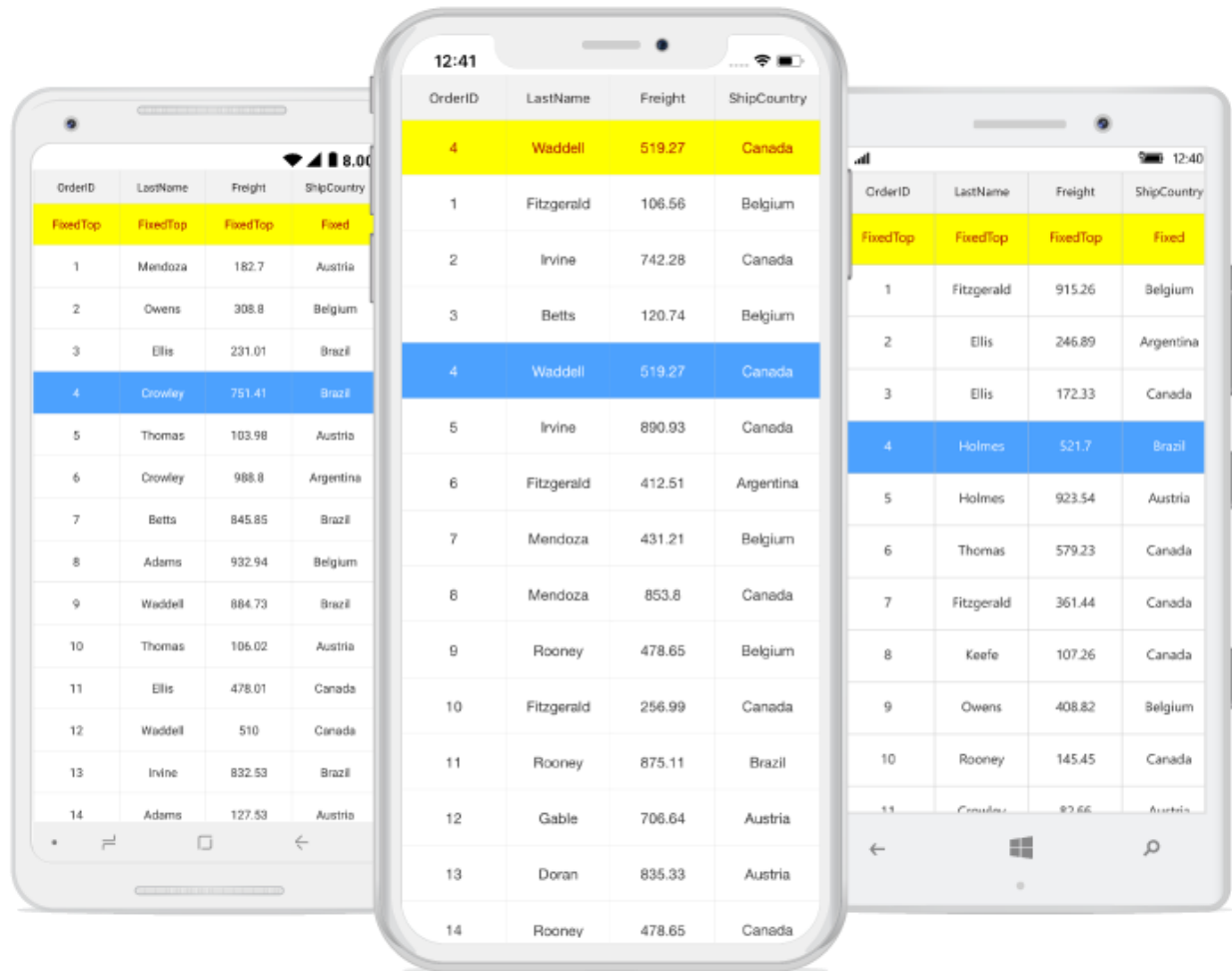
### *Override DataGrid style*

The Xamarin.Forms DataGrid applies style for unboundRow elements by writing a Style class overriding from DataGridStyle, and assigning it to the [SfDataGrid.GridStyle](#) property.

To apply custom style for UnboundRow, follow the code example:

### **C#**

```
SfDataGrid dataGrid = new SfDataGrid();  
dataGrid.GridStyle = new Dark ();  
public class UnboundRowStyle : DataGridStyle  
{  
    public UnboundRowStyle()  
    {  
    }  
    public override Color GetUnboundRowForegroundColor()  
    {  
        return Color.Red;  
    }  
    public override Color GetUnboundRowBackgroundColor()  
    {  
        return Color.Yellow;  
    }  
    public override FontAttributes GetUnboundRowFontAttribute()  
    {  
        return FontAttributes.Bold;  
    }  
}
```



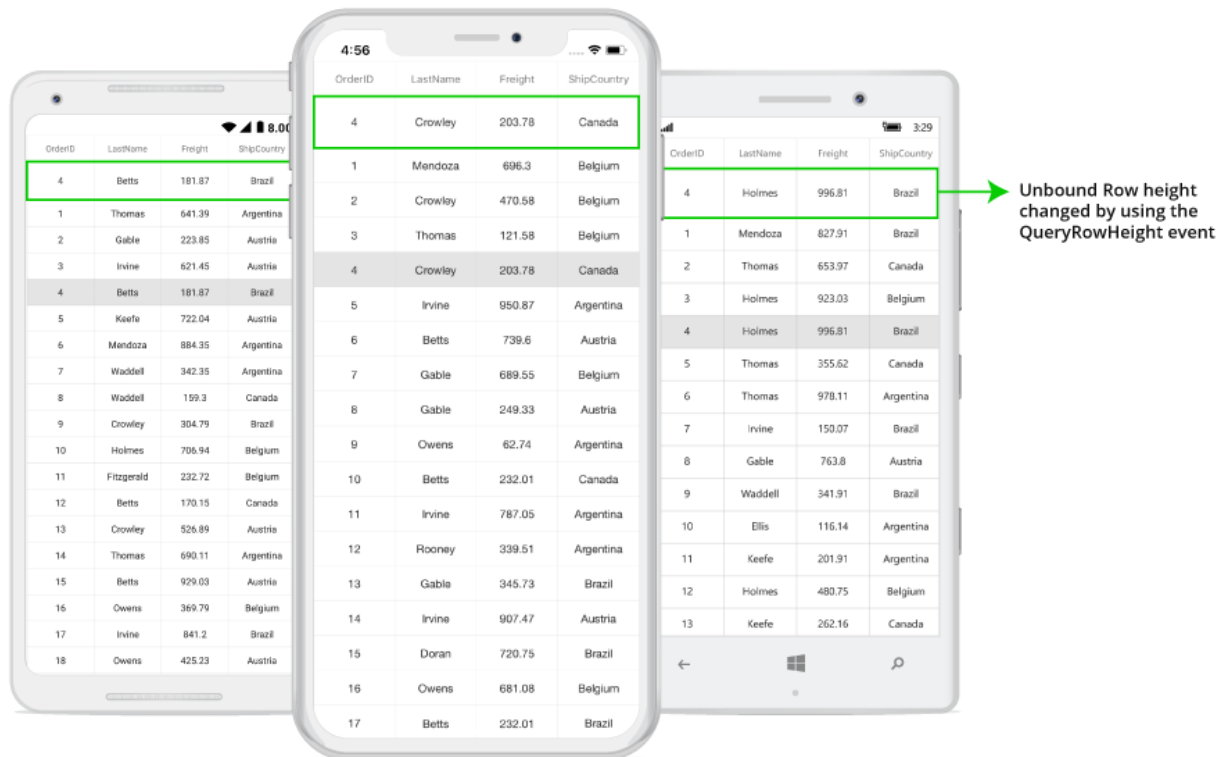
**Note:** You can also apply style to the unbound row using the `SfDataGrid.QueryCellStyle` and `SfDataGrid.QueryRowStyle` event by referring this [link](#).

### Changing unbound row height

You can change the height of unbound row using `SfDataGrid.QueryRowHeight` event.

### C#

```
using Syncfusion.SfDataGrid.XForms;
this.dataGrid.QueryRowHeight += DataGrid_QueryRowHeight;
private void DataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (dataGrid.IsUnboundRow(e.RowIndex))
    {
        e.Height = 60;
        e.Handled = true;
    }
}
```



### Get unbound rows

You can get the unbound row at the specified row index using [GetUnboundRow](#) method.

#### C#

```
using Syncfusion.SfDataGrid.XForms;
var unboundRow = dataGrid.GetUnboundRowAtRowIndex(1);
```

### Unbound Column

The data grid allows adding additional columns which are not bound with data object from the underlying data source. The unbound column can be added using the [SfDataGrid.GridUnboundColumn](#) class.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridUnboundColumn MappingName="DiscountPrice"
HeaderText="Discount Price"
Expression="UnitPrice*Discount"
Format="C">
</syncfusion:GridUnboundColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

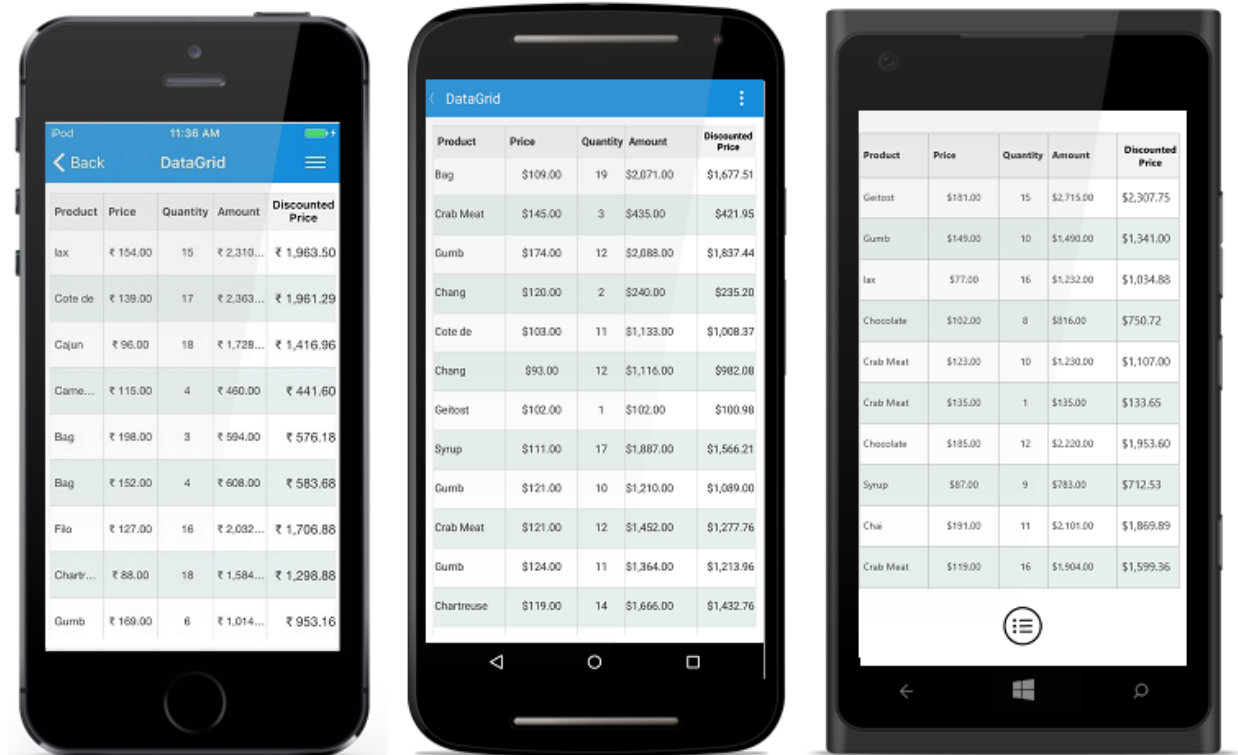
#### C#

```
SfDataGrid dataGrid = new SfDataGrid();
```

```

ViewModel viewModel = new ViewModel();
dataGrid.ItemsSource = viewModel.Orders;
GridUnboundColumn DiscountColumn = new GridUnboundColumn()
{
    MappingName = "DiscountPrice",
    HeaderText = "Discount Price",
    Expression = "UnitPrice*Discount",
    Format = "C"
};
this.dataGrid.Columns.Add(DiscountColumn);

```



**Note:** It is mandatory to specify the [GridColumn.MappingName](#) for `SfDataGrid.GridUnboundColumn` with some name to identify the column. It is not necessary to define name of the field in the data object.

### Populating data for the unbound column

Data for the unbound column can be populated by setting the [Expression](#) or [Format](#) property.

#### Using expression

The arithmetic or logic expression can be specified by using the `Expression` property to compute the display value. By default, `GridUnboundColumn` evaluates the expression with casing. The casing will be disabled while evaluating the expression by setting the [CaseSensitive](#) property to `false`.

List of supported arithmetic and logical operations are as follows:

Arithmetic operations	Operator
Add	+

Subtract	-
Multiply	*
Divide	/
Power	^
Mod	%
Greater Than	>
Less Than	<
Equal	=
GreaterThanOrEqual	>=
LessThanOrEqual	<=

### Logical Operations

Logical operations	Operators
AND	(char)135
OR	(char)136
NOT	(char)137

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridUnboundColumn MappingName="UnboundColumn"
HeaderText="Unbound Column"
Expression="UnitPrice*Discount" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
SfDataGrid dataGrid = new SfDataGrid();
ViewModel viewModel = new ViewModel();
dataGrid.ItemsSource = viewModel.Orders;
GridUnboundColumn DiscountColumn = new GridUnboundColumn()
{
    MappingName = "DiscountPrice",
    HeaderText = "Discount Price",
    Expression = "UnitPrice*Discount"
};
this.dataGrid.Columns.Add(DiscountColumn);
```

*Using format*

Value of other columns can be formatted, and the formatted value displayed in the unbound column using the `Format` property.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridUnboundColumn MappingName="DiscountPrice"
HeaderText="Discount Price"
Format="'{Discount}% for {OrderID}'" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

**C#**

```
SfDataGrid dataGrid = new SfDataGrid();
ViewModel viewModel = new ViewModel();
dataGrid.ItemsSource = viewModel.Orders;
GridUnboundColumn DiscountColumn = new GridUnboundColumn()
{
    MappingName = "DiscountPrice",
    HeaderText = "Discount Price",
    Format = "'{Discount}% for {OrderID}'"
};
this.dataGrid.Columns.Add(DiscountColumn);
```

*Using QueryUnboundColumnValue event*

The [QueryUnboundColumnValue](#) event is fired when value for the unbound column is queried. It provides the information about the cell that triggered this event. So, you can set the desired value for the grid cells of the unbound column. This event is triggered with the [GridUnboundColumnEventArgs](#).

The `GridUnboundColumnEventArgs` provides the following properties:

- [Column](#): Gets `GridColumn` of the cell that triggers this event.
- [OriginalSender](#): Gets the data grid raising this event.
- [Record](#): Gets the underlying row data.
- [UnboundAction](#): Defines the action for triggering this event.
- [Value](#): Gets or sets the value for `GridUnboundColumn` cell based on `UnboundAction`.

---

**Note:** `UnboundActions.CommitData` and `UnboundActions.PasteData` are currently not supported, and likely to be supported in future.

---

Populate the data for the unbound column by handling `QueryUnboundColumnValue` event which allows customizing the value of the `GridUnboundColumn`. `GridUnboundColumnEventArgs` exposes `Value` property, by which you can set the value for the grid cells of the unbound column based on the `UnboundAction`.

Refer to the following code example in which data for the unbound column is populated by handling the `QueryUnboundColumnValue` event:

**C#**



```

dataGrid.QueryUnboundColumnValue += DataGrid_QueryUnboundColumnValue;
private void DataGrid_QueryUnboundColumnValue(object sender,
GridUnboundColumnEventArgs e)
{
    if (e.UnboundAction == UnboundActions.QueryData)
    {
        var quantity =
        Convert.ToInt16(e.Record.GetType().GetProperty("Quantity").GetValue(e.Record));
        var unitPrice =
        Convert.ToInt16(e.Record.GetType().GetProperty("UnitPrice").GetValue(e.Record));
        var amount = quantity * unitPrice;
        e.Value = amount;
    }
}

```

## Export to Excel

The SfDataGrid supports exporting the data to Excel with several customization options like custom appearance, excluding specific columns, excluding headers, setting custom row height, setting custom column width, etc.

The following assemblies should be added for exporting the SfDataGrid to Excel file.

Project	Required assemblies
PCL	pcl\Syncfusion.SfGridConverter.XForms.dll pcl\Syncfusion.Compression.Portable.dll pcl\Syncfusion.Pdf.Portable.dll pcl\Syncfusion.XlsIO.Portable.dll

If you are using nuget package in the package, the following NuGet package should be installed to export the SfDataGrid to Excel file.

Project	Required package
Xamarin.Forms	Syncfusion.Xamarin.DataGridExport

The following code sample demonstrates how to create and display a SfDataGrid in view.

## XML

```

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition />
</Grid.RowDefinitions>
<StackLayout Grid.Row="0" Orientation="Vertical">
<Button x:Name="Excel" Text="ExportExcel" Clicked ="ExportToExcel_Clicked"
HeightRequest="50" />
</StackLayout>
<sfgrid:SfDataGrid x:Name="dataGrid"
AllowGroupExpandCollapse="True"

```

```
AllowSorting="True"
Grid.Row="1"
SelectionMode="Multiple"
ColumnSizer="None"
ItemsSource="{Binding OrdersInfo}"
AutoGenerateColumns="False" >
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn HeaderText="Order ID" MappingName="OrderID"/>
<sfgrid:GridTextColumn HeaderText="First Name" MappingName="FirstName"/>
<sfgrid:GridTextColumn HeaderText="Employee ID" MappingName="EmployeeID"/>
<sfgrid:GridNumericColumn HeaderText="Freight" MappingName="Freight"/>
<sfgrid:GridTextColumn HeaderText="Is Closed" MappingName="IsClosed"/>
<sfgrid:GridTextColumn HeaderText="Ship City" MappingName="ShipCity"/>
<sfgrid:GridDateTimeColumn HeaderText="Shipping Date"
MappingName="ShippingDate" />
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
</Grid>
```

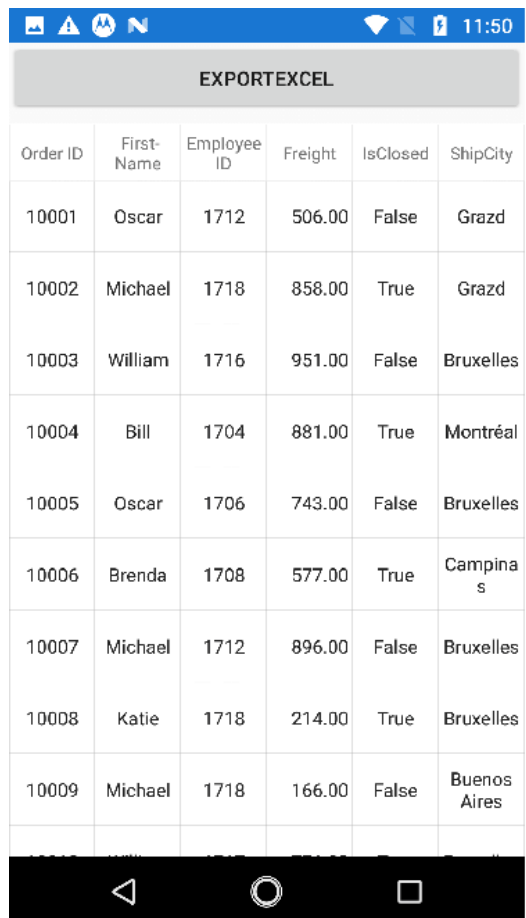
### C#

```
private void ExportToExcel_Clicked(object sender, EventArgs e)
{
    // Perform exporting to Excel sheet operations here.
}
```


You can export the data to Excel by using the [DataGridExcelExportingController.ExportToExcel](#) method by passing the SfDataGrid as an argument.


### C#

```
private void ExportToExcel_Clicked(object sender, EventArgs e)
{
    DataGridExcelExportingController excelExport = new
    DataGridExcelExportingController();
    var excelEngine = excelExport.ExportToExcel(this.dataGrid);
    var workbook = excelEngine.Excel.Workbooks[0];
    MemoryStream stream = new MemoryStream();
    workbook.SaveAs(stream);
    workbook.Close();
    excelEngine.Dispose();
    Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.xlsx",
    "application/msexcel", stream);
}
```



SfDataGrid  
to Excel





**Note:** SfDataGrid cannot export the GridTemplateColumn to PDF or Excel, since we cannot get the loaded views and draw them with the particular range, values etc from GridTemplateColumn.

### Exporting Options

You can also export the data to Excel with various customizing options by passing the grid and [DataGridExcelExportingOption](#) as arguments to the `ExportToExcel` method.

### C#

```
DataGridExcelExportingController excelExport = new
DataGridExcelExportingController ();
DataGridExcelExportingOption exportOption = new DataGridExcelExportingOption
();
exportOption.ExportColumnWidth = false;
exportOption.DefaultColumnWidth = 150;
var excelEngine = excelExport.ExportToExcel (this.dataGrid, exportOption);
```

The SfDataGrid provides several properties in `DataGridExcelExportingOption` class to customize the grid while exporting it to Excel.

#### Exporting formatted text and actual value

By default, the actual value will only be exported to Excel. To export the display text, set the `ExportMode` property as `Text`.

{% tabs%}

**C#**

```
DataGridExcelExportingController excelExport = new
DataGridExcelExportingController ();
DataGridExcelExportingOption options = new DataGridExcelExportingOption ();
options.ExportMode = ExportMode.Text;
var excelEngine = excelExport.ExportToExcel(dataGrid, options);
var workBook = excelEngine.Excel.Workbooks[0];
```

*Getting RowIndex, ColumnIndex and GridColumns for customization**ExcelColumnIndex*

The [ExcelColumnIndex Property](#) gets or internally sets the column index being exported to the Excel. Each column is exported based on this index to identify the current exporting column index.

*ExcelRowIndex*

The [ExcelRowIndex Property](#) gets the row index being exported to the Excel. Each row is exported based on this index to identify the current exporting row index.

*GridColumns*

Using the property [System.Collections.IEnumerable columns](#) you can get or set the [ExcludedColumns](#) columns collection which contains all the columns that are to be exported. The columns in the ExcludedColumns List will not be a member of the GridColumns collection.

*Customize header, groups and table summary when exporting**Export groups*

By default, all the groups in the data grid will be exported to Excel sheet. To export the data grid without groups, set the [DataGridExcelExportingOption.ExportGroups](#) property to `false`.

**C#**

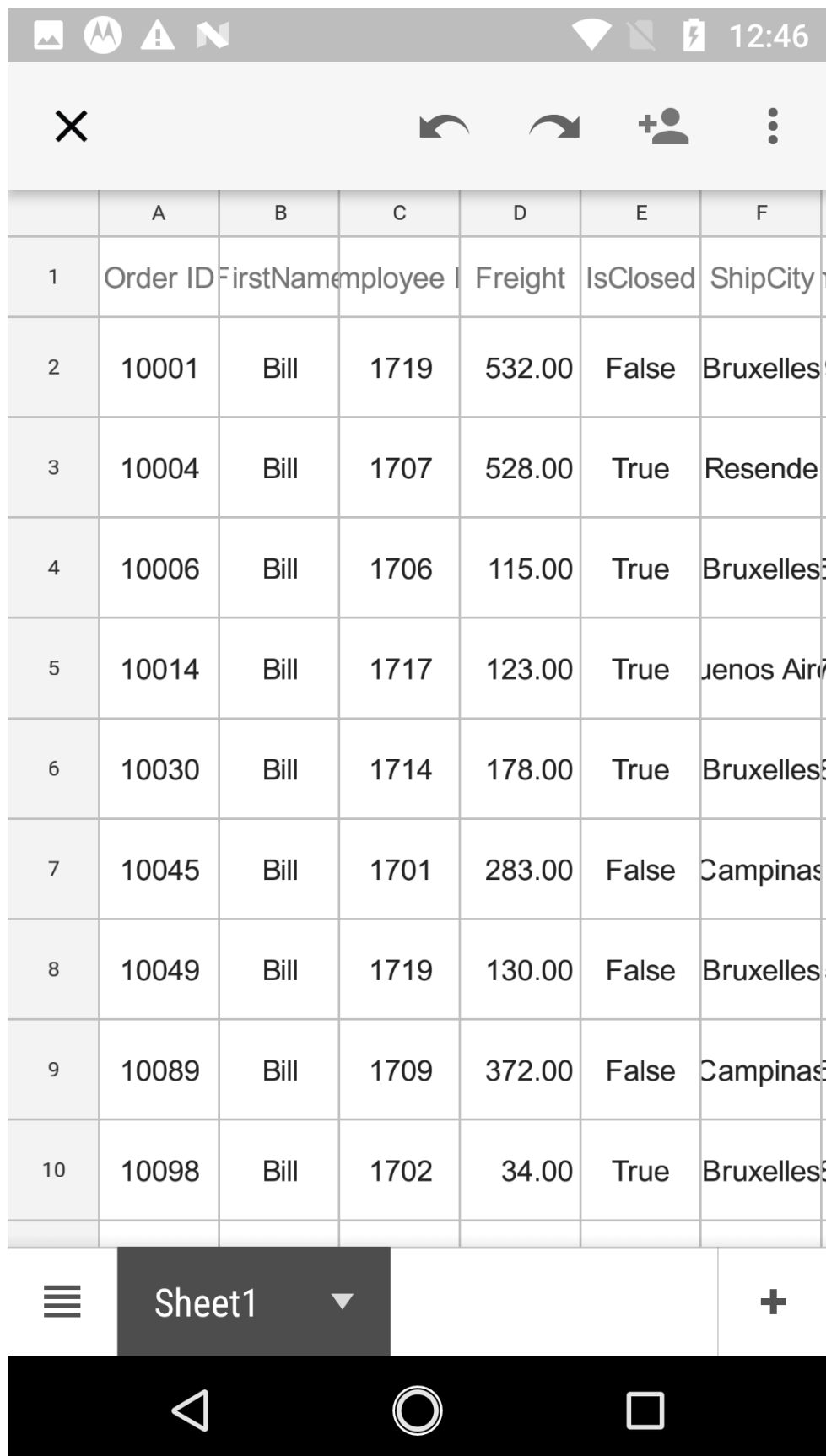
```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
option.ExportGroups = true;
```

- `ExportGroups = true;`

	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	Total Freight :\$96,432.00					
3	Total Salary :\$5,858.00 for 9 members					
4	10032	Bill	1710	830.00	True	Bruxelles
5	10064	Bill	1701	587.00	True	Montréal
6	10067	Bill	1714	108.00	False	o de Jane
7	10070	Bill	1705	898.00	True	Bruxelles7
8	10085	Bill	1706	329.00	False	Bruxelles3
9	10091	Bill	1709	672.00	False	Bruxelles3
10	10165	Bill	1718	812.00	False	Bruxelles

Sheet1

- `ExportGroups = false;`



	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	10001	Bill	1719	532.00	False	Bruxelles
3	10004	Bill	1707	528.00	True	Resende
4	10006	Bill	1706	115.00	True	Bruxelles
5	10014	Bill	1717	123.00	True	Jenos Air
6	10030	Bill	1714	178.00	True	Bruxelles
7	10045	Bill	1701	283.00	False	Campinas
8	10049	Bill	1719	130.00	False	Bruxelles
9	10089	Bill	1709	372.00	False	Campinas
10	10098	Bill	1702	34.00	True	Bruxelles

*Export header*

By default, the column headers will be exported to Excel sheet. To export the SfDataGrid without column headers, set the [DataGridExcelExportingOption.ExportHeader](#) property to `false`.

**C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.ExportHeader = false;
```





	A	B	C	D	E	F
1	10001	Frank	1712	493.00	False	o de Jane
2	10002	Ralph	1710	93.00	True	uenos Air
3	10003	Fiona	1714	771.00	False	awasse
4	10004	Michael	1710	874.00	True	Montréal
5	10005	Danielle	1711	20.00	False	Grazd
6	10006	Bill	1712	773.00	True	Grazd
7	10007	Ralph	1711	243.00	False	Montréal
8	10008	Oscar	1712	533.00	True	awasse
9	10009	Oscar	1700	961.00	False	o de Jane
10	10010	Ralph	1717	145.00	True	Grazd

### Export table summary

By default, table summaries in the data grid will be exported to Excel. To export the SfDataGrid without table summaries, set the [DataGridExcelExportingOption.ExportTableSummary](#) property to `false`.

### C#

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.ExportTableSummary = true;
```

- `ExportTableSummary = true;`

	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	Total Freight :\$96,432.00					
3	Total Salary :\$5,858.00 for 9 members					
4	10032	Bill	1710	830.00	True	Bruxelles
5	10064	Bill	1701	587.00	True	Montréal
6	10067	Bill	1714	108.00	False	o de Jane
7	10070	Bill	1705	898.00	True	Bruxelles7
8	10085	Bill	1706	329.00	False	Bruxelles3
9	10091	Bill	1709	672.00	False	Bruxelles5
10	10165	Bill	1718	812.00	False	Bruxelles

Sheet1

- `ExportTableSummary = false;`



	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	10001	Bill	1719	532.00	False	Bruxelles
3	10004	Bill	1707	528.00	True	Resende
4	10006	Bill	1706	115.00	True	Bruxelles
5	10014	Bill	1717	123.00	True	Jenos Air
6	10030	Bill	1714	178.00	True	Bruxelles
7	10045	Bill	1701	283.00	False	Campinas
8	10049	Bill	1719	130.00	False	Bruxelles
9	10089	Bill	1709	372.00	False	Campinas
10	10098	Bill	1702	34.00	True	Bruxelles

*Export groups with outlines*

To export the data grid with applied grouping, enable the group expand or collapse option in the Excel sheet by setting the [DataGridExcelExportingOption.AllowOutlining](#) to true. The default value of this property is false so, you cannot expand or collapse the group in the Excel sheet.

**C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
option.AllowOutlining = true;
```

*Exclude columns when exporting*

By default, all the columns (including hidden columns) in the SfDataGrid will be exported to Excel. To exclude some particular columns when exporting to Excel, add those columns to the [DataGridExcelExportingOption.ExcludeColumns](#) property in [DataGridExcelExportingOption](#) list.

**C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
var list = new List<string>();
list.Add("OrderID");
list.Add("LastName");
option.ExcludedColumns = list;
```

**Excluded Columns**

Order ID	First-Name	Employee ID	Freight	IsClosed	ShipCity
10001	Irene	1700	508.00	False	Grazd
10002	Irene	1702	913.00	True	Tsawassen
10003	Gina	1719	241.00	False	Bruxelles
10004	Oscar	1715	953.00	True	Bruxelles
10005	Oscar	1711	523.00	False	Grazd
10006	Fiona	1715	56.00	True	Grazd
10007	Ralph	1704	743.00	False	Grazd
10008	Kyle	1701	384.00	True	Buenos Aires
10009	Brenda	1700	293.00	False	Montréal

	A	B	C	D	E	F
1	First-Name	Employee ID	Freight	IsClosed	ShipCity	ShippingDate
2	Irene	1700	508.00	False	Grazd	9/7/2008
3	Irene	1702	913.00	True	Tsawassen	6/3/2009
4	Gina	1719	241.00	False	Bruxelles	4/13/2010
5	Oscar	1715	953.00	True	Bruxelles	0/21/2006
6	Oscar	1711	523.00	False	Grazd	3/18/2002
7	Fiona	1715	56.00	True	Grazd	5/12/2004
8	Ralph	1704	743.00	False	Grazd	9/10/2000
9	Kyle	1701	384.00	True	Jenos Aires	3/26/2011
10	Brenda	1700	293.00	False	Montréal	4/3/2005

### Customize Exporting Excel Version

The SfDataGrid allows exporting the data to Excel in specific versions by using the [DataGridExcelExportingOption.ExcelVersion](#) property.

#### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.ExcelVersion = Syncfusion.XlsIO.ExcelVersion.Excel2013;
```

### *Exporting the grid from a specified row and column index*

#### *StartColumnIndex*

By default, the exported SfDataGrid will start from the 0th column in the Excel sheet. You can specify the starting column by using the [DataGridExcelExportingOption.StartColumnIndex](#) property.

#### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.StartColumnIndex = 4;
```



	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7			Order ID	First Name	employee I	Freight I
8			10001	Irene	1716	665.00
9			10002	Torrey	1717	827.00
10			10003	Danielle	1718	477.00
11			10004	Oscar	1710	674.00
12			10005	Daniel	1700	926.00
13			10006	Gina	1703	72.00
14			10007	William	1718	707.00
15			10008	Oscar	1703	431.00



### StartRowIndex

By default, the exported SfDataGrid will start from the 0th row in the Excel sheet. You can specify the starting row by using the [DataGridExcelExportingOption.StartRowIndex](#) property.

### C#

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.StartRowIndex = 10;
```



	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7			Order ID	FirstName	employee I	Freight I
8			10001	Irene	1716	665.00
9			10002	Torrey	1717	827.00
10			10003	Danielle	1718	477.00
11			10004	Oscar	1710	674.00
12			10005	Daniel	1700	926.00
13			10006	Gina	1703	72.00
14			10007	William	1718	707.00
15			10008	Oscar	1703	431.00

*Exporting with sorting and filtering*

The SfDataGrid allows exporting the data grid to Excel with sorting and filtering options enabled on the column header in Excel sheet by setting the [DataGridExcelExportingOption.AllowSortingAndFiltering](#) to `true`. The default value of this property is false.

**C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.AllowSortingAndFiltering = true;
```

AllowSorting AndFiltering

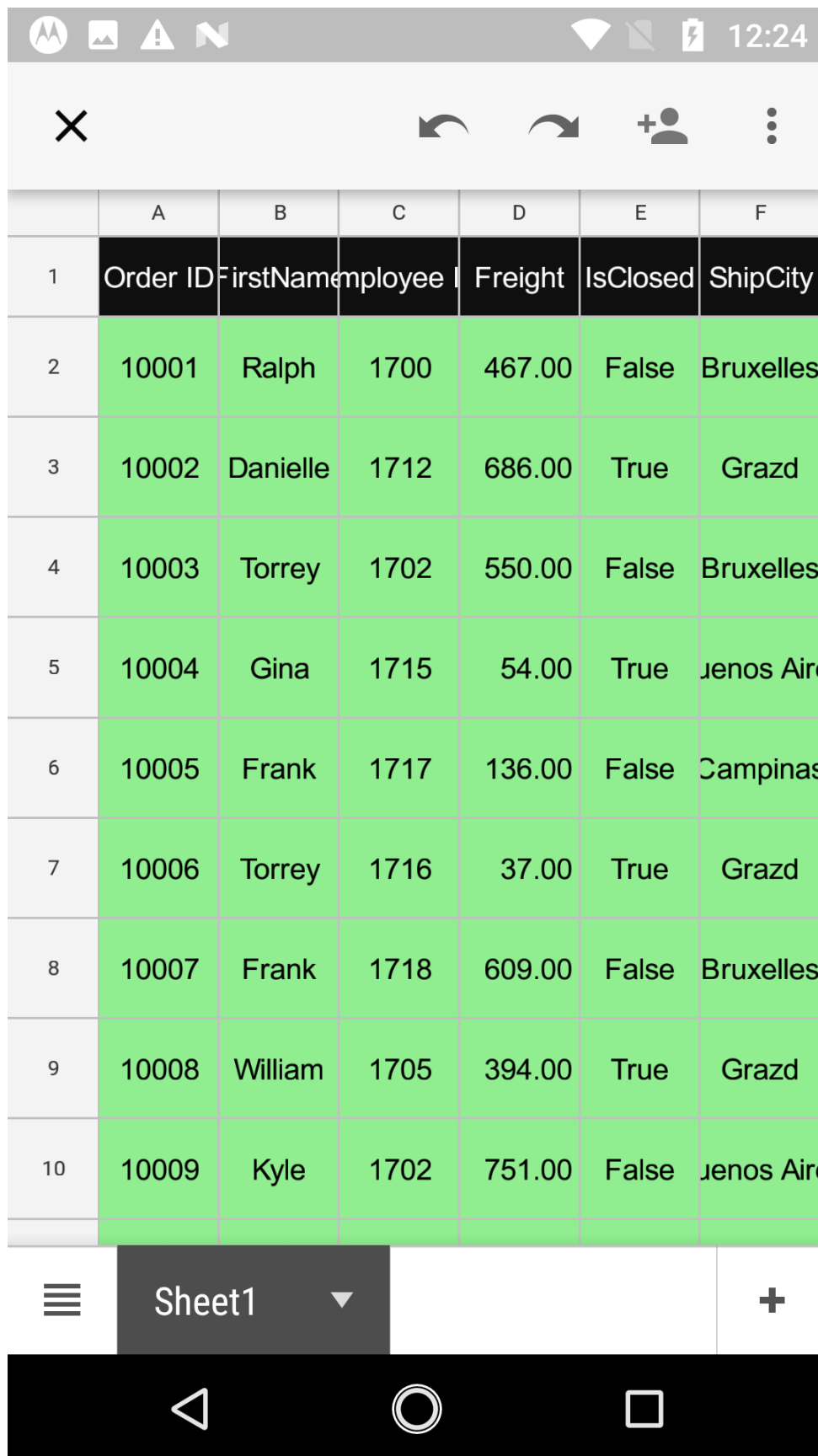
	A	B	C	D	E	F
1	OrderID	FirstName	LastName	Freight	Close	ShipCity
2	10001	Katie	1716	472.00	False	Campinas
3	10002	Bill	1701	831.00	True	Buenos Aires
4	10003	Daniel	1707	760.00	False	Grazd
5	10004	Ralph	1707	472.00	True	sawasse
6	10005	Ralph	1718	115.00	False	sawasse
7	10006	Irene	1705	887.00	True	Bruxelles
8	10007	Kyle	1705	927.00	False	sawasse
9	10008	Daniel	1701	444.00	True	sawasse
10	10009	Gina	1718	196.00	False	Grazd

*Applying styles while exporting*

The SfDataGrid allows exporting the data with the applied GridStyle by setting the [DataGridExcelExportingOption.ApplyGridStyle](#) to `true`. By default, the data will be exported without the GridStyle.

**C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.ApplyGridStyle = true;
```



	A	B	C	D	E	F
1	Order ID	FirstName	employee ID	Freight	IsClosed	ShipCity
2	10001	Ralph	1700	467.00	False	Bruxelles
3	10002	Danielle	1712	686.00	True	Grazd
4	10003	Torrey	1702	550.00	False	Bruxelles
5	10004	Gina	1715	54.00	True	Jenos Air
6	10005	Frank	1717	136.00	False	Campinas
7	10006	Torrey	1716	37.00	True	Grazd
8	10007	Frank	1718	609.00	False	Bruxelles
9	10008	William	1705	394.00	True	Grazd
10	10009	Kyle	1702	751.00	False	Jenos Air

### BottomTableSummaryStyle

The SfDataGrid supports exporting the bottom table summary with custom style by using the [DataGridExcelExportingOption.BottomTableSummaryStyle](#) property.

#### C#

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
option.BottomTableSummaryStyle = new ExportCellStyle()
{
    BackgroundColor = Xamarin.Forms.Color.Yellow,
    BorderColor = Xamarin.Forms.Color.Red,
    ForegroundColor = Xamarin.Forms.Color.Green,
};
```

	A	B	C	D	E	F
207	10118	William	1715	847.00	True	Jenos Air
208	10120	William	1711	547.00	True	Isawasse
209	10140	William	1716	33.00	True	Montréal
210	10141	William	1716	443.00	False	Grazd
211	10145	William	1703	991.00	False	Grazd
212	10147	William	1716	661.00	False	Bruxelles
213	10161	William	1715	746.00	False	Grazd
214	10165	William	1716	644.00	False	Montréal
215	10174	William	1714	446.00	True	Isawasse
216	10182	William	1713	233.00	True	Bruxelles
217	10186	William	1712	364.00	True	Jenos Air
218	Total Freight :\$97,824.00 for 200 OrderCount					
219						



### GroupCaptionStyle

The SfDataGrid supports exporting the GroupCaptionSummaries with custom style by using the [DataGridExcelExportingOption.GroupCaptionStyle](#) property.

#### C#

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
option.GroupCaptionStyle = new ExportCellStyle()
{
    BackgroundColor = Xamarin.Forms.Color.Yellow,
    BorderColor = Xamarin.Forms.Color.Red,
    ForegroundColor = Xamarin.Forms.Color.Green,
};
```




	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	Total Freight :\$97,824.00					
3	Total Salary :\$6,832.00 for 16 members					
4	10006	Bill	1716	479.00	True	sawasse
5	10012	Bill	1708	474.00	True	Grazd
6	10018	Bill	1718	284.00	True	Campinas
7	10023	Bill	1702	731.00	False	Bruxelles
8	10040	Bill	1718	688.00	True	Bruxelles
9	10048	Bill	1717	855.00	True	Grazd
10	10049	Bill	1700	222.00	False	sawasse

### HeaderStyle

The SfDataGrid allows exporting the column headers with custom style by using the [DataGridExcelExportingOption.HeaderStyle](#) property.

#### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
option.HeaderStyle = new ExportCellStyle()
{
    BackgroundColor = Xamarin.Forms.Color.Yellow,
    BorderColor = Xamarin.Forms.Color.Red,
    ForegroundColor = Xamarin.Forms.Color.Green,
};
```



	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	Total Freight :\$102,654.00					
3	Total Salary :\$7,671.00 for 12 members					
4	10001	Bill	1711	86.00	False	uenos Aires
5	10081	Bill	1707	908.00	False	Grazd
6	10097	Bill	1713	305.00	False	Grazd
7	10119	Bill	1703	629.00	False	Montréal
8	10128	Bill	1701	755.00	True	Bruxelles
9	10135	Bill	1703	419.00	False	sawasse
10	10143	Bill	1709	954.00	False	Bruxelles

### RecordStyle

The SfDataGrid allows exporting the records with custom style by using the [DataGridExcelExportingOption.RecordStyle](#) property.

### C#

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
option.RecordStyle = new ExportCellStyle()
{
    BackgroundColor = Xamarin.Forms.Color.Yellow,
    BorderColor = Xamarin.Forms.Color.Red,
    ForegroundColor = Xamarin.Forms.Color.Green,
};
```

	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	Total Freight :\$102,654.00					
3	Total Salary :\$7,671.00 for 12 members					
4	10001	Bill	1711	86.00	False	uenos Aires
5	10081	Bill	1707	908.00	False	Grazd
6	10097	Bill	1713	305.00	False	Grazd
7	10119	Bill	1703	629.00	False	Montréal
8	10128	Bill	1701	755.00	True	Bruxelles
9	10135	Bill	1703	419.00	False	sawasse
10	10143	Bill	1709	954.00	False	Bruxelles

### *TopTableSummaryStyle*

The SfDataGrid supports exporting the top table summary with custom style by using the [DataGridExcelExportingOption.TopTableSummaryStyle](#) property.

#### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
option.TopTableSummaryStyle = new ExportCellStyle()
{
    BackgroundColor = Xamarin.Forms.Color.Yellow,
    BorderColor = Xamarin.Forms.Color.Red,
    ForegroundColor = Xamarin.Forms.Color.Green,
};
```

	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	Total Freight :\$97,824.00					
3	Total Salary :\$6,832.00 for 16 members					
4	10006	Bill	1716	479.00	True	sawasse
5	10012	Bill	1708	474.00	True	Grazd
6	10018	Bill	1718	284.00	True	Campinas
7	10023	Bill	1702	731.00	False	Bruxelles
8	10040	Bill	1718	688.00	True	Bruxelles
9	10048	Bill	1717	855.00	True	Grazd
10	10049	Bill	1700	222.00	False	sawasse

Sheet1



### Exporting Unbound rows

By default, the Unbound rows will not be exported to the excel document. However, you can export the unbound rows to excel by setting the [DataGridExcelExportingOption.ExportUnboundRows](#) property as `true`.

#### C#

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.ExportUnboundRows = true;
```

### Exporting unbound columns

The `SfDataGrid.GridUnboundColumns` will be exported as [SfDataGrid.GridTextColumns](#) without specifying any code. You can customize the `SfDataGrid.GridUnboundColumns` as `SfDataGrid.GridTextColumns` by using the `CellExporting` and `RowExporting` events.

#### XML

```
<sfgrid:GridUnboundColumn Expression="OrderID * 12"  
HeaderFontAttribute="Bold"  
HeaderText="Unbound"  
HeaderTextAlignment="Start"  
MappingName="Unbound"  
Padding="5, 0, 0, 0"  
TextAlignment="Start">  
</sfgrid:GridUnboundColumn>
```

The following screenshot shows that the unbound column is exported to excel sheet along with text columns.

Carrier 


1:03 PM



Done

DataGrid.xlsx

CustomerID	Frieght	Country	Unbound
Maria Anders	ALFKI	Germany	1001
Ana Trujilo	ANATR	Mexico	1002
Ant Fuller	ANTON	Mexico	1003
Thomas Hardy	AROUT	UK	1004
Tim Adams	BERGS	Sweden	1005
Hanna Moos	BLAUS	Germany	1006
Andrew Fuller	BLONP	France	1007
Martin King	BOLID	Spain	1008
Lenny Lin	BONAP	France	1009
John Carter	BOTTM	Canada	1010
Lauro King	AROUT	UK	1011
Anne Wilson	BLAUS	Germany	1012
Maria Anders	ALFKI	Germany	1013
Ana Trujilo	ANATR	Mexico	1014
Ant Fuller	ANTON	Mexico	1015



### *ExportGroupSummary*

By default, the **GroupSummary** rows in the data grid will be exported to Excel. To export the **SfDataGrid** without group summaries, set the [DataGridExcelExportingOption.ExportGroupSummary](#) property to **false**.

### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
// Set false here to export the DataGrid without GroupSummary rows. The  
// default value is true.  
// option.ExportGroupSummary = false;
```

fx

OrderID

▼

	A	B	C	D	E
1	OrderID	CustomerID	ShipCountry	Customer	ShipCity
2	OrderID : 1001 - 1 Items				
3	1001	Ana Hardy	Mexico	ANATR	Mexico D.
4	Total OrderID: 1001 for 1 members				
5	OrderID : 1002 - 1 Items				
6	1002	Ant Fuller	Mexico	ANTON	Mexico D.
7	Total OrderID: 1002 for 1 members				
8	OrderID : 1003 - 1 Items				
9	1003	Thomas Hardy	UK	AROUT	London
10	Total OrderID: 1003 for 1 members				

Sheet1

+

*GroupSummaryStyle*

SfDataGrid supports exporting the GroupSummary rows with custom style by using the [DataGridExcelExportingOption.GroupSummaryStyle](#) property.

**C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
option.GroupSummaryStyle = new ExportCellStyle()
{
    BackgroundColor = Xamarin.Forms.Color.Red,
    BorderColor = Xamarin.Forms.Color.Yellow,
    ForegroundColor = Xamarin.Forms.Color.White,
};
```



## Saving options

### *Save as stream*

#### **C#**

```
var excelExport = new DataGridExcelExportingController();
var options = new DataGridExcelExportingOption();
var excelEngine = excelExport.ExportToExcel(dataGrid, options);
var workBook = excelEngine.Excel.Workbooks[0];
MemoryStream stream = new MemoryStream();
workBook.SaveAs(stream);
```

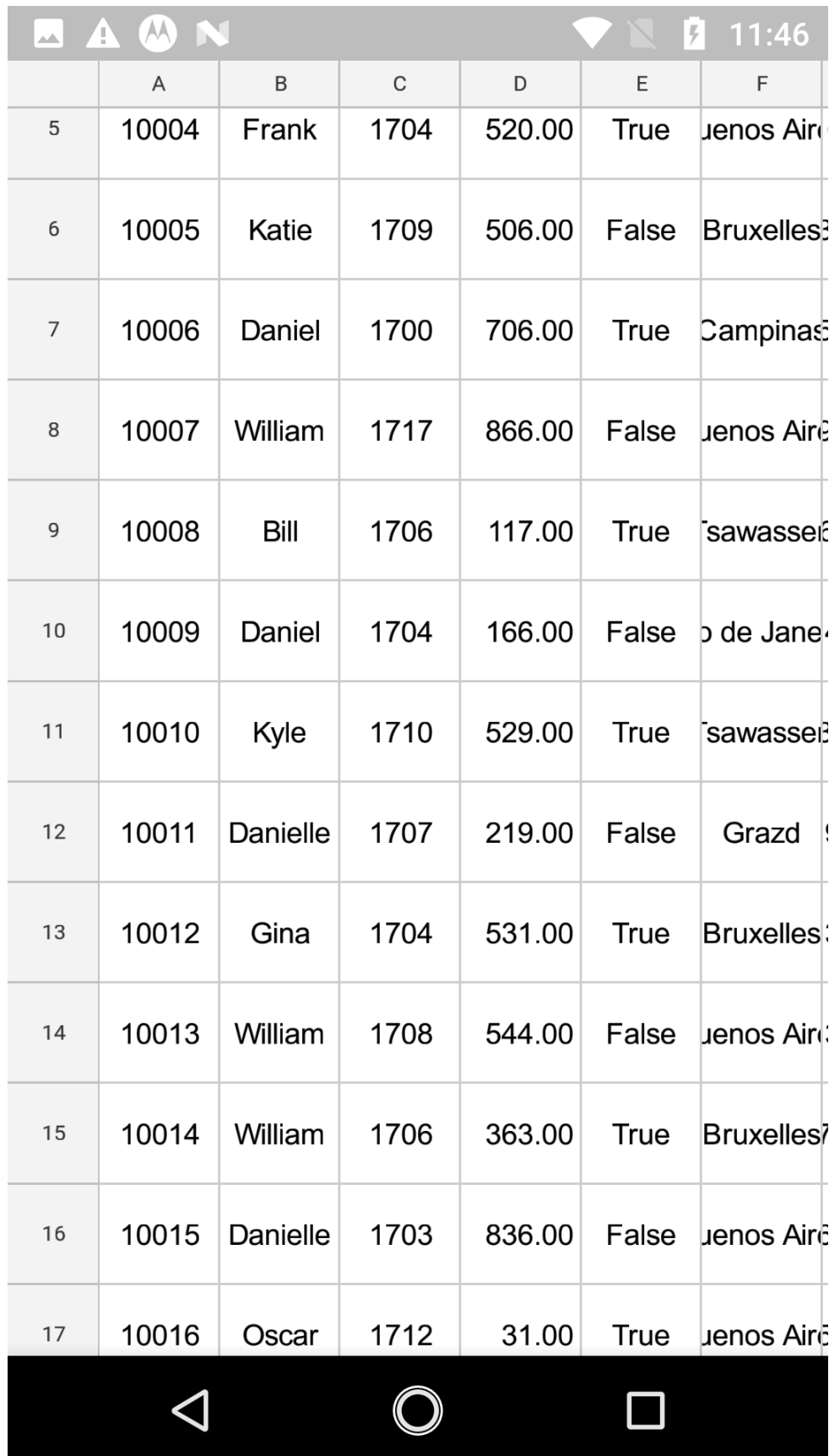
## Export AllPages in datagrid

When exporting to Excel using SfDataPager inside the SfDataGrid, it will only export the current page by default. However, you can export all the pages by setting the [DataGridExcelExportingOption.ExportAllPages](#) property to **true**.

#### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();
option.ExportAllPages = true;
```

- ExportAllPages = true;



	A	B	C	D	E	F
5	10004	Frank	1704	520.00	True	uenos Air
6	10005	Katie	1709	506.00	False	Bruxelles3
7	10006	Daniel	1700	706.00	True	Campinas5
8	10007	William	1717	866.00	False	uenos Air5
9	10008	Bill	1706	117.00	True	Isawasse6
10	10009	Daniel	1704	166.00	False	o de Jane6
11	10010	Kyle	1710	529.00	True	Isawasse3
12	10011	Danielle	1707	219.00	False	Grazd 1
13	10012	Gina	1704	531.00	True	Bruxelles:
14	10013	William	1708	544.00	False	uenos Air:
15	10014	William	1706	363.00	True	Bruxelles7
16	10015	Danielle	1703	836.00	False	uenos Air5
17	10016	Oscar	1712	31.00	True	uenos Air5



- `ExportAllPages = false;`



	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	10001	Danielle	1713	530.00	False	uenos Air
3	10002	Frank	1704	219.00	True	Resende
4	10003	Bill	1715	347.00	False	uenos Air
5	10004	Frank	1710	65.00	True	o de Jane
6	10005	Torrey	1718	759.00	False	uenos Air
7	10006	Gina	1716	585.00	True	Resende
8	10007	Oscar	1702	75.00	False	Grazd
9	10008	Danielle	1703	335.00	True	Bruxelles
10	10009	William	1707	376.00	False	Bruxelles

## Row Height and Column Width customization

### *DefaultColumnWidth*

The SfDataGrid allows customizing the column width in Excel file by using the [DataGridExcelExportingOption.DefaultColumnWidth](#) property. The `DefaultColumnWidth` value will be applied to all the columns in the Excel sheet.

#### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.DefaultColumnWidth = 100;
```

### *DefaultRowHeight*

The SfDataGrid allows customizing the row height in Excel file by using the [DataGridExcelExportingOption.DefaultRowHeight](#) property. The `DefaultRowHeight` value will be applied to all the rows in the Excel sheet.

#### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.DefaultRowHeight = 50;
```



	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	10001	Danielle	1713	530.00	False	uenos Air
3	10002	Frank	1704	219.00	True	Resende
4	10003	Bill	1715	347.00	False	uenos Air
5	10004	Frank	1710	65.00	True	o de Jane
6	10005	Torrey	1718	759.00	False	uenos Air
7	10006	Gina	1716	585.00	True	Resende
8	10007	Oscar	1702	75.00	False	Grazd
9	10008	Danielle	1703	335.00	True	Bruxelles
10	10009	William	1707	376.00	False	Bruxelles

### *ExportColumnWidth*

By default, the data grid columns will be exported to Excel with `DataGridExcelExportingOption.DefaultColumnWidth` value. You can also export the data grid to Excel with exact column widths by setting the [DataGridExcelExportingOption.ExportColumnWidth](#) property to `true`.

#### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.ExportColumnWidth = true;
```

### *ExportRowHeight*

By default, the data grid rows will be exported to Excel with `DataGridExcelExportingOption.DefaultRowHeight` value. You can also export the data grid to Excel with exact row heights by setting the [DataGridExcelExportingOption.ExportRowHeight](#) property to `true`.

#### **C#**

```
DataGridExcelExportingOption option = new DataGridExcelExportingOption();  
option.ExportRowHeight = true;
```



	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	10001	Irene	1703	930.00	False	Bruxelles
3	10002	Oscar	1709	134.00	True	sawassen
4	10003	Oscar	1710	693.00	False	uenos Air
5	10004	Brenda	1700	463.00	True	Bruxelles
6	10005	Danielle	1708	439.00	False	Grazd

## Events

### *Styling cells based on cell type.*

You can customize the row style based on the cell type when exporting to Excel by handling the **RowExporting** event.

#### **C#**

```
DataGridExcelExportingController excelExport = new
DataGridExcelExportingController();
excelExport.RowExporting += ExcelExport_RowExporting;
private void ExcelExport_RowExporting(object sender,
DataGridRowExcelExportingEventArgs e)
{
    if (!(e.Record.Data is OrderInfo))
        return;
    if (e.RowType == ExportRowType.Record)
    {
        e.Range.CellStyle.ColorIndex = Syncfusion.XlsIO.ExcelKnownColors.Red;
    }
}
```

## Cell customization in Excel while exporting

### *Customize cell value while exporting*

You can customize the cell values when exporting to Excel by handling the **CellExporting** event.

#### **C#**

```
DataGridExcelExportingController excelExport = new
DataGridExcelExportingController();
excelExport.CellExporting += ExcelExport_CellExporting;
private void ExcelExport_CellExporting(object sender,
DataGridCellExcelExportingEventArgs e)
{
    if (e.CellType == ExportCellType.RecordCell && e.ColumnName == "IsClosed")
    {
        if ((bool)e.CellValue)
            e.CellValue = "Y";
        else
            e.CellValue = "N";
    }
}
```

### *Changing row style in Excel based on data*

You can customize the row style based on the row data when exporting to Excel by handling the **RowExporting** event.

#### **C#**

```
DataGridExcelExportingController excelExport = new
DataGridExcelExportingController();
excelExport.RowExporting += ExcelExport_RowExporting;
private void ExcelExport_RowExporting(object sender,
DataGridRowExcelExportingEventArgs e)
{
    if (!(e.Record.Data is OrderInfo))
```

```

return;
if (e.RowType == ExportRowType.Record)
{
    if ((e.Record.Data as OrderInfo).IsClosed)
        e.Range.CellStyle.ColorIndex = Syncfusion.XlsIO.ExcelKnownColors.Yellow;
    else
        e.Range.CellStyle.ColorIndex = Syncfusion.XlsIO.ExcelKnownColors.Red;
}
}

```

### *Customize the cells based on column name*

You can customize the column style based on the row data when exporting to Excel by handling the **CellExporting** event.

### **C#**

```

DataGridExcelExportingController excelExport = new
DataGridExcelExportingController();
excelExport.RowExporting += ExcelExport_RowExporting;
private void ExcelExport_RowExporting(object sender,
DataGridRowExcelExportingEventArgs e)
{
    if (!(e.Record.Data is OrderInfo))
        return;
    if (e.CellType == ExportCellType.RecordCell && e.ColumnName == "FirstName")
    {
        e.Range.CellStyle.ColorIndex = Syncfusion.XlsIO.ExcelKnownColors.Red;
    }
}

```

The SfDataGrid provides the following events when exporting to Excel:

- [RowExporting](#): Raised when exporting a row at the execution time.
- [CellExporting](#): Raised when exporting a cell at the execution time.

### *Row exporting*

The [DataGridRowExcelExportingEventHandler](#) delegate allows customizing the styles for record rows and group caption rows. The **RowExporting** event is triggered with [DataGridRowExcelExportingEventArgs](#) that contains the following properties:

- [Range](#): Specifies the Excel range to be exported. It provides full access to the exporting cell in Excel.
- [Record](#): Gets the collection of the exported underlying data objects.
- [RowType](#): Specifies the row type by using **ExportRowType** enum. You can use this property to check the row type and apply different styles based on the row type.
- [Worksheet](#): Sets the **Worksheet** properties such as sheet protection, gridlines, and so on.

You can use these events to customize the properties of exported grid rows. The following code example illustrates how to change the background color of the record rows and caption summary rows when exporting.



**C#**

```
//HandlingRowExportingEvent for exporting to excel
DataGridExcelExportingController excelExport = new
DataGridExcelExportingController ();
excelExport.RowExporting += excelExport_RowExporting;
void excelExport_RowExporting (object sender,
DataGridRowExcelExportingEventArgs e)
{
if (e.RowType == ExportRowType.Record) {
if ((e.Record.Data as OrderInfo).IsClosed)
e.Range.CellStyle.ColorIndex = Syncfusion.XlsIO.ExcelKnownColors.Yellow;
else
e.Range.CellStyle.ColorIndex = Syncfusion.XlsIO.ExcelKnownColors.LightGreen;
}
if (e.RowType == ExportRowType.CaptionSummary) {
e.Range.CellStyle.ColorIndex =
Syncfusion.XlsIO.ExcelKnownColors.Grey_25_percent;
}
}
```



	A	B	C	D	E	F
1	Order ID	FirstName	employee I	Freight	IsClosed	ShipCity
2	10001	Torrey	1700	372.00	False	uenos Air
3	10002	Michael	1710	677.00	True	Bruxelles
4	10003	Michael	1705	659.00	False	Resende
5	10004	Gina	1719	984.00	True	uenos Air
6	10005	Danielle	1707	775.00	False	Campinas
7	10006	Irene	1708	961.00	True	Grazd
8	10007	Ralph	1707	325.00	False	uenos Air
9	10008	Gina	1716	160.00	True	Montréal
10	10009	Kyle	1705	629.00	False	o de Jane

### CellExporting

The [DataGridCellExcelExportingEventHandler](#) delegate allows customizing the styles for header cells, record cells, and group caption cells. The [CellExporting](#) event is triggered with [DataGridCellExcelExportingEventArgs](#) that contains the following properties:

- [CellType](#): Specifies the cell type by using [ExportCellType](#) enum. Checks the cell type and apply different cell styles based on the cell type.
- [CellValue](#): Contains the exported actual value. Applies formatting in Excel by using the [Range](#) property.
- [ColumnName](#): Specifies the column name (MappingName) of the exporting cell. You can apply formatting for a particular column by checking the [column name](#).
- [Handled](#): Determines whether the cell is exported to Excel or not.
- [Range](#): Specifies the Excel range to be exported. It provides full access to the exporting cell in Excel.
- [Record](#): Gets the collection of the exported underlying data objects.

You can use these events to customize the properties of the grid cells exported to excel. The following code example illustrates how to customize the background color, foreground color, and cell value of the header cells, record cells, and caption summary cells when exporting.

### C#

```
//HandlingCellExportingEvent for exporting to Excel
DataGridExcelExportingController excelExport = new
DataGridExcelExportingController ();
excelExport.CellExporting += excelExport_CellExporting;
void excelExport_CellExporting(object sender,
DataGridCellExcelExportingEventArgs e)
{
    if (e.CellType == ExportCellType.HeaderCell) {
        e.Range.CellStyle.ColorIndex = Syncfusion.XlsIO.ExcelKnownColors.Blue;
        e.Range.CellStyle.PatternColorIndex =
        Syncfusion.XlsIO.ExcelKnownColors.White;
        e.Range.CellStyle.BeginUpdate();
        e.Range.CellStyle.Borders.LineStyle =
        Syncfusion.XlsIO.ExcelLineStyle.Dash_dot;
        e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalDown].LineStyle =
        ExcelLineStyle.Dash_dot;
        e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalUp].LineStyle =
        ExcelLineStyle.Dash_dot_dot;
        e.Range.CellStyle.Borders.Color = ExcelKnownColors.Black;
        e.Range.CellStyle.EndUpdate();
    }
    if (e.CellType == ExportCellType.RecordCell) {
        e.Range.CellStyle.ColorIndex = Syncfusion.XlsIO.ExcelKnownColors.Yellow;
        e.Range.CellStyle.PatternColorIndex =
        Syncfusion.XlsIO.ExcelKnownColors.Black;
        e.Range.CellStyle.BeginUpdate();
        e.Range.CellStyle.Borders.LineStyle =
        Syncfusion.XlsIO.ExcelLineStyle.Dash_dot;
        e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalDown].LineStyle =
        ExcelLineStyle.Dash_dot;
    }
}
```

```

e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalUp].LineStyle =
ExcelLineStyle.Dash_dot_dot;
e.Range.CellStyle.Borders.Color = ExcelKnownColors.Black;
e.Range.CellStyle.EndUpdate();
}
if (e.CellType == ExportCellType.GroupCaptionCell) {
e.Range.CellStyle.ColorIndex =
Syncfusion.XlsIO.ExcelKnownColors.Grey_25_percent;
e.Range.CellStyle.PatternColorIndex =
Syncfusion.XlsIO.ExcelKnownColors.Blue;
e.Range.CellStyle.BeginUpdate();
e.Range.CellStyle.Borders.Linestyle =
Syncfusion.XlsIO.ExcelLineStyle.Dash_dot;
e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalDown].LineStyle =
ExcelLineStyle.Dash_dot;
e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalUp].LineStyle =
ExcelLineStyle.Dash_dot_dot;
e.Range.CellStyle.Borders.Color = ExcelKnownColors.Black;
e.Range.CellStyle.EndUpdate();
}
// You can also set the desired values for the CaptionSummary rows and
// GroupSummary rows as shown below
//if (e.CellType == ExportCellType.GroupCaptionCell) {
//    e.Range.CellStyle.ColorIndex =
//        Syncfusion.XlsIO.ExcelKnownColors.Grey_25_percent;
//    e.Range.CellStyle.PatternColorIndex =
//        Syncfusion.XlsIO.ExcelKnownColors.Blue;
//    e.Range.CellStyle.BeginUpdate();
//    e.Range.CellStyle.Borders.Linestyle =
//        Syncfusion.XlsIO.ExcelLineStyle.Dash_dot;
//    e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalDown].LineStyle =
//        ExcelLineStyle.Dash_dot;
//    e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalUp].LineStyle =
//        ExcelLineStyle.Dash_dot_dot;
//    e.Range.CellStyle.Borders.Color = ExcelKnownColors.Black;
//    e.Range.CellStyle.EndUpdate();
//}
//if (e.CellType == ExportCellType.GroupSummaryCell){
//    e.Range.CellStyle.ColorIndex =
//        Syncfusion.XlsIO.ExcelKnownColors.Light_yellow;
//    e.Range.CellStyle.PatternColorIndex =
//        Syncfusion.XlsIO.ExcelKnownColors.Black;
//    e.Range.CellStyle.BeginUpdate();
//    e.Range.CellStyle.Borders.Linestyle =
//        Syncfusion.XlsIO.ExcelLineStyle.Dashed;
//    e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalDown].LineStyle =
//        ExcelLineStyle.Dashed;
//    e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalUp].LineStyle =
//        ExcelLineStyle.Dashed;
//    e.Range.CellStyle.Borders.Color = ExcelKnownColors.Red;
//    e.Range.CellStyle.EndUpdate();
//}
}

```

	A	B	C	D	E	F
1	Order ID	FirstName	employee	Freight	IsClosed	ShipCity
2	10001	Torrey	1712	568.00	False	Montréal
3	10002	Fiona	1701	924.00	True	Grazd
4	10003	Bill	1702	550.00	False	Bruxelles
5	10004	Bill	1701	806.00	True	Grazd
6	10005	Katie	1719	387.00	False	Campinas
7	10006	Fiona	1717	676.00	True	Grazd
8	10007	Gina	1705	98.00	False	Montréal
9	10008	Katie	1715	968.00	True	sawasse
10	10009	Ralph	1715	596.00	False	Grazd

Customize exported workbook and worksheet

#### *Saving a workbook*

The following code snippet explains how to save the converted Excel sheet in our local device.

#### **C#**

```
// Need to define the Interfaces in-order to use it in platform wise using  
Dependency Service  
public interface ISave  
{  
    void Save(string filename, string contentType, MemoryStream stream);  
}  
public interface ISaveWindows  
{  
    Task Save(string filename, string contentType, MemoryStream stream);  
}  
// In Android renderer project  
public class SaveAndroid : ISave  
{  
    public void Save(string filename, string contentType, MemoryStream stream)  
    {  
        string exception = string.Empty;  
        string root = null;  
        if (Android.OS.Environment.IsExternalStorageEmulated)  
        {  
            root = Android.OS.Environment.ExternalStorageDirectory.ToString();  
        }  
        else  
            root = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);  
        Java.IO.File myDir = new Java.IO.File(root + "/Syncfusion");  
        myDir.Mkdir();  
        Java.IO.File file = new Java.IO.File(myDir, filename);  
        if (file.Exists()) file.Delete();  
        try  
        {  
            FileOutputStream outs = new FileOutputStream(file);  
            outs.Write(stream.ToArray());  
            outs.Flush();  
            outs.Close();  
        }  
        catch (Exception e)  
        {  
            exception = e.ToString();  
        }  
        if (file.Exists() && contentType != "application/html")  
        {  
            Android.Net.Uri path = Android.Net.Uri.FromFile(file);  
            string extension =  
                Android.Webkit.MimeTypeMap.GetFileExtensionFromUrl(Android.Net.Uri.FromFile(  
                    file).ToString());  
            string mimeType =  
                Android.Webkit.MimeTypeMap.Singleton.GetMimeTypeFromExtension(extension);  
            Intent intent = new Intent(Intent.ActionView);  
            intent.SetDataAndType(path, mimeType);  
            Forms.Context.StartActivity(Intent.CreateChooser(intent, "Choose App"));  
        }  
    }  
}
```

```
}
// In iOS renderer project
public class SaveIOS : ISave
{
    void ISave.Save(string filename, string contentType, MemoryStream stream)
    {
        string exception = string.Empty;
        string path = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        string filePath = Path.Combine(path, filename);
        try
        {
            FileStream fileStream = File.Open(filePath, FileMode.Create);
            stream.Position = 0;
            stream.CopyTo(fileStream);
            fileStream.Flush();
            fileStream.Close();
        }
        catch (Exception e)
        {
            exception = e.ToString();
        }
        if (contentType == "application/html" || exception != string.Empty)
            return;
        UIViewController currentController =
            UIApplication.SharedApplication.KeyWindow.RootViewController;
        while (currentController.PresentedViewController != null)
            currentController = currentController.PresentedViewController;
        UIView currentView = currentController.View;
        QLPreviewController previewController = new QLPreviewController();
        QLPreviewItem item = new QLPreviewItemBundle(filename, filePath);
        previewController.DataSource = new PreviewControllerDS(item);
        currentController.PresentViewController((UIViewController)previewController,
            true, (Action)null);
    }
}

public class PreviewControllerDS : QLPreviewControllerDataSource
{
    private QLPreviewItem _item;
    public PreviewControllerDS(QLPreviewItem item)
    {
        _item = item;
    }
    public override nint PreviewItemCount (QLPreviewController controller)
    {
        return (nint)1;
    }
    public override IQLPreviewItem GetPreviewItem (QLPreviewController
        controller, nint index)
    {
        return _item;
    }
}

public class QLPreviewItemFileSystem : QLPreviewItem
{
    string _fileName, _filePath;
    public QLPreviewItemFileSystem(string fileName, string filePath)
    {

```

```
_fileName = fileName;
_filePath = filePath;
}
public override string ItemTitle
{
    get
    {
        return _fileName;
    }
}
public override NSURL ItemUrl
{
    get
    {
        return NSURL.FromFilename(_filePath);
    }
}
}
public class QLPreviewItemBundle : QLPreviewItem
{
    string _fileName, _filePath;
    public QLPreviewItemBundle(string fileName, string filePath)
    {
        _fileName = fileName;
        _filePath = filePath;
    }
    public override string ItemTitle
    {
        get
        {
            return _fileName;
        }
    }
    public override NSURL ItemUrl
    {
        get
        {
            var documents = NSBundle.MainBundle.BundlePath;
            var lib = Path.Combine(documents, _filePath);
            var url = NSURL.FromFilename(lib);
            return url;
        }
    }
}
// In UWP renderer project
public class SaveWindows : ISaveWindows
{
    public async Task Save(string filename, string contentType, MemoryStream stream)
    {
        if (Device.Idiom != TargetIdiom.Desktop)
        {
            StorageFolder local = Windows.Storage.ApplicationData.Current.LocalFolder;
            StorageFile outFile = await local.CreateFileAsync(filename,
                CreationCollisionOption.ReplaceExisting);
            using (Stream outputStream = await outFile.OpenStreamForWriteAsync())
            {

```



```

outStream.Write(stream.ToArray(), 0, (int)stream.Length);
}
if (contentType != "application/html")
await Windows.System.Launcher.LaunchFileAsync(outFile);
}
else
{
StorageFile storageFile = null;
FileSavePicker savePicker = new FileSavePicker();
savePicker.SuggestedStartLocation = PickerLocationId.Desktop;
savePicker.SuggestedFileName = filename;
switch (contentType)
{
case "application/vnd.openxmlformats-officedocument.presentationml.presentation":
savePicker.FileTypeChoices.Add("PowerPoint Presentation", new List<string>()
{ ".pptx", });
break;
case "application/msexcel":
savePicker.FileTypeChoices.Add("Excel Files", new List<string>() { ".xlsx",
});
break;
case "application/msword":
savePicker.FileTypeChoices.Add("Word Document", new List<string>() { ".docx"
});
break;
case "application/pdf":
savePicker.FileTypeChoices.Add("Adobe PDF Document", new List<string>() {
".pdf" });
break;
case "application/html":
savePicker.FileTypeChoices.Add("HTML Files", new List<string>() { ".html"
});
break;
}
storageFile = await savePicker.PickSaveFileAsync();
using (Stream outStream = await storageFile.OpenStreamForWriteAsync())
{
outStream.Write(stream.ToArray(), 0, (int)stream.Length);
outStream.Flush();
outStream.Dispose();
}
stream.Flush();
stream.Dispose();
await Windows.System.Launcher.LaunchFileAsync(storageFile);
}
}
}

```

### [Worksheet customization](#)

#### [Setting borders](#)

The SfDataGrid allows exporting the data to Excel with custom borders style in Excel sheet by handling the `CellExporting` event.

#### **C#**

```
private void ExcelExport_CellExporting(object sender,
DataGridCellExcelExportingEventArgs e)
{
    if (e.CellType == ExportCellType.RecordCell)
    {
        e.Range.CellStyle.BeginUpdate();
        e.Range.CellStyle.Borders.LineStyle =
        Syncfusion.XlsIO.ExcelLineStyle.Dash_dot;
        e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalDown].LineStyle =
        ExcelLineStyle.Dash_dot;
        e.Range.CellStyle.Borders[ExcelBordersIndex.DiagonalUp].LineStyle =
        ExcelLineStyle.Dash_dot_dot;
        e.Range.CellStyle.Borders.Color = ExcelKnownColors.Black;
        e.Range.CellStyle.EndUpdate();
    }
}
```



	A	B	C	D	E	F
1	Order ID	FirstName	employee	Freight	IsClosed	ShipCity
2	10001	Torrey	1712	568.00	False	Montréal
3	10002	Fiona	1701	924.00	True	Grazd
4	10003	Bill	1702	550.00	False	Bruxelles
5	10004	Bill	1701	806.00	True	Grazd
6	10005	Katie	1719	387.00	False	Campinas
7	10006	Fiona	1717	676.00	True	Grazd
8	10007	Gina	1705	98.00	False	Montréal
9	10008	Katie	1715	968.00	True	sawasse
10	10009	Ralph	1715	596.00	False	Grazd

### Enabling filters

You can show filters in exported worksheet by enabling filter for the exported range in the worksheet.

#### C#

```
var excelExport = new DataGridExcelExportingController();
var options = new DataGridExcelExportingOption();
var excelEngine = excelExport.ExportToExcel(dataGrid, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.Worksheets[0].AutoFilters.FilterRange =
workBook.Worksheets[0].UsedRange;
MemoryStream stream = new MemoryStream();
workBook.SaveAs(stream);
```

While using stacked headers, you can specify the range based on Stacked headers count.

#### C#

```
var excelExport = new DataGridExcelExportingController();
var options = new DataGridExcelExportingOption();
options.ExportStackedHeaders = true;
var excelEngine = excelExport.ExportToExcel(dataGrid, options);
var workBook = excelEngine.Excel.Workbooks[0];
var range = "A" + (dataGrid.StackedHeaderRows.Count + 1).ToString() + ":" +
workBook.Worksheets[0].UsedRange.End.AddressLocal;
excelEngine.Excel.Workbooks[0].Worksheets[0].AutoFilters.FilterRange =
workBook.Worksheets[0].Range[range];
workBook.SaveAs("Sample.xlsx");
MemoryStream stream = new MemoryStream();
workBook.SaveAs(stream);
```

### Customize the range of cells

You can customize the range of cells after exporting to Excel by directly manipulating worksheet.

#### C#

```
var excelExport = new DataGridExcelExportingController();
var options = new DataGridExcelExportingOption();
options.ExportStackedHeaders = true;
var excelEngine = excelExport.ExportToExcel(dataGrid, 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;
MemoryStream stream = new MemoryStream();
workBook.SaveAs(stream);
```

### Performance improvement

Using [ExcelExportingOptions.CellsExportingEventHandler](#) to customize settings of each cell will consume more memory and time. So, avoid using `CellsExportingEventHandler` when customizing large number of cells and instead of you can do the required customizations in the exported sheet.

### *Formatting column without using CellsExportingEventHandler*

You can perform cell level customization such as row-level styling and formatting a particular column in the exported worksheet.

In the following code snippet, NumberFormat for GridNumericColumn column is changed in the exported sheet after exporting without using `CellsExportingEventHandler`.

Reference: <http://help.syncfusion.com/file-formats/xlsio/working-with-cell-or-range-formatting>

#### **C#**

```
var excelExport = new DataGridExcelExportingController();
var options = new DataGridExcelExportingOption();
options.ExportMode = ExportMode.Value;
var excelEngine = excelExport.ExportToExcel(dataGrid, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.ActiveSheet.Columns[4].NumberFormat = "0.0";
```

### *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 excelExport = new DataGridExcelExportingController();
var options = new DataGridExcelExportingOption();
options.ExportMode = ExportMode.Value;
var excelEngine = excelExport.ExportToExcel(dataGrid, options);
var workBook = excelEngine.Excel.Workbooks[0];
IConditionalFormats condition =
workBook.ActiveSheet.Range[2, 1, this.dataGrid.View.Records.Count+1, this.dataGrid.Columns.Count].ConditionalFormats;
IConditionalFormat condition1 = condition.AddCondition();
condition1.FormatType = ExcelCfType.Formula;
condition1.FirstFormula = "MOD(ROW(), 2)=0";
condition1.BackColorRGB = System.Drawing.Color.Pink;
IConditionalFormat condition2 = condition.AddCondition();
condition2.FormatType = ExcelCfType.Formula;
condition2.FirstFormula = "MOD(ROW(), 2)=1";
condition2.BackColorRGB = System.Drawing.Color.LightGray;
```

### *Exporting the selected rows of SfDataGrid*

SfDataGrid allows you to export only the currently selected rows in the grid to the worksheet using the [DataGridExcelExportingController.ExportToExcel](#) method by passing the instance of the SfDataGrid and [SfDataGrid.SelectedItems](#) collection as an argument.

Refer the below code to export the selected rows alone to Excel.

#### **C#**

```
private void ExportToExcel(object sender, EventArgs e)
{
    DataGridExcelExportingController excelExport = new
    DataGridExcelExportingController();
```

```
ObservableCollection<object> selectedItems = dataGrid.SelectedItems;
var excelEngine = excelExport.ExportToExcel(this.dataGrid, selectedItems);
var workbook = excelEngine.Excel.Workbooks[0];
MemoryStream stream = new MemoryStream();
workbook.SaveAs(stream);
workbook.Close();
excelEngine.Dispose();
if (Device.RuntimePlatform == Device.UWP)
    Xamarin.Forms.DependencyService.Get<ISaveWindowsPhone>().Save("DataGrid.xlsx",
        "application/msexcel", stream);
else
    Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.xlsx",
        "application/msexcel", stream);
}
```

Order ID	Customer	Ship Country	Ship City	Customer ID
1001	ANATR	Mexico	Mexico D.F.	Ana Hardy
1002	ANTON	Mexico	Mexico D.F.	Ant Fuller
1003	AROUT	UK	London	Thomas Hardy
1004	BERGS	Sweden	Berlin	Tim Adams
1005	BLAUS	Germany	Mannheim	Hanna Mos
1006	BLONP	France	Strasbourg	Andrew Fuller
1007	BOLID	Spain	Madrid	Martin King
1008	BONAP	France	Marseille	Lenny Lin

[illegible]

Export to PDF

The SfDataGrid supports exporting the data to PDF with several customization options like custom appearance, excluding specific columns, excluding headers, setting custom row height, setting custom column width, and so on.

The following assemblies should be added for exporting to PDF file.

Project	Required assemblies
PCL	<p>pcl\Syncfusion.SfGridConverter.XForms.dll</p> <p>pcl\Syncfusion.Compression.Portable.dll</p> <p>pcl\Syncfusion.Pdf.Portable.dll</p> <p>pcl\Syncfusion.XlsIO.Portable.dll</p>

If you are using nuget package in the package, the following NuGet package should be installed to export the SfDataGrid to PDF file.

Project	Required package
Xamarin.Forms	Syncfusion.Xamarin.DataGridExport

You can export the SfDataGrid to PDF by using the following extension methods present in the Syncfusion.SfDataGrid.XForms.Exporting namespace.

- [ExportToPdf](#)
- [ExportToPdfGrid](#)

The following code illustrates how to create and display a SfDataGrid in view.

#### XML

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition />
</Grid.RowDefinitions>
<StackLayout Grid.Row="0" Orientation="Vertical">
<Button x:Name="Export" Text="ExportPDF" Clicked ="PDFExport_Clicked"
HeightRequest="50" />
</StackLayout>
<sfgrid:SfDataGrid x:Name="dataGrid"
AllowGroupExpandCollapse="True"
AllowSorting="True"
Grid.Row="1"
SelectionMode="Multiple"
ColumnSizer="None"
ItemsSource="{Binding OrdersInfo}"
AutoGenerateColumns="False" >
<sfgrid:SfDataGrid.Columns>
<sfgrid:GridTextColumn HeaderText="Order ID" MappingName="OrderID"/>
<sfgrid:GridTextColumn HeaderText="FirstName" MappingName="FirstName"/>
<sfgrid:GridTextColumn HeaderText="Employee ID" MappingName="EmployeeID"/>
<sfgrid:GridNumericColumn HeaderText="Freight" MappingName="Freight"/>
<sfgrid:GridTextColumn HeaderText="IsClosed" MappingName="IsClosed"/>
<sfgrid:GridTextColumn HeaderText="ShipCity" MappingName="ShipCity"/>
<sfgrid:GridDateTimeColumn HeaderText="ShippingDate"
MappingName="ShippingDate" />
</sfgrid:SfDataGrid.Columns>
</sfgrid:SfDataGrid>
</Grid>
```

#### ExportToPdf

You can export the data to PDF by using the [DataGridPdfExportingController.ExportToPdf](#) method by passing the SfDataGrid as an argument.

#### C#

```
private void PDFExport_Clicked(object sender, EventArgs e)
```

```
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    var exportToPdf = pdfExport.ExportToPdf(this.dataGrid, new
    DataGridPdfExportOption()
    {
        FitAllColumnsInOnePage = true,
    });
    exportToPdf.Save(stream);
    exportToPdf.Close(true);
    if (Device.OS == TargetPlatform.WinPhone || Device.OS ==
    TargetPlatform.Windows)
        Xamarin.Forms.DependencyService.Get<ISaveWindowsPhone>().Save("DataGrid.pdf",
        "application/pdf", stream);
    else
        Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.pdf",
        "application/pdf", stream);
}
```

### ExportToPdfGrid

You can also export the data to PDF by using the [DataGridPdfExportingController.ExportToPdfGrid](#) method by passing the SfDataGrid as an argument.

### C#

```
private void PDFExport_Clicked(object sender, EventArgs e)
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    var pdfDoc = new PdfDocument();
    PdfPage page = pdfDoc.Pages.Add();
    var exportToPdfGrid = pdfExport.ExportToPdfGrid(this.dataGrid,
    this.dataGrid.View, new DataGridPdfExportOption()
    {
        FitAllColumnsInOnePage = true,
    }, pdfDoc);
    exportToPdfGrid.Draw(page, new PointF(10, 10));
    pdfDoc.Save(stream);
    pdfDoc.Close(true);
    if (Device.OS == TargetPlatform.WinPhone || Device.OS ==
    TargetPlatform.Windows)
        Xamarin.Forms.DependencyService.Get<ISaveWindowsPhone>().Save("DataGrid.pdf",
        "application/pdf", stream);
    else
        Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.pdf",
        "application/pdf", stream);
}
```



Carrier  12:21 PM 

Done DataGrid.pdf



OrderID	CustomerID	Frieght	Country
1001	Maria Anders	ALFKI	Germany
1002	Ana Trujilo	ANATR	Mexico
1003	Ant Fuller	ANTON	Mexico
1004	Thomas Hardy	AROUT	UK
1005	Tim Adams	BERGS	Sweden
1006	Hanna Moos	BLAUS	Germany
1007	Andrew Fuller	BLONP	France
1008	Martin King	BOLID	Spain
1009	Lenny Lin	BONAP	France
1010	John Carter	BOTTM	Canada
1011	Lauro King	AROUT	UK
1012	Anne Wilson	BLAUS	Germany
1013	Maria Anders	ALFKI	Germany
1014	Ana Trujilo	ANATR	Mexico
1015	Ant Fuller	ANTON	Mexico
1016	Thomas Hardy	AROUT	UK
1017	Tim Adams	BERGS	Sweden
1018	Hanna Moos	BLAUS	Germany
1019	Andrew Fuller	BLONP	France
1020	Martin King	BOLID	Spain
1021	Lauro King	AROUT	UK
1022	Anne Wilson	BLAUS	Germany



---

**Note:** SfDataGrid cannot export the GridTemplateColumn to PDF or Excel, since we cannot get the loaded views and draw them with the particular range, values etc from GridTemplateColumn.

---

### Exporting options

#### *Exclude columns when exporting*

By default, all the columns (including hidden columns) in the SfDataGrid will be exported to PDF. To exclude some particular columns when exporting to PDF, add those columns to [DataGridPdfExportOption.ExcludeColumns](#) list.

### **C#**

```
DataGridPdfExportingController pdfExport = new
DataGridPdfExportingController ();
DataGridPdfExportOption exportOption = new DataGridPdfExportOption ();
exportOption.FitAllColumnsInOnePage = true;
var list = new List<string>();
list.Add("OrderID");
list.Add("LastName");
exportOption.ExcludedColumns = list;
var doc = pdfExport.ExportToPdf (this.dataGrid, exportOption);
```



### Getting PDF document

The [DataGridPdfExportOption.PdfDocument](#) allows exporting the SfDataGrid to an existing or a new PDF document.

#### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();
PdfDocument pdfDocument = new PdfDocument();
pdfDocument.Pages.Add();
pdfDocument.Pages.Add();
pdfDocument.Pages.Add();
option.StartPageIndex = 1;
option.PdfDocument = pdfDocument;
```

### Getting GridColumns for customization

Using the property [GridColumns](#) you can get or set the [System.Collections.IEnumerable](#) columns collection which contains all the columns that are to be exported. The columns in the ExcludedColumns List will not be a member of the GridColumns collection.








### Column header on each page

You can show or hide the column headers on each page of the exported PDF document by using the [DataGridPdfExportOption.RepeatHeaders](#) property. The default value is true.





#### C#

```
private void PDFExport_Clicked(object sender, EventArgs e)
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    DataGridPdfExportOption option = new DataGridPdfExportOption();
    option.RepeatHeaders = true;
    var exportToPdf = pdfExport.ExportToPdf(this.dataGrid, option);
    exportToPdf.Save(stream);
    exportToPdf.Close(true);
    Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.pdf",
    "application/pdf", stream);
}
```



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10001	Gina	1704	101.00	False	Grazd	9/7/2008
10002	Ralph	1719	122.00		Buenos Aires	6/3/2009
10003	Irene	1716	370.00	False	Rio de Janeiro	4/13/2010
10004	Brenda	1717	305.00		Bruxelles	10/21/2006
10005	Katie	1709	362.00	False	Grazd	3/18/2002
10006	Torrey	1718	585.00		Grazd	5/12/2004
10007	Danielle	1701	249.00	False	Campinas	9/10/2000
10008	Frank	1718	962.00		Grazd	6/26/2011
10009	Kyle	1708	665.00	False	Bruxelles	4/3/2005
10010	Gina	1714	319.00		Tsawassen	3/18/2010
10011	Irene	1703	361.00	False	Buenos Aires	9/1/2010
10012	Gina	1710	693.00		Rio de Janeiro	3/7/2012
10013	Ralph	1710	800.00	False	Rio de Janeiro	3/9/2007
10014	Katie	1716	917.00		Buenos Aires	7/27/2013



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10015	Gina	1718	492.00	False	Bruxelles	6/19/2012
10016	Michael	1706	863.00		Tsawassen	5/18/2001
10017	Fiona	1703	298.00	False	Montréal	8/8/2008
10018	Brenda	1719	22.00		Campinas	4/17/2013
10019	Katie	1709	151.00	False	Tsawassen	8/20/2011
10020	Frank	1711	898.00		Grazd	3/19/2010
10021	Kyle	1711	186.00	False	Campinas	3/26/2007
10022	Torrey	1708	391.00		Bruxelles	5/3/2007

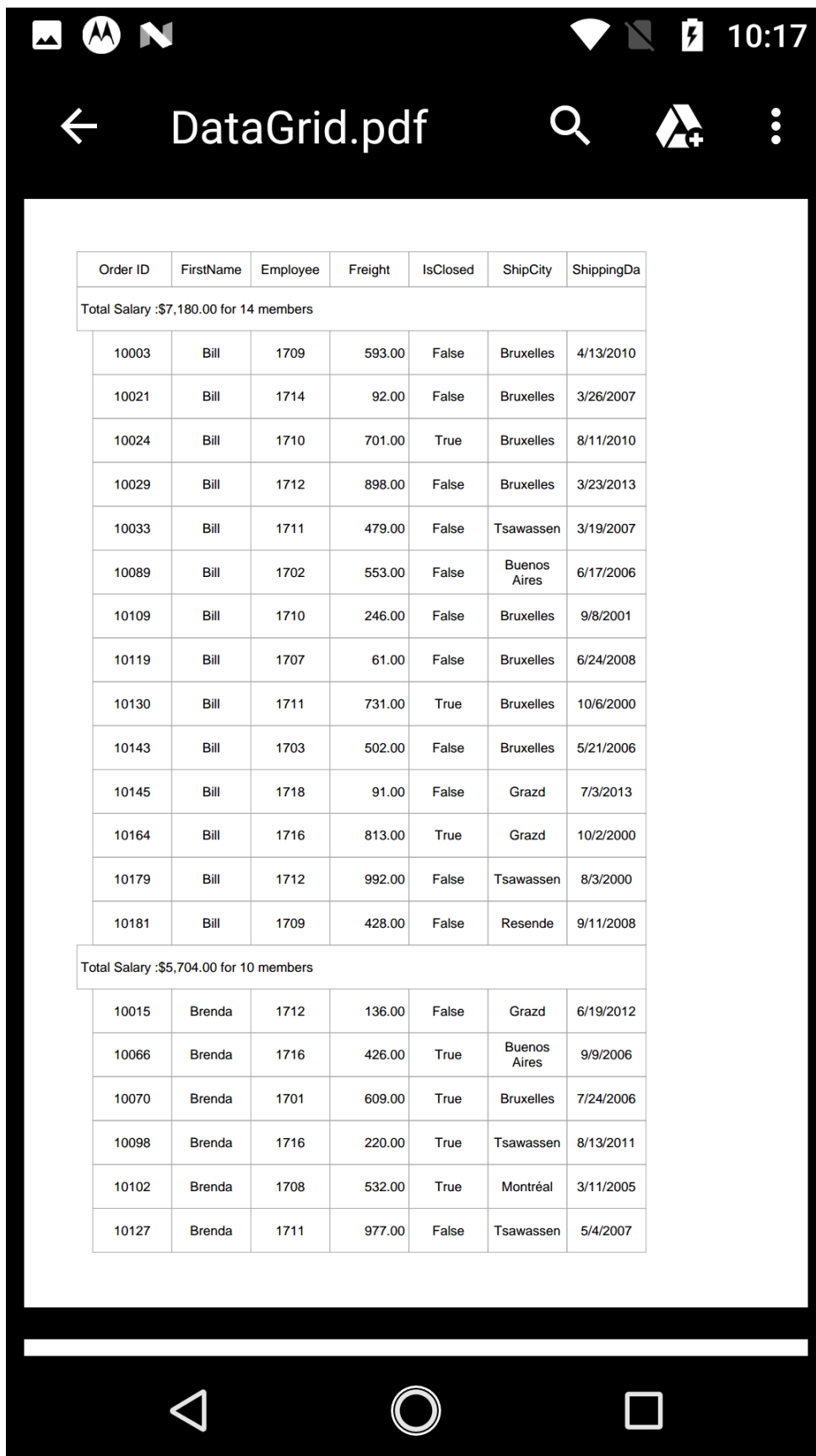
*Customize header, groups and table summary when exporting*

Exclude groups while exporting

By default, all the groups in the data grid will be exported to PDF document. To export the data grid without groups, set the [DataGridPdfExportOption.ExportGroups](#) property to `false`.

#### C#

```
private void PDFExport_Clicked(object sender, EventArgs e)
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    DataGridPdfExportOption option = new DataGridPdfExportOption();
    option.ExportGroups = true;
    var exportToPdf = pdfExport.ExportToPdf(this.dataGrid, option);
    exportToPdf.Save(stream);
    exportToPdf.Close(true);
    Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.pdf",
    "application/pdf", stream);
}
```



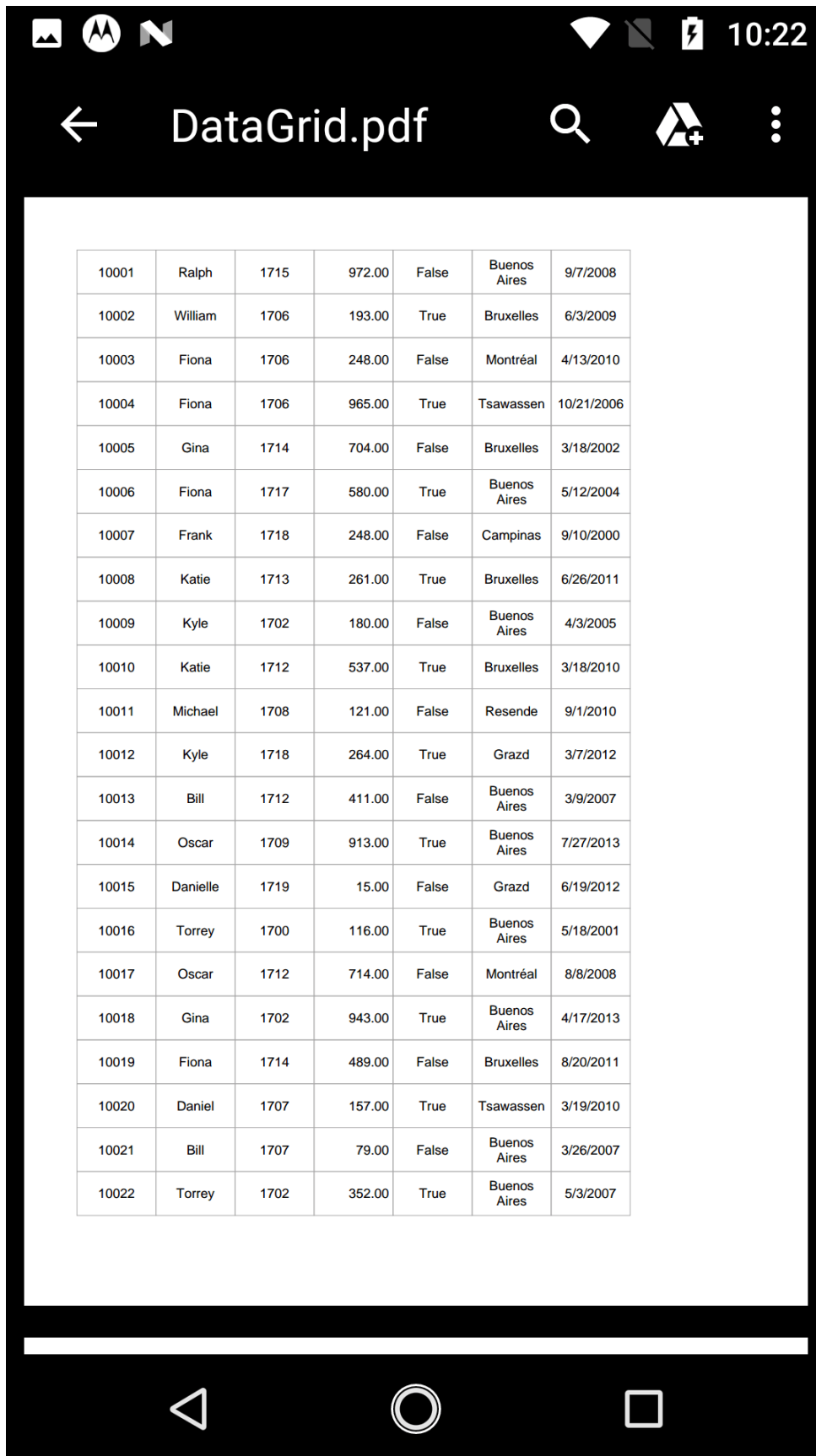
### Exclude Column header while exporting

By default, the column headers will be exported to PDF document. To export the SfDataGrid without the column headers, set the [DataGridPdfExportOption.ExportHeader](#) property to `false`.

### C#

```
private void PDFExport_Clicked(object sender, EventArgs e)
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    DataGridPdfExportOption option = new DataGridPdfExportOption();
    option.ExportHeader = false;
    var exportToPdf = pdfExport.ExportToPdf(this.dataGrid, option);
    exportToPdf.Save(stream);
    exportToPdf.Close(true);
    Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.pdf",
    "application/pdf", stream);
}
```





### Exclude table summaries while exporting

By default, table summaries in the data grid will be exported to PDF. To export the SfDataGrid without table summaries, set the [DataGridPdfExportOption.ExportTableSummary](#) property to `false`.

### C#

```
private void PDFExport_Clicked(object sender, EventArgs e)
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    DataGridPdfExportOption option = new DataGridPdfExportOption();
    option.ExportTableSummary = false;
    var exportToPdf = pdfExport.ExportToPdf(this.dataGrid, option);
    exportToPdf.Save(stream);
    exportToPdf.Close(true);
    Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.pdf",
    "application/pdf", stream);
}
```



Order ID    FirstName    Employee    Freight    IsClosed    ShipCity    ShippingDa

Total Freight :\$99,744.00

10001	Ralph	1711	851.00	False	Grazd	9/7/2008
10002	Oscar	1709	721.00	True	Rio de Janeiro	6/3/2009
10003	Torrey	1702	860.00	False	Montréal	4/13/2010
10004	Bill	1711	817.00	True	Grazd	10/21/2006
10005	Daniel	1710	62.00	False	Buenos Aires	3/18/2002
10006	Kyle	1717	419.00	True	Buenos Aires	5/12/2004
10007	Ralph	1716	679.00	False	Grazd	9/10/2000
10008	Bill	1719	707.00	True	Campinas	6/26/2011
10009	Frank	1705	542.00	False	Bruxelles	4/3/2005
10010	Irene	1717	842.00	True	Grazd	3/18/2010
10011	Frank	1711	632.00	False	Resende	9/1/2010
10012	Danielle	1706	174.00	True	Bruxelles	3/7/2012
10013	Danielle	1706	19.00	False	Rio de Janeiro	3/9/2007
10014	Irene	1702	533.00	True	Bruxelles	7/27/2013
10015	Brenda	1707	23.00	False	Bruxelles	6/19/2012
10016	Daniel	1704	270.00	True	Buenos Aires	5/18/2001
10017	Irene	1702	857.00	False	Grazd	8/8/2008
10018	Ralph	1704	707.00	True	Montréal	4/17/2013
10019	Fiona	1703	364.00	False	Buenos Aires	8/20/2011
10020	William	1713	619.00	True	Campinas	3/19/2010
10021	Frank	1710	118.00	False	Grazd	3/26/2007

### *Export all columns in one page*

Gets or sets a value indicating whether all the columns should be fitted on a page or not. To export all the columns in one page, set the [DataGridPdfExportOption.FitAllColumnsInOnePage](#) property to `true`.

### *Exporting the grid from a specified page and point*

The SfDataGrid allows exporting the data to a particular starting position on a particular PDF page using the following options:

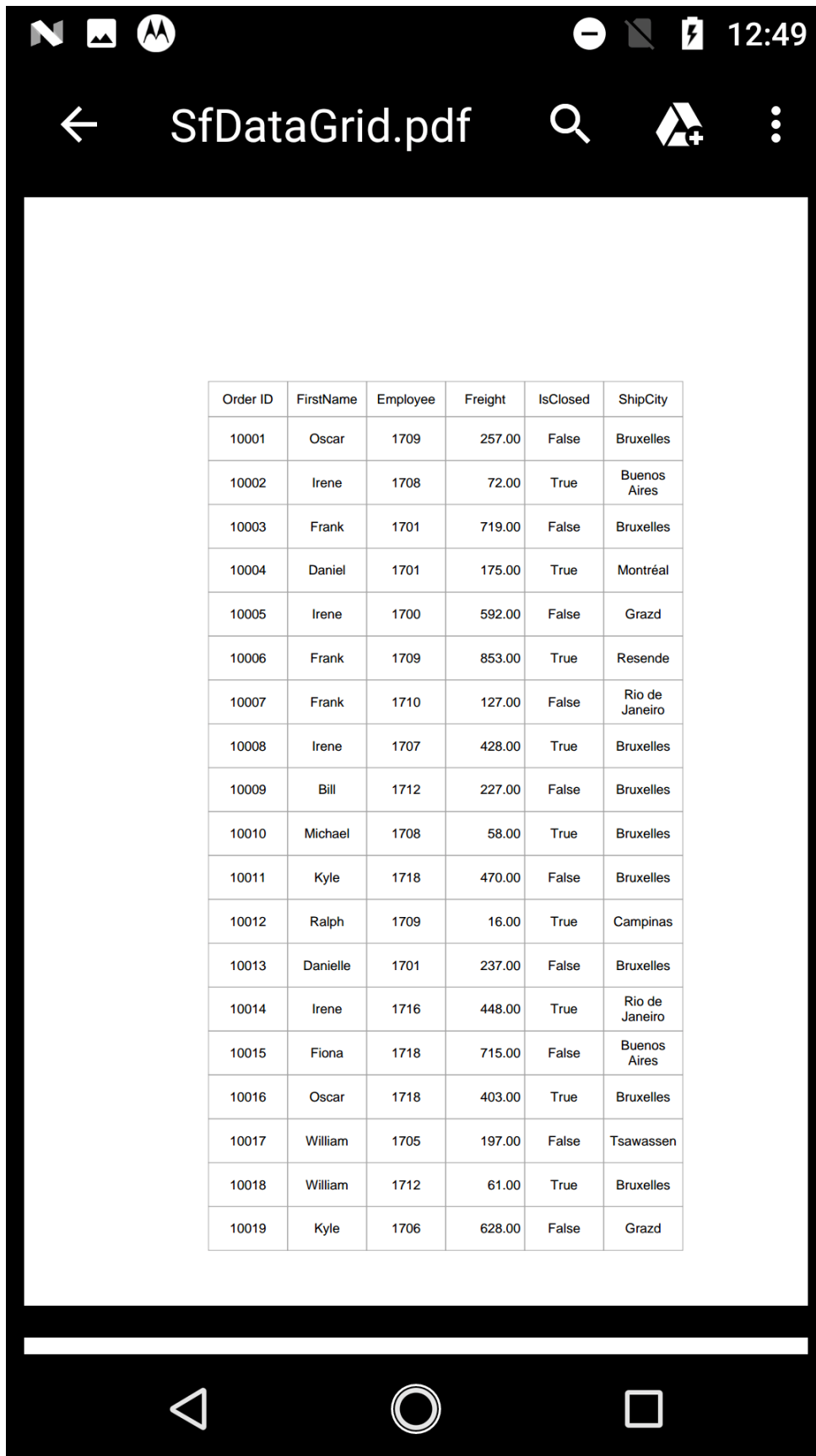
- StartPageIndex
- StartPoint

### *StartPageIndex*

The SfDataGrid allows exporting the data to a particular page in the PDF document by using the [DataGridPdfExportOption.StartPageIndex](#) property.

### **C#**

```
private void PDFExport_Clicked(object sender, EventArgs e)
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    PdfDocument pdfDocument = new PdfDocument();
    pdfDocument.Pages.Add();
    pdfDocument.Pages.Add();
    pdfDocument.Pages.Add();
    DataGridPdfExportOption option = new DataGridPdfExportOption();
    option.PdfDocument = pdfDocument;
    option.StartPageIndex = 1;
    var exportToPdf = pdfExport.ExportToPdf(this.dataGrid, option);
    exportToPdf.Save(stream);
    exportToPdf.Close(true);
    Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.pdf",
    "application/pdf", stream);
}
```



### StartPoint

The SfDataGrid allows exporting the data to a particular x,y starting point in the PDF page by using the [DataGridPdfExportOption.StartPoint](#) property.

### C#

```
private void PDFExport_Clicked(object sender, EventArgs e)
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    DataGridPdfExportOption option = new DataGridPdfExportOption();
    option.StartPoint = new PointF(0, 500);
    var exportToPdf = pdfExport.ExportToPdf(this.dataGrid, option);
    exportToPdf.Save(stream);
    exportToPdf.Close(true);
    Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid1.pdf",
    "application/pdf", stream);
}
```

Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10001	Brenda	1704	880.00	False	Bruxelles	9/7/2008
10002	Kyle	1700	614.00	True	Bruxelles	6/3/2009
10003	Gina	1713	362.00	False	Campinas	4/13/2010
10004	Danielle	1716	64.00	True	Buenos Aires	10/21/2006
10005	Torrey	1717	438.00	False	Grazd	3/18/2002
10006	Irene	1719	262.00	True	Rio de Janeiro	5/12/2004
10007	Ralph	1717	664.00	False	Buenos Aires	9/10/2000
10008	Irene	1718	456.00	True	Bruxelles	6/26/2011
10009	Gina	1703	461.00	False	Buenos Aires	4/3/2005
10010	Daniel	1713	815.00	True	Bruxelles	3/18/2010
10011	Torrey	1707	376.00	False	Buenos Aires	9/1/2010
10012	Katie	1710	884.00	True	Bruxelles	3/7/2012

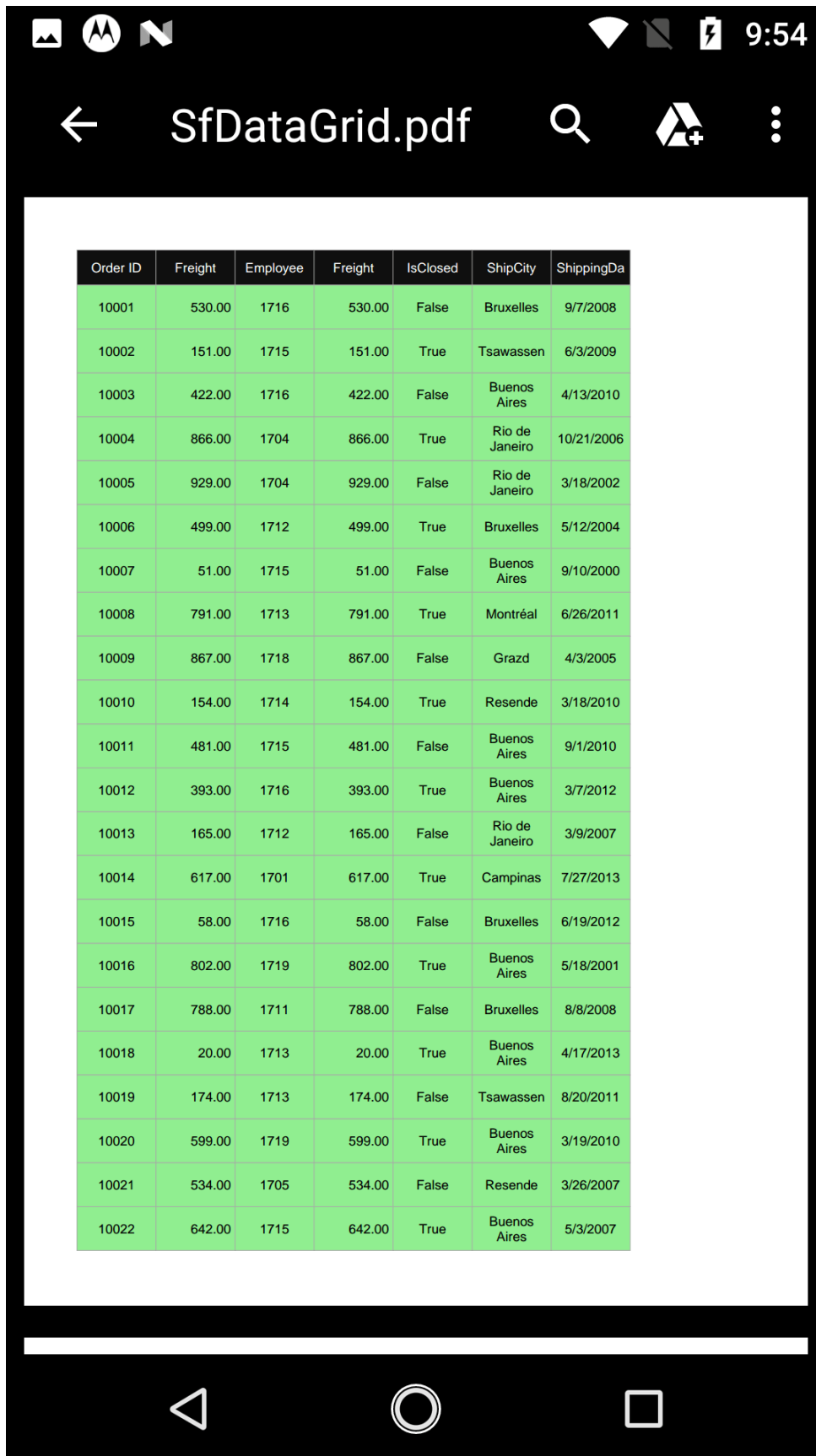
### *Applying styles while exporting*

The SfDataGrid allows exporting the data with the applied GridStyle by setting the [DataGridPdfExportOption.ApplyGridStyle](#) property to `true`. By default, data will be exported without the GridStyle.

### **C#**

```
private void PDFExport_Clicked(object sender, EventArgs e)
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    DataGridPdfExportOption option = new DataGridPdfExportOption();
    option.ApplyGridStyle = true;
    var exportToPdf = pdfExport.ExportToPdf(this.dataGrid, option);
    exportToPdf.Save(stream);
    exportToPdf.Close(true);
    Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid1.pdf",
    "application/pdf", stream);
}
```





You can also customize the following styles while exporting to PDF:

- BottomTableSummaryStyle
- GroupCaptionStyle
- HeaderStyle
- RecordStyle
- TopTableSummaryStyle

#### BottomTableSummaryStyle

The SfDataGrid supports exporting the bottom table summary with custom style by using the [DataGridPdfExportOption.BottomTableSummaryStyle](#) property.

#### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();
option.BottomTableSummaryStyle = new PdfGridCellStyle()
{
    BackgroundBrush = PdfBrushes.Gray,
    Borders = new PdfBorders() { Bottom = PdfPens.Aqua, Left =
    PdfPens.AliceBlue, Right = PdfPens.Red, Top = PdfPens.RoyalBlue },
    CellPadding = new PdfPaddings(2, 2, 2, 2),
    TextBrush = PdfBrushes.Yellow,
    TextPen = PdfPens.Green,
    StringFormat = new PdfStringFormat() { Alignment = PdfTextAlignment.Right,
    CharacterSpacing = 3f, WordSpacing = 10f }
};
```

Irene	1712	871.00	False	Buenos Aires	3/9/2001
Kyle	1705	971.00	True	Bruxelles	9/24/2006
Gina	1705	255.00	False	Grazd	5/18/2012
Frank	1705	891.00	True	Tsawassen	9/12/2013
William	1706	550.00	False	Montréal	4/12/2004
Oscar	1711	143.00	True	Buenos Aires	9/15/2001
Michael	1700	18.00	False	Buenos Aires	8/10/2013
Fiona	1715	505.00	True	Montréal	4/11/2008
Daniel	1708	878.00	False	Campinas	10/24/2006



FirstNam	Employee ID	Freight	IsClosed	ShipCity	Shipping Date
Irene	1701	984.00	True	Campinas	5/9/2000
Katie	1718	256.00	False	Buenos Aires	4/9/2013
Bill	1709	505.00	True	Resende	6/19/2005
Daniel	1704	764.00	False	Grazd	9/23/2003
Oscar	1716	201.00	True	Grazd	7/7/2005
Total Freight :\$102,061.00 for 200 OrderCount					

### GroupCaptionStyle

The SfDataGrid supports exporting the group caption summaries with custom style by using the [DataGridPdfExportOption.GroupCaptionStyle](#) property.

#### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();
option.GroupCaptionStyle = new PdfGridCellStyle
{
    BackgroundBrush = PdfBrushes.White,
    Borders = new PdfBorders() { Bottom = PdfPens.Aqua, Left =
    PdfPens.AliceBlue, Right = PdfPens.Red, Top = PdfPens.RoyalBlue },
    CellPadding = new PdfPaddings(3, 3, 3, 3),
    TextBrush = PdfBrushes.Red,
    TextPen = PdfPens.Green,
    StringFormat = new PdfStringFormat() { Alignment = PdfTextAlignment.Right,
    CharacterSpacing = 3f, WordSpacing = 10f }
};
```

FirstNam e	Employee ID	Freight	IsClosed	ShipCity	Shipping Date
Total Freight :\$102,061.00					
Kyle	1713	352.00	False	Buenos Aires	9/7/2008
Oscar	1702	452.00	True	Rio de Janeiro	6/3/2009
Torrey	1710	311.00	False	Resende	4/13/2010
Ralph	1703	336.00	True	Rio de Janeiro	10/21/2006
Brenda	1716	650.00	False	Buenos Aires	3/18/2002
Michael	1709	729.00	True	Grazd	5/12/2004
Kyle	1709	404.00	False	Tsawassen	9/10/2000
Bill	1703	568.00	True	Buenos Aires	6/26/2011
Irene	1708	743.00	False	Rio de Janeiro	4/3/2005
Kyle	1715	595.00	True	Grazd	3/18/2010
Bill	1709	974.00	False	Tsawassen	9/1/2010
Torrey	1704	200.00	True	Bruxelles	3/7/2012
Bill	1700	74.00	False	Bruxelles	3/9/2007

### HeaderStyle

The SfDataGrid allows exporting the column headers with custom style by using the [DataGridPdfExportOption.HeaderStyle](#) property.

### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();
option.HeaderStyle = new PdfGridCellStyle()
{
    BackgroundBrush = PdfBrushes.Yellow,
    Borders = new PdfBorders() { Bottom = PdfPens.Aqua, Left =
    PdfPens.AliceBlue, Right = PdfPens.Red, Top = PdfPens.RoyalBlue },
    CellPadding = new PdfPaddings(2, 2, 2, 2),
    TextBrush = PdfBrushes.Red,
    TextPen = PdfPens.Green,
    StringFormat = new PdfStringFormat() { Alignment = PdfTextAlignment.Right,
    CharacterSpacing = 3f, WordSpacing = 10f }
};
```

10:42

←

SfDataGrid.pdf

🔍

+

⋮

◀

○

◻

### RecordStyle

The SfDataGrid allows exporting the records with custom style by using the [DataGridPdfExportOption.RecordStyle](#) property.

### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();
option.RecordStyle = new PdfGridCellStyle()
{
    BackgroundBrush = PdfBrushes.Red,
    Borders = new PdfBorders() { Bottom = PdfPens.Aqua, Left =
    PdfPens.AliceBlue, Right = PdfPens.Red, Top = PdfPens.RoyalBlue },
    CellPadding = new PdfPaddings(2, 2, 2, 2),
    TextBrush = PdfBrushes.White,
    TextPen = PdfPens.Green,
    StringFormat = new PdfStringFormat() { Alignment = PdfTextAlignment.Right,
    CharacterSpacing = 3f, WordSpacing = 10f }
};
```



10:42

←

SfDataGrid.pdf

🔍

+

⋮

Order ID	FirstName	Employee	Freight	IsClosed	ShipCity
Total Freight :\$102,394.00					
Total Salary :\$12,146.00 for 20 members					
10008	Bill	1712	989.00	True	Bueno Alre
10026	Bill	1702	594.00	True	Graz
10063	Bill	1706	558.00	False	Montréal
10078	Bill	1706	423.00	True	Bueno Alre
10098	Bill	1718	681.00	True	Campin
10109	Bill	1713	487.00	False	Bueno Alre
10114	Bill	1713	774.00	True	Graz
10115	Bill	1715	155.00	False	Montréal
10117	Bill	1710	622.00	False	Graz
10121	Bill	1716	907.00	False	Bueno Alre
10122	Bill	1704	155.00	True	Bueno Alre
10130	Bill	1706	972.00	True	Rio d Janelr

### TopTableSummaryStyle

The SfDataGrid supports exporting the top table summary with custom style by using the [DataGridPdfExportOption.TopTableSummaryStyle](#) property.

#### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();
option.TopTableSummaryStyle = new PdfGridCellStyle()
{
    BackgroundBrush = PdfBrushes.Gray,
    Borders = new PdfBorders() { Bottom = PdfPens.Aqua, Left =
    PdfPens.AliceBlue, Right = PdfPens.Red, Top = PdfPens.RoyalBlue },
    CellPadding = new PdfPaddings(2, 2, 2, 2),
    TextBrush = PdfBrushes.Yellow,
    TextPen = PdfPens.Green,
    StringFormat = new PdfStringFormat() { Alignment = PdfTextAlignment.Right,
    CharacterSpacing = 3f, WordSpacing = 10f }
};
```

FirstNam e	Employee ID	Freight	IsClosed	ShipCity	Shipping Date
Total Freight :\$102,061.00					
Kyle	1713	352.00	False	Buenos Aires	9/7/2008
Oscar	1702	452.00	True	Rio de Janeiro	6/3/2009
Torrey	1710	311.00	False	Resende	4/13/2010
Ralph	1703	336.00	True	Rio de Janeiro	10/21/2006
Brenda	1716	650.00	False	Buenos Aires	3/18/2002
Michael	1709	729.00	True	Grazd	5/12/2004
Kyle	1709	404.00	False	Tsawassen	9/10/2000
Bill	1703	568.00	True	Buenos Aires	6/26/2011
Irene	1708	743.00	False	Rio de Janeiro	4/3/2005
Kyle	1715	595.00	True	Grazd	3/18/2010
Bill	1709	974.00	False	Tsawassen	9/1/2010
Torrey	1704	200.00	True	Bruxelles	3/7/2012
Bill	1700	74.00	False	Bruxelles	3/9/2007

*GroupSummaryStyle*

SfDataGrid supports exporting the GroupSummary rows with custom style by using the [DataGridPdfExportOption.GroupSummaryStyle](#) property.

**C#**

```
DataGridPdfExportOption pdfOption = new DataGridPdfExportOption();
pdfOption.GroupSummaryStyle = new PdfGridCellStyle()
{
    BackgroundBrush = PdfBrushes.Green,
    TextBrush = PdfBrushes.Yellow,
    TextPen = PdfPens.White,
    StringFormat = new PdfStringFormat() { Alignment = PdfTextAlignment.Right,
    CharacterSpacing = 3f, WordSpacing = 10f }
};
```

**DataGrid.pdf**

OrderID	CustomerID	ShipCountry	Customer	ShipCity	IsChecked
OrderID : 1001 - 1 Items					
1001	Ana Hardy	Mexico	ANATR	Mexico D.F.	<input checked="" type="checkbox"/>
Total OrderID: 1001 for 1 members					
OrderID : 1002 - 1 Items					
1002	Ant Fuller	Mexico	ANTON	Mexico D.F.	<input checked="" type="checkbox"/>
Total OrderID: 1002 for 1 members					
OrderID : 1003 - 1 Items					
1003	Thomas Hardy	UK	AROUT	London	<input checked="" type="checkbox"/>
Total OrderID: 1003 for 1 members					
OrderID : 1004 - 1 Items					
1004	Tim Adams	Sweden	BERGS	Berlin	<input checked="" type="checkbox"/>
Total OrderID: 1004 for 1 members					
OrderID : 1005 - 1 Items					
1005	Hanna Moos	Germany	BLAUS	Mannheim	<input checked="" type="checkbox"/>
Total OrderID: 1005 for 1 members					
OrderID : 1006 - 1 Items					
1006	Andrew Fuller	France	BLONP	Strasbourg	<input checked="" type="checkbox"/>
Total OrderID: 1006 for 1 members					
OrderID : 1007 - 1 Items					
1007	Martin King	Spain	BOLID	Madrid	<input checked="" type="checkbox"/>
Total OrderID: 1007 for 1 members					
OrderID : 1008 - 1 Items					
1008	Lennv Lin	France	BONAP	Marseille	<input type="checkbox"/>

### *Customizing borders*

The SfDataGrid allows customizing the grid borders as follows using the [DataGridPdfExportOption.GridLineType]

([https://help.syncfusion.com/cr/cref\\_files/xamarin/sfgridconverter/Syncfusion.SfGridConverter.XForms~Syncfusion.SfDataGrid.XForms.Exporting.DataGridPdfExportOption~GridLineType.html](https://help.syncfusion.com/cr/cref_files/xamarin/sfgridconverter/Syncfusion.SfGridConverter.XForms~Syncfusion.SfDataGrid.XForms.Exporting.DataGridPdfExportOption~GridLineType.html)) property:

- Both
- Horizontal
- Vertical
- None

### *Both*

Set the [DataGridPdfExportOption.GridLineType](#) as `GridLineType.Both` to export the data grid with both horizontal and vertical borders.

### **C#**

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.GridLineType = GridLineType.Both;
```



### Horizontal

Set the [DataGridPdfExportOption.GridLineType](#) as `GridLineType.Horizontal` to export the data grid with horizontal border.

### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.GridLineType = GridLineType.Horizontal;
```



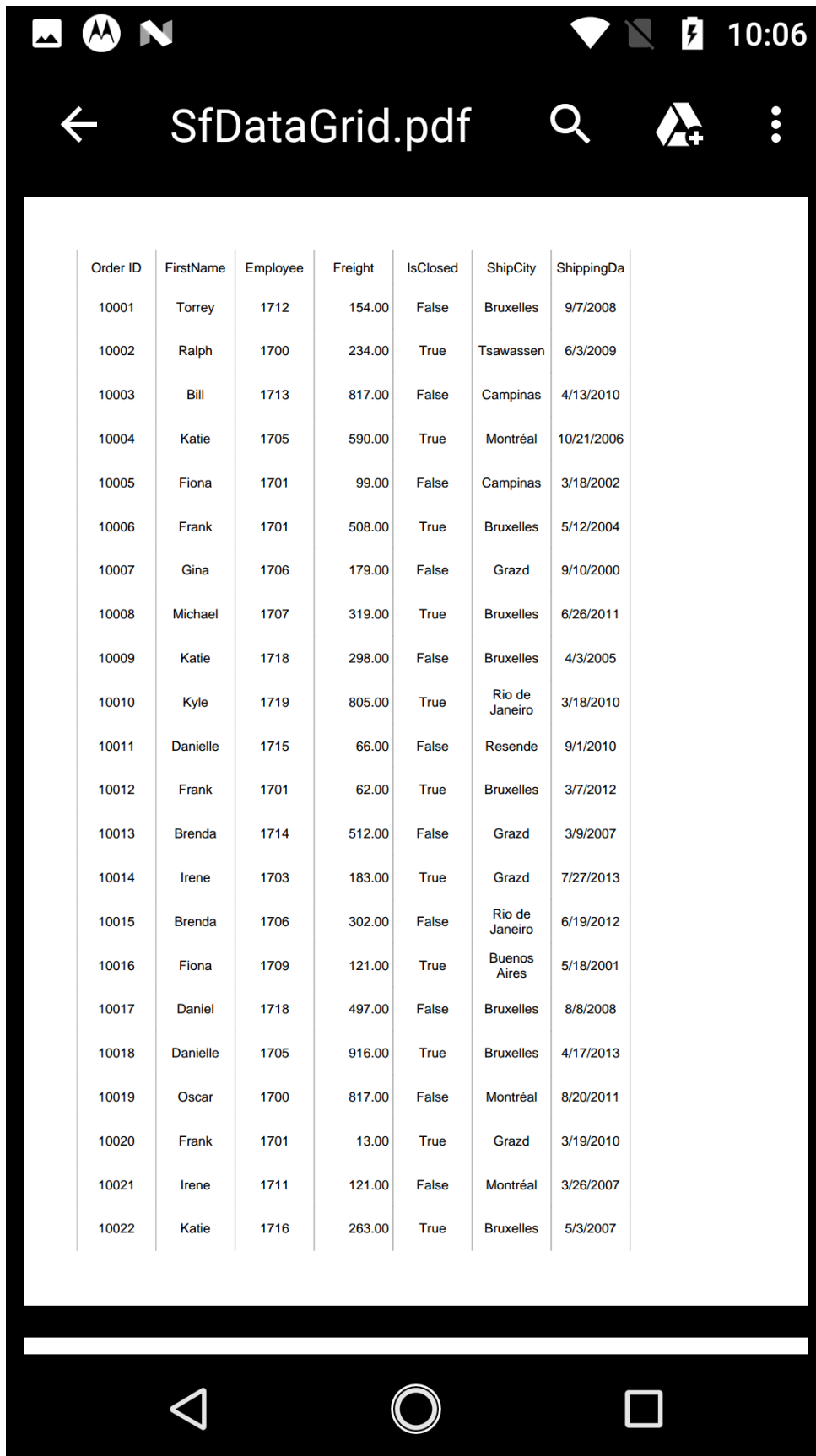


### Vertical

Set the [DataGridPdfExportOption.GridLineType](#) to `GridLineType.Vertical` to export the data grid with vertical border.

### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.GridLineType = GridLineType.Vertical;
```



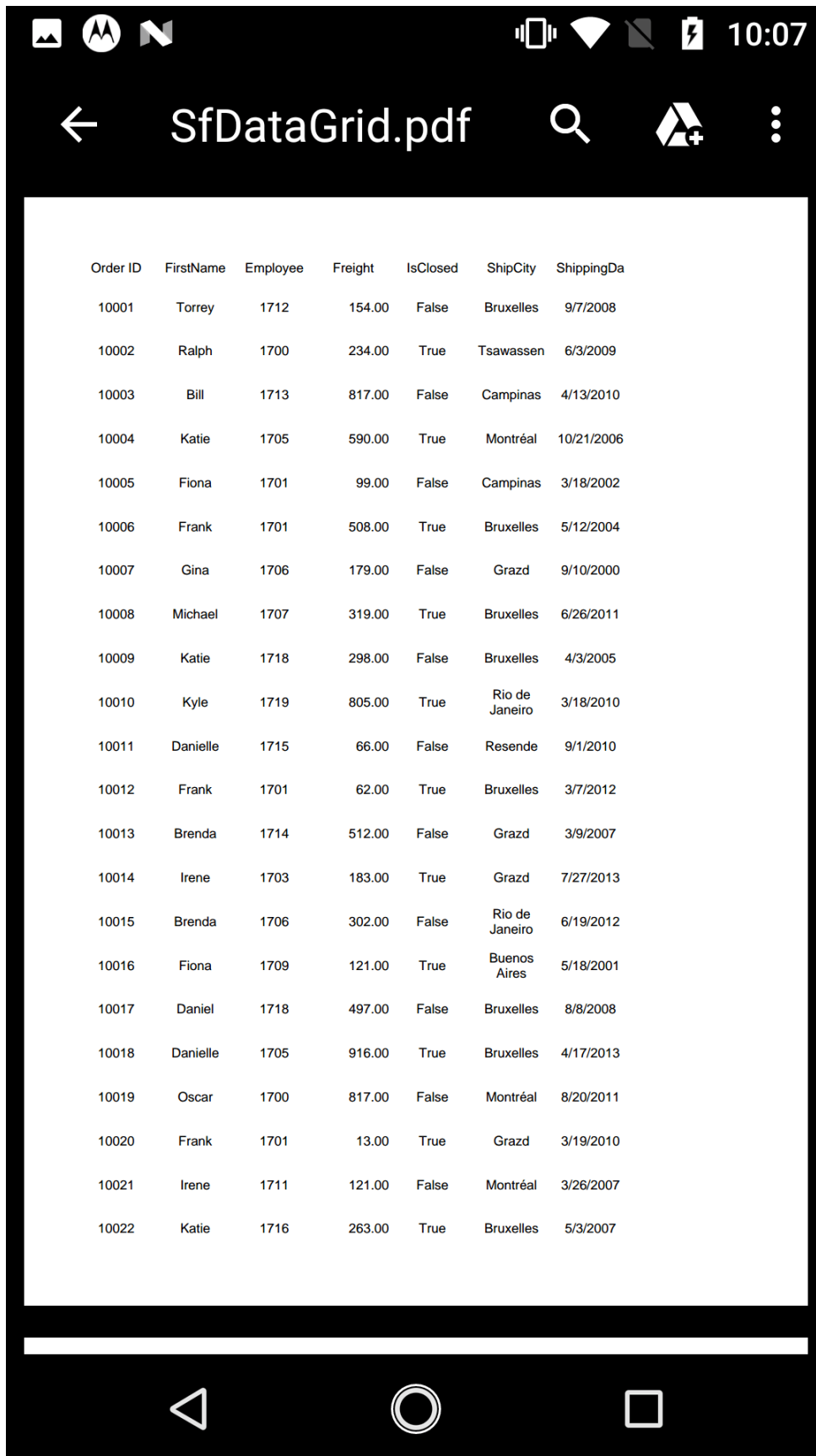
Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10001	Torrey	1712	154.00	False	Bruxelles	9/7/2008
10002	Ralph	1700	234.00	True	Tsawassen	6/3/2009
10003	Bill	1713	817.00	False	Campinas	4/13/2010
10004	Katie	1705	590.00	True	Montréal	10/21/2006
10005	Fiona	1701	99.00	False	Campinas	3/18/2002
10006	Frank	1701	508.00	True	Bruxelles	5/12/2004
10007	Gina	1706	179.00	False	Grazd	9/10/2000
10008	Michael	1707	319.00	True	Bruxelles	6/26/2011
10009	Katie	1718	298.00	False	Bruxelles	4/3/2005
10010	Kyle	1719	805.00	True	Rio de Janeiro	3/18/2010
10011	Danielle	1715	66.00	False	Resende	9/1/2010
10012	Frank	1701	62.00	True	Bruxelles	3/7/2012
10013	Brenda	1714	512.00	False	Grazd	3/9/2007
10014	Irene	1703	183.00	True	Grazd	7/27/2013
10015	Brenda	1706	302.00	False	Rio de Janeiro	6/19/2012
10016	Fiona	1709	121.00	True	Buenos Aires	5/18/2001
10017	Daniel	1718	497.00	False	Bruxelles	8/8/2008
10018	Danielle	1705	916.00	True	Bruxelles	4/17/2013
10019	Oscar	1700	817.00	False	Montréal	8/20/2011
10020	Frank	1701	13.00	True	Grazd	3/19/2010
10021	Irene	1711	121.00	False	Montréal	3/26/2007
10022	Katie	1716	263.00	True	Bruxelles	5/3/2007

None

Set the [DataGridPdfExportOption.GridLineType](#) to `GridLineType.None` to export the data grid without borders.

### **C#**

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.GridLineType = GridLineType.None;
```



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10001	Torrey	1712	154.00	False	Bruxelles	9/7/2008
10002	Ralph	1700	234.00	True	Tsawassen	6/3/2009
10003	Bill	1713	817.00	False	Campinas	4/13/2010
10004	Katie	1705	590.00	True	Montréal	10/21/2006
10005	Fiona	1701	99.00	False	Campinas	3/18/2002
10006	Frank	1701	508.00	True	Bruxelles	5/12/2004
10007	Gina	1706	179.00	False	Grazd	9/10/2000
10008	Michael	1707	319.00	True	Bruxelles	6/26/2011
10009	Katie	1718	298.00	False	Bruxelles	4/3/2005
10010	Kyle	1719	805.00	True	Rio de Janeiro	3/18/2010
10011	Danielle	1715	66.00	False	Resende	9/1/2010
10012	Frank	1701	62.00	True	Bruxelles	3/7/2012
10013	Brenda	1714	512.00	False	Grazd	3/9/2007
10014	Irene	1703	183.00	True	Grazd	7/27/2013
10015	Brenda	1706	302.00	False	Rio de Janeiro	6/19/2012
10016	Fiona	1709	121.00	True	Buenos Aires	5/18/2001
10017	Daniel	1718	497.00	False	Bruxelles	8/8/2008
10018	Danielle	1705	916.00	True	Bruxelles	4/17/2013
10019	Oscar	1700	817.00	False	Montréal	8/20/2011
10020	Frank	1701	13.00	True	Grazd	3/19/2010
10021	Irene	1711	121.00	False	Montréal	3/26/2007
10022	Katie	1716	263.00	True	Bruxelles	5/3/2007

### Export paging

When exporting to PDF in the SfDataGrid using SfDataPager, by default it will export only the current page. You can export all the pages by setting the [DataGridPdfExportOption.ExportAllPages](#) to `true`.

### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.ExportAllPages = true;
```



- `ExportAllPages = true;`



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10001	Danielle	1719	969.00	False	Rio de Janeiro	9/7/2008
10002	Brenda	1714	311.00	True	Grazd	6/3/2009
10003	Brenda	1716	414.00	False	Montréal	4/13/2010
10004	Gina	1703	844.00	True	Grazd	10/21/2006
10005	Oscar	1707	188.00	False	Tsawassen	3/18/2002
10006	Bill	1714	284.00	True	Tsawassen	5/12/2004
10007	Irene	1715	264.00	False	Grazd	9/10/2000
10008	Irene	1707	643.00	True	Campinas	6/26/2011
10009	Ralph	1712	674.00	False	Bruxelles	4/3/2005
10010	Gina	1718	596.00	True	Bruxelles	3/18/2010

### Setting header and footer

The SfDataGrid provides a way to display additional content at the top (header) or bottom (footer) of the page when exporting to PDF by handling the

`DataGridPdfExportingController.HeaderAndFooterExporting` event.

You can insert string in the header and footer in PdfHeaderFooterEventHandler. Setting PdfPageTemplateElement as PdfHeaderFooterEventArgs.PdfDocumentTemplate.Top loads the content at the top of the page. Setting the PdfPageTemplateElement as PdfHeaderFooterEventArgs.PdfDocumentTemplate.Bottom loads the content at the bottom of the page.

### C#

```
DataGridPdfExportingController pdfExport = new
DataGridPdfExportingController();
pdfExport.HeaderAndFooterExporting += PdfExport_HeaderAndFooterExporting;
private void PdfExport_HeaderAndFooterExporting(object sender,
PdfHeaderFooterEventArgs e)
{
PdfFont font = new PdfStandardFont(PdfFontFamily.TimesRoman, 20f,
PdfFontStyle.Bold);
var width = e.PdfPage.GetClientSize().Width;
PdfPageTemplateElement header = new PdfPageTemplateElement(width, 38);
header.Graphics.DrawString("Order Details", font, PdfPens.Black, 70, 3);
e.PdfDocumentTemplate.Top = header;
PdfPageTemplateElement footer = new PdfPageTemplateElement(width, 38);
footer.Graphics.DrawString("Order Details", font, PdfPens.Black, 70, 3);
e.PdfDocumentTemplate.Bottom = footer;
}
```

## Order Details

Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10001	Torrey	1702	818.00	False	Bruxelles	9/7/2008
10002	Daniel	1711	180.00	True	Rio de Janeiro	6/3/2009
10003	Brenda	1707	471.00	False	Campinas	4/13/2010
10004	Irene	1700	634.00	True	Grazd	10/21/2006
10005	Kyle	1701	796.00	False	Resende	3/18/2002
10006	Fiona	1718	96.00	True	Rio de Janeiro	5/12/2004
10007	Irene	1715	731.00	False	Grazd	9/10/2000
10008	Danielle	1712	676.00	True	Rio de Janeiro	6/26/2011
10009	Gina	1713	590.00	False	Buenos Aires	4/3/2005
10010	Irene	1707	391.00	True	Bruxelles	3/18/2010
10011	Torrey	1702	332.00	False	Bruxelles	9/1/2010
10012	Torrey	1713	577.00	True	Bruxelles	3/7/2012
10013	Michael	1718	317.00	False	Montréal	3/9/2007
10014	Kyle	1713	425.00	True	Rio de Janeiro	7/27/2013
10015	Gina	1700	912.00	False	Resende	6/19/2012
10016	Daniel	1715	865.00	True	Buenos Aires	5/18/2001
10017	Bill	1715	43.00	False	Bruxelles	8/8/2008
10018	Fiona	1709	832.00	True	Bruxelles	4/17/2013
10019	Daniel	1718	310.00	False	Resende	8/20/2011

## Order Details

## Order Details

Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10020	Katie	1710	718.00	True	Tsawassen	3/19/2010
10021	Brenda	1718	139.00	False	Grazd	3/26/2007
10022	Irene	1704	650.00	True	Bruxelles	5/3/2007

### Change PDF page orientation

You can change the page orientation of the PDF document when exporting. Default page orientation is portrait.

To change the page orientation, export PDF grid value by using the `ExportToPdfGrid` method. Then, draw that PDF grid into a PDF document by changing the `PageSettings.Orientation` property of PDF document.

#### **C#**

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
PdfDocument pdfDocument = new PdfDocument();  
pdfDocument.PageSettings.Orientation = PdfPageOrientation.Landscape;  
//pdfDocument.PageSettings.Orientation = PdfPageOrientation.Portrait;  
option.PdfDocument = pdfDocument;
```



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10001	Irene	1703	893.00	N	Tsawassen	9/7/2008
10002	Oscar	1712	358.00	Y	Buenos Aires	6/3/2009
10003	Irene	1703	736.00	N	Bruxelles	4/13/2010
10004	Oscar	1704	937.00	Y	Bruxelles	10/21/2006
10005	Ralph	1708	331.00	N	Grazd	3/18/2002
10006	Katie	1716	620.00	Y	Campinas	5/12/2004
10007	Ralph	1716	365.00	N	Grazd	9/10/2000
10008	Oscar	1718	177.00	Y	Buenos Aires	6/26/2011
10009	Ralph	1716	146.00	N	Buenos Aires	4/3/2005
10010	Irene	1700	676.00	Y	Buenos Aires	3/18/2010
10011	Ralph	1718	31.00	N	Buenos Aires	9/1/2010
10012	Torrey	1705	749.00	Y	Montréal	3/7/2012
10013	Michael	1708	81.00	N	Grazd	3/9/2007
10014	Kyle	1704	508.00	Y	Rio de Janeiro	7/27/2013



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10015	Bill	1709	621.00	N	Bruxelles	6/19/2012
10016	Frank	1707	850.00	Y	Campinas	5/18/2001
10017	Danielle	1719	473.00	N	Montréal	8/8/2008
10018	Oscar	1704	223.00	Y	Bruxelles	4/17/2013
10019	Daniel	1703	397.00	N	Bruxelles	8/20/2011
10020	William	1717	513.00	Y	Tsawassen	3/19/2010
10021	Daniel	1714	673.00	N	Buenos Aires	3/26/2007
10022	William	1701	175.00	Y	Buenos Aires	5/3/2007

Saving options

[Save directly as file](#)

[Save directly as file](#)

The following code snippet explains how to save the converted PDF document in our local device.

### C#

```
// Need to define the Interfaces in-order to use it in platform wise using
Dependency Service
public interface ISave
{
    void Save(string filename, string contentType, MemoryStream stream);
}
public interface ISaveWindows
{
    Task Save(string filename, string contentType, MemoryStream stream);
}
// In Android renderer project
public class SaveAndroid : ISave
{
    public void Save(string filename, string contentType, MemoryStream stream)
    {
        string exception = string.Empty;
        string root = null;
        if (Android.OS.Environment.IsExternalStorageEmulated)
        {
            root = Android.OS.Environment.ExternalStorageDirectory.ToString();
        }
        else
        {
            root = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
            Java.IO.File myDir = new Java.IO.File(root + "/Syncfusion");
            myDir.Mkdir();
            Java.IO.File file = new Java.IO.File(myDir, filename);
            if (file.Exists()) file.Delete();
            try
            {
                FileOutputStream outs = new FileOutputStream(file);
                outs.Write(stream.ToArray());
                outs.Flush();
                outs.Close();
            }
            catch (Exception e)
            {
                exception = e.ToString();
            }
            if (file.Exists() && contentType != "application/html")
            {
                Android.Net.Uri path = Android.Net.Uri.FromFile(file);
                string extension =
                Android.Webkit.MimeTypeMap.GetFileExtensionFromUrl(Android.Net.Uri.FromFile(
                file).ToString());
                string mimeType =
                Android.Webkit.MimeTypeMap.Singleton.GetMimeTypeFromExtension(extension);
                Intent intent = new Intent(Intent.ActionView);
                intent.SetDataAndType(path, mimeType);
                Forms.Context.StartActivity(Intent.CreateChooser(intent, "Choose App"));
            }
        }
    }
}
```

```
}
}
}
// In iOS renderer project
public class SaveIOS : ISave
{
    void ISave.Save(string filename, string contentType, MemoryStream stream)
    {
        string exception = string.Empty;
        string path = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        string filePath = Path.Combine(path, filename);
        try
        {
            FileStream fileStream = File.Open(filePath, FileMode.Create);
            stream.Position = 0;
            stream.CopyTo(fileStream);
            fileStream.Flush();
            fileStream.Close();
        }
        catch (Exception e)
        {
            exception = e.ToString();
        }
        if (contentType == "application/html" || exception != string.Empty)
            return;
        UIViewController currentController =
            UIApplication.SharedApplication.KeyWindow.RootViewController;
        while (currentController.PresentedViewController != null)
            currentController = currentController.PresentedViewController;
        UIView currentView = currentController.View;
        QLPreviewController previewController = new QLPreviewController();
        QLPreviewItem item = new QLPreviewItemBundle(filename, filePath);
        previewController.DataSource = new PreviewControllerDS(item);
        currentController.PresentViewController((UIViewController)previewController,
            true, (Action)null);
    }
}

public class PreviewControllerDS : QLPreviewControllerDataSource
{
    private QLPreviewItem _item;
    public PreviewControllerDS(QLPreviewItem item)
    {
        _item = item;
    }
    public override nint PreviewItemCount (QLPreviewController controller)
    {
        return (nint)1;
    }
    public override IQLPreviewItem GetPreviewItem (QLPreviewController
        controller, nint index)
    {
        return _item;
    }
}

public class QLPreviewItemFileSystem : QLPreviewItem
{
    string _fileName, _filePath;
```

```
public QLPreviewItemFileSystem(string fileName, string filePath)
{
    _fileName = fileName;
    _filePath = filePath;
}
public override string ItemTitle
{
    get
    {
        return _fileName;
    }
}
public override NSURL ItemUrl
{
    get
    {
        return NSURL.FromFilename(_filePath);
    }
}
}

public class QLPreviewItemBundle : QLPreviewItem
{
    string _fileName, _filePath;
    public QLPreviewItemBundle(string fileName, string filePath)
    {
        _fileName = fileName;
        _filePath = filePath;
    }
    public override string ItemTitle
    {
        get
        {
            return _fileName;
        }
    }
    public override NSURL ItemUrl
    {
        get
        {
            var documents = NSBundle.MainBundle.BundlePath;
            var lib = Path.Combine(documents, _filePath);
            var url = NSURL.FromFilename(lib);
            return url;
        }
    }
}

// In UWP renderer project
public class SaveWindows : ISaveWindows
{
    public async Task Save(string filename, string contentType, MemoryStream stream)
    {
        if (Device.Idiom != TargetIdiom.Desktop)
        {
            StorageFolder local = Windows.Storage.ApplicationData.Current.LocalFolder;
            StorageFile outFile = await local.CreateFileAsync(filename,
                CreationCollisionOption.ReplaceExisting);
        }
    }
}
```



```

using (Stream outputStream = await outFile.OpenStreamForWriteAsync())
{
    outputStream.Write(stream.ToArray(), 0, (int)stream.Length);
}
if (contentType != "application/html")
    await Windows.System.Launcher.LaunchFileAsync(outFile);
}
else
{
    StorageFile storageFile = null;
    FileSavePicker savePicker = new FileSavePicker();
    savePicker.SuggestedStartLocation = PickerLocationId.Desktop;
    savePicker.SuggestedFileName = filename;
    switch (contentType)
    {
        case "application/vnd.openxmlformats-officedocument.presentationml.presentation":
            savePicker.FileTypeChoices.Add("PowerPoint Presentation", new List<string>() { ".pptx", });
            break;
        case "application/msexcel":
            savePicker.FileTypeChoices.Add("Excel Files", new List<string>() { ".xlsx", });
            break;
        case "application/msword":
            savePicker.FileTypeChoices.Add("Word Document", new List<string>() { ".docx", });
            break;
        case "application/pdf":
            savePicker.FileTypeChoices.Add("Adobe PDF Document", new List<string>() { ".pdf", });
            break;
        case "application/html":
            savePicker.FileTypeChoices.Add("HTML Files", new List<string>() { ".html", });
            break;
    }
    storageFile = await savePicker.PickSaveFileAsync();
    using (Stream outputStream = await storageFile.OpenStreamForWriteAsync())
    {
        outputStream.Write(stream.ToArray(), 0, (int)stream.Length);
        outputStream.Flush();
        outputStream.Dispose();
    }
    stream.Flush();
    stream.Dispose();
    await Windows.System.Launcher.LaunchFileAsync(storageFile);
}
}
}

```

#### *Save as stream*

You can also save the manipulated PDF document to stream by using overloads of the Save method.

#### **C#**

```
//Load an existing PDF document  
PdfLoadedDocument loadedDocument = new PdfLoadedDocument("Input.pdf");  
//To-Do some manipulation  
//To-Do some manipulation  
//Creates an instance of memory stream  
MemoryStream stream = new MemoryStream();  
//Save the document stream  
loadedDocument.Save(stream);
```

## Row height and column width customization

### *ExportColumnWidth*

By default, the data grid columns will be exported to PDF with

`DataGridPdfExportOption.DefaultColumnWidth` value. To export the data grid to PDF with exact column widths, set the [DataGridPdfExportOption.ExportColumnWidth](#) to `true`.

### **C#**

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.ExportColumnWidth = true;
```

### *ExportRowHeight*

By default, the data grid rows will be exported to PDF with

`DataGridPdfExportOption.DefaultRowHeight` value. To export the data grid to PDF with exact row heights, set the [DataGridPdfExportOption.ExportRowHeight](#) to `true`.

### **C#**

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.ExportRowHeight = true;
```



### *DefaultColumnWidth*

The SfDataGrid allows customizing column width in the PDF document using the [DataGridPdfExportOption.DefaultColumnWidth](#) property. The `DefaultColumnWidth` value will be applied to all the columns in the document.

#### **C#**

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.DefaultColumnWidth = 100;
```

### *DefaultRowHeight*

The SfDataGrid allows customizing row height in the PDF document using the [DataGridPdfExportOption.DefaultRowHeight](#) property. The `DefaultRowHeight` value will be applied to all the rows in the document.

#### **C#**

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.DefaultRowHeight = 50;
```



## Events

### Exporting customization

#### *Styling cells based on CellType in PDF*

The SfDataGrid provides you the following events for exporting:

- [RowExporting](#): Raised when exporting a row at the execution time.
- [CellExporting](#): Raised when exporting a cell at the execution time.

#### *RowExporting*

The [DataGridRowPdfExportingEventHandler](#) delegate allows customizing the styles of record rows and group caption rows. The **RowExporting** event is triggered with [DataGridRowPdfExportingEventArgs](#) that contains the following properties:

- [PdfGrid](#): Customizes the pdfGrid properties such as **Background**, **CellPadding**, **CellSpacing**, and so on.
- [PdfRow](#): Specifies the **PDFGridRow** to be exported. Customizes the properties of a particular row.
- [Record](#): Gets the collection of exported underlying data objects.
- [RowType](#): Specifies the row type using **ExportRowType** enum. Checks the row type and applies different styles based on the row type.

You can use this event to customize the properties of the grid rows exported to PDF. The following code example illustrates how to change the background color of the record rows and caption summary rows when exporting.

#### **C#**

```
//HandlingRowExportingEvent for exporting to PDF
DataGridPdfExportingController pdfExport = new
DataGridPdfExportingController ();
pdfExport.RowExporting += pdfExport_RowExporting;
void pdfExport_RowExporting (object sender, DataGridRowPdfExportingEventArgs
e)
{
    if (e.RowType == ExportRowType.Record) {
        if ((e.Record.Data as OrderInfo).IsClosed)
            e.PdfRow.Style.BackgroundBrush = PdfBrushes.Yellow;
        else
            e.PdfRow.Style.BackgroundBrush = PdfBrushes.LightGreen;
    }
    // You can also set the desired background colors for the CaptionSummary row
    // and GroupSummary row as shown below
    // if (e.RowType == ExportRowType.CaptionSummary) {
    //     e.PdfRow.Style.BackgroundBrush = PdfBrushes.LightGray;
    // }
    // if (e.RowType == ExportRowType.GroupSummary) {
    //     e.PdfRow.Style.BackgroundBrush = PdfBrushes.Red;
    // }
}
```



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity
10001	Daniel	1707	469.00	False	Resende
10002	Kyle	1711	731.00	True	Resende
10003	William	1714	574.00	False	Rio de Janeiro
10004	Bill	1703	134.00	True	Buenos Aires
10005	William	1707	567.00	False	Rio de Janeiro
10006	Danielle	1711	341.00	True	Tsawassen
10007	Katie	1700	123.00	False	Rio de Janeiro
10008	Torrey	1702	623.00	True	Montréal
10009	Fiona	1718	854.00	False	Rio de Janeiro
10010	Daniel	1705	369.00	True	Grazd
10011	Katie	1710	991.00	False	Buenos Aires
10012	Gina	1701	530.00	True	Resende
10013	Bill	1717	500.00	False	Tsawassen
10014	Torrey	1702	678.00	True	Grazd

### CellExporting

The [DataGridCellPdfExportingEventHandler](#) delegate allows customizing the styles for the header cells, record cells, group caption cells, and group summary cells. The [CellExporting](#) event is triggered with [DataGridCellPdfExportingEventArgs](#) that contains the following properties:

The [DataGridCellPdfExportingEventHandler](#) delegate allows customizing the styles for header cells, record cells, and group caption cells. The [CellExporting](#) event is triggered with [DataGridCellPdfExportingEventArgs](#) that contains the following properties:

- [CellType](#): Specifies the cell type using [ExportCellType](#) enum. Checks the cell type and applies different cell styles based on the cell type.
- [CellValue](#): Contains the exported actual value to format the PDF using the [Range](#) property.
- [ColumnName](#): Specifies the column name (MappingName) of the exporting cell. Applies formatting for a particular column by checking the [ColumnName](#).
- [Handled](#): Determines whether the cell is exported to PDF or not.
- [PdfGrid](#): Specifies the [PDFGridCell](#) to be exported. Customizes Background, Foreground, Font, Alignment, etc., properties of a particular cell.
- [Record](#): Gets the collection of exported underlying data objects.

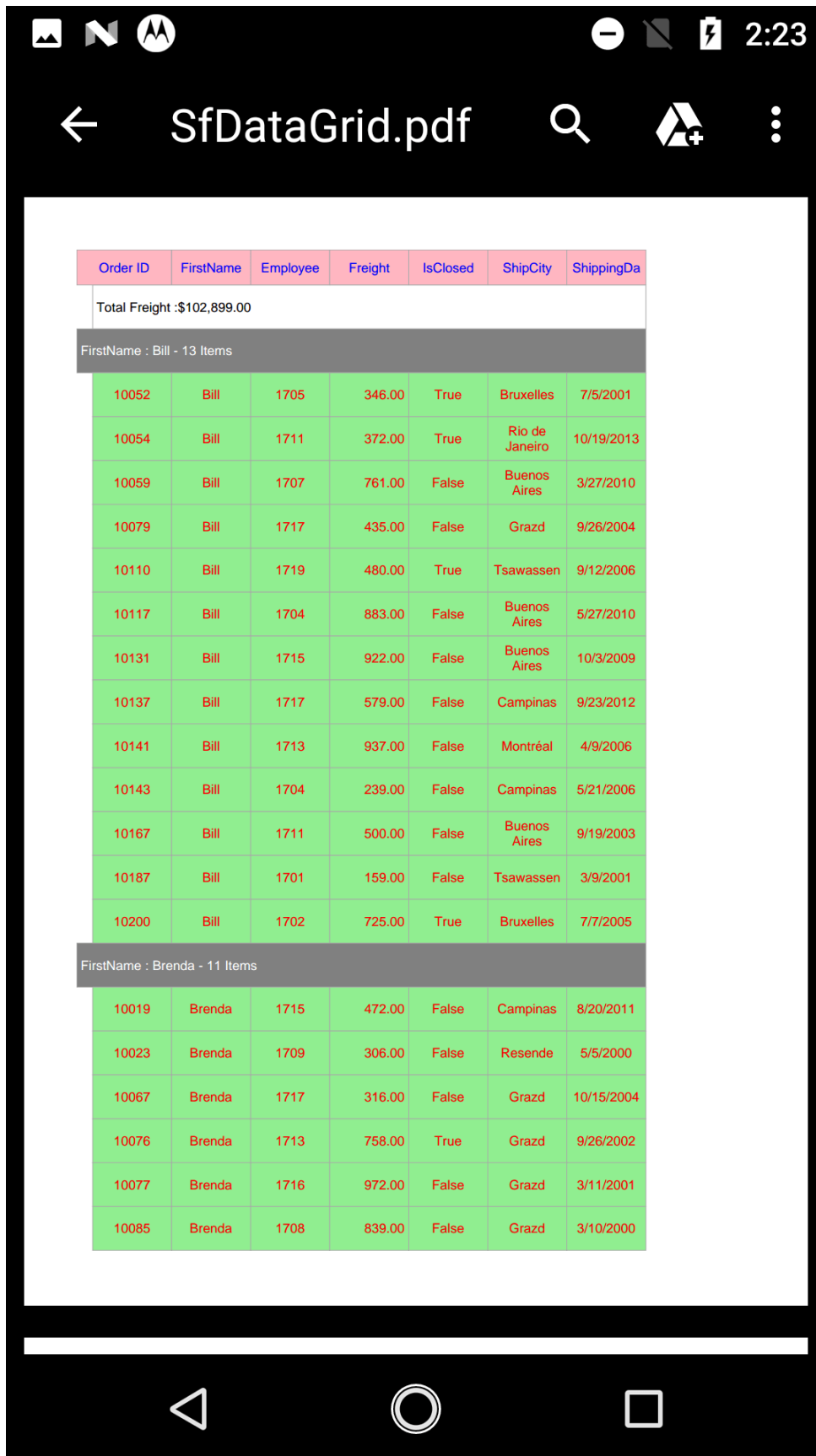
You can use this event to customize the properties of the grid cells exported to PDF. The following code example illustrates how to customize the background color, foreground color, and cell value of the header cells, record cells, and caption summary cells when exporting.

### C#

```
//HandlingCellExportingEvent for exporting to PDF
DataGridPdfExportingController pdfExport = new
DataGridPdfExportingController ();
pdfExport.CellExporting += pdfExport_CellExporting;
void pdfExport_CellExporting(object sender,
DataGridCellPdfExportingEventArgs e)
{
    if (e.CellType == ExportCellType.HeaderCell)
    {
        e.PdfGridCell.Style.BackgroundBrush = PdfBrushes.LightPink;
        e.PdfGridCell.Style.TextBrush = PdfBrushes.Blue;
    }
    if (e.CellType == ExportCellType.RecordCell)
    {
        e.PdfGridCell.Style.BackgroundBrush = PdfBrushes.LightGreen;
        e.PdfGridCell.Style.TextBrush = PdfBrushes.Red;
    }
    // You can also set the desired values for the CaptionSummary rows and
    // GroupSummary rows as shown below
    // if (e.CellType == ExportCellType.GroupCaptionCell)
    // {
    //     e.PdfGridCell.Style.BackgroundBrush = PdfBrushes.Gray;
    //     e.PdfGridCell.Style.TextBrush = PdfBrushes.White;
    // }
    // if (e.CellType == ExportCellType.GroupSummaryCell)
    // {
    //     e.PdfGridCell.Style.BackgroundBrush = PdfBrushes.Blue;
    //     e.PdfGridCell.Style.TextBrush = PdfBrushes.Orange;
    // }
```



```
// }  
}
```



### Exporting Unbound rows

By default the unbound rows will not be exported to pdf document. However, You can export the unbound rows to PDF by setting the [DataGridPdfExportOption.ExportUnboundRows](#) property as `true`.

#### C#

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
option.ExportUnboundRows = true;
```

### Exporting unbound columns

The `SfDataGrid.GridUnboundColumns` will be exported as [SfDataGrid.GridTextColumns](#) without any specific code. You can customize the `SfDataGrid.GridUnboundColumns` as `SfDataGrid.GridTextColumns` by using the `CellExporting` and `RowExporting` events.

#### XML

```
<sfgrid:GridUnboundColumn Expression="OrderID * 12"  
HeaderFontAttribute="Bold"  
HeaderText="Unbound"  
HeaderTextAlignment="Start"  
MappingName="Unbound"  
Padding="5, 0, 0, 0"  
TextAlignment="Start">  
</sfgrid:GridUnboundColumn>
```

The following screenshot shows that the unbound column is exported to PDF document with text columns.

Carrier  12:46 PM 

Done DataGrid.pdf



OrderID	CustomerID	Frieght	Country	Unbound
1001	Maria Anders	ALFKI	Germany	1001
1002	Ana Trujilo	ANATR	Mexico	1002
1003	Ant Fuller	ANTON	Mexico	1003
1004	Thomas Hardy	AROUT	UK	1004
1005	Tim Adams	BERGS	Sweden	1005
1006	Hanna Moos	BLAUS	Germany	1006
1007	Andrew Fuller	BLONP	France	1007
1008	Martin King	BOLID	Spain	1008
1009	Lenny Lin	BONAP	France	1009
1010	John Carter	BOTTM	Canada	1010
1011	Lauro King	AROUT	UK	1011
1012	Anne Wilson	BLAUS	Germany	1012
1013	Maria Anders	ALFKI	Germany	1013
1014	Ana Trujilo	ANATR	Mexico	1014
1015	Ant Fuller	ANTON	Mexico	1015
1016	Thomas Hardy	AROUT	UK	1016
1017	Tim Adams	BERGS	Sweden	1017
1018	Hanna Moos	BLAUS	Germany	1018
1019	Andrew Fuller	BLONP	France	1019
1020	Martin King	BOLID	Spain	1020
1021	Lauro King	AROUT	UK	1021
1022	Anne Wilson	BLAUS	Germany	1022



### *ExportGroupSummary*

By default, the **GroupSummary** rows in the data grid will be exported to PDF. To export the **SfDataGrid** without group summaries, set the [DataGridPdfExportingOption.ExportGroupSummary](#) property to **false**.

### **C#**

```
DataGridPdfExportOption option = new DataGridPdfExportOption();  
// Set false here to export the DataGrid without GroupSummary rows. The  
// default value is true.  
// option.ExportGroupSummary = false;
```

Carrier  12:21 PM 

Done DataGrid.pdf



OrderID	CustomerID	Frieght	Country
1001	Maria Anders	ALFKI	Germany
1002	Ana Trujilo	ANATR	Mexico
1003	Ant Fuller	ANTON	Mexico
1004	Thomas Hardy	AROUT	UK
1005	Tim Adams	BERGS	Sweden
1006	Hanna Moos	BLAUS	Germany
1007	Andrew Fuller	BLONP	France
1008	Martin King	BOLID	Spain
1009	Lenny Lin	BONAP	France
1010	John Carter	BOTTM	Canada
1011	Lauro King	AROUT	UK
1012	Anne Wilson	BLAUS	Germany
1013	Maria Anders	ALFKI	Germany
1014	Ana Trujilo	ANATR	Mexico
1015	Ant Fuller	ANTON	Mexico
1016	Thomas Hardy	AROUT	UK
1017	Tim Adams	BERGS	Sweden
1018	Hanna Moos	BLAUS	Germany
1019	Andrew Fuller	BLONP	France
1020	Martin King	BOLID	Spain
1021	Lauro King	AROUT	UK
1022	Anne Wilson	BLAUS	Germany

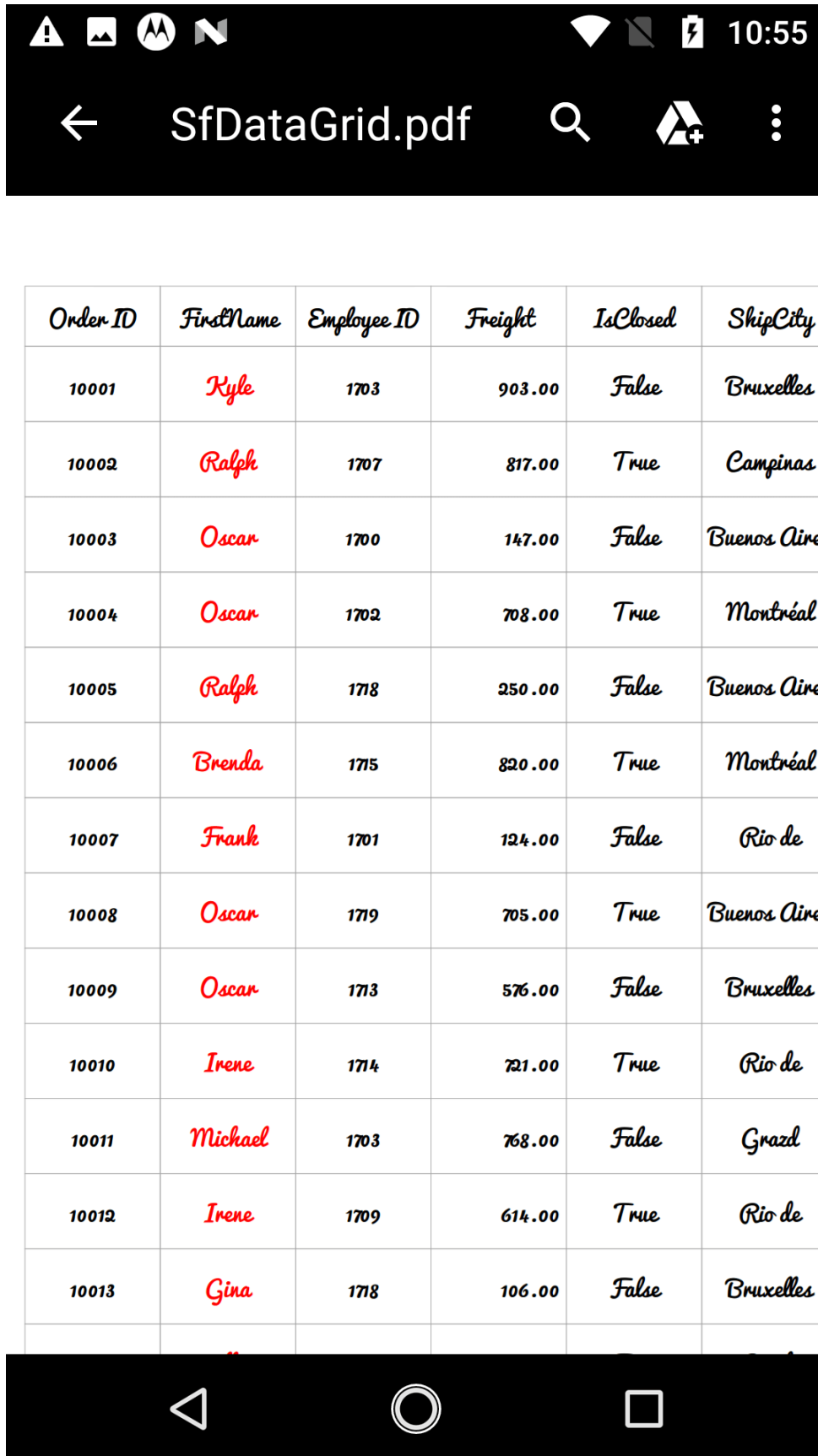


*Embedding fonts in PDF file*

By default, some fonts (such as Unicode font) are not supported in PDF. In this case, embed the font in the PDF document with the help of PdfTrueTypeFont.

**C#**

```
Stream fontStream =
typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("SfDataGridSample.Pacifico.ttf");
private void PdfExport_CellExporting(object sender,
DataGridCellPdfExportingEventArgs e)
{
if (e.CellValue != null)
{
PdfFont font = e.PdfGridCell.Style.Font;
if (font != null)
{
PdfTrueTypeFont unicodeFont = new PdfTrueTypeFont(fontStream, font.Size,
font.Style);
e.PdfGridCell.Style.Font = unicodeFont;
}
}
}
```



Order ID	FirstName	Employee ID	Freight	IsClosed	ShipCity
10001	Kyle	1703	903.00	False	Bruxelles
10002	Ralph	1707	817.00	True	Campinas
10003	Oscar	1700	147.00	False	Buenos Aires
10004	Oscar	1702	708.00	True	Montréal
10005	Ralph	1718	250.00	False	Buenos Aires
10006	Brenda	1715	820.00	True	Montréal
10007	Frank	1701	124.00	False	Rio de
10008	Oscar	1719	705.00	True	Buenos Aires
10009	Oscar	1713	576.00	False	Bruxelles
10010	Irene	1714	721.00	True	Rio de
10011	Michael	1703	768.00	False	Grazd
10012	Irene	1709	614.00	True	Rio de
10013	Gina	1718	106.00	False	Bruxelles



## Cell customization in PDF while exporting

### *Customize cell values while exporting*

You can customize the cell values when exporting to PDF by handling the **CellExporting** event.

#### **C#**

```
DataGridPdfExportingController pdfExport = new
DataGridPdfExportingController();
pdfExport.CellExporting += PdfExport_CellExporting;
private void PdfExport_CellExporting(object sender,
DataGridCellPdfExportingEventArgs e)
{
    if (e.CellType == ExportCellType.RecordCell && e.ColumnName == "IsClosed")
    {
        if ((bool)e.CellValue)
            e.CellValue = "Y";
        else
            e.CellValue = "N";
    }
}
```

In the following screenshot, IsClosed column value has been changed based on the condition.



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10001	Irene	1703	893.00	N	Tsawassen	9/7/2008
10002	Oscar	1712	358.00	Y	Buenos Aires	6/3/2009
10003	Irene	1703	736.00	N	Bruxelles	4/13/2010
10004	Oscar	1704	937.00	Y	Bruxelles	10/21/2006
10005	Ralph	1708	331.00	N	Grazd	3/18/2002
10006	Katie	1716	620.00	Y	Campinas	5/12/2004
10007	Ralph	1716	365.00	N	Grazd	9/10/2000
10008	Oscar	1718	177.00	Y	Buenos Aires	6/26/2011
10009	Ralph	1716	146.00	N	Buenos Aires	4/3/2005
10010	Irene	1700	676.00	Y	Buenos Aires	3/18/2010
10011	Ralph	1718	31.00	N	Buenos Aires	9/1/2010
10012	Torrey	1705	749.00	Y	Montréal	3/7/2012
10013	Michael	1708	81.00	N	Grazd	3/9/2007
10014	Kyle	1704	508.00	Y	Rio de Janeiro	7/27/2013



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10015	Bill	1709	621.00	N	Bruxelles	6/19/2012
10016	Frank	1707	850.00	Y	Campinas	5/18/2001
10017	Danielle	1719	473.00	N	Montréal	8/8/2008
10018	Oscar	1704	223.00	Y	Bruxelles	4/17/2013
10019	Daniel	1703	397.00	N	Bruxelles	8/20/2011
10020	William	1717	513.00	Y	Tsawassen	3/19/2010
10021	Daniel	1714	673.00	N	Buenos Aires	3/26/2007
10022	William	1701	175.00	Y	Buenos Aires	5/3/2007

*Changing row style in PDF based on data*

You can customize the row style based on the row data when exporting to PDF by handling the **RowExporting** event.

**C#**

```
DataGridPdfExportingController pdfExport = new
DataGridPdfExportingController();
pdfExport.RowExporting += PdfExport_RowExporting;
private void PdfExport_RowExporting(object sender,
DataGridRowPdfExportingEventArgs e)
{
    if (!(e.Record.Data is OrderInfo))
        return;
    if (e.RowType == ExportRowType.Record)
    {
        if ((e.Record.Data as OrderInfo).IsClosed)
            e.PdfRow.Style.BackgroundBrush = PdfBrushes.Yellow;
        else
            e.PdfRow.Style.BackgroundBrush = PdfBrushes.LightGreen;
    }
}
```



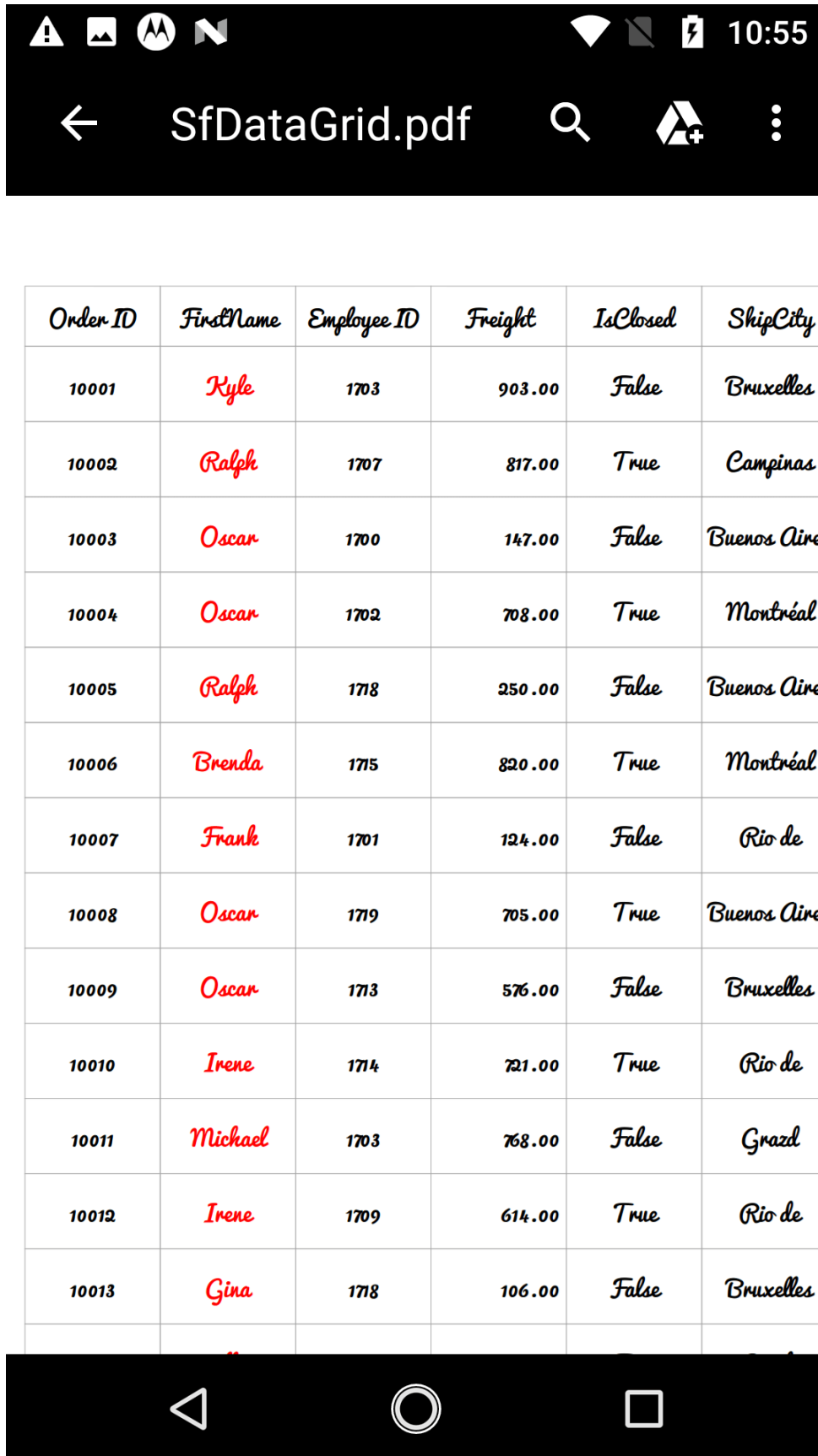
Order ID	FirstName	Employee	Freight	IsClosed	ShipCity
10001	Daniel	1707	469.00	False	Resende
10002	Kyle	1711	731.00	True	Resende
10003	William	1714	574.00	False	Rio de Janeiro
10004	Bill	1703	134.00	True	Buenos Aires
10005	William	1707	567.00	False	Rio de Janeiro
10006	Danielle	1711	341.00	True	Tsawassen
10007	Katie	1700	123.00	False	Rio de Janeiro
10008	Torrey	1702	623.00	True	Montréal
10009	Fiona	1718	854.00	False	Rio de Janeiro
10010	Daniel	1705	369.00	True	Grazd
10011	Katie	1710	991.00	False	Buenos Aires
10012	Gina	1701	530.00	True	Resende
10013	Bill	1717	500.00	False	Tsawassen
10014	Torrey	1702	678.00	True	Grazd

*Customize the cells based on column name*

You can customize the column style based on the row data when exporting to PDF by handling the `CellExporting` event.

**C#**

```
DataGridPdfExportingController pdfExport = new
DataGridPdfExportingController();
pdfExport.CellExporting += PdfExport_CellExporting;
private void PdfExport_CellExporting(object sender,
DataGridCellPdfExportingEventArgs e)
{
    if (e.CellType == ExportCellType.RecordCell && e.ColumnName == "FirstName")
    {
        e.PdfGridCell.Style.TextBrush = PdfBrushes.Red;
    }
}
```



The image shows a mobile application interface. At the top is a black header bar with a back arrow, the title 'SfDataGrid.pdf', a search icon, a share icon, and a menu icon. Below the header is a table with 6 columns: Order ID, FirstName, Employee ID, Freight, IsClosed, and ShipCity. The table contains 13 rows of data. At the bottom is a black navigation bar with three icons: a back arrow, a circle, and a square.

Order ID	FirstName	Employee ID	Freight	IsClosed	ShipCity
10001	Kyle	1703	903.00	False	Bruxelles
10002	Ralph	1707	817.00	True	Campinas
10003	Oscar	1700	147.00	False	Buenos Aires
10004	Oscar	1702	708.00	True	Montréal
10005	Ralph	1718	250.00	False	Buenos Aires
10006	Brenda	1715	820.00	True	Montréal
10007	Frank	1701	124.00	False	Rio de
10008	Oscar	1719	705.00	True	Buenos Aires
10009	Oscar	1713	576.00	False	Bruxelles
10010	Irene	1714	721.00	True	Rio de
10011	Michael	1703	768.00	False	Graz
10012	Irene	1709	614.00	True	Rio de
10013	Gina	1718	106.00	False	Bruxelles

*Exporting middle Eastern languages (Arabic and Hebrew) from SfDataGrid to PDF*

By default, [Middle Eastern languages](#) (Arabic and Hebrew) in the SfDataGrid are exported as left to right in PDF. You can export them as displayed (export from right to left) by enabling the `RightToLeft` property in PdfStringFormat class and apply the format to the PdfGridCell by using CellsExportingEventHandler.

**C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.CellsExportingEventHandler = CellsExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
private void CellsExportingEventHandler(object sender,
GridCellPdfExportingEventArgs e)
{
    if (e.CellType != ExportCellType.RecordCell)
        return;
    PdfStringFormat format = new PdfStringFormat();
    //format the string from right to left.
    format.RightToLeft = true;
    e.PdfGridCell.StringFormat = format;
}
```

*Exporting images to PDF document*

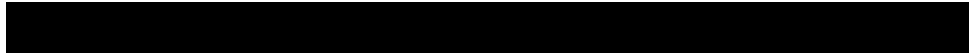
By default, images loaded in the GridTemplateColumn will not be exported to PDF. You can export it by handling the CellExporting event. In DataGridCellPdfExportingEventHandler, the required image is loaded in the PdfGridCell.








**C#**

```
private void PdfExport_CellExporting(object sender,
DataGridCellPdfExportingEventArgs e)
{
    if (e.CellType == ExportCellType.RecordCell && e.ColumnName == "IsClosed")
    {
        var style = new PdfGridCellStyle();
        PdfPen normalBorder = new PdfPen(PdfBrushes.DarkGray, 0.2f);
        Xamarin.Forms.Image trueImage = new Xamarin.Forms.Image();
        Xamarin.Forms.Image falseImage = new Xamarin.Forms.Image();
        if ((bool)e.CellValue)
            trueImage.Source = ImageSource.FromResource("SfDataGridSample.True.jpg");
        else
            falseImage.Source = ImageSource.FromResource("SfDataGridSample.False.jpg");
        Assembly assembly = null;
        assembly = e.Record.Data.GetType().GetTypeInfo().Assembly;
        var streamImageSource = trueImage.Source as StreamImageSource;
        if (streamImageSource != null)
        {
            var imageStream =
                assembly.GetManifestResourceStream(ExportingHelper.GetImagePath(streamImageSource));
            if (imageStream != null)
            {
                PdfImage pdfImage = PdfImage.FromStream(imageStream);
                style.BackgroundImage = pdfImage;
            }
        }
    }
}
```





```
e.PdfGridCell.ImagePosition = PdfGridImagePosition.Fit;  
e.PdfGridCell.Style = style;  
imageStream.Flush();  
e.CellValue = null;  
}  
}  
}  
}
```





Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10001	Gina	1704	101.00	False	Grazd	9/7/2008
10002	Ralph	1719	122.00		Buenos Aires	6/3/2009
10003	Irene	1716	370.00	False	Rio de Janeiro	4/13/2010
10004	Brenda	1717	305.00		Bruxelles	10/21/2006
10005	Katie	1709	362.00	False	Grazd	3/18/2002
10006	Torrey	1718	585.00		Grazd	5/12/2004
10007	Danielle	1701	249.00	False	Campinas	9/10/2000
10008	Frank	1718	962.00		Grazd	6/26/2011
10009	Kyle	1708	665.00	False	Bruxelles	4/3/2005
10010	Gina	1714	319.00		Tsawassen	3/18/2010
10011	Irene	1703	361.00	False	Buenos Aires	9/1/2010
10012	Gina	1710	693.00		Rio de Janeiro	3/7/2012
10013	Ralph	1710	800.00	False	Rio de Janeiro	3/9/2007
10014	Katie	1716	917.00		Buenos Aires	7/27/2013



Order ID	FirstName	Employee	Freight	IsClosed	ShipCity	ShippingDa
10015	Gina	1718	492.00	False	Bruxelles	6/19/2012
10016	Michael	1706	863.00		Tsawassen	5/18/2001
10017	Fiona	1703	298.00	False	Montréal	8/8/2008
10018	Brenda	1719	22.00		Campinas	4/17/2013
10019	Katie	1709	151.00	False	Tsawassen	8/20/2011
10020	Frank	1711	898.00		Grazd	3/19/2010
10021	Kyle	1711	186.00	False	Campinas	3/26/2007
10022	Torrey	1708	391.00		Bruxelles	5/3/2007

### Exporting the selected rows of SfDataGrid

SfDataGrid allows you to export only the currently selected rows in the grid to the document using the [DataGridPdfExportingController.ExportToPdf](#) method by passing the instance of the SfDataGrid and [SfDataGrid.SelectedItems](#) collection as an argument.

Refer the below code to export the selected rows alone to the PDF document.

#### C#

```
private void ExportToPDF(object sender, EventArgs e)
{
    DataGridPdfExportingController pdfExport = new
    DataGridPdfExportingController();
    MemoryStream stream = new MemoryStream();
    ObservableCollection<object> selectedItem = dataGrid.SelectedItems;
    var doc = pdfExport.ExportToPdf(this.dataGrid, selectedItem);
    doc.Save(stream);
    doc.Close(true);
    if (Device.RuntimePlatform == Device.UWP)
        Xamarin.Forms.DependencyService.Get<ISaveWindowsPhone>().Save("DataGrid.pdf",
        "application/pdf", stream);
    else
        Xamarin.Forms.DependencyService.Get<ISave>().Save("DataGrid.pdf",
        "application/pdf", stream);
}
```

Order ID	Customer	Ship Country	Ship City	Customer ID
1001	ANATR	Mexico	Mexico D.F.	Ana Hardy
1002	ANTON	Mexico	Mexico D.F.	Ant Fuller
1003	AROUT	UK	London	Thomas Hardy
1004	BERGS	Sweden	Berlin	Tim Adams
1005	BLAUS	Germany	Mannheim	Hanna Moos
1006	BLONP	France	Strasbourg	Andrew Fuller
1007	BOLID	Spain	Madrid	Martin King
1008	BONAP	France	Marseille	Lenny Lin



Created with a trial version of Syncfusion Essential PDF

Order ID	Customer	Ship Country	Ship City	Customer ID
1001	ANATR	Mexico	Mexico D.F.	Ana Hardy
1003	AROUT	UK	London	Thomas Hardy
1007	BOLID	Spain	Madrid	Martin King

### Localization

Localization is the process of translating application resources into different languages for specific cultures. SfDataGrid uses the following static text that can be localized in application level:

- LOAD MORE ITEMS
- Drop below
- Drop above
- Cancel drop

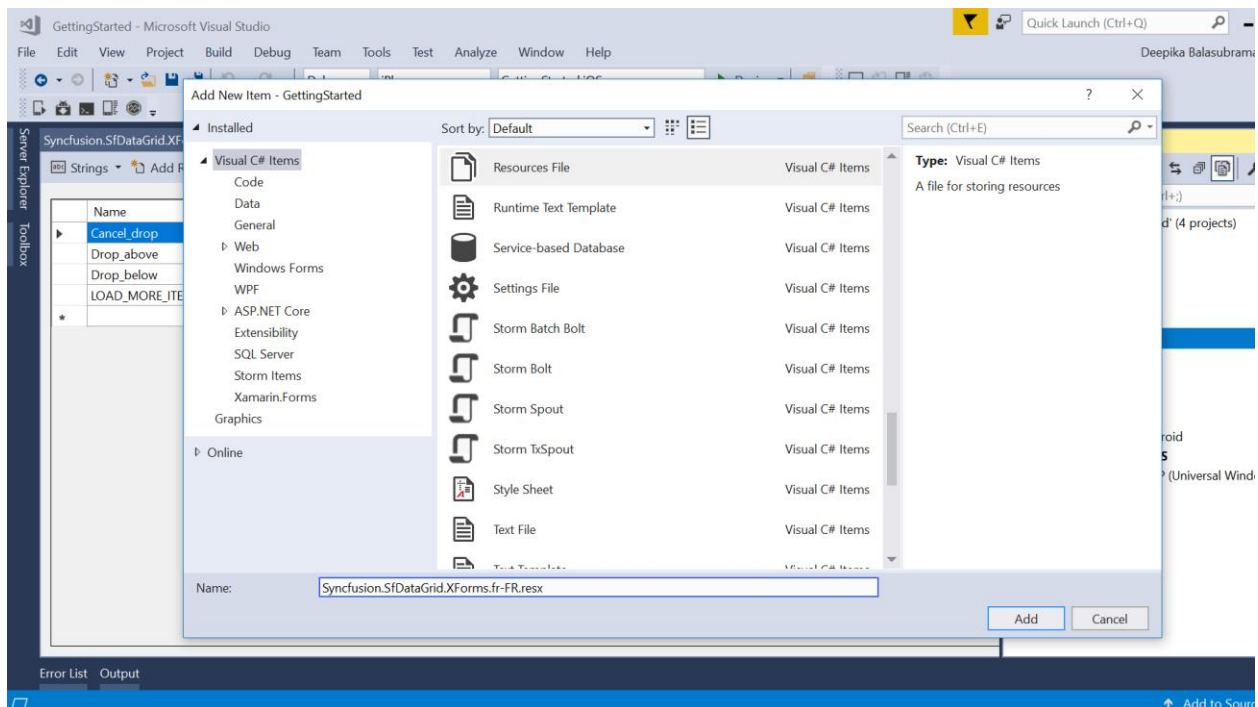
To localize the SfDataGrid, follow the steps in application level:

1. Add a .resx file.
2. Convert the platform specific language format to .NET format.
3. Apply the converted format.

### Add a .resx file

In the portable project of your application, add a .resx file inside the resources folder with **Build Action -> EmbeddedResource**. The file name should be **Syncfusion control's Namespace + language code** format.

For example, to set the culture as French, the file should be named as **Syncfusion.SfDataGrid.XForms.fr-FR.resx**.

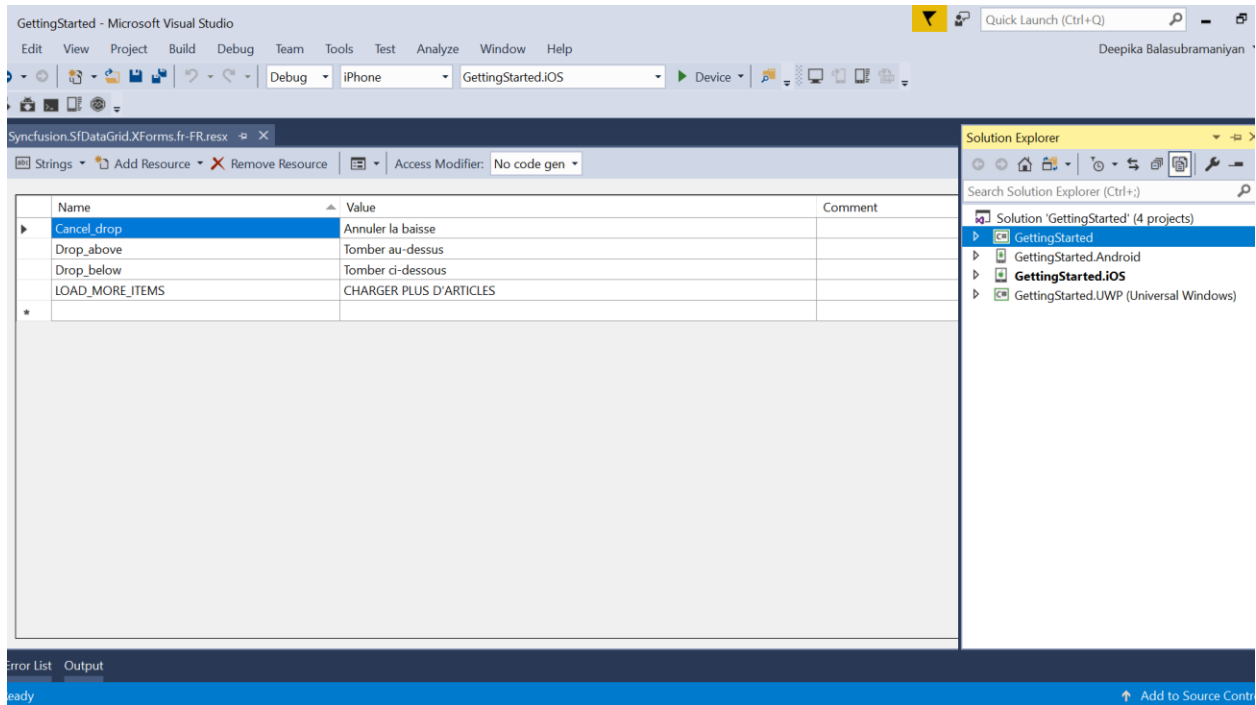


Based on the language, set the appropriate equivalent text to the static text in the .resx file.

---

**Note:** You should create and add separate .resx files for the individual languages.

---



Convert the platform specific language format to .NET format

To get the localized text from the added .resx file, declare an interface named `ILocalize` in your PCL project and implement the interface in each platform renderer. This will query the language set in the device using platform specific code and convert this platform specific format to .NET format.

Refer to the following code snippet to declare the interface in PCL project.

### C#

```
public interface ILocalize
{
    CultureInfo GetCurrentCultureInfo();
    void SetLocale(CultureInfo cultureInfo);
}

public class PlatformCulture
{
    public PlatformCulture(string platformCultureString)
    {
        if (String.IsNullOrEmpty(platformCultureString))
            throw new ArgumentException("Expected culture identifier",
                "platformCultureString"); // in C# 6 use nameof(platformCultureString)
        PlatformString = platformCultureString.Replace("_", "-"); // .NET expects
        // dash, not underscore
        var dashIndex = PlatformString.IndexOf("-", StringComparison.Ordinal);
        if (dashIndex > 0)
        {
            var parts = PlatformString.Split('-');
            LanguageCode = parts[0];
            LocaleCode = parts[1];
        }
        else
        {
            LanguageCode = PlatformString;
        }
    }
}
```

```
LocaleCode = "";
}
}
public string PlatformString
{
    get; private set;
}
public string LanguageCode
{
    get; private set;
}
public string LocaleCode
{
    get; private set;
}
public override string ToString()
{
    return PlatformString; ;
}
}
```

Refer to the following code to implement the interface in Android renderer project.

### C#

```
public class Localize : ILocalize
{
    public void SetLocale(CultureInfo cultureInfo)
    {
        Thread.CurrentThread.CurrentCulture = cultureInfo;
        Thread.CurrentThread.CurrentUICulture = cultureInfo;
    }
    public CultureInfo GetCurrentCultureInfo()
    {
        var netLanguage = "en";
        var androidLocale = Java.Util.Locale.Default;
        netLanguage = AndroidToDotnetLanguage(androidLocale.ToString().Replace("_",
        "-"));
        // this gets called a lot - try/catch can be expensive so consider caching
        or something
        CultureInfo cultureInfo = null;
        try
        {
            cultureInfo = new CultureInfo(netLanguage);
        }
        catch
        {
            // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
            // fallback to first characters, in this case "en"
            try
            {
                var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
                cultureInfo = new CultureInfo(fallback);
            }
            catch
            {

```

```

// iOS language not valid .NET culture, falling back to English
cultureInfo = new CultureInfo("en");
}
}
return cultureInfo;
}
private string AndroidToDotnetLanguage(string androidLanguage)
{
var netLanguage = androidLanguage;
//certain languages need to be converted to CultureInfo equivalent
switch (androidLanguage)
{
case "ms-BN": // "Malaysian (Brunei)" not supported .NET culture
case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture
case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture
netLanguage = "ms"; // closest supported
break;
case "in-ID": // "Indonesian (Indonesia)" has different code in .NET
netLanguage = "id-ID"; // correct code for .NET
break;
case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET
culture
netLanguage = "de-CH"; // closest supported
break;
// add more application-specific cases here (if required)
// ONLY use cultures that have been tested and known to work
}
return netLanguage;
}
private string ToDotnetFallbackLanguage(PlatformCulture platformCulture)
{
var netLanguage = platformCulture.LanguageCode; // use the first part of the
identifier (two chars, usually);
switch (platformCulture.LanguageCode)
{
case "gsw":
netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
break;
// add more application-specific cases here (if required)
// ONLY use cultures that have been tested and known to work
}
return netLanguage;
}
}

```

Refer to the following code to implement the interface in iOS renderer project.

### C#

```

public class Localize : ILocalize
{
public void SetLocale(CultureInfo cultureInfo)
{
Thread.CurrentThread.CurrentCulture = cultureInfo;
Thread.CurrentThread.CurrentUICulture = cultureInfo;
}
}

```

```

public CultureInfo GetCurrentCultureInfo()
{
    var netLanguage = "en";
    if (NSLocale.PreferredLanguages.Length > 0)
    {
        var pref = NSLocale.PreferredLanguages[0];
        netLanguage = iOSToDotnetLanguage(pref);
    }
    // this gets called a lot - try/catch can be expensive so consider caching
    // or something
    CultureInfo cultureInfo = null;
    try
    {
        cultureInfo = new CultureInfo(netLanguage);
    }
    catch
    {
        // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
        // fallback to first characters, in this case "en"
        try
        {
            var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
            cultureInfo = new CultureInfo(fallback);
        }
        catch
        {
            // iOS language not valid .NET culture, falling back to English
            cultureInfo = new CultureInfo("en");
        }
    }
    return cultureInfo;
}

private string iOSToDotnetLanguage(string iOSLanguage)
{
    var netLanguage = iOSLanguage;
    //certain languages need to be converted to CultureInfo equivalent
    switch (iOSLanguage)
    {
        case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture
        case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture
            netLanguage = "ms"; // closest supported
            break;
        case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET
            culture
            netLanguage = "de-CH"; // closest supported
            break;
        // add more application-specific cases here (if required)
        // ONLY use cultures that have been tested and known to work
    }
    return netLanguage;
}

private string ToDotnetFallbackLanguage(PlatformCulture platCulture)
{
    var netLanguage = platCulture.LanguageCode; // use the first part of the
    identifier (two chars, usually);
    switch (platCulture.LanguageCode)
    {

```

```
//
case "pt":
netLanguage = "pt-PT"; // fallback to Portuguese (Portugal)
break;
case "gsw":
netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
break;
// add more application-specific cases here (if required)
// ONLY use cultures that have been tested and known to work
}
return netLanguage;
}
}
```

Implementation of the interface is not required for UWP project, since the resources automatically recognizes the selected language.

#### Apply the converted format

After setting the root/main page of the application in your App.Xaml.cs file of the PCL project, initialize a new instance of the `ResourceManager` class and set it to the [DataGridResourceManager.Manager](#) property to look up into the resources with specified root name in the given assembly. Using `DependencyService`, call `SetLocale()` of the implemented interface with necessary language code as parameter.

#### C#

```
public partial class App : Application
{
    public App()
    {
        InitializeComponent();
        MainPage = new GettingStarted.MainPage();
        if (Device.RuntimePlatform == Device.iOS || Device.RuntimePlatform == Device.Android)
        {
            DataGridResourceManager.Manager = new
            ResourceManager("GettingStarted.Resources.Syncfusion.SfDataGrid.XForms",
            this.GetType().GetTypeInfo().Assembly);
            // the ResourceManager class constructor has two parameters.
            // 1. ResXPath => Full path of the resx file in the application. Here in the
            // above line GettingStarted refers to the namespace of the Application
            // 2. Assembly => Application assembly (PCL)
            // Sets the required culture to the static texts in the control.
            DependencyService.Get<ILocalize>().SetLocale(new CultureInfo("fr-FR"));
        }
    }
}
```

For Android and iOS, it is mandatory to implement the previous steps. However, to set a specific language to the application irrespective of the selected language in the device, use `CultureInfo.CurrentCulture` in a specific project of UWP platform.

Refer to the following code example to localize the text in UWP platform.



MainPage.Xaml.cs

**C#**

```
public MainPage()
{
    this.InitializeComponent();
    SfDataGridRenderer.Init();
    // Applying localization for UWP
    CultureInfo.CurrentUICulture = new CultureInfo("fr");
    LoadApplication(new GettingStarted.App());
}
```

OrderID	EmployeeID	CustomerID	FirstName
CHARGER PLUS D'ARTICLES			
10002	9002	Linod	Kyle
10003	9003	Welli	Daniel
10004	9004	Frans	Danielle
10005	9005	Seves	Daniel
10006	Annuler la baisse		Fiona
10007	9007	Folig	Kyle
10008	9008	Welli	William
10009	9009	Welli	Kyle
10010	9010	Linod	Frank
10011	9011	Blonp	Frank
10012	9012	Welli	William

You can download the sample [here](#).

### AutomationId

SfDataGrid and SfDataPager support built-in [AutomationId](#) for all their inner elements. These **AutomationId** values allow the automation framework to find and interact with the inner elements when the test scripts are run. A unique **AutomationId** is maintained for each inner element by prefixing the control's **AutomationId** with the inner element's Id.

### DataGrid

The below table illustrates the predefined automation values set internally which can be used to identify the SfDataGrid elements.

Element	Value	Example
Header Row	"Row" +RowIndex	R0
Header Cell	"R" +RowIndex + "C" + ColumnIndex	R0C2
Row	"Row" +RowIndex	R4
Grid Cell	"R" +RowIndex + "C" + ColumnIndex	R4C2
Group Header	"Row" +RowIndex	R5
LoadMore View	"LOAD MORE ITEMS"	LOAD MORE ITEMS

The following screenshots illustrate the **AutomationId** values of grid cells, rows, and other inner elements of SfDataGrid.

Order Shipment Details				SyncfusionGrid Row0
Order Details		Freight Details		
Order ID ↑	LastName	Freight	IsClosed	SyncfusionGrid R2C1
Total Count of OrderId:1				
10001	Betts	24016	<input type="checkbox"/>	
Total Count of OrderId:1				
10002	Doran	24020	<input checked="" type="checkbox"/>	
Total Count of OrderId:1				SyncfusionGrid Row7
10003	Holmes	24024	<input type="checkbox"/>	
Total Count of OrderId:1				
10004	Jefferson	24028	<input checked="" type="checkbox"/>	SyncfusionGrid R10C2
Total Count of OrderId:1				
10005	Landry	24032	<input type="checkbox"/>	
Total Count of OrderId:1				
10006	Newberry	24036	<input checked="" type="checkbox"/>	SyncfusionGrid Row14
Total Count of OrderId:1				
10007	Perez	24040	<input type="checkbox"/>	

Order ID	LastName	Freight	IsClosed
10086	Mendoza	24204	<input checked="" type="checkbox"/>
10087	Owens	24208	<input type="checkbox"/>
10088	Rooney	24212	<input checked="" type="checkbox"/>
10089	Waddell	24178	<input type="checkbox"/>
10090	Thomas	24182	<input checked="" type="checkbox"/>
10091	Betts	24186	<input type="checkbox"/>
10092	Doran	24190	<input checked="" type="checkbox"/>
10093	Holmes	24194	<input type="checkbox"/>
10094	Jefferson	24198	<input checked="" type="checkbox"/>
10095	Landry	24202	<input type="checkbox"/>
10096	Newberry	24206	<input checked="" type="checkbox"/>
10097	Perez	24210	<input type="checkbox"/>
10098	Spencer	24214	<input checked="" type="checkbox"/>
10099	Vargas	24218	<input type="checkbox"/>
10100	LOAD MORE ITEMS		<input checked="" type="checkbox"/>

SyncfusionGrid LOAD MORE ITEMS

The following code snippet demonstrates how to set the AutomationId to data grid.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo,Mode=TwoWay}"
AutomationId="SyncfusionGrid"
AllowGroupExpandCollapse="True">
<sfgrid:SfDataGrid.GroupColumnDescriptions>
<sfgrid:GroupColumnDescription ColumnName="OrderID" />
</sfgrid:SfDataGrid.GroupColumnDescriptions>
<sfgrid:SfDataGrid.CaptionSummaryRow>
<sfgrid:GridGroupSummaryRow Title="Total Count of OrderId:{OrderID}"
ShowSummaryInRow="True">
<sfgrid:GridGroupSummaryRow.SummaryColumns>
<sfgrid:GridSummaryColumn Name="OrderID"
```

```

MappingName="OrderID"
Format="{Count}"
SummaryType="CountAggregate" />
</sfgrid:GridGroupSummaryRow.SummaryColumns>
</sfgrid:GridGroupSummaryRow>
</sfgrid:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```

**C#**

```

ViewModel viewModel = new ViewModel();
SfDataGrid dataGrid = new SfDataGrid();
dataGrid.ItemsSource = viewModel.OrderInfoCollection;
dataGrid.AutomationId = "SyncfusionGrid";
dataGrid.AllowGroupExpandCollapse = true;
this.sfGrid.GroupColumnDescriptions.Add(new GroupColumnDescription()
{
    ColumnName = "OrderID",
});
GridGroupSummaryRow summaryRow = new GridGroupSummaryRow();
summaryRow.Title = "Total Count of OrderId:{OrderID}";
summaryRow.ShowSummaryInRow = true;
summaryRow.SummaryColumns.Add(new GridSummaryColumn()
{
    Name = "OrderID",
    MappingName = "OrderID",
    Format = "{Count}",
    SummaryType = SummaryType.CountAggregate
});
sfGrid.CaptionSummaryRow = summaryRow;

```

Refer to the following code snippet to access the inner elements of data grid from the automation script.

**C#**

```

[Test]
[Description("SfDataGrid Automation Id")]
public void SfDataGrid_AutomationId()
{
    // To enter edit mode for the grid cell at fourth row and second column.
    App.DoubleTap("SyncfusionGrid R4C1");
    // To tap group expand and collapse icon
    App.Tap("SyncfusionGrid Row5");
    // To apply sorting
    App.Tap("SyncfusionGrid R2C1");
    // To click LoadMoreView for loading more items
    App.Tap("SyncfusionGrid LOAD MORE ITEMS");
}

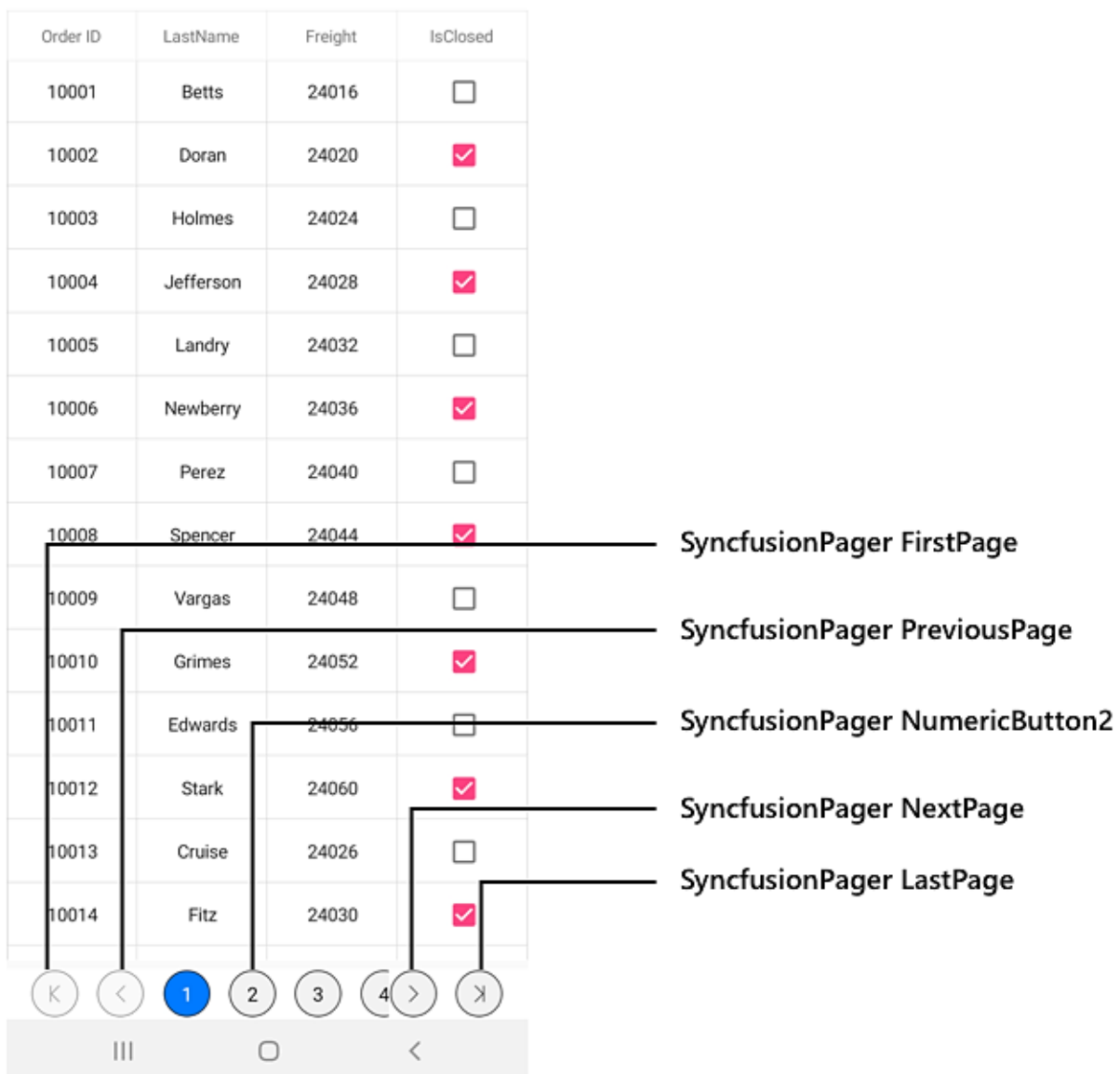
```

**DataPager**

The below table illustrates the predefined automation values set internally which can be used to identify the SfDataPager elements.

Element	Value
First page button	"FirstPage"
Previous page button	"PreviousPage"
Numeric buttons	"NumericButton" + NumericButtonIndex Example : NumericButton3
Next page button	"NextPage"
Last page button	"LastPage"

The following screenshot illustrates the **AutomationId** values of the pager buttons in SfDataPager.



The following code snippet demonstrates how to set AutomationId to dataPager.

### XML

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>
  <sfPager:SfDataPager x:Name="dataPager"
    Grid.Row="1"
    PageSize="10"
    HeightRequest="50"
    NumericButtonCount="20"
    Source="{Binding OrdersInfo}"
    AutomationId="SyncfusionPager">
  </sfPager:SfDataPager>
  <sfgrid:SfDataGrid x:Name="dataGrid"
    Grid.Row="0"
    ItemsSource="{Binding PagedSource, Source={x:Reference dataPager}}" >
  </sfgrid:SfDataGrid>
</Grid>
```

### C#

```
SfDataGrid sfGrid = new SfDataGrid();
SfDataPager sfPager = new SfDataPager();
ViewModel viewModel = new ViewModel();
sfPager.PageSize = 15;
sfPager.Source = viewModel.Info;
sfGrid.ItemsSource = sfPager.PagedSource;
sfPager.AutomationId = "SyncfusionPager";
Grid gridLayout = new Grid();
gridLayout.HorizontalOptions = LayoutOptions.FillAndExpand;
gridLayout.RowDefinitions = new RowDefinitionCollection
{
    new RowDefinition { },
    new RowDefinition { Height = 50 },
};
gridLayout.Children.Add(sfGrid, 0, 0);
gridLayout.Children.Add(sfPager, 0, 1);
this.Content = gridLayout;
```

Refer to the following code snippet to access the inner elements of data pager from automation script.

### C#

```
[Test]
[Description("SfDataPager Automation Id")]
public void SfDataPager_AutomationId()
{
    // To tap the first page numeric button
    App.Tap("SyncfusionPager FirstPage");
    // To tap the previous page numeric button
    App.Tap("SyncfusionPager PreviousPage");
    // To tap the next page numeric button
```



```

App.Tap ("SyncfusionPager NextPage");
// To tap the next last page numeric button
App.Tap ("SyncfusionPager LastPage");
// To tap the second numeric button
App.Tap ("SyncfusionPager NumericButton2");
}

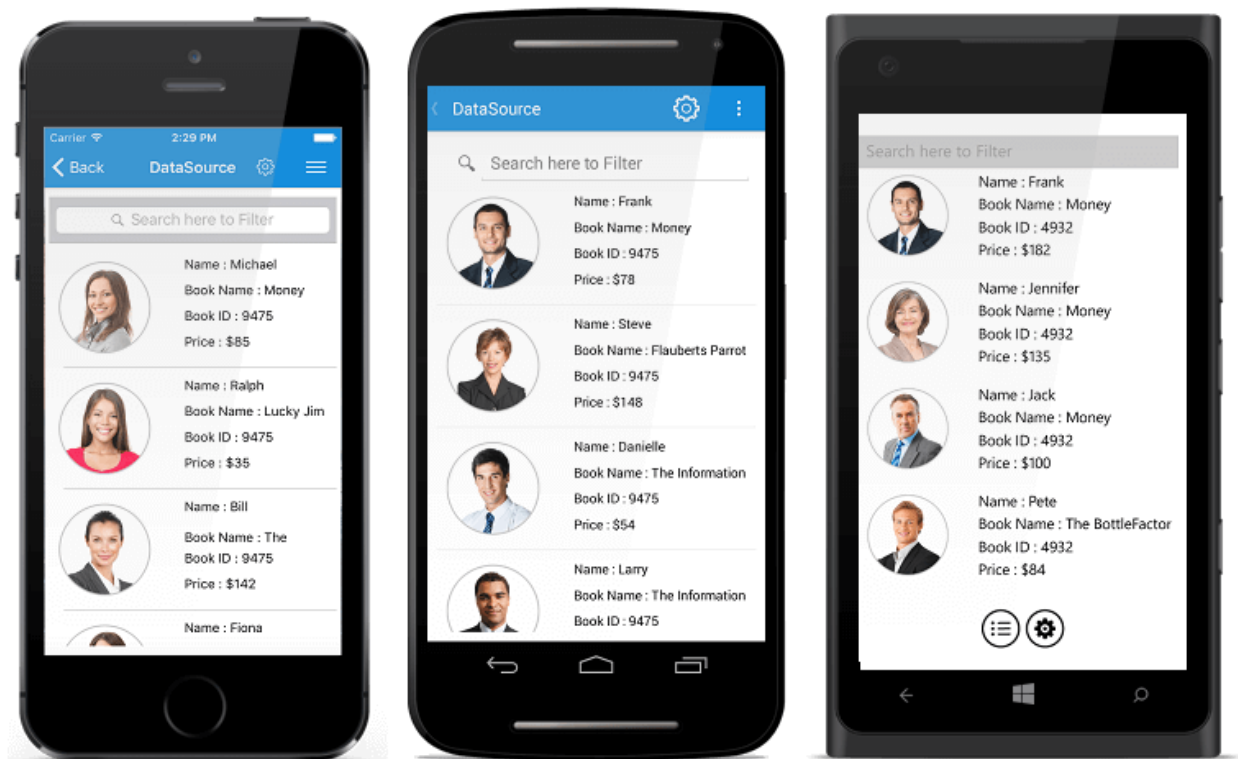
```

## DataSource

### DataSource

#### Overview

DataSource is a non UI component that consumes raw data and processes data operations such as sorting, filtering and grouping saving developers' time and efforts in building the functionality themselves. We can apply DataSource to any data bound control which can be further processed using the bound DataSource.



#### Getting started

##### Assembly deployment

After installing Essential Studio for Xamarin, you can find all the required assemblies in the following installation folders,

{Syncfusion Essential Studio Installed location}\Essential Studio\{Syncfusion release version}\lib

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

**Note:** Assemblies can be found in an unzipped package location in Mac.

### Adding DataSource reference

You can add DataSource reference using one of the following methods:

#### Method 1: Adding DataSource reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org). To add DataSource to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.DataSource](#), and then install it.

![Adding DataSource reference from NuGet](DataSource-GettingStarted\_images/Adding DataSource reference.png)

---

**Note:** Install the same version of DataSource NuGet in all the projects.

---

#### Method 2: Adding DataSource reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the DataSource control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding DataSource assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.DataSource.Portable.dll Syncfusion.Licensing.dll
-----	--

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Creating your first DataSource in Xamarin.Forms

- This is how the final output will look like on iOS, Android and Windows Phone devices. You can also download the entire source code of this demo from [here](#).



- Create a new blank ([Xamarin.Forms.NET Standard](#)) application in Xamarin Studio or Visual Studio for Xamarin.Forms.
- Now, create a simple data source as shown in the following code example. Add the following code example in a newly created class file and save it as **Contacts.cs** file.

### C#

```
public class Contacts : INotifyPropertyChanged
{
    private string contactName;
    public Contacts(string name)
    {
        contactName = name;
    }
    public string ContactName
    {
        get { return contactName; }
        set
        {
            if (contactName != value)
            {
                contactName = value;
                this.RaisedOnPropertyChanged("ContactName");
            }
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    public void RaisedOnPropertyChanged(string _PropertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(_PropertyName));
        }
    }
}
```

- Add the following code example in a newly created class file and save it as **ContactList.cs** file.

**C#**

```
public class ContactsList : ObservableCollection<Contacts>,
INotifyPropertyChanged
{
    public ContactsList()
    {
        foreach (var customerName in CustomerNames)
        {
            var contact = new Contacts(customerName);
            this.Add(contact);
        }
    }
    string[] CustomerNames = new string[] {
        "Kyle",
        "Gina",
        "Irene",
        "Katie",
        "Michael",
        "Oscar",
        "Ralph",
        "Torrey",
        "William",
        "Bill",
        "Daniel",
        "Frank",
        "Brenda",
        "Danielle",
        "Fiona",
        "Howard",
        "Jack",
        "Larry",
    };
}
```

- You can set the source for the DataSource by using the [DataSource.Source](#) property. You can bind the [DataSource.DisplayItems](#) as `ItemsSource` for any data bound control.

**C#**

```
public App()
{
    DataSource dataSource = new DataSource();
    dataSource.Source = new ContactsList();
}
```

**Sorting**

DataSource allows sorting the bound source by using the [DataSource.SortDescriptors](#) property. You can create a [SortDescriptor](#) for the property to be sorted and add it in the `DataSource.SortDescriptors` collection.

The **SortDescriptor** object holds following three properties:

- **PropertyName**: Specifies name of the sorted property.
- **Direction**: Specifies an object of type [ListSortDirection](#) that defines the sorting direction.
- **Comparer**: Specifies a comparer to be applied when sorting take place.

The following code illustrates this.

#### C#

```
dataSource.SortDescriptors.Add(new SortDescriptor("ContactName"));
```

### Grouping

DataSource allows sorting the bound source by using the [DataSource.GroupDescriptors](#) property. You can create a [GroupDescriptor](#) for the property to be grouped and add it in the **DataSource.GroupDescriptors** collection.

**GroupDescriptor** object holds following two properties:

- **PropertyName**: Specifies name of the grouped property.
- **KeySelector**: Sets the [KeySelector](#) for grouping.
- **Comparer**: Comparer to be applied in when sorting take place

The following code example illustrates this without **KeySelector**.

#### C#

```
dataSource.GroupDescriptors.Add(new GroupDescriptor("ContactName"));
```

The following code example illustrates this with **KeySelector**.

#### C#

```
dataSource.GroupDescriptors.Add(new GroupDescriptor()  
{  
    PropertyName = "ContactName",  
    KeySelector = (object obj1) =>  
    {  
        var item = (obj1 as Contacts);  
        return item.ContactName[0].ToString();  
    }  
});
```

### Binding DataSource to a ListView

Please refer the below code example that illustrates binding the created DataSource to a ListView control.

#### C#

```
public App()  
{  
    DataSource dataSource = new DataSource();  
    dataSource.Source = new ContactsList();  
}
```

```

dataSource.SortDescriptors.Add(new SortDescriptor("ContactName"));
dataSource.GroupDescriptors.Add(new GroupDescriptor()
{
    PropertyName = "ContactName",
    KeySelector = (object obj1) =>
    {
        var item = (obj1 as Contacts);
        return item.ContactName[0].ToString();
    }
});
StackLayout stack = new StackLayout();
stack.Children.Add(new Label()
{
    TextColor = Color.Black,
    FontSize = 14,
    HeightRequest = 50,
    Text = "Contact List",
    HorizontalTextAlignment = TextAlignment.Center,
    VerticalTextAlignment = TextAlignment.Center,
    BackgroundColor = Color.Gray
});
listView = new ListView();
listView.ItemTemplate = new DataTemplate(() =>
{
    var label = new Label()
    {
        TextColor = Color.Black,
        FontSize = 12,
        VerticalTextAlignment = TextAlignment.Center,
        BackgroundColor = Color.White,
    };
    label.SetBinding(Label.TextProperty, new Binding("ContactName"));
    var viewCell = new ViewCell() { View = label };
    viewCell.BindingContextChanged += ViewCell_BindingContextChanged;
    return viewCell;
});
listView.ItemsSource = dataSource.DisplayItems;
stack.Children.Add(listView);
MainPage = new ContentPage { Content = stack };
Device.OnPlatform(iOS:() => MainPage.Padding = new Thickness(0, 20, 0, 0));
}
//Will be executed only in grouping case. Existing view in view cell will be
replaced by label.
private void ViewCell_BindingContextChanged(object sender, EventArgs e)
{
    var viewCell = sender as ViewCell;
    if (viewCell.BindingContext is GroupResult)
    {
        var label = new Label()
        {
            TextColor = Color.Black,
            FontSize = 14,
            HeightRequest = 50,
            HorizontalTextAlignment = TextAlignment.Center,
            VerticalTextAlignment = TextAlignment.Center,
            BackgroundColor = Color.Gray
        };
    }
}

```

```
label.SetBinding(Label.TextProperty, new Binding("Key"));  
viewCell.View = label;  
}  
}
```

### Defining the LiveDataUpdateMode

**DataSource** listens manipulation operations such as add, delete, and data update (property change) at runtime and responds based on the [LiveDataUpdateMode](#) property. Default value is **LivDataUpdateMode.Default**. The **LivDataUpdateMode** property holds the following two enumeration values:

- Default: Refreshes an item in view when the underlying property is changed.
- AllowDataShaping: Refreshes an item in view and also updates the collection when the underlying property is changed.

### C#

```
private void UpdateData_Clicked(object sender, EventArgs e)  
{  
    DataSource.LiveDataUpdateMode = LiveDataUpdateMode.AllowDataShaping;  
    ViewModel.Items[0].Title = "DataSourceItem_0";  
}
```

### Defining the source data type

When the [Source](#) property binds with different types of underlying collection derived from the same type, set the [SourceType](#) as base type for all different types.

### C#

```
DataSource.SourceType = typeof(Contacts);
```

## SfDateTimeRangeNavigator

### Overview

The Date Time Range Navigator control provides an intuitive interface for selecting a smaller date range from a larger collection. It is commonly used in financial dashboards to filter the date range for which the data needs to be visualized.

### Key features

Main capabilities of the control are listed below

- Major scale displays a date time scale that is one or more level higher than the minor scale.
- Visualize data with the available built-in chart types.
- Intuitive user interface to select a particular time range.



## Getting started

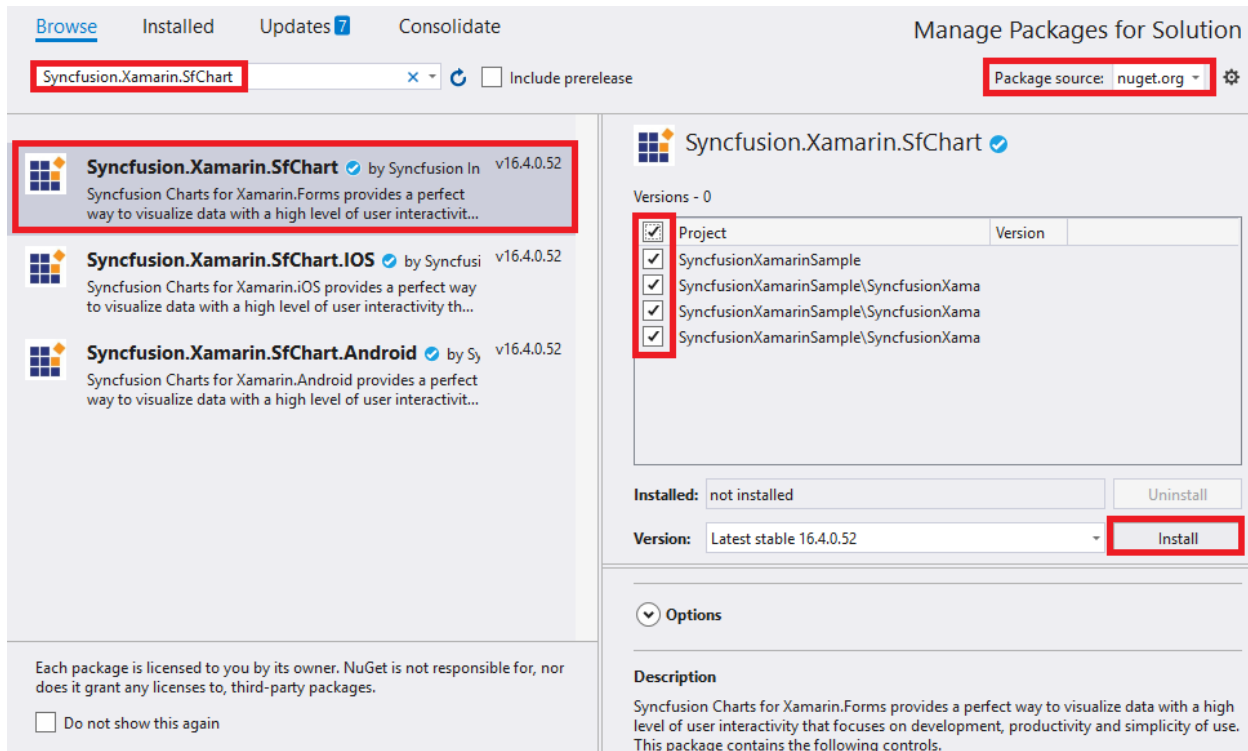
This section walks you through the steps required to add [SfDateTimeRangeNavigator](#) and populate it with data, and also explains how to respond to range selection performed in the control.

### Adding SfDateTimeRangeNavigator reference

You can add SfDateTimeRangeNavigator reference in one of the following methods:

#### Method 1: Adding SfDateTimeRangeNavigator reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfChart). To add SfDateTimeRangeNavigator to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfChart](#), and then install it.



**Note:** Install the same version of the SfChart NuGet in all the projects.

#### Method 2: Adding SfDateTimeRangeNavigator reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfDateTimeRangeNavigator control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfDateTimeRangeNavigator assemblies manually from the installed location



If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location : {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfChart.XForms.Android.dll Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfChart.XForms.iOS.dll Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfChart.UWP.dll Syncfusion.SfChart.XForms.UWP.dll Syncfusion.SfChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the application on each platform with range navigator

To use the range navigator inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, below steps are not needed.

#### iOS

To launch the range navigator in iOS, call the `SfRangeNavigatorRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms framework initialization and before the `LoadApplication` method is called as demonstrated in the following code sample:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
```

```
{  
...  
global::Xamarin.Forms.Forms.Init();  
Syncfusion.RangeNavigator.XForms.iOS.SfRangeNavigatorRenderer.Init();  
LoadApplication(new App());  
...  
}
```

### Universal Windows Platform (UWP)

To launch the range navigator in UWP, call the `SfRangeNavigatorRenderer.Init()` method in the constructor of `MainPage` before the `LoadApplication` method is called as demonstrated in the following code sample.

#### C#

```
public MainPage()  
{  
...  
Syncfusion.RangeNavigator.XForms.UWP.SfRangeNavigatorRenderer.Init();  
LoadApplication (new App ());  
...  
}
```

In addition to the above configurations, you need to initialize the control assemblies in `App.xaml.cs` in UWP project as shown in the below code snippets. This is required to deploy application with range navigator in **Release** mode in UWP platform.

#### C#

```
// In App.xaml.cs  
protected override void OnLaunched(LaunchActivatedEventArgs e)  
{  
...  
if (rootFrame == null)  
{  
List<Assembly> assembliesToInclude = new List<Assembly>();  
assembliesToInclude.Add(typeof(Syncfusion.RangeNavigator.XForms.UWP.  
SfRangeNavigatorRenderer).GetTypeInfo().Assembly);  
Xamarin.Forms.Forms.Init(e, assembliesToInclude);  
}  
...  
}
```

### Android

The Android platform does not require any additional configuration to render the range navigator.

### Adding and configuring SfDateTimeRangeNavigator

First, let us initialize the control with major and minor date time scales by specifying the minimum and maximum date to be visualized in the control using [Minimum](#) and [Maximum](#) properties.

Following code example illustrates this,

#### XML

```
<rangnavigator:SfDateTimeRangeNavigator Minimum="2015,01,01"
Maximum="2016,01,01"/>
```

**C#**

```
[C#]
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator();
rangeNavigator.Minimum = new DateTime(2015, 01, 01);
rangeNavigator.Maximum = new DateTime(2016, 01, 01);
```



**Note:** If you don't specify [Minimum](#) and [Maximum](#) properties, minimum and maximum dates will be chosen automatically based on the provided data using [ItemsSource](#) property, which is explained in the next step in this section.

Next, create a data model representing the list of sample data.

**C#**

```
[C#]
public class DataModel
{
    public ObservableCollection<Model> DateTimeData;
    DataModel()
    {
        DateTimeData = new ObservableCollection<Model>()
        {
            new Model (new DateTime(2015, 01, 01), 14),
            new Model (new DateTime(2015, 02, 01), 54),
            new Model (new DateTime(2015, 03, 01), 23),
            new Model (new DateTime(2015, 04, 01), 53),
            new Model (new DateTime(2015, 05, 01), 25),
            new Model (new DateTime(2015, 06, 01), 32),
            new Model (new DateTime(2015, 07, 01), 78),
            new Model (new DateTime(2015, 08, 01), 100),
            new Model (new DateTime(2015, 09, 01), 55),
            new Model (new DateTime(2015, 10, 01), 38),
            new Model (new DateTime(2015, 11, 01), 27),
            new Model (new DateTime(2015, 12, 01), 56),
            new Model (new DateTime(2016, 01, 01), 33)
        };
    }
}

public class Model
{
    public DateTime Date { get; set; }
    public double Value { get; set; }
    public Model(DateTime dateTime, double value)
    {
        Date = dateTime;
    }
}
```

```
Value = value;
}
}
```

Then, let us populate the chart, which is displayed inside the [SfDateTimeRangeNavigator](#), by setting the above data using [ItemsSource](#) property. And then specify the property names which contain the x and y values in the model using [XBindingPath](#) and [YBindingPath](#) properties.

**Note:** By default, data is visualized using line series. You can change the chart type or add more series by accessing the SfChart instance using [SfDateTimeRangeNavigator.Content](#) property.

### XML

```
<rangnavigator:SfDateTimeRangeNavigator ItemsSource="{Binding
DateTimeData}" XBindingPath="Date" YBindingPath="Value"/>
```

### C#

```
[C#]
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()
{
    rangeNavigator.ItemsSource = dataModel.DateTimeData,
    rangeNavigator.XBindingPath = "Date",
    rangeNavigator.YBindingPath = "Value"
};
```



### Handle range selection

In real time, other controls like chart, grid etc., are updated in response to the range selection performed in SfDateTimeRangeNavigator. You can handle the selection using [RangeChanged](#) event and update other controls based on the selected date time or perform some other tasks using the selected data.

**Note:** You can get the selected start and end date using [ViewRangeStart](#) and [ViewRangeEnd](#) properties or get the collection of selected data using [SelectedData](#) property.

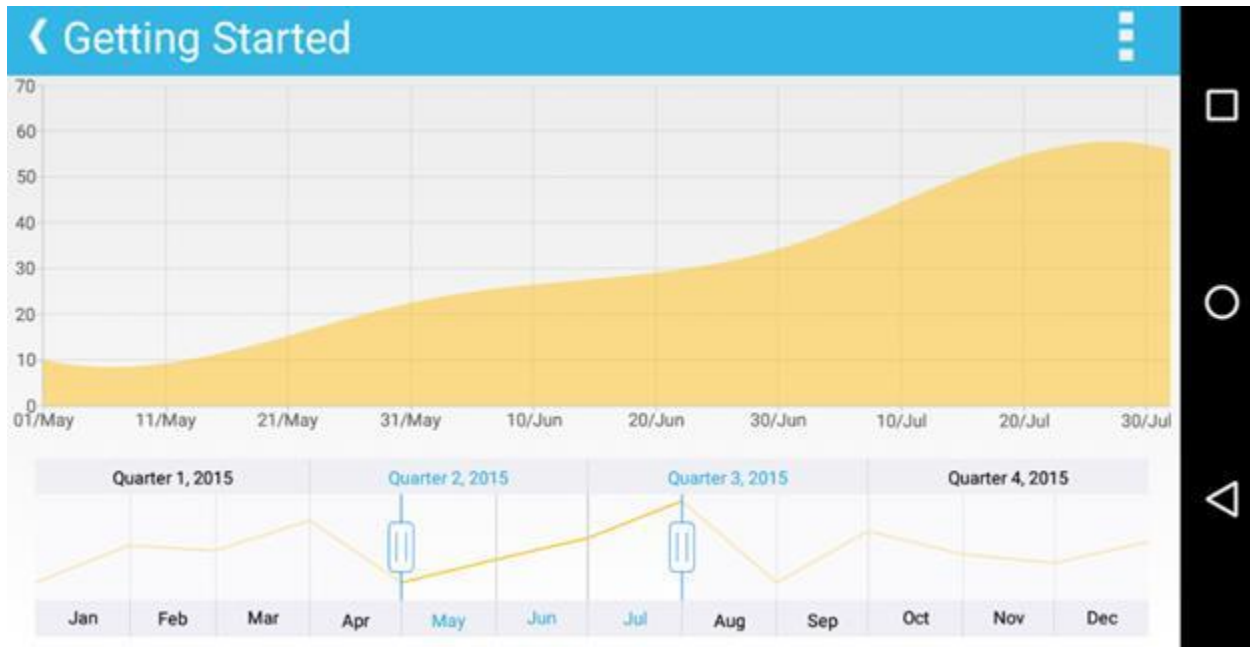
Following code example illustrates how to handle range selection and update chart's date time axis range,

### C#

```
[C#]
rangeNavigator.RangeChanged += rangeNavigator_RangeChanged;
private void rangeNavigator_RangeChanged(object sender,
RangeChangedEventArgs e)
{
    //Updating chart's date time range
    dateTimeAxis.Minimum = e.ViewRangeStartDate;
    dateTimeAxis.Maximum = e.ViewRangeEndDate;
}
```

```
}
```

You can find the complete getting started sample from this [link](#).



## Content

[SfDateTimeRangeNavigator](#) allows you to set [SfChart](#) as its content, explicitly, using [Content](#) property. However, if you provide data source using [ItemsSource](#) property, the Chart with line series will be created for the provided [ItemsSource](#) and will be set as the content of range navigator internally, by default. But, if you configure the range navigator using [Minimum](#) and [Maximum](#) properties, you have to manually configure the Chart with data source.

**Note:** Though the [Content](#) property's data type is View and it can accept any View as its value, but currently [SfDateTimeRangeNavigator](#) can accept only [SfChart](#) as its content.

The following code snippet shows how to configure the range navigator using [ItemsSource](#) property.

### XML

```
Namespace:
xmlns:rangenavigator="clr-
namespace:Syncfusion.RangeNavigator.XForms;assembly=Syncfusion.SfChart.XForms
s"
...
<rangenavigator:SfDateTimeRangeNavigator ItemsSource="{Binding
DateTimeRangeData}" XBindingPath="XValue" YBindingPath="YValue"/>
```

### C#

```
Namespace:
using Syncfusion.RangeNavigator.XForms;
...
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator();
ViewModel viewModel = new ViewModel();
```

```
rangeNavigator.ItemsSource = viewModel.DateTimeRangeData;
rangeNavigator.XBindingPath = "XValue";
rangeNavigator.YBindingPath = "YValue";
```

**C#**

```
[C#]
public class ViewModel
{
    public ObservableCollection<ChartDataPoint> DateTimeRangeData { get; set; }
    public ViewModel()
    {
        DateTimeRangeData = new ObservableCollection<ChartDataPoint>
        {
            new ChartDataPoint(new DateTime(2015, 01, 1), 14),
            new ChartDataPoint(new DateTime(2015, 02, 1), 54),
            new ChartDataPoint(new DateTime(2015, 03, 1), 23),
            new ChartDataPoint(new DateTime(2015, 04, 1), 53),
            new ChartDataPoint(new DateTime(2015, 05, 1), 25),
            new ChartDataPoint(new DateTime(2015, 06, 1), 32),
            new ChartDataPoint(new DateTime(2015, 07, 1), 78),
            new ChartDataPoint(new DateTime(2015, 08, 1), 100),
            new ChartDataPoint(new DateTime(2015, 09, 1), 55),
            new ChartDataPoint(new DateTime(2015, 10, 1), 38),
            new ChartDataPoint(new DateTime(2015, 11, 1), 27),
            new ChartDataPoint(new DateTime(2015, 12, 1), 56),
            new ChartDataPoint(new DateTime(2015, 12, 31), 35),
        };
    }
}
```

The following code snippet shows how to configure the range navigator using [Minimum](#) and [Maximum](#) properties.

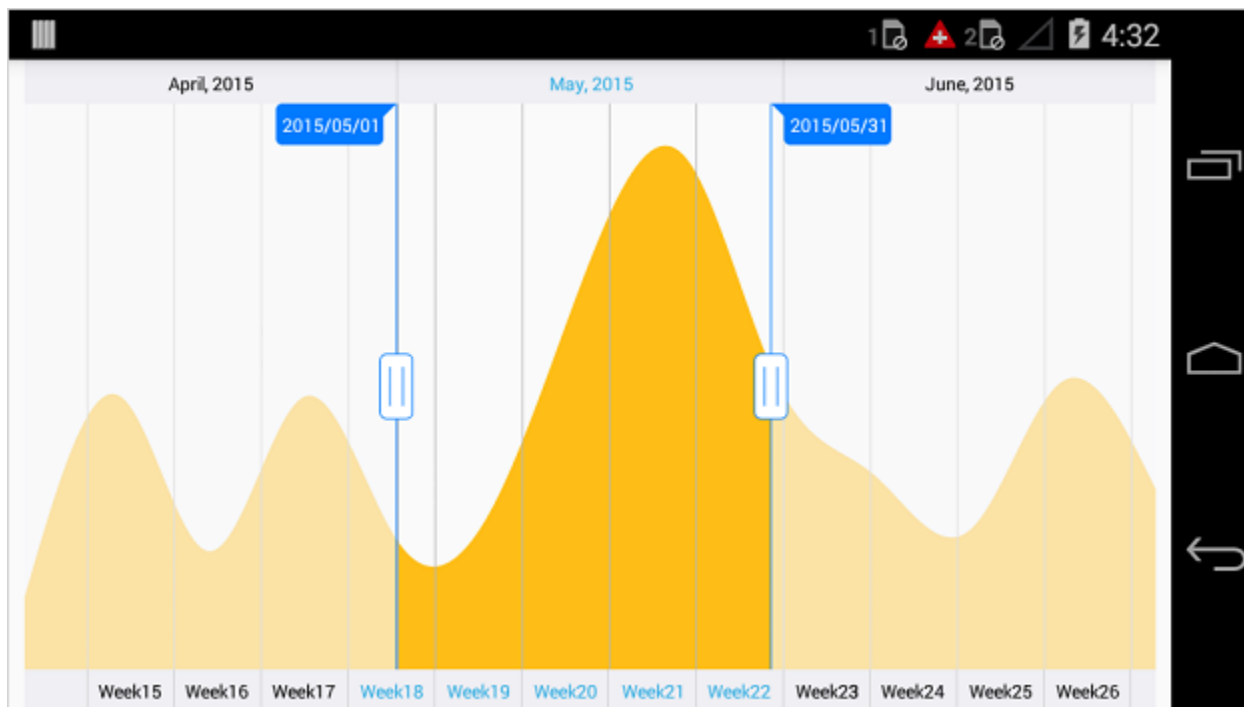
**XML**

```
Namespace:
xmlns:rangenavigator="clr-
namespace:Syncfusion.RangeNavigator.XForms;assembly=Syncfusion.SfChart.XForm
s"
xmlns:chart="clr-
namespace:Syncfusion.SfChart.XForms;assembly=Syncfusion.SfChart.XForms"
...
<rangenavigator:SfDateTimeRangeNavigator ViewRangeStart="5/1/2015"
ViewRangeEnd="5/30/2015" Minimum="4/1/2015" Maximum="6/30/2015">
<rangenavigator:SfDateTimeRangeNavigator.Content>
<chart:SfChart>
. . .
</chart:SfChart>
</rangenavigator:SfDateTimeRangeNavigator.Content>
```

**C#**

```
Namespace:
using Syncfusion.RangeNavigator.XForms;
```

```
using Syncfusion.SfChart.XForms;
...
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator();
rangeNavigator.Minimum = new DateTime(2015, 4, 1);
rangeNavigator.Maximum = new DateTime(2015, 6, 30);
rangeNavigator.ViewRangeStart = new DateTime(2015, 5, 1);
rangeNavigator.ViewRangeEnd = new DateTime(2015, 5, 31);
SfChart chart = new SfChart();
...
rangeNavigator.Content = chart;
```



### Selecting Range

The left and right thumb of [SfDateTimeRangeNavigator](#) are used to indicate the selected range in the large collection of data. Following are the ways you can select a range.

- By dragging the thumbs.
- By tapping on the minor and major labels.
- By setting the [ViewRangeStart](#) and [ViewRangeEnd](#) properties.

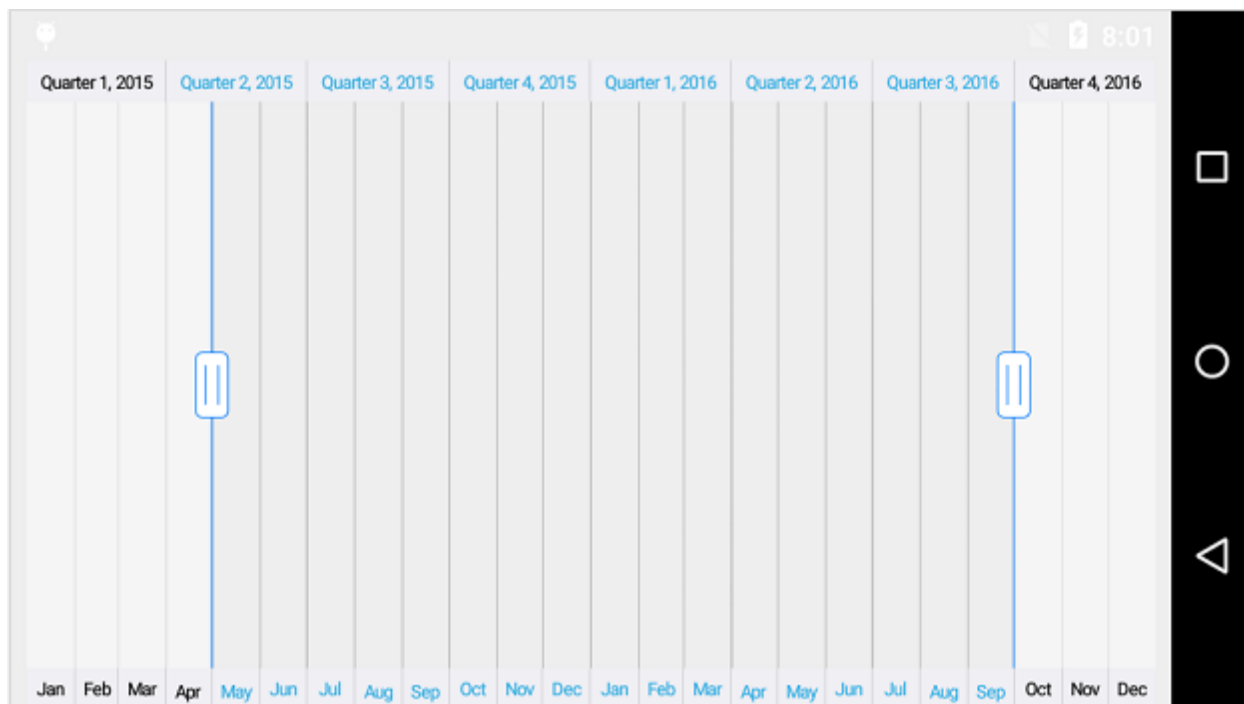
Following code example shows how to configure the selected range using [ViewRangeStart](#) and [ViewRangeEnd](#) properties of [SfDateTimeRangeNavigator](#).

#### XML

```
<rangnavigator:SfDateTimeRangeNavigator x:Name ="dateTime"
Minimum="1/1/2015"
Maximum="1/1/2017" ViewRangeStart="5/1/2015" ViewRangeEnd="10/1/2016"/>
```

#### C#

```
SfDateTimeRangeNavigator dateTime = new SfDateTimeRangeNavigator();
dateTime.ViewRangeStart = new DateTime(2015, 5, 1);
dateTime.ViewRangeEnd = new DateTime(2016, 10, 1);
```



### Selected Data

The [SelectedData](#) property of [SfDateTimeRangeNavigator](#) used to get the underlying data collection of the selected range. You can directly bind this property to other data visualization control to visualize the selected range of data.

### Overlay Color

The [OverlayColor](#) property of [SfDateTimeRangeNavigator](#) is used to customize the color of unselected area in date-time range navigator.

### Deferred Update

[RangeChanged](#) event will be fired whenever you drag the thumbs. If you are doing some long running tasks in this event handler, then dragging the thumbs will not be smooth. You can delay this event by enabling [EnableDeferredUpdate](#) property. If this property is set to true, the [RangeChanged](#) event will get fired only when you stop dragging or if the thumb is held for more than 500 milliseconds. If it is false, the range will be updated for every movement of the thumb. However, It is true by default.

The delay of update is 500 milliseconds by default. However, it can be changed using [DeferredUpdateDelay](#) property of [SfDateTimeRangeNavigator](#).

### XML

```
<rangnavigator:SfDateTimeRangeNavigator x:Name ="dateTime"
Minimum="1/1/2015"
Maximum="1/1/2017" EnableDeferredUpdate="False" DeferredUpdateDelay="600" />
```

### C#



```
SfDateTimeRangeNavigator dateTime = new SfDateTimeRangeNavigator();
dateTime.EnableDeferredUpdate = false;
dateTime.DeferredUpdateDelay = 600;
```

### RangeChanged Event

This event is triggered when the selected range of the [SfDateTimeRangeNavigator](#) is changed. The argument contains the following information.

- [ViewRangeStartDate](#) – used to get the start date of the selected range.
- [ViewRangeEndDate](#) – used to get the end date of the selected range.

### Thumb

The [LeftThumbStyle](#) and [RightThumbStyle](#) properties are used to configure the left and right thumb of the [SfDateTimeRangeNavigator](#). Following properties are available in thumb style to configure left and right thumb individually.

- [BorderColor](#) – used to change the stroke color of the thumb.
- [BackgroundColor](#) – used to change the background color of the thumb.
- [BorderWidth](#) – used to change the stroke width of the thumb.
- [Width](#) – used to change the width of the thumb.
- [Height](#) – used to change the height of the thumb.
- [LineColor](#) – used to change the line color of the thumb.
- [LineWidth](#) – used to change the line width of the thumb.
- [LineDashArray](#) – used to change the dash array of the thumb line.

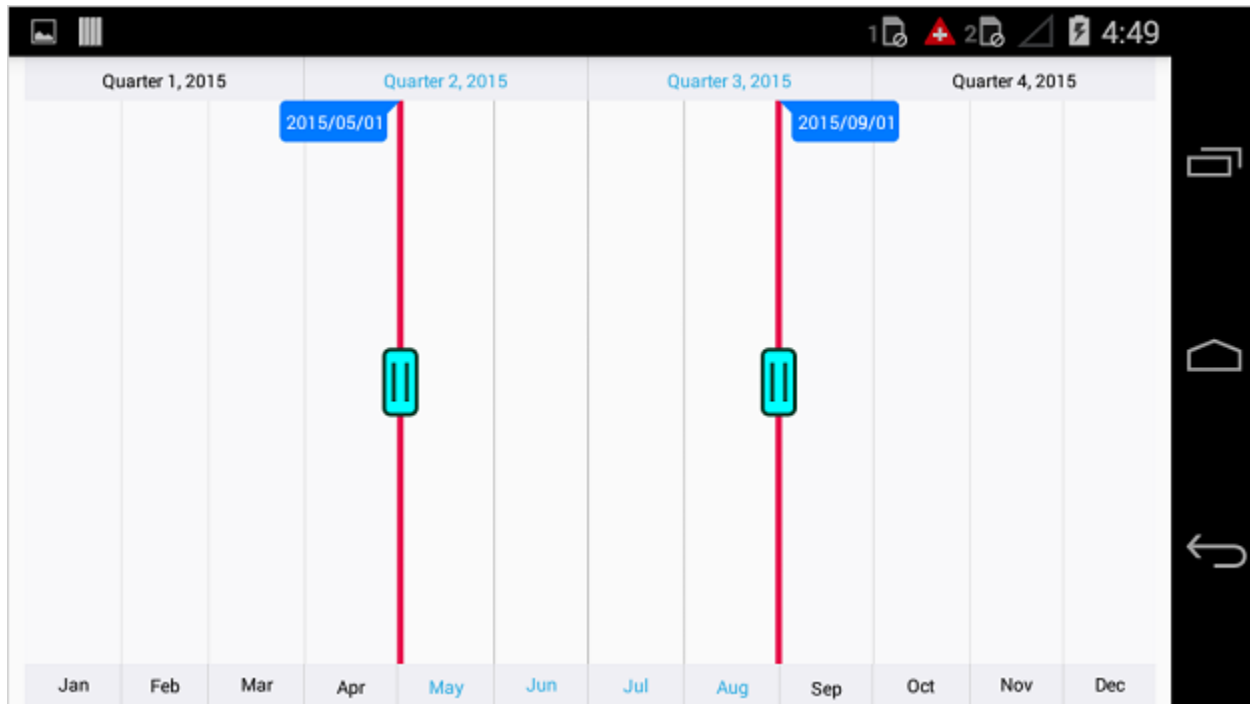
### XML

```
<rangnavigator:SfDateTimeRangeNavigator HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" Minimum="1/1/2015"
Maximum="12/31/2015">
  <rangnavigator:SfDateTimeRangeNavigator.LeftThumbStyle>
    <rangnavigator:ThumbStyle BorderColor="#083928" BackgroundColor="Aqua"
    BorderWidth="3" LineColor="#E70E49"
    LineWidth="5"/>
  </rangnavigator:SfDateTimeRangeNavigator.LeftThumbStyle>
  <rangnavigator:SfDateTimeRangeNavigator.RightThumbStyle>
    <rangnavigator:ThumbStyle BorderColor="#083928" BackgroundColor="Aqua"
    BorderWidth="3" LineColor="#E70E49"
    LineWidth="5"/>
  </rangnavigator:SfDateTimeRangeNavigator.RightThumbStyle>
</rangnavigator:SfDateTimeRangeNavigator>
```

### C#

```
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator();
...
rangeNavigator.LeftThumbStyle.BackgroundColor = Color.Aqua;
rangeNavigator.LeftThumbStyle.BorderColor = Color.FromHex("#083928");
rangeNavigator.LeftThumbStyle.BorderWidth = 3;
rangeNavigator.LeftThumbStyle.LineColor = Color.FromHex("#E70E49");
rangeNavigator.LeftThumbStyle.LineWidth = 5;
```

```
rangeNavigator.RightThumbStyle.BackgroundColor = Color.Aqua;
rangeNavigator.RightThumbStyle.BorderColor = Color.FromHex("#083928");
rangeNavigator.RightThumbStyle.BorderWidth = 3;
rangeNavigator.RightThumbStyle.LineColor = Color.FromHex("#E70E49");
rangeNavigator.RightThumbStyle.LineWidth = 5;
```



### Grid Lines

The [MinorScaleStyle](#) and [MajorScaleStyle](#) properties of [SfDateTimeRangeNavigator](#) used to customize the minor and major grid lines. Following properties are available in each scale style to configure the grid lines.

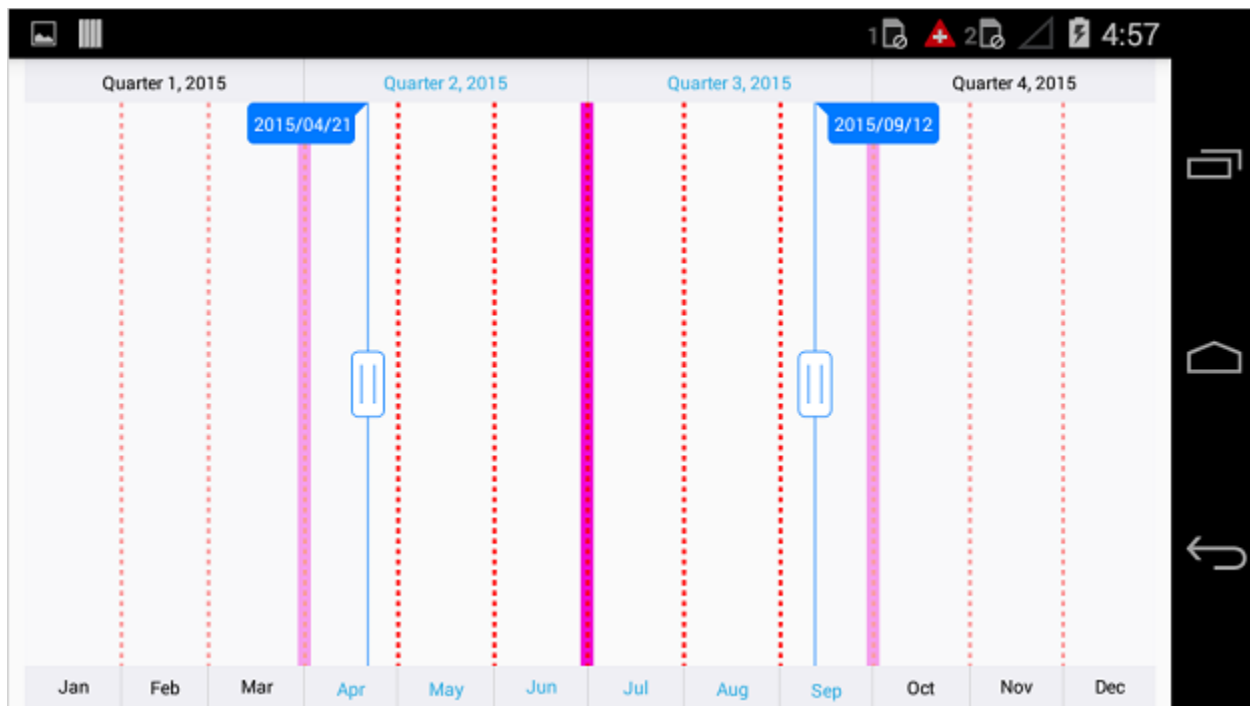
- [ShowGridLines](#) – used to set the visibility of grid lines.
- [GridLineWidth](#) – used to set the width for grid lines.
- [GridLineColor](#) – used to set the color for grid lines.
- [GridLineDashes](#) – used to set dashes for grid lines.

### XML

```
<rangnavigator:SfDateTimeRangeNavigator HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" Minimum="1/5/2015"
Maximum="11/7/2015">
  <rangnavigator:SfDateTimeRangeNavigator.MajorScaleStyle>
    <rangnavigator:ScaleStyle GridLineColor="#F109D7" GridLineWidth="10"
ShowGridLines="True"/>
  </rangnavigator:SfDateTimeRangeNavigator.MajorScaleStyle >
  <rangnavigator:SfDateTimeRangeNavigator.MinorScaleStyle>
    <rangnavigator:ScaleStyle GridLineColor="Red" GridLineWidth="3"
ShowGridLines="True"/>
  </rangnavigator:SfDateTimeRangeNavigator.MinorScaleStyle >
</rangnavigator:SfDateTimeRangeNavigator>
```

**C#**

```
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator();
...
rangeNavigator.MajorScaleStyle.ShowGridLines = true;
rangeNavigator.MajorScaleStyle.GridLineColor = Color.FromHex("#F109D7");
rangeNavigator.MajorScaleStyle.GridLineWidth = 10;
rangeNavigator.MinorScaleStyle.ShowGridLines = true;
rangeNavigator.MinorScaleStyle.GridLineColor = Color.Red;
rangeNavigator.MinorScaleStyle.GridLineWidth = 3;
rangeNavigator.MinorScaleStyle.GridLineDashArray = new double[2] { 4, 4 };
```

**Tooltip**

The tooltip is used to show the selected range start and end value of the [SfDateTimeRangeNavigator](#).

**Tooltip Visibility**

The [EnableTooltip](#) property of [SfDateTimeRangeNavigator](#) is used to control the visibility of the left and right tooltip. It is true by default.

**XML**

```
<rangnavigator:SfDateTimeRangeNavigator Minimum="1/1/2015"
Maximum="1/1/2016" ViewRangeStart="5/1/2015"
ViewRangeEnd="9/1/2016" EnableTooltip="False"/>
```

**C#**

```
SfDateTimeRangeNavigator dateTimeRangeNavigator = new
SfDateTimeRangeNavigator();
dateTimeRangeNavigator.EnableTooltip = false;
```



### Tooltip Format

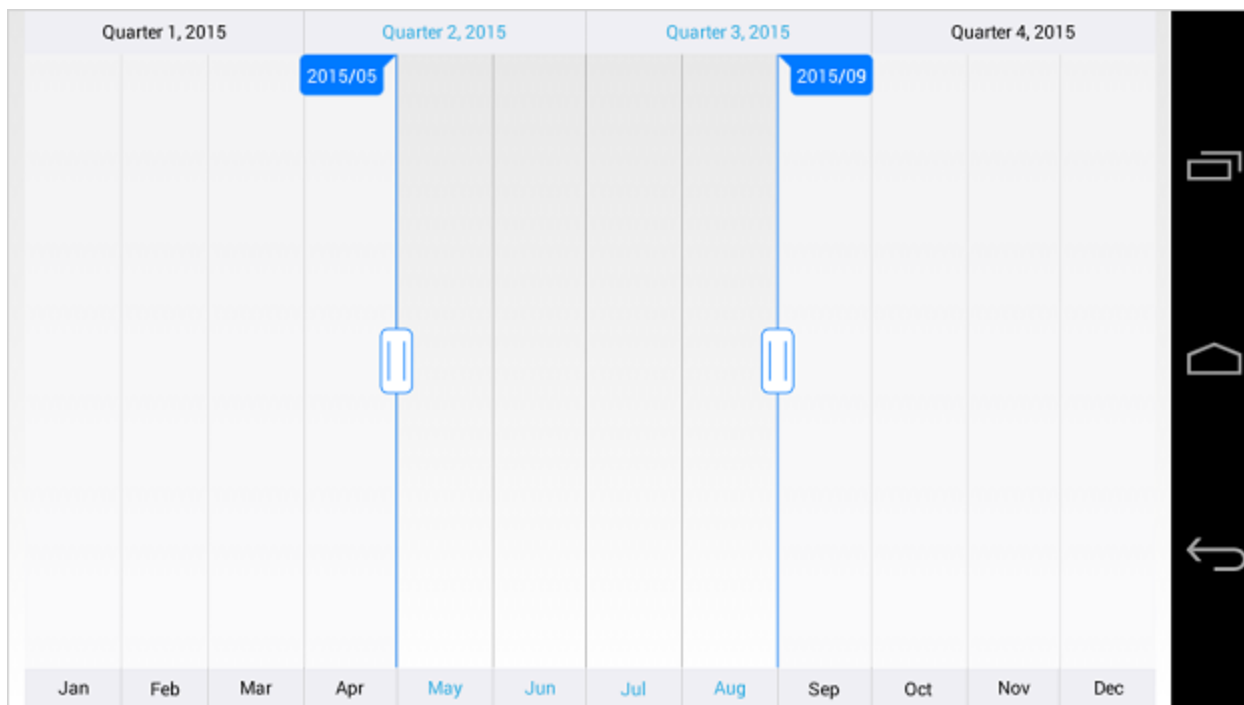
You can format the date value of the tooltip using [TooltipFormat](#) property of **SfDateTimeRangeNavigator**.

### XML

```
<rangnavigator:SfDateTimeRangeNavigator TooltipFormat="yyyy/MM" />
```

### C#

```
dateTimeRangeNavigator.TooltipFormat = "yyyy/MM";
```



### Appearance Customization

The [LeftTooltipStyle](#) and [RightTooltipStyle](#) properties of [SfDateTimeRangeNavigator](#) are used to customize the left and right tooltip. Following properties are available in each tooltip style to customize the appearance of the tooltip.

- [TextColor](#) – used to change the color of the tooltip text.
- [BackgroundColor](#) – used to change the background color of the tooltip.
- [BorderColor](#) – used to change the border color of the tooltip.
- [BorderWidth](#) – used to change the width of the tooltip border.
- [FontFamily](#) – used to change the font family of the tooltip text.
- [FontSize](#) – used to change the font size of the tooltip text.
- [FontAttributes](#) – used to change the font attribute of the tooltip text.
- [Margin](#) - used to change the margin size of the tooltip text.

### XML

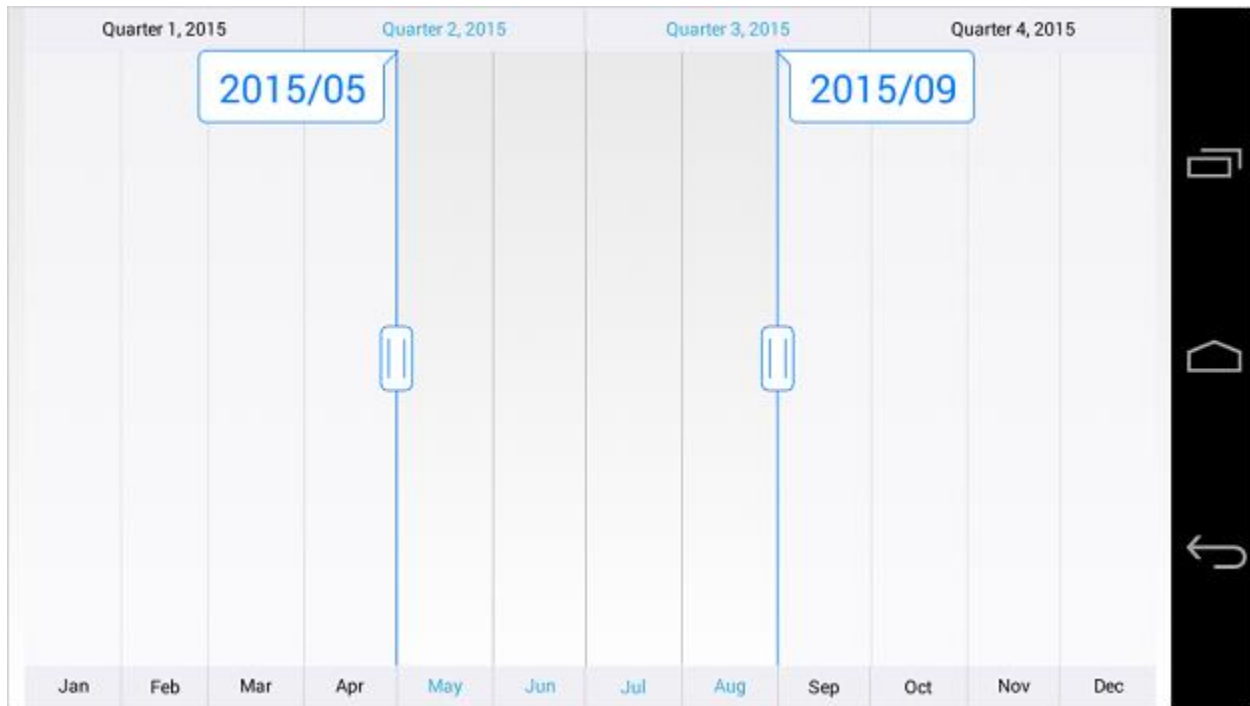
```
<rangnavigator:SfDateTimeRangeNavigator Minimum="1/1/2015"
Maximum="1/1/2016" ViewRangeStart="5/1/2015" ViewRangeEnd="9/1/2016">
  <rangnavigator:SfDateTimeRangeNavigator.LeftTooltipStyle>
    <rangnavigator:TooltipStyle TextColor="Blue" BackgroundColor="White"
    BorderColor="Blue" BorderWidth="2"
    FontSize="30" Margin="15"/>
  </rangnavigator:SfDateTimeRangeNavigator.LeftTooltipStyle>
  <rangnavigator:SfDateTimeRangeNavigator.RightTooltipStyle>
    <rangnavigator:TooltipStyle TextColor="Blue" BackgroundColor="White"
    BorderColor="Blue" BorderWidth="2"
    FontSize="30" Margin="15"/>
  </rangnavigator:SfDateTimeRangeNavigator.RightTooltipStyle>
</rangnavigator:SfDateTimeRangeNavigator>
```

**C#**

```

dateTimeRangeNavigator.LeftTooltipStyle.TextColor = Color.Blue;
dateTimeRangeNavigator.LeftTooltipStyle.BackgroundColor = Color.White;
dateTimeRangeNavigator.LeftTooltipStyle.BorderColor = Color.Blue;
dateTimeRangeNavigator.LeftTooltipStyle.BorderWidth = 2;
dateTimeRangeNavigator.LeftTooltipStyle.FontSize = 30;
dateTimeRangeNavigator.LeftTooltipStyle.Margin = 15;
dateTimeRangeNavigator.RightTooltipStyle.TextColor = Color.Blue;
dateTimeRangeNavigator.RightTooltipStyle.BackgroundColor = Color.White;
dateTimeRangeNavigator.RightTooltipStyle.BorderColor = Color.Blue;
dateTimeRangeNavigator.RightTooltipStyle.BorderWidth = 2;
dateTimeRangeNavigator.RightTooltipStyle.FontSize = 30;
dateTimeRangeNavigator.RightTooltipStyle.Margin = 15;

```

**Major and Minor Scales**

The SfDateTimeRangeNavigator control displays major and minor scales at the top and bottom position of the control.

**Intervals**

By default, best possible interval component will be chosen for both major and minor scales based on the available size of the view. For example, if the available space is sufficient to show only year labels without overlapping, interval will be displayed in years. However, you can also set specific [DateTimeIntervalType](#) components using [Intervals](#) property as demonstrated in the below code snippet.

**XML**

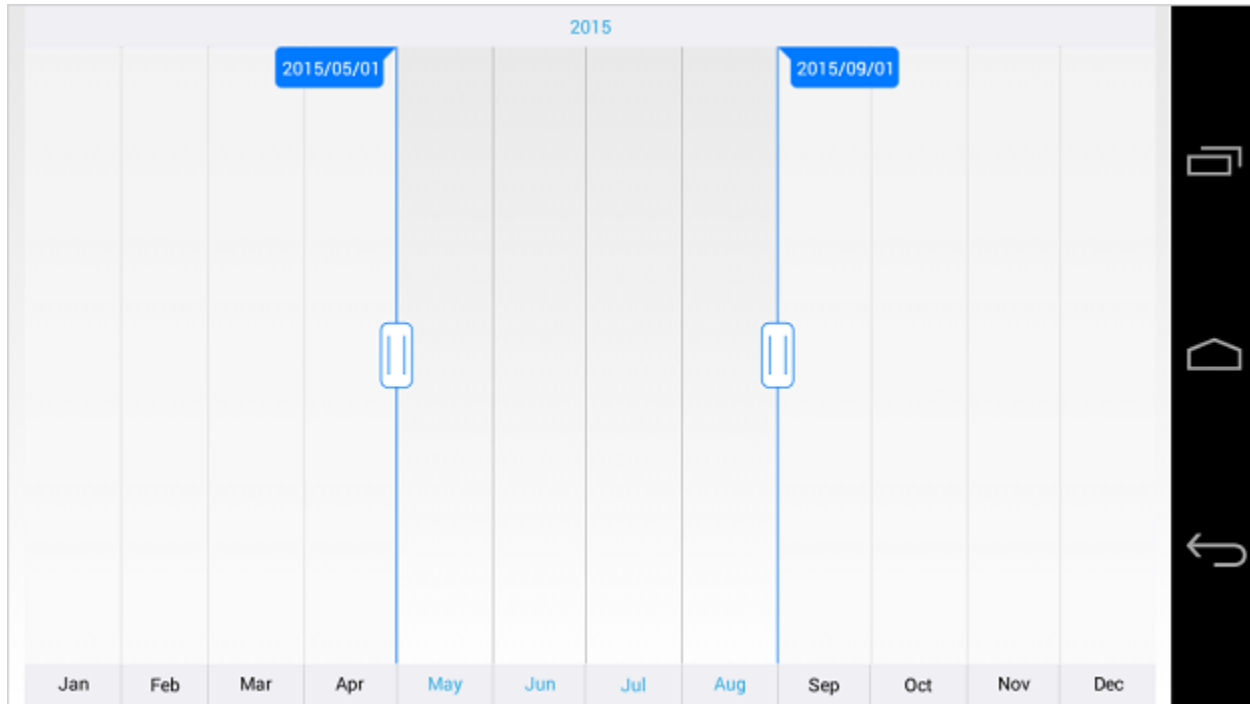
```

<rangenavigator:SfDateTimeRangeNavigator Minimum="1/1/2015"
Maximum="1/1/2016" ViewRangeStart="5/1/2015"
ViewRangeEnd="9/1/2016" Intervals="Year,Month" />

```

**C#**

```
SfDateTimeRangeNavigator dateTime = new SfDateTimeRangeNavigator();
dateTime.Intervals = DateTimeIntervalType.Year | DateTimeIntervalType.
Month;
```

**Appearance Customization**

The [MajorScaleStyle](#) and [MinorScaleStyle](#) properties of [SfDateTimeRangeNavigator](#) are used to customize the appearance of ticks and labels.

- [Position](#) – used to position the labels and ticks [inside](#) or [outside](#) of the range navigator.
- [LabelAlignment](#) – used to align the label at [Left](#), [Center](#) and [Right](#).
- [LabelTextColor](#) – used to change the text color of the labels.
- [LabelFontSize](#) – used to change the font size of the labels.
- [LabelFontAttributes](#) – used to change the font attribute of the labels.
- [LabelFontFamily](#) – used to change the font family of the labels.
- [SelectedLabelFontSize](#) – used to change the font size of the selected labels.
- [SelectedLabelFontAttributes](#) – used to change the font attribute of the selected labels.
- [SelectedLabelFontFamily](#) – used to change the font family of the selected labels.
- [LabelMargin](#) – used to change the margin size of the labels.
- [SelectedLabelTextColor](#) – used to change the text color of the selected labels.
- [SelectedLabelMargin](#) – used to change the margin of the selected labels.
- [TickLineWidth](#) - used to change the thickness of the tick line.
- [TickLineColor](#) - used to change the color of the tick line.

**XML**

```
<rangnavigator:SfDateTimeRangeNavigator Minimum="1/1/2015"
Maximum="1/1/2016" ViewRangeStart="5/1/2015"
```

```
ViewRangeEnd="9/1/2016">
<rangnavigator:SfDateTimeRangeNavigator.MajorScaleStyle>
<rangnavigator:ScaleStyle Position="Inside" LabelAlignment="Right"
SelectedLabelTextColor="Blue"
SelectedLabelFontSize="20" SelectedLabelMargin="15" LabelTextColor="Black"
LabelFontSize="20" LabelMargin="15"/>
</rangnavigator:SfDateTimeRangeNavigator.MajorScaleStyle>
<rangnavigator:SfDateTimeRangeNavigator.MinorScaleStyle>
<rangnavigator:ScaleStyle Position="Inside" LabelAlignment="Left"
SelectedLabelTextColor="Black"
SelectedLabelFontSize="20" SelectedLabelMargin="15" LabelTextColor="Red"
LabelFontSize="20" LabelMargin="15" />
</rangnavigator:SfDateTimeRangeNavigator.MinorScaleStyle>
</rangnavigator:SfDateTimeRangeNavigator>
```

## C#

```
SfDateTimeRangeNavigator dateTime = new SfDateTimeRangeNavigator();
dateTime.MajorScaleStyle.Position = ScalePosition.Inside;
dateTime.MajorScaleStyle.LabelTextColor = Color.Black;
dateTime.MajorScaleStyle.LabelMargin = 15;
dateTime.MajorScaleStyle.LabelFontSize = 20;
dateTime.MajorScaleStyle.LabelAlignment = LabelAlignment.Right;
dateTime.MajorScaleStyle.SelectedLabelTextColor = Color.Red;
dateTime.MajorScaleStyle.SelectedLabelMargin = 15;
dateTime.MajorScaleStyle.SelectedLabelFontSize = 20;
dateTime.MinorScaleStyle.Position = ScalePosition.Inside;
dateTime.MinorScaleStyle.LabelTextColor = Color.Red;
dateTime.MinorScaleStyle.LabelMargin = 15;
dateTime.MinorScaleStyle.LabelFontSize = 20;
dateTime.MinorScaleStyle.LabelAlignment = LabelAlignment.Left;
dateTime.MinorScaleStyle.SelectedLabelTextColor = Color.Black;
dateTime.MinorScaleStyle.SelectedLabelMargin = 15;
dateTime.MinorScaleStyle.SelectedLabelFontSize = 20;
```





### Scale visibility

You can also control the visibility of minor scale and major scale using the [MajorScaleStyle.IsVisible](#) and [MinorScaleStyle.IsVisible](#) properties.

### Hide minor scale

#### XML

```
<rangenavigator:SfDateTimeRangeNavigator Minimum="2015,01,01"
Maximum="2019,01,01">
  <rangenavigator:SfDateTimeRangeNavigator.MinorScaleStyle>
    <rangenavigator:ScaleStyle IsVisible="false" />
  </rangenavigator:SfDateTimeRangeNavigator.MinorScaleStyle>
</rangenavigator:SfDateTimeRangeNavigator>
```

#### C#

```
SfDateTimeRangeNavigator dateTime = new SfDateTimeRangeNavigator();
dateTime.Minimum = new DateTime(2015, 01, 01);
dateTime.Maximum = new DateTime(2019, 01, 01);
dateTime.MinorScaleStyle.IsVisible = false;
```



*Hide major scale*

### **XML**

```
<rangenavigator:SfDateTimeRangeNavigator Minimum="2015,01,01"
Maximum="2019,01,01">
  <rangenavigator:SfDateTimeRangeNavigator.MajorScaleStyle>
    <rangenavigator:ScaleStyle IsVisible="false" />
  </rangenavigator:SfDateTimeRangeNavigator.MajorScaleStyle>
</rangenavigator:SfDateTimeRangeNavigator>
```

### **C#**

```
SfDateTimeRangeNavigator dateTime = new SfDateTimeRangeNavigator();
dateTime.Minimum = new DateTime(2015, 01, 01);
dateTime.Maximum = new DateTime(2019, 01, 01);
dateTime.MajorScaleStyle.IsVisible = false;
```



### MinorScaleLabelsCreated event

This event occurs when the minor scale labels are created initially. The argument of this event contains the following information:

- **MinorScaleLabels** - Gets the **Content** of the each minor scale label.

### XML

```
<rangnavigator:SfDateTimeRangeNavigator Minimum="2015,01,01"
Maximum="2019,01,01" MinorScaleLabelsCreated="MinorScaleLabelsCreated" >
```

### C#

```
public MainPage()
{
    InitializeComponent();
    SfDateTimeRangeNavigator dateTime = new SfDateTimeRangeNavigator();
    dateTime.Minimum = new DateTime(2015, 01, 01);
    dateTime.Maximum = new DateTime(2019, 01, 01);
    dateTime.MinorScaleLabelsCreated += MinorScaleLabelsCreated;
}
private void MinorScaleLabelsCreated(object sender,
Syncfusion.RangeNavigator.XForms.MinorScaleLabelsCreatedEventArgs e)
{
    // handle event action.
}
```

### MajorScaleLabelsCreated event

This event occurs when the major scale labels are created initially. The argument of this event contains the following information:

- **MajorScaleLabels** - Gets the **Content** of the each major scale label.

### XML

```
<rangnavigator:SfDateTimeRangeNavigator Minimum="2015,01,01"
Maximum="2016,01,01" MajorScaleLabelsCreated="MajorScaleLabelsCreated" >
```

### C#

```
public MainPage()
{
    InitializeComponent();
    SfDateTimeRangeNavigator dateTime = new SfDateTimeRangeNavigator();
    dateTime.Minimum = new DateTime(2015, 01, 01);
    dateTime.Maximum = new DateTime(2016, 01, 01);
    dateTime.MajorScaleLabelsCreated += MajorScaleLabelsCreated;
}
private void MajorScaleLabelsCreated(object sender,
Syncfusion.RangeNavigator.XForms.MajorScaleLabelsCreatedEventArgs e)
{
    // handle event action.
}
```

```
}

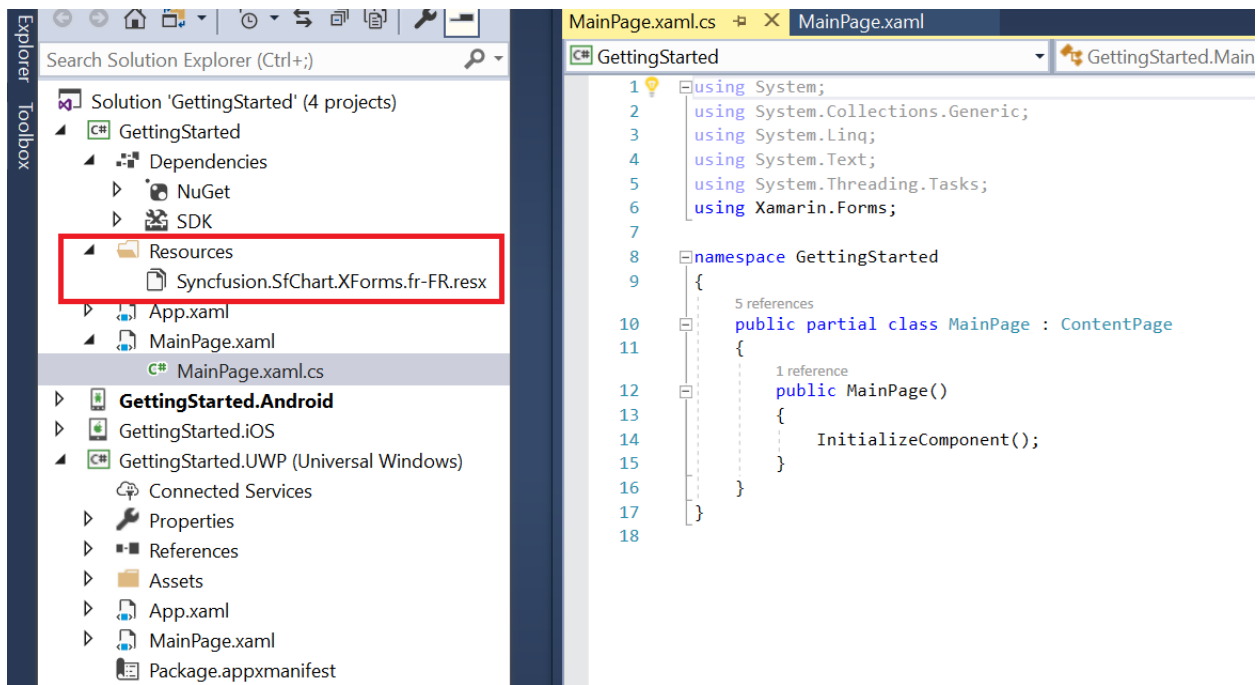
```

## Localization

You can localize [SfDateTimeRangeNavigator](#) in all the platforms by adding a .resx file in a .NET Standard project alone. The following steps describe how to localize SfDateTimeRangeNavigator in a project.

**Note:** Here, the resources have been already created for some cultures and shared them on [Syncfusion GitHub](#) for your convenience.

1. Add a new folder in the .NET Standard project named Resources.
2. Add resource files for the languages you wish to support and set their Build Action to EmbeddedResource. The name of the resource file should be \$name of the Syncfusion component\$+\$language code\$+.resx. For example, if you add a resource file for the French culture, add the Syncfusion.SfChart.XForms.fr-FR.resx file to Resources folder as illustrated in the following screenshot.



3. Provide the French values for each key in the respective .resx files. Here, “Quarter” and “Week” are the keys, and “Trimestre” and “La semaine” are their respective French values.

## XML

```
<data name="Quarter" xml:space="preserve">
<value>Trimestre</value>
</data>
<data name="Week" xml:space="preserve">
<value>La semaine</value>
</data>
```

4. Set resource manager to '[RangeNavigatorResourceManager.Manager](#)' to get the resource manager from users as demonstrated in the following code sample. For more details, please refer [Localization](#).

**C#**

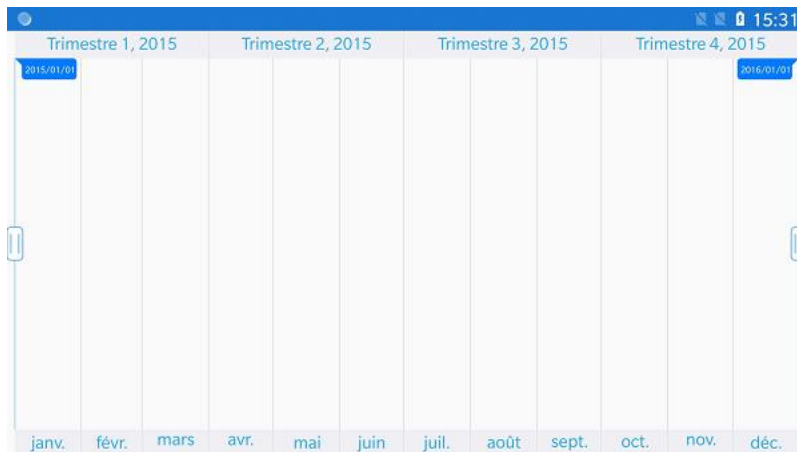
```
RangeNavigatorResourceManager.Manager = new  
ResourceManager("GettingStarted.Resources.Syncfusion.SfChart.XForms",  
Application.Current.GetType().Assembly);
```

**Localize at application level**

You can also localize the text at application-level regardless of the language selected on the device. The following platform-specific codes are needed to localize the text at application-level. Use the [DependencyServices](#) to set this from .NET Standard project.

**C#**

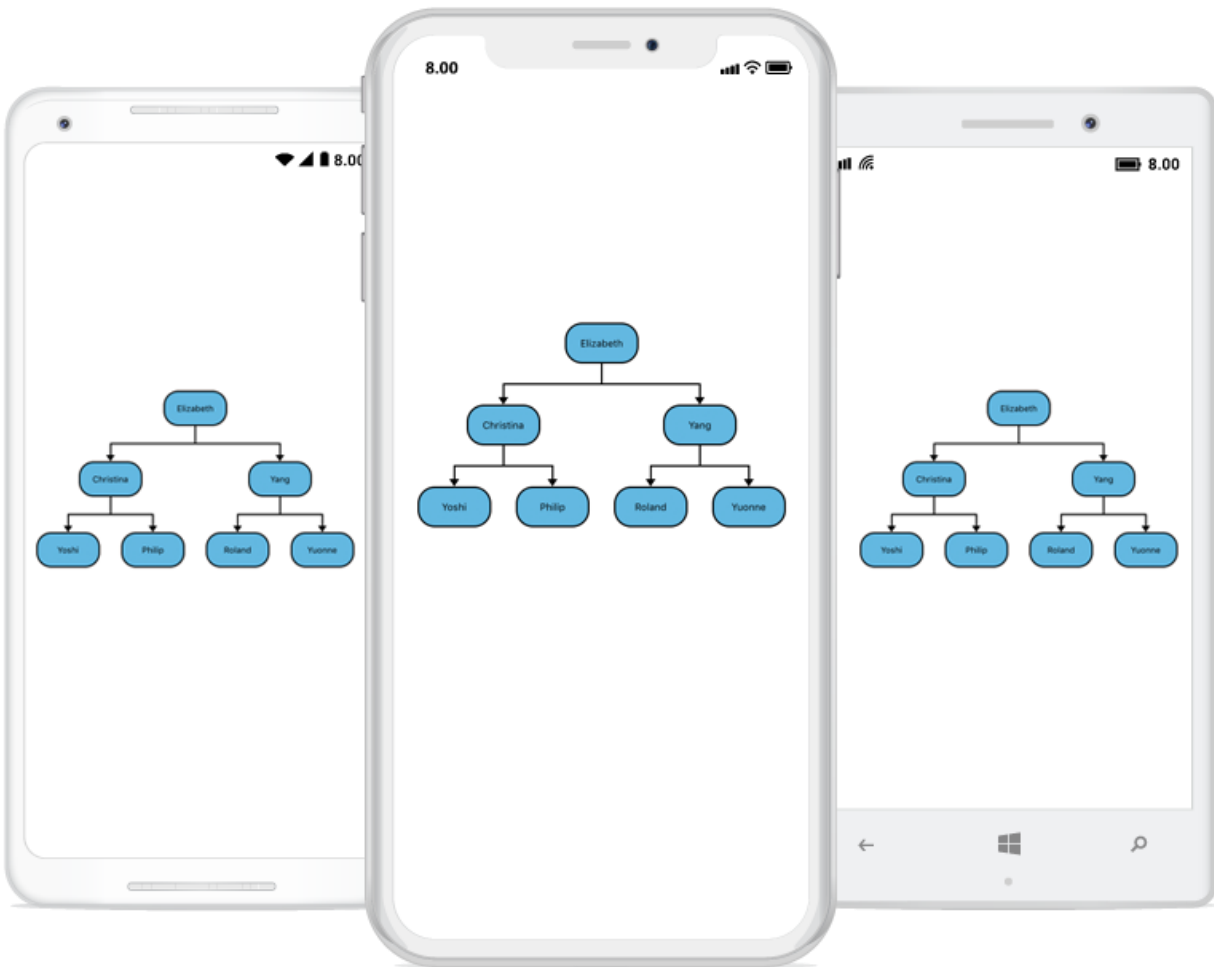
```
//For Android and iOS,  
Thread.CurrentThread.CurrentUICulture = new CultureInfo("fr-FR");  
//For UWP,  
CultureInfo.CurrentUICulture = new CultureInfo("fr-FR");
```



## SfDiagram

### Overview

The diagram control allows to create different types of diagrams such as flowcharts, use case diagrams, workflow process diagrams, etc.



## Key features

- **Node, Connector, Port:** Elements that is used to compose diagrams.
- **Interaction:** Zoom, pan.
- **Layouts:** Arrange nodes in a tree like structure based on the relationship on Nodes.
- **Clipboard Commands:** Performs cut, copy, and paste operations.
- **Undo/Redo:** Performs correction in the recent changes.
- **Serialization:** Save the current state of the diagram, and load it back when needed.
- **Stencil:** It holds a list of symbols that is dropped over the diagram.

## Getting started

This section provides a quick overview for working with Diagram for Xamarin.Forms. This walkthrough demonstrates that, how to create a simple flow chart and an organization chart.

### Adding SfDiagram reference

You can add SfDiagram reference using one of the following methods:

#### Method 1: Adding SfDiagram reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfDiagram). To add SfDiagram to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfDiagram](https://www.nuget.org/packages/Syncfusion.Xamarin.SfDiagram), and then install it.

![Adding SfDiagram reference from NuGet](Getting-Started\_images/Adding SfDiagram reference.png)

**Note:** Install the same version of SfDiagram NuGet in all the projects.

### Method 2: Adding SfDiagram reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfDiagram control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfDiagram assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

.NET Standard	Syncfusion.SfDiagram.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfDiagram.XForms.Android.dll Syncfusion.SfDiagram.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfDiagram.XForms.iOS.dll Syncfusion.SfDiagram.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfDiagram.UWP.dll Syncfusion.SfDiagram.XForms.UWP.dll Syncfusion.SfDiagram.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** If you are adding the references from toolbox, this step is not needed.

### Additional step for iOS

To launch SfDiagram in iOS, call the `SfDiagramRenderer.Init()` in `FinishedLaunching` overridden method of `AppDelegate` class in iOS Project, as demonstrated in the following code example.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    LoadApplication(new App());
    SfDiagramRenderer.Init();
    return base.FinishedLaunching(app, options);
}
```

---

**Information:** We need to create an instance of the `SfDiagramRenderer` in iOS and UWP projects as shown in this [KB article](#).

---

### Additional step for UWP

This step is required only if the application is deployed in Release mode with .NET native tool chain enabled and it is for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the SfDiagram assembly at `OnLaunched` overridden method of the `App` class in UWP project is the suggested work around, as demonstrated in the following code example.

#### C#

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    .....
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add 'using System.Reflection;'
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfDiagramRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    .....
}
```

---

**Information:** You can refer to the [KB article](#) for more details.

---

### Basic building blocks of Diagram

- **Diagram-** It represents the drawing surface where all the graphical elements like nodes and connectors resides, can be used to display various types of diagrams and it is the root instance of the diagram control. A Diagram instance contains a collection of nodes and connectors to represent the graphical diagram.
- **Nodes-** This represents the geometric shapes such as flowchart elements, network diagram elements, use case elements, etc.
- **Connectors-** These are the objects used to create link between two nodes, to represent the relationships between them in the diagram.



- **Ports**-It represents a point in the node, where the connectors can be connected. A Node can contain any number of ports.
- **Annotation**-It is a block of the text that can be displayed over a Node or Connector. Annotation is used to textually represent an object with a string that can be edited at run time.

### Creating a simple flow chart

Create a new cross platform app (Xamarin.Forms) with portable class library in the Visual Studio and name the project as "GettingStarted" and refer to the above mentioned assemblies to the respective projects.

An additional step is required to render the SfDiagram control in iOS project. You need to create an instance of the SfDiagramRenderer class within FinishedLaunching method of AppDelegate class in iOS project as shown as follows

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    LoadApplication(new App());
    new SfDiagramRenderer();
    return base.FinishedLaunching(app, options);
}
```

### Adding SfDiagram in Xamarin.Forms

1. Import SfDiagram control namespace as xmlns:syncfusion="clr-namespace:Syncfusion.SfDiagram.XForms;assembly=Syncfusion.SfDiagram.XForms in XAML Page.
2. Set the SfDiagram control as content to the ContentPage.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
xmlns:syncfusion="clr-namespace:Syncfusion.SfDiagram.XForms;assembly=Syncfusion.SfDiagram.XForms"
x:Class="GettingStarted.Sample">
    <ContentPage.Content>
        <!--Initializes the SfDiagram-->
        <syncfusion:SfDiagram x:Name="diagram" />
    </ContentPage.Content>
</ContentPage>
```

#### C#

```
using Syncfusion.SfDiagram.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
```

```
public class App : Application
{
    SfDiagram diagram;
    public App()
    {
        //Initializes the SfDiagram
        diagram= new SfDiagram();
        MainPage = new ContentPage { Content = diagram};
    }
}
```

The following code snippet illustrates the creation of Nodes and Connectors in the diagram.

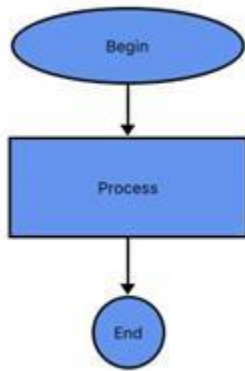
### C#

```
public GettingStarted()
{
    InitializeComponent();
    //Initializes the SfDiagram
    SfDiagram diagram = new SfDiagram();
    Node Begin = AddNode("Begin", 150, 60, 120, 40, "Begin", ShapeType.Ellipse);
    Node Process = AddNode("Process", 150, 140, 120, 60, "Process",
        ShapeType.Rectangle);
    Node End = AddNode("End", 190, 225, 40, 40, "End", ShapeType.Ellipse);
    //Add nodes to the SfDiagram
    diagram.AddNode(Begin);
    diagram.AddNode(Process);
    diagram.AddNode(End);
    Connector connector1 = new Connector()
    {
        SourceNode = Begin,
        TargetNode = Process,
    };
    Connector connector2 = new Connector()
    {
        SourceNode = Process,
        TargetNode = End,
    };
    //Add connectors to the SfDiagram
    diagram.AddConnector(connector1);
    diagram.AddConnector(connector2);
    this.Content = diagram;
}

///
///
///
public Node AddNode(string id, float offsetX, float offsetY, float width,
    float height, string text, ShapeType shape)
{
    Node node = new Node();
    node.OffsetX = offsetX;
    node.OffsetY = offsetY;
    node.Height = height;
    node.Width = width;
    node.ShapeType = shape;
}
```

```
node.Style.Brush = new SolidBrush(Color.FromRgb(100, 149, 237));
node.Annotations.Add(new Annotation() { Content = text});
return node;
}
```

The flow chart will get displayed in the SfDiagram as follows



This demo project can be downloaded from the following link [GettingStarted Demo](#).

### Create a simple organizational chart

SfDiagram provides support to auto-arrange the nodes based on hierarchical relation. Organization chart is an example of displaying hierarchical information.

Now, you have to create a class named “Employee” to store the employee’s information like name, designation, ID, reporting person ID, etc. Also, create a collection class that stores a collection of the employees.

### C#

```
//Employee Business Object
public class Employee
{
    public string ParentId { get; set; }
    public string Name { get; set; }
    public string Designation { get; set; }
    public int EmpId { get; set; }
}

//Employee Collection
public class Employees : ObservableCollection<Employee>
{
}
```

### Initialize Employee data

Define Employee Information as a Collection. The below code example shows an employee array whose,

- Name is used as a unique identifier and
- ParentId is used to identify the person to whom an employee report to, in the organization.

### XML

```

<!-- Initializes the employee collection-->
<local:Employees x:Key="employees">
<local:Employee Name="Elizabeth" Employee_Id="1" ParentId=""
Designation="CEO"/>
<local:Employee Name="Christina" Employee_Id="2" ParentId="1"
Designation="Manager"/>
<local:Employee Name="Yang" Employee_Id="3" ParentId="1"
Designation="Manager"/>
<local:Employee Name="Yoshi" Employee_Id="4" ParentId="2" Designation="Team
Lead"/>
<local:Employee Name="Philip" Employee_Id="5" ParentId="2" Designation="S/w
Developer"/>
<local:Employee Name="Roland" Employee_Id="6" ParentId="3"
Designation="TeamLead"/>
<local:Employee Name="Yvonne" Employee_Id="7" ParentId="3"
Designation="Testing Engineer"/>
</local:Employees>
<!--Initializes the DataSourceSettings -->
<syncfusion:DataSourceSettings x:Key="DataSourceSettings"
DataSource="{StaticResource employees}" ParentId="ParentId" Id="Employee_Id"
Root="1"/>
<!--Initializes the Layout-->
<syncfusion:DirectedTreeLayout x:Key="treeLayout" HorizontalSpacing="80"
VerticalSpacing="50" SpaceBetweenSubTrees="20" Orientation="TopToBottom"/>
<syncfusion:LayoutManager x:Key="layoutManager" Layout="{StaticResource
treeLayout}"/>
<!--Initializes the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram" LayoutManager="{StaticResource
layoutManager}" DataSourceSettings="{StaticResource DataSourceSettings}">

```

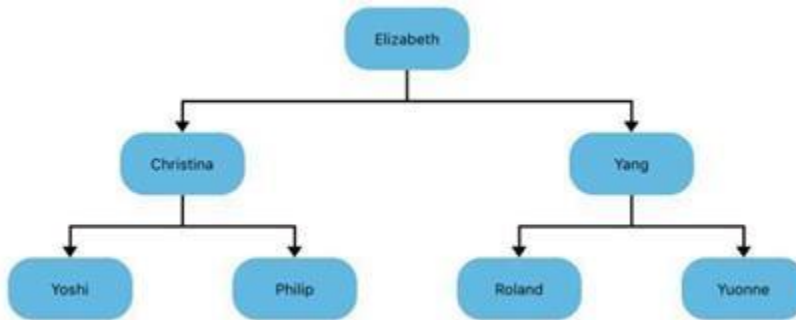
## C#

```

//Initializes the employee collection
ObservableCollection<Employee> employees = new
ObservableCollection<Employee>();
employees.Add(new Employee() { Name = "Elizabeth", Employee_Id = "1",
ParentId = "", Designation = "CEO" });
employees.Add(new Employee() { Name = "Christina", Employee_Id = "2",
ParentId = "1", Designation = "Manager" });
employees.Add(new Employee() { Name = "Yang", Employee_Id = "3", ParentId =
"1", Designation = "Manager" });
employees.Add(new Employee() { Name = "Yoshi", Employee_Id = "4", ParentId =
"2", Designation = "Team Lead" });
employees.Add(new Employee() { Name = "Philip", Employee_Id = "5", ParentId
= "2", Designation = "S/w Developer" });
employees.Add(new Employee() { Name = "Roland", Employee_Id = "6", ParentId
= "3", Designation = "TeamLead" });
employees.Add(new Employee() { Name = "Yvonne", Employee_Id = "7", ParentId
= "3", Designation = "Testing Engineer" });
//Initializes the DataSourceSettings
diagram.DataSourceSettings = new DataSourceSettings() { DataSource =
employees, Id = "Employee_Id", ParentId = "ParentId" };
//Initializes the Layout
DirectedTreeLayout treeLayout = new DirectedTreeLayout() { HorizontalSpacing
= 80, VerticalSpacing = 50, TreeOrientation = TreeOrientation.TopToBottom };
diagram.LayoutManager = new LayoutManager() { Layout = treeLayout };

```

The Employee data is displayed in the SfDiagram as follows



This demo project can be downloaded from the following link [OrganizationalChart Demo](#).

### Diagram

Diagram control allows to create different types of diagrams such as flow charts, use case diagrams, workflow process diagrams, and more.

### Page settings

Page settings enable to customize the appearance, width, and height of the Diagram page. The size and appearance of the Diagram pages can be customized with the PageSettings property.

The PageWidth and PageHeight properties of page settings define the size of the page. You can also customize the appearance of off-page regions with the property BackgroundColor.

The following code illustrates how to customize the page size and the appearance of page and off-page.

### XML

```
<diagram:SfDiagram x:Name="diagram" BackgroundColor="Lime">
  <diagram:SfDiagram.PageSettings>
    <diagram:PageSettings PageWidth="500" PageHeight="50" PageBackGround="White"
    />
  </diagram:SfDiagram.PageSettings>
</diagram:SfDiagram>
```

### C#

```
//Sets Page background
diagram.BackgroundColor = Color.Lime;
//Sets Page size
diagram.PageSettings.PageWidth = 500;
diagram.PageSettings.PageHeight = 500;
//Customizes the appearance of Page
diagram.PageSettings.PageBackGround= Color.White;
```

### Stencil:

Stencil has a collection of Symbols. Stencil is used to clone the desired symbol by dragging it from the Stencil and dropping it into the SfDiagram.

The following code snippet illustrates to add the stencil.

### XML

```
<syncfusion:Stencil x:Name="stencil" >
</syncfusion:Stencil>
```

**C#**

```
Stencil stencil = new Stencil();
this.Content = stencil;
```

**Node template:**

You can replace the entire node template with your own design using SfDiagram.NodeTemplate property.

The following code snippet and screenshot illustrates this.

**XML**

```
<DataTemplate x:Key="template">
<Grid WidthRequest="80" HeightRequest="80">
<Image Source="diagram.png"/>
</Grid>
</DataTemplate>
<control:SfDiagram x:Name="diagram" NodeTemplate="{StaticResource
template}">
```

**C#**

```
var template = new DataTemplate(() =>
{
    Grid grid = new Grid();
    grid.WidthRequest = 80;
    grid.HeightRequest = 80;
    Image image = new Image();
    image.Source = "employee.png";
    grid.Children.Add(image);
    return grid;
});
diagram.NodeTemplate = template;
```

**Diagram constraints**

The constraints property of Diagram allows you to enable/disable certain features.

Diagram Constraints allow to enable or disable the following behaviors of Node.

Drag

Resize

Rotate

AnnotationEditing

EnableSelectors

EnableZoomAndPan

IsReadOnly

## Example

The following code illustrates how to disable the item dragging.

### XML

```
<syncfusion:SfDiagram x:Name="diagram" EnableDrag="False">
</syncfusion:SfDiagram>
```

### C#

```
SfDiagram diagram = new SfDiagram();
// Disable the item dragging
diagram.EnableDrag = false;
this.Content = diagram;
```

## Diagram style settings

It is easier to apply default rendering styles to all shapes, connectors, stencil symbol, and stencil header in a diagram. Pass the following arguments to diagram style setting constructor:

- **DefaultNodeStyle** argument: Defines the node style properties.
- **DefaultConnectorStyle** argument: Defines the connector style properties.
- **DefaultSymbolStyle** argument: Defines the symbol style properties.
- **DefaultHeaderStyle** argument: Defines the header style properties.

The following code shows how to define the diagram style settings for the diagram object.

### C#

```
// Diagram style settings
Syncfusion.SfDiagram.XForms.Style NodeStyle = new
Syncfusion.SfDiagram.XForms.Style() { Brush = new SolidBrush(Color.Blue),
StrokeBrush = new SolidBrush(Color.Brown), StrokeWidth = 2, StrokeStyle =
StrokeStyle.Dashed };
Syncfusion.SfDiagram.XForms.Style ConnectorStyle = new
Syncfusion.SfDiagram.XForms.Style() { StrokeBrush = new
SolidBrush(Color.DeepSkyBlue), StrokeWidth = 3.0f, StrokeStyle =
StrokeStyle.Dotted };
SymbolStyle SymbolStyle = new SymbolStyle() { Width = 70, Height = 70,
BorderThickness = 4, BorderBrush = Color.Gray }; HeaderStyle HeaderStyle =
new HeaderStyle() { FontSize = 24, TextBrush = Color.White, Fill =
Color.SteelBlue, FontStyle = FontStyle.Italic, HorizontalAlignment =
HorizontalAlignment.Center };
DiagramStyleSettings diagramStyleSettings = new
DiagramStyleSettings(NodeStyle, ConnectorStyle, SymbolStyle, HeaderStyle);
//Passing diagram style settings instance to SfDiagram constructor
SfDiagram diagram = new SfDiagram(diagramStyleSettings);
```

## Zooming enhancement

Supports customizing the zoom levels. You can set minimum zoom level value to 0.01f and maximum to 'infinite'.

### XML

```
<!-- Define the minimum and maximum zoom factor value -->
<control:SfDiagram x:Name="diagram" MinimumZoomFactor="0.01"
MaximumZoomFactor="5.00" />
```

## C#

```
// Define the minimum and maximum zoom factor value
diagram.MinimumZoomFactor = 0.01f;
diagram.MaximumZoomFactor = 5.00f;
```

**Note:** Diagram supports zooming with custom option in Xamarin.Forms.Android and Xamarin.Forms.iOS alone.

## Node

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

### Create node

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

The following code snippet illustrates how to create the node.

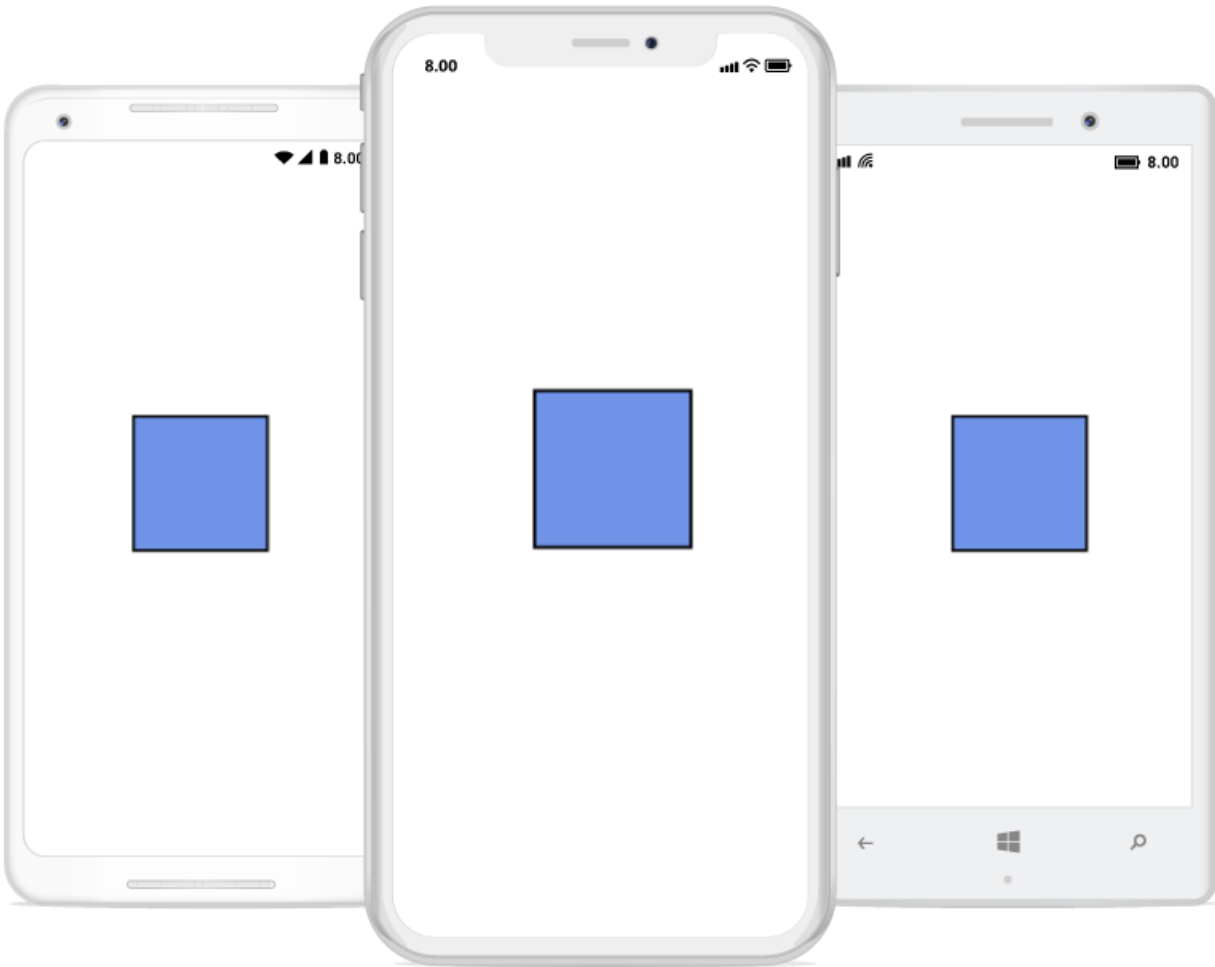
## XML

```
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize NodeCollection-->
  <syncfusion:SfDiagram.Nodes>
    <syncfusion:NodeCollection>
      <!--Initialize Node-->
      <syncfusion:Node Width="100" Height="100" OffsetX="200" OffsetY="200">
      </syncfusion:Node>
    </syncfusion:NodeCollection>
  </syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>
```

## C#

```
//Initialize Node
Node n = new Node() { Width = 120, Height = 40, OffsetX = 300, OffsetY = 60,
ShapeType = ShapeType.Ellipse };
//Adds the node to the SfDiagram
diagram.AddNode(n);
```





Create a node with custom path/shape

Node can be created with different custom shapes and path using SfGraphics.

The following code snippet is used to create the Node with custom shape/path.

#### C#

```
//Initialize Node
Node node = new Node();
//Initialize SfGraphics
SfGraphics graphics = new SfGraphics();
Pen pen = new Pen();
pen.StrokeBrush = new SolidBrush(Color.Red);
pen.StrokeWidth = 2;
SolidBrush brush = new SolidBrush(Color.Yellow);
brush.FillColor = Color.Yellow;
pen.Brush = brush2;
graphics.DrawRectangle(pen, new Rectangle(0, 0, 50, 50));
//Update the SfGraphics to the node
node.UpdateSfGraphics(graphics);
```

### Accessing a node from the diagram instance

We can access the Node from the diagram instance using the following code snippet.

#### C#

```
//Access the node from the diagram instance.  
Node node = diagram.Nodes[0];
```

### Remove a node

The following code snippet is used to remove the node from the diagram.

#### C#

```
//Remove the node from the diagram.  
Node node = new Node();  
diagram.RemoveNode(node);
```

### Customization:

We can Customize the entire node with our own customized design using Template property.

#### XML

```
<!--Initialize the DataTemplate-->  
<DataTemplate x:Key="template">  
  <Grid WidthRequest="80" HeightRequest="80">  
    <Image Source="diagram.png"/>  
  </Grid>  
</DataTemplate>  
<!--Initialize node-->  
<diagram:Node OffsetX="300" OffsetY="300" Width="60" Height="70"  
  Template="{StaticResource template}" >
```

#### C#

```
//Initialize the template  
var template = new DataTemplate(() =>  
{  
  Grid grid = new Grid();  
  grid.WidthRequest = 80;  
  grid.HeightRequest = 80;  
  Image image = new Image();  
  image.Source = "employee.png";  
  grid.Children.Add(image);  
  return grid;  
});  
//Initialize node with template.  
Node node = new Node() { Width = 120, Height = 40, OffsetX = 300, OffsetY =  
60, Template = template};
```

### Constraints

Node Constraints allow us to enable or disable the following behaviors of Node.

- Drag
- Resize

- Rotate
- AnnotationEditing
- IsLocked.

#### Example

The following code illustrates how to disable node dragging.

#### XML

```
<!--Initialize the node with constraints-->
<syncfusion:Node OffsetX="300" OffsetY="60" Width="120" Height="40"
ShapeType="Ellipse" EnableDrag="False">
```

#### C#

```
//Initialize the node with constraints
Node node = new Node() {EnableDrag = false;, Width = 50, Height = 50,
OffsetX = 100, OffsetY = 100};
```

### Connector

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

#### Create connector

Connector can be created by defining the start and end points.

#### XML

```
<!--creating connector instance-->
<control:SfDiagram.Connectors>
<control:ConnectorCollection>
<control:Connector SourcePoint="100,100" TargetPoint="300,300">
</control:Connector>
</control:ConnectorCollection>
</control:SfDiagram.Connectors>
```

#### C#

```
// creating connector instance
var connector1 = new Connector()
{
SourcePoint = new Point(100,100),
TargetPoint = new Point(300,300)
};
diagram.AddConnector(connector1);
```

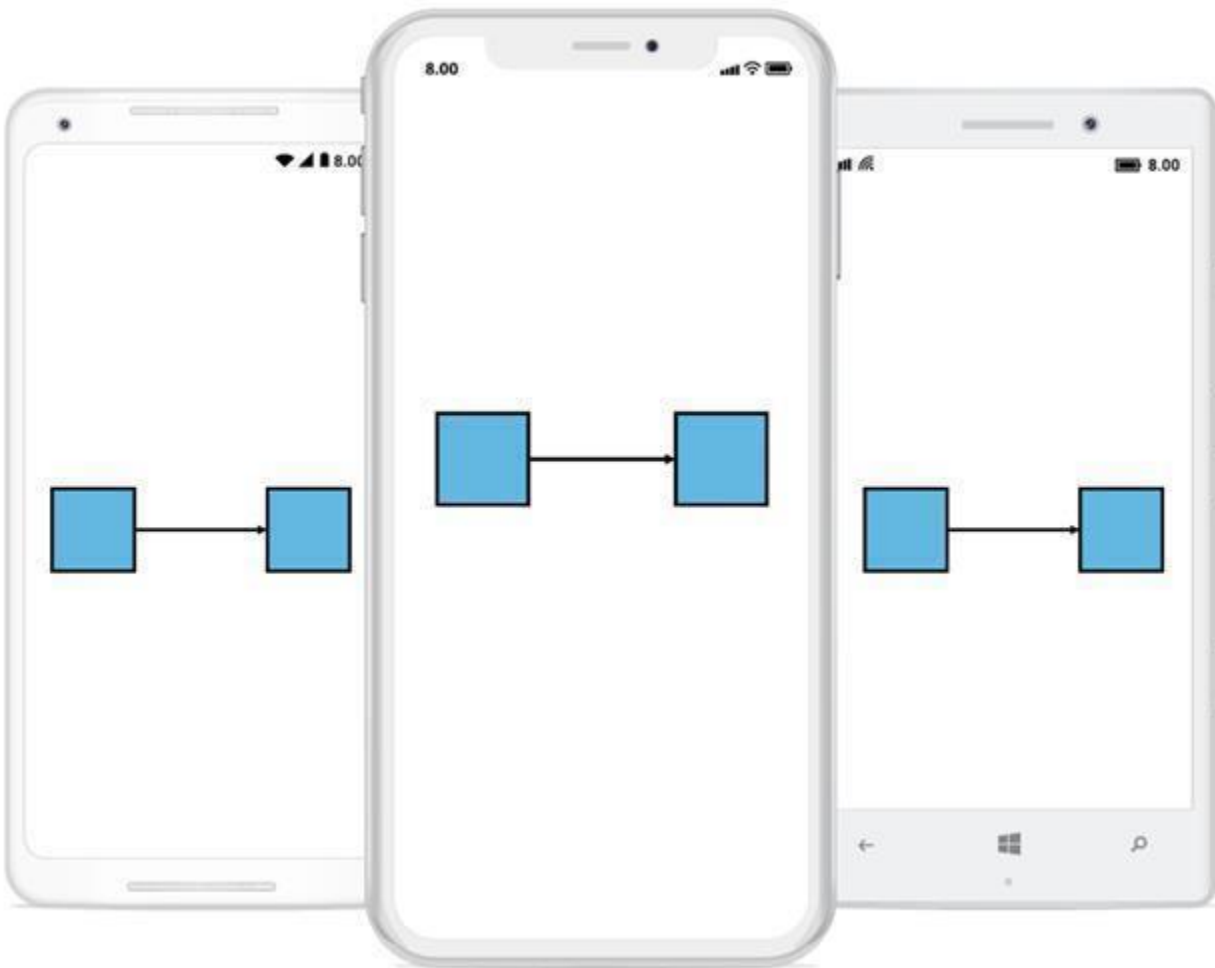
### Connections with nodes

The SourceNode and TargetNode properties allow to define the Nodes to be connected. The following code example illustrates how to connect two Nodes.

#### C#

```
// creating node instance
```

```
var node1 = new Node(100, 300, 100, 100);  
diagram.AddNode(node1);  
var node2 = new Node(300, 300, 100, 100);  
diagram.AddNode(node2);  
// creating connector instance and connecting nodes with it  
var connector1 = new Connector()  
{  
    SourceNode = node1,  
    TargetNode = node2  
};  
diagram.AddConnector(connector1);
```



### Connections with ports

The SourcePort and TargetPort properties allow to create connections between some specific points of Source/Target Nodes. The following code examples illustrates how to use Port to Port connection.

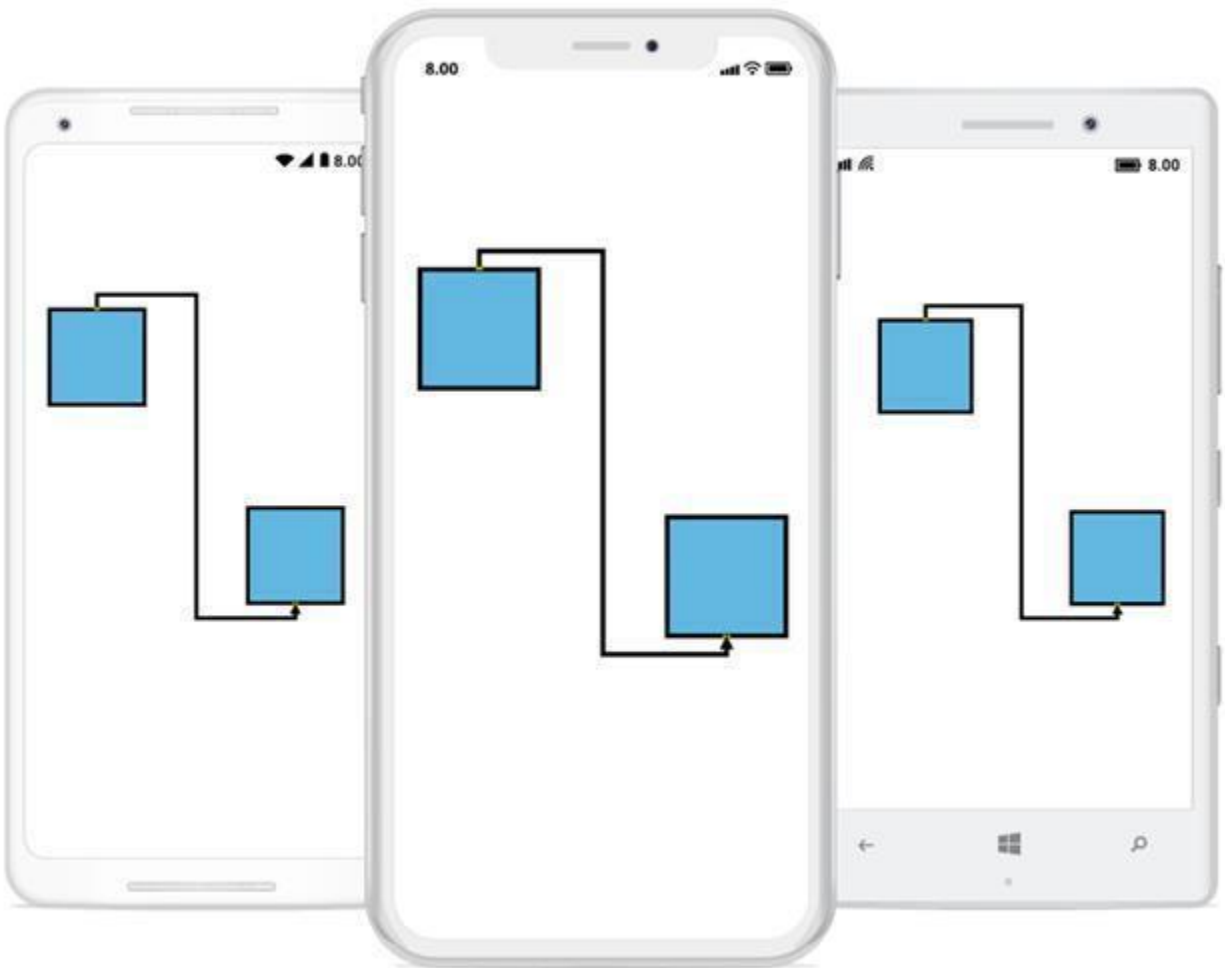
### Using port

The following code example illustrates how to create Port and connect using that ports.

### C#

```
// creating node instance
```

```
var node1 = new Node(100, 100, 100, 100);  
//adding port instance to the node  
node1.Ports.Add(new Port() { NodeOffsetX = 0.5, NodeOffsetY = 0 });  
//adding node to the diagram  
diagram.AddNode(node1);  
// creating node instance  
var node2 = new Node(300, 300, 100, 100);  
//adding port instance to the node  
node2.Ports.Add(new Port() { NodeOffsetX = 0.5, NodeOffsetY = 1 });  
//adding node to the diagram  
diagram.AddNode(node2);  
//creating and connecting the ports with connector  
var connector1 = new Connector()  
{  
    SourceNode = node1,  
    TargetNode = node2,  
    SourcePort=node1.Ports[0],  
    TargetPort=node2.Ports[0],  
    SegmentType = SegmentType.OrthoSegment  
};  
diagram.AddConnector(connector1);
```



## Segments

The path of the Connector is defined with a collection of segments.

### Straight

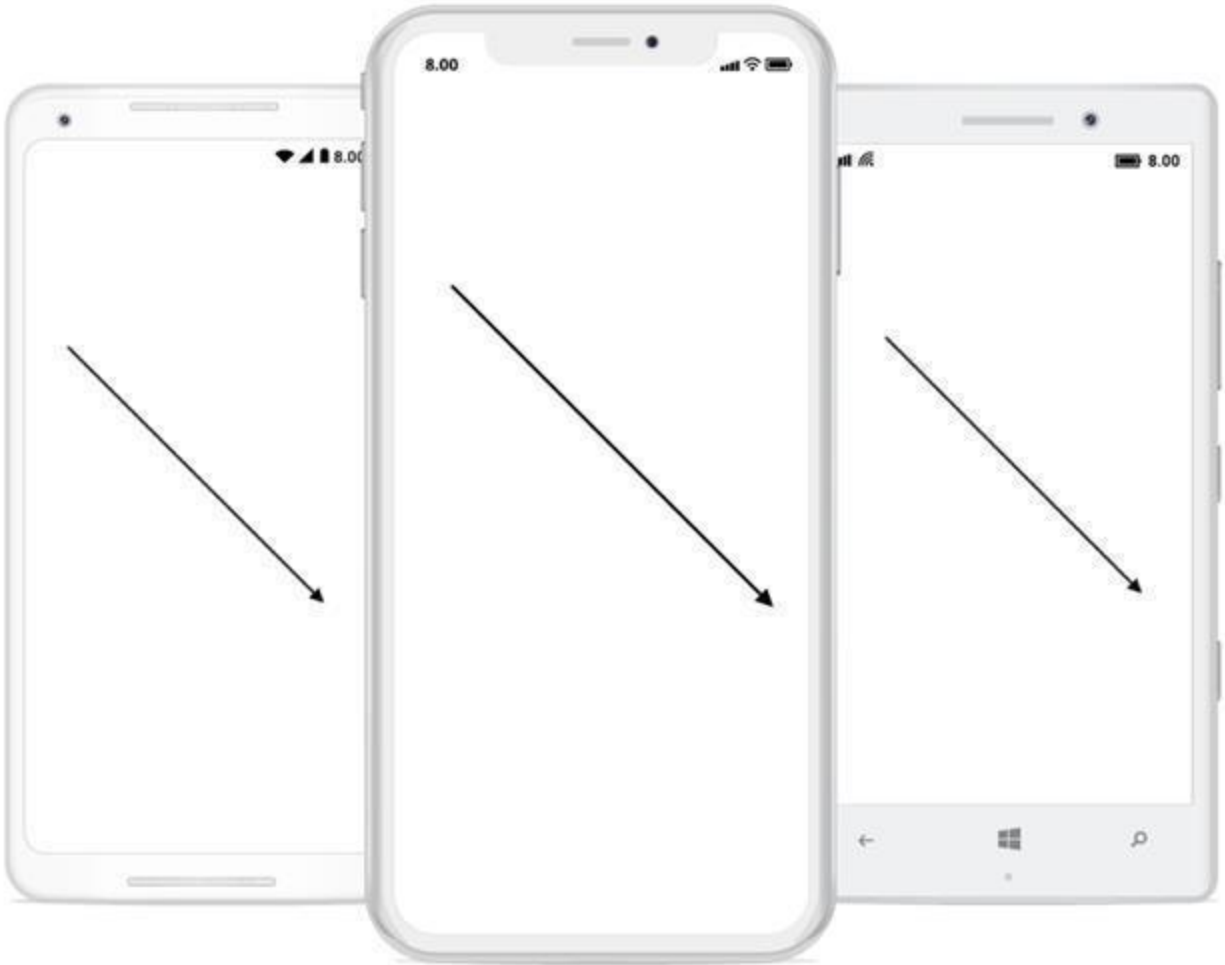
Straight segment allows to create a straight line. To create a straight line, you should specify the segment as `StraightSegment`. The following code example illustrates how to create a default straight segment.

#### XML

```
<!--creating connector instance with define its segment type-->  
<control:SfDiagram.Connectors>  
  <control:ConnectorCollection>  
    <control:Connector SourcePoint="100,100" TargetPoint="300,300"  
      SegmentType="StraightSegment">  
    </control:Connector>  
  </control:ConnectorCollection>  
</control:SfDiagram.Connectors>
```

#### C#

```
// creating connector instance with define its segment type  
var connector1 = new Connector()  
{  
    SourcePoint = new Point(100,100),  
    TargetPoint = new Point(300,300),  
    SegmentType = SegmentType.StraightSegment  
};  
diagram.AddConnector(connector1);
```



### Orthogonal

Orthogonal segments are used to create segments that are perpendicular to each other.

Set the segment as `OrthogonalSegment` to create the default orthogonal segment. The following code example illustrates how to create a default orthogonal segment.

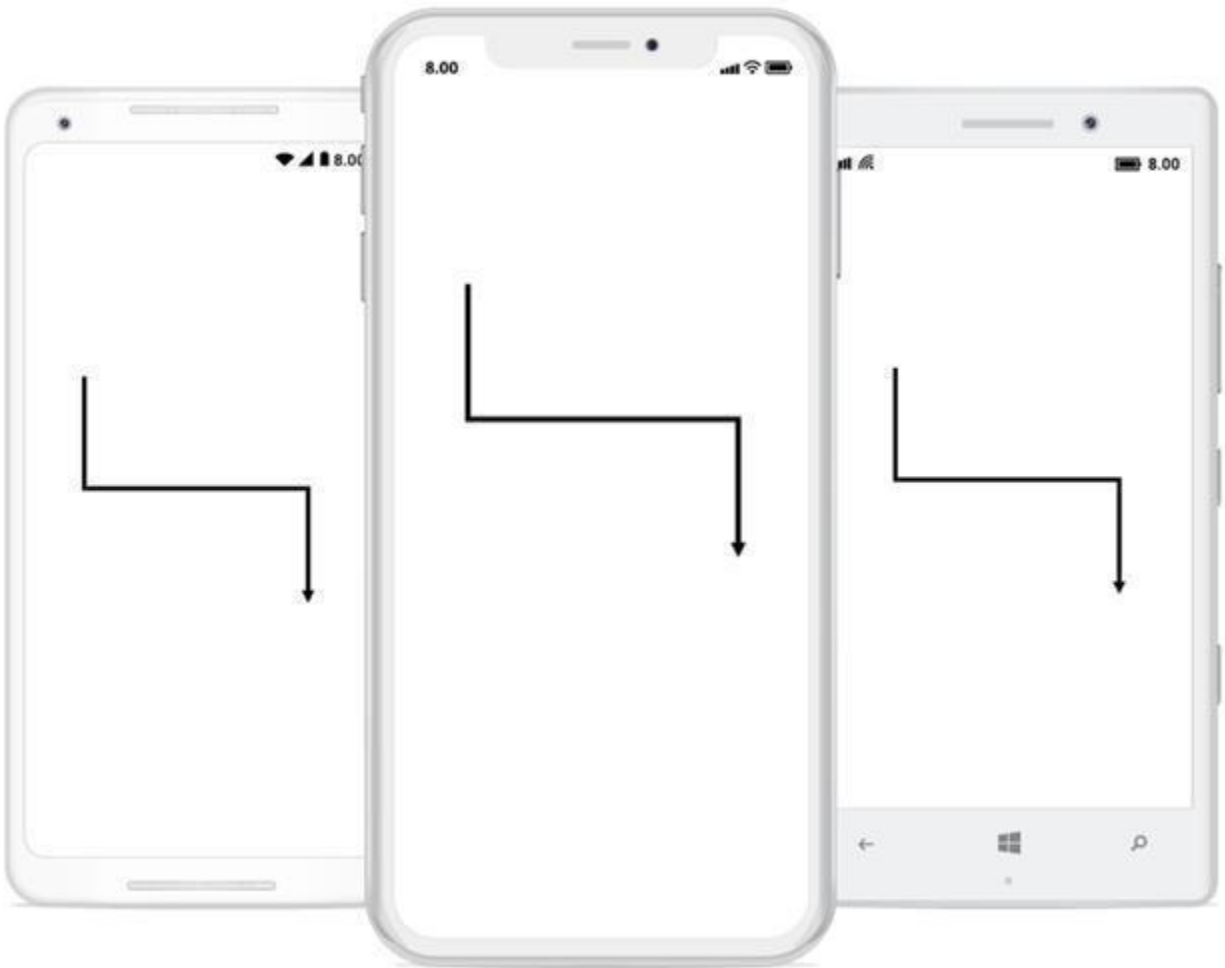
#### XML

```
<!--creating connector instance with define its segment type-->
<control:SfDiagram.Connectors>
  <control:ConnectorCollection>
    <control:Connector SourcePoint="100,100" TargetPoint="300,300"
      SegmentType="OrthoSegment">
    </control:Connector>
  </control:ConnectorCollection>
</control:SfDiagram.Connectors>
```

#### C#

```
// creating connector instance with define its segment type
var connector1 = new Connector()
{
```

```
SourcePoint = new Point(100, 100),  
TargetPoint = new Point(300, 300),  
SegmentType = SegmentType.OrthoSegment  
};  
diagram.AddConnector(connector1);
```



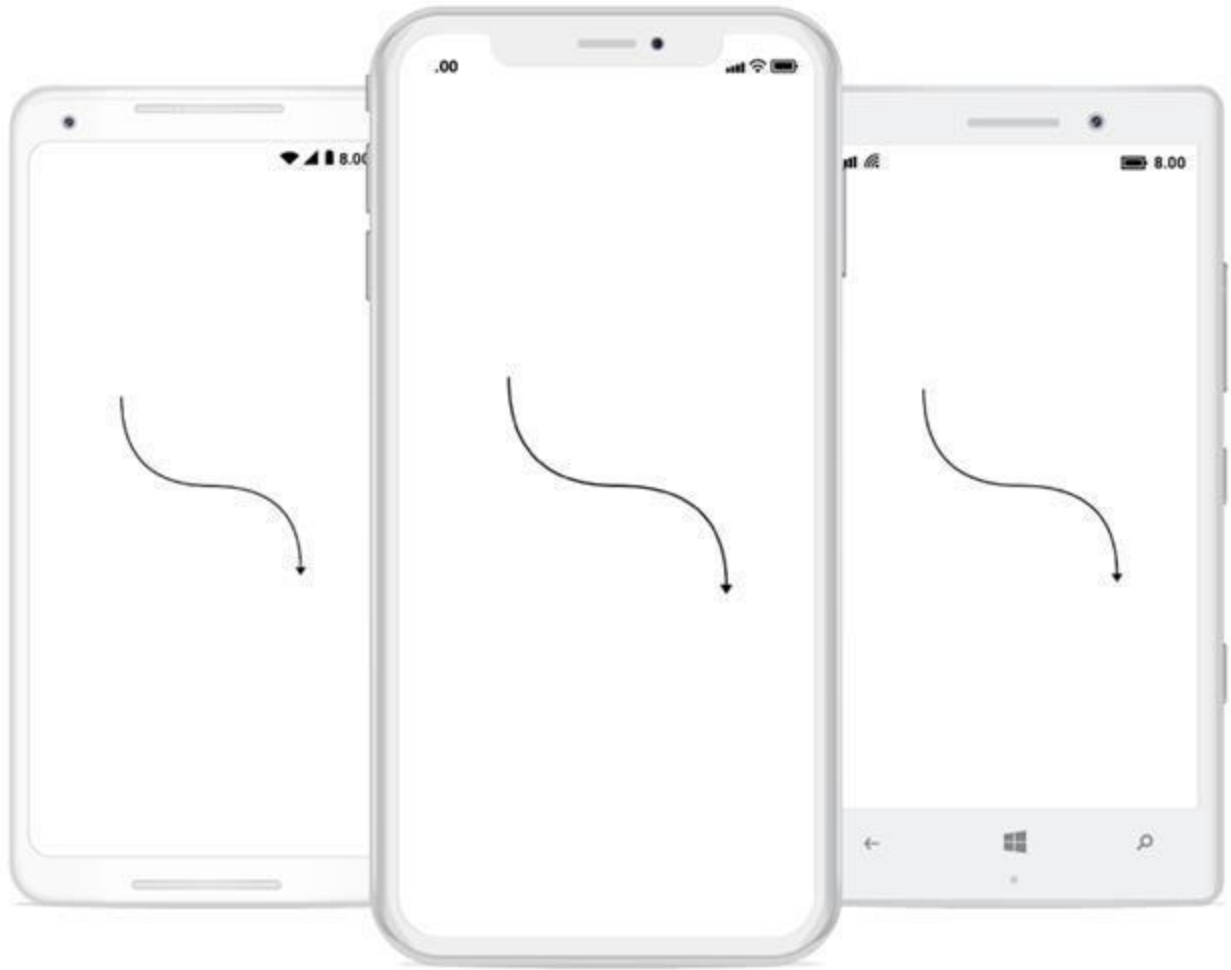
## Curve

Curve segments are used to create links between two points, nodes or ports with curve segments. The following code example illustrates how to create a default curve segment.

### C#

```
// creating connector instance with define its segment type  
Connector connector = new Connector(this);  
connector.SourcePoint = new Point(100, 100);  
connector.TargetPoint = new Point(300, 300);  
connector.SegmentType = SegmentType.CurveSegment;  
diagram.AddConnector(connector);
```





### Decorator

Start and end points of a Connector can be decorated with some customizable shapes like arrows, circles, diamond and square. You can decorate the connection end points with the `SourceDecorator` and `TargetDecorator` properties of `Connector`.

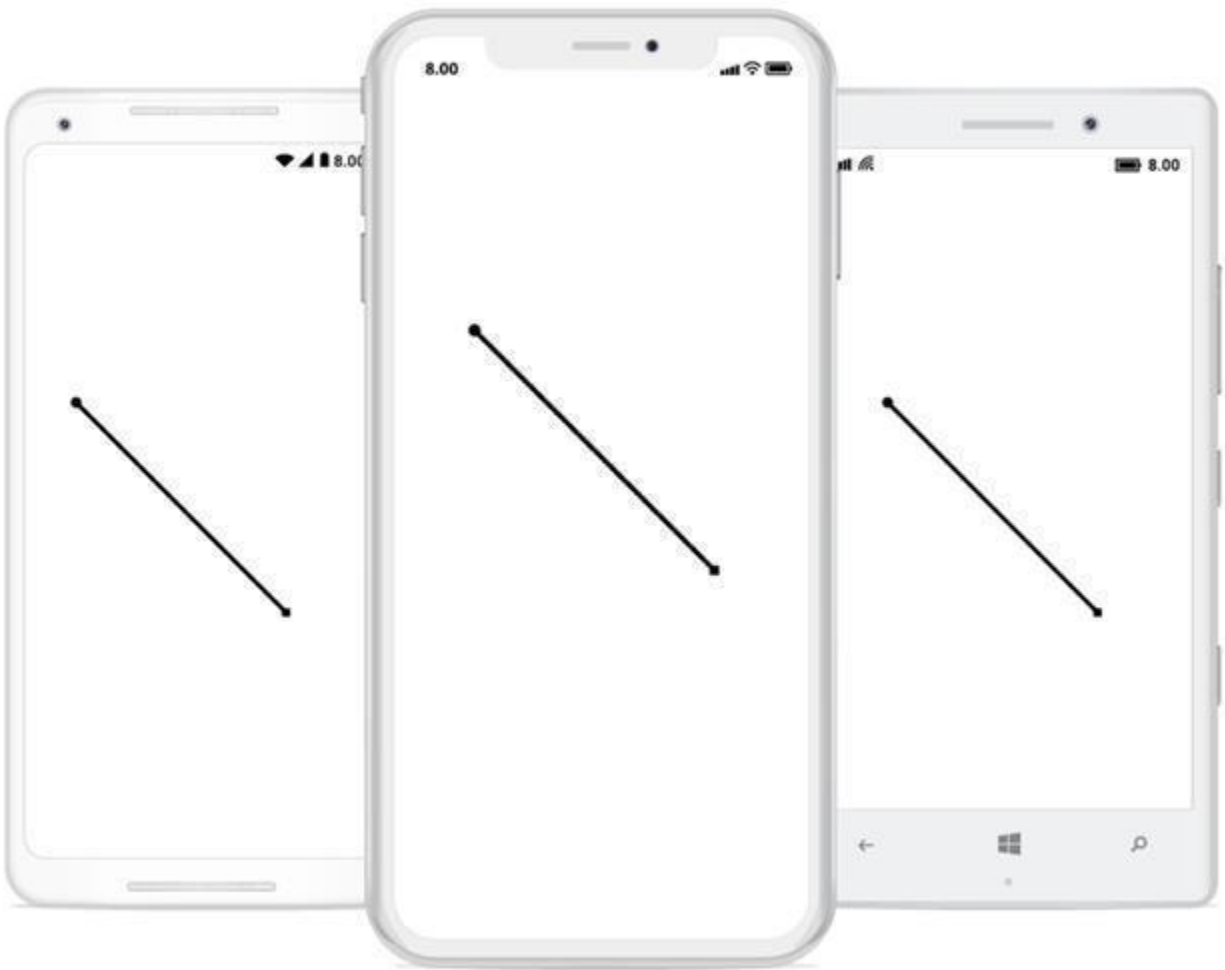
The `SourceDecoratorStyle` and `TargetDecoratorStyle` properties allows to define the shape of the decorators. The following code example illustrates how to create decorators of various shapes.

### XML

```
<!--creating connector instance with decorator-->
<control:SfDiagram.Connectors>
  <control:ConnectorCollection>
    <control:Connector SourcePoint="100,100" TargetPoint="300,300"
      SegmentType="StraightSegment" TargetDecoratorType="Diamond"
      SourceDecoratorType="Circle">
    </control:Connector>
  </control:ConnectorCollection>
</control:SfDiagram.Connectors>
```

### C#

```
//creating connector instance with decorator-->  
var connector1 = new Connector()  
{  
    SourcePoint = new Point(100,100),  
    TargetPoint = new Point(300,300),  
    SourceDecoratorType = DecoratorType.Circle,  
    TargetDecoratorType = DecoratorType.Diamond,  
    SegmentType= SegmentType.StraightSegment  
};  
diagram.AddConnector(connector1);
```



### Remove connector

Connector can be removed or detached from connection in two ways.

1. passing the connector as parameter to Remove connector method in diagram

The following code example illustrates how to remove a connector from connection

### C#

```
//creating connector instance
```

```

var connector1 = new Connector()
{
    SourcePoint = new Point(100,100),
    TargetPoint = new Point(300,300)
};
//adding connector to the diagram
diagram.AddConnector(connector1);
//removing connector from the diagram using object.
diagram.RemoveConnector(connector1);

```

## 2. passing the index value of the connector to connector collection (using RemoveAt method)

The following code example illustrates how to remove a connector from connection

### C#

```

//creating connector instance
var connector1 = new Connector()
{
    SourcePoint = new Point(100,100),
    TargetPoint = new Point(300,300)
};
//adding connector to the diagram
diagram.AddConnector(connector1);
//removing connector from the diagram using index.
diagram.Connectors.RemoveAt(0);

```

### Appearance

StrokeThickness, Stroke and style of the LineConnector and Decorators can be customized with a set of defined properties.

### Connector appearance

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

### XML

```

<!--creating connector instance-->
<control:SfDiagram.Connectors>
<control:ConnectorCollection>
<control:Connector SourcePoint="100,100" TargetPoint="300,300">
<!--defining connector styles-->
<control:Connector.Style>
<control:Style StrokeStyle="Dashed" StrokeWidth="4"/>
</control:Connector.Style>
</control:Connector>
</control:ConnectorCollection>
</control:SfDiagram.Connectors>

```

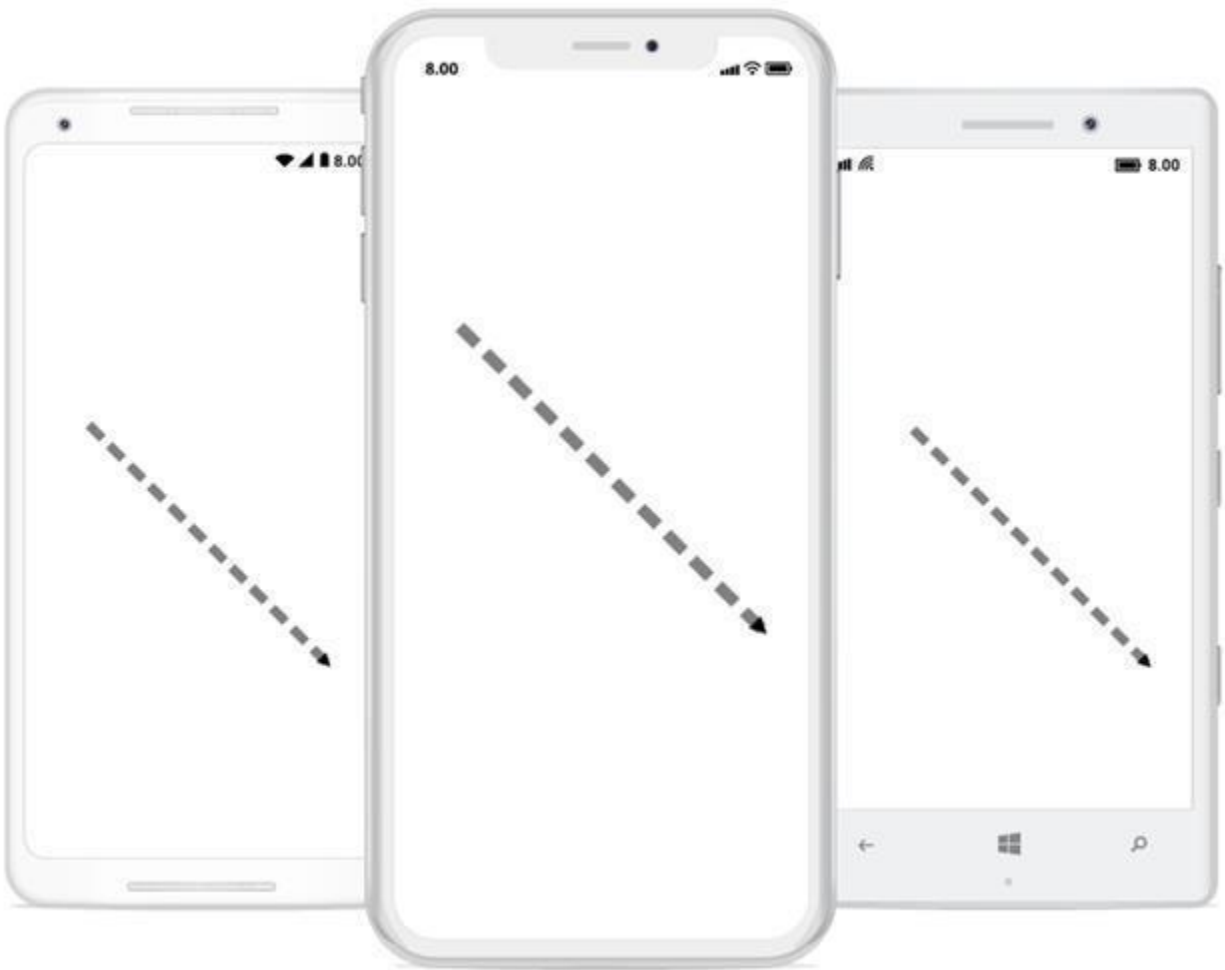
### C#

```

// creating connector instance
var connector1 = new Connector()
{
    SourcePoint = new Point(100,100),

```

```
TargetPoint = new Point(300, 300),
SegmentType = SegmentType.StraightSegment
};
//defining connector styles
connector1.Style = new Syncfusion.SfDiagram.XForms.Style()
{
    StrokeBrush = new SolidBrush(Color.Gray),
    StrokeStyle = StrokeStyle.Dashed,
    StrokeWidth = 4
};
diagram.AddConnector(connector1);
```



### Decorator appearance

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

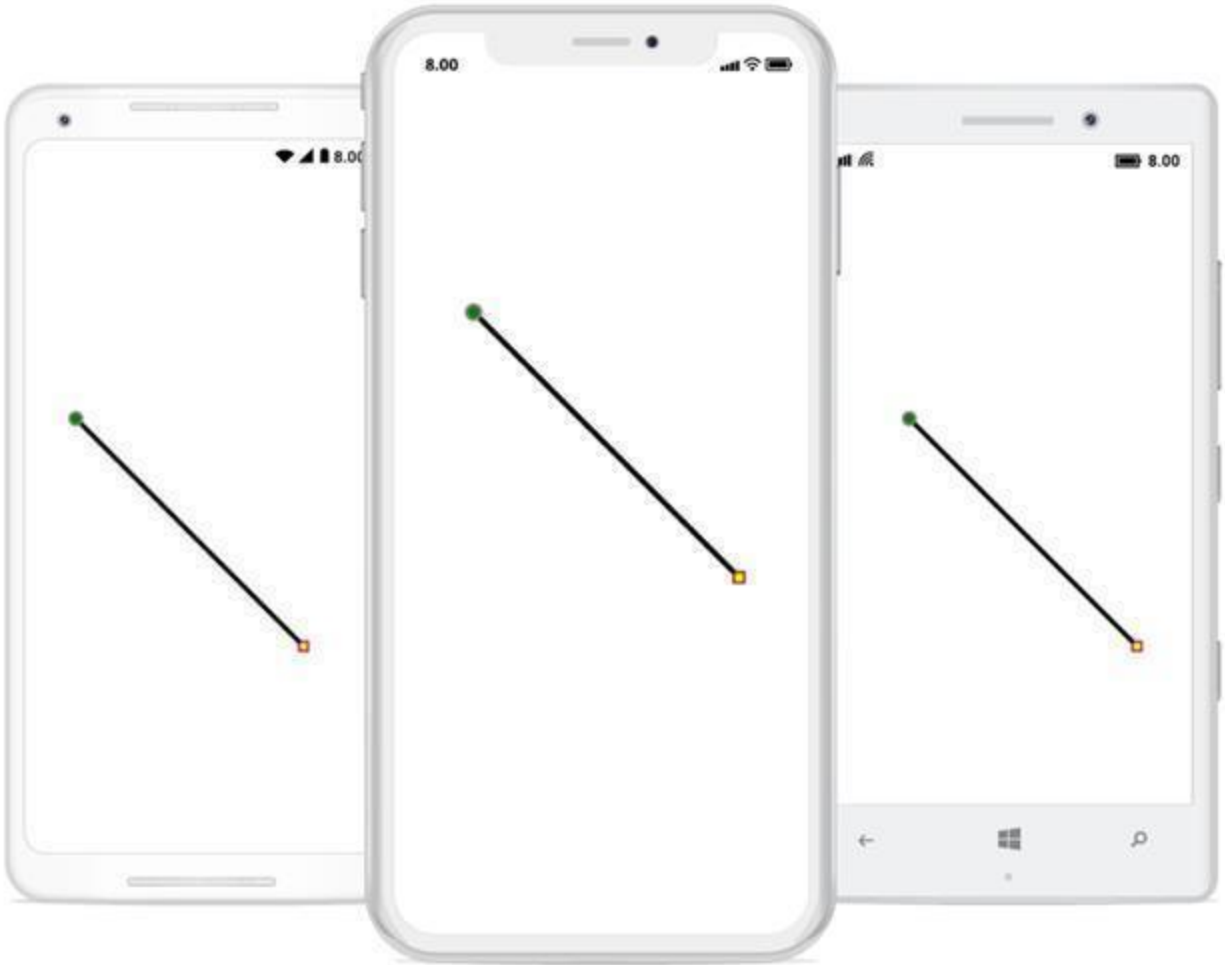
#### XML

```
<!--creating connector instance with decorator-->
<control:SfDiagram.Connectors>
  <control:ConnectorCollection>
    <control:Connector SourcePoint="100,100" TargetPoint="300,300"
      TargetDecoratorType="Diamond" SourceDecoratorType="Circle">
```

```
<!--defining decorator style for a connector-->
<control:Connector.SourceDecoratorStyle>
<control:DecoratorStyle Fill="Green" Stroke="Grey" StrokeThickness="5"
Width="12"/>
</control:Connector.SourceDecoratorStyle>
<!--defining decorator style for a connector-->
<control:Connector.TargetDecoratorStyle>
<control:DecoratorStyle Fill="Yellow" Stroke="Brown" StrokeThickness="4"
Width="10"/>
</control:Connector.TargetDecoratorStyle>
</control:Connector>
</control:ConnectorCollection>
</control:SfDiagram.Connectors>
```

## C#

```
// creating connector instance with decorator
var connector1 = new Connector()
{
    SourcePoint=new Point(100,100),
    TargetPoint= new Point(300,300),
    SegmentType= SegmentType.StraightSegment,
    SourceDecoratorType = DecoratorType.Circle,
    TargetDecoratorType = DecoratorType.Diamond
};
//defining decorator style for a connector
connector1.TargetDecoratorStyle = new DecoratorStyle()
{
    Fill = Color.Yellow,
    Stroke = Color.Brown,
    StrokeThickness = 4,
    Width = 12
};
// defining decorator style for a connector
connector1.SourceDecoratorStyle = new DecoratorStyle()
{
    Fill = Color.Green,
    Stroke = Color.Gray,
    StrokeThickness = 5,
    Width = 12
};
diagram.AddConnector(connector1);
```



## Annotations

Annotation is a block of text that can be displayed over a Node or Connector. Annotation is used to textually represent an object with a string that can be edited at run time.

You can add Multiple Labels to a Node/Connector.

### Create annotation

You can add an Annotation to a Node/Connector by defining the Annotation and adding that to the Annotations property of Node/Connector. The Content property of Annotation defines the object to be displayed. The following code illustrates how to create an Annotation.

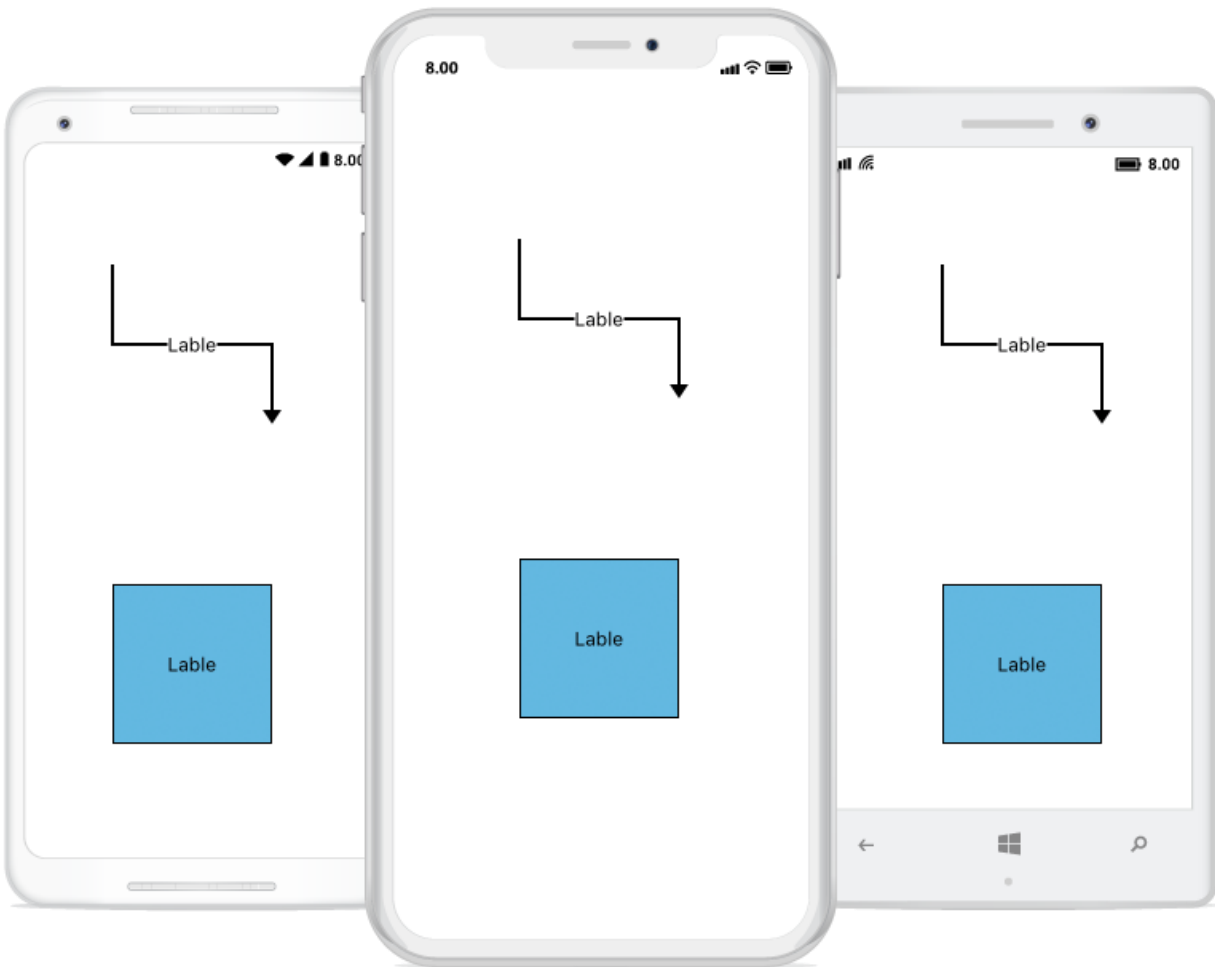
### XML

```
<!-- To Add Annotation for Node -->
<control:SfDiagram x:Name="diagram">
  <control:SfDiagram.Nodes>
    <control:Node OffsetX="100" OffsetY="300" Width="100" Height="100">
      <control:Node.Annotations>
        <control:AnnotationCollection>
          <control:Annotation Content="Lable"/>
        </control:AnnotationCollection>
      </control:Node.Annotations>
    </control:Node>
  </control:SfDiagram.Nodes>
</control:SfDiagram>
```

```
</control:Node>
</control:SfDiagram.Nodes>
<!-- To Add Annotation for Connector-->
<control:SfDiagram.Connectors>
<control:Connector SourcePoint="100,100" TargetPoint="200,200"
SegmentType="OrthoSegment">
<control:Connector.Annotations>
<control:AnnotationCollection>
<control:Annotation Content="Lable"/>
</control:AnnotationCollection>
</control:Connector.Annotations>
</control:Connector>
</control:SfDiagram.Connectors>
</control:SfDiagram>
```

## **C#**

```
//To Add Annotation for Node
Node node1 = new Node(100, 300, 100, 100);
node1.Annotations.Add(new Annotation() { Content = "Label" });
diagram.AddNode(node1);
// To Add Annotation for Connector
Connector Connector1 = new Connector();
Connector1.SourcePoint = new Point(100, 100);
Connector1.TargetPoint = new Point(200, 200);
Connector1.Annotations.Add(new Annotation() { Content = "Label"
});diagram.AddConnector(Connector1);
```



### Accessing an annotation from node and connector instance

User can able to access the node or connector annotation using annotation name. The following code illustrates how to access an Annotation.

#### C#

```
// Accessing Annotation
Connector Connector1 = new Connector();
Connector1.SourcePoint = new Point(100, 100);
Connector1.TargetPoint = new Point(300, 300);
var label = new Annotation() { Content = "Label" };
Connector1.Annotations.Add(label);
diagram.AddConnector(Connector1);
Connector1.Annotations.Remove(label);
```

### Remove an annotation

User can able to remove the annotation using index value. The following code illustrates how to remove an Annotation.

#### C#

```
//Remove Annotation using Index
```



```
Connector Connector1 = new Connector();
Connector1.SourcePoint = new Point(100, 100);
Connector1.TargetPoint = new Point(300, 300);
var label = new Annotation() { Content = "Label" };
Connector1.Annotations.Add(label);
diagram.AddConnector(Connector1);
Connector1.Annotations.RemoveAt(0);
```

### Annotation customization

User can able to customize the annotation using properties. The following code illustrates how to remove an Annotation.

#### XML

```
<!-- Annotation customization for node -->
<control:SfDiagram.Nodes>
<control:Node OffsetX="100" OffsetY="300" Width="100" Height="100">
<control:Node.Annotations>
<control:AnnotationCollection>
<control:Annotation Content="Lable" FontFamily="Arial" FontSize="14"/>
</control:AnnotationCollection>
</control:Node.Annotations>
</control:Node>
</control:SfDiagram.Nodes>
<!-- Annotation customization for connector -->
<control:SfDiagram.Connectors>
<control:Connector SourcePoint="100,100" TargetPoint="200,200"
SegmentType="OrthoSegment">
<control:Connector.Annotations>
<control:AnnotationCollection>
<control:Annotation Content="Lable" FontFamily="Arial" FontSize="14"/>
</ontrol:AnnotationCollection>
</control:Connector.Annotations>
</control:Connector>
</control:SfDiagram.Connectors>
```

#### C#

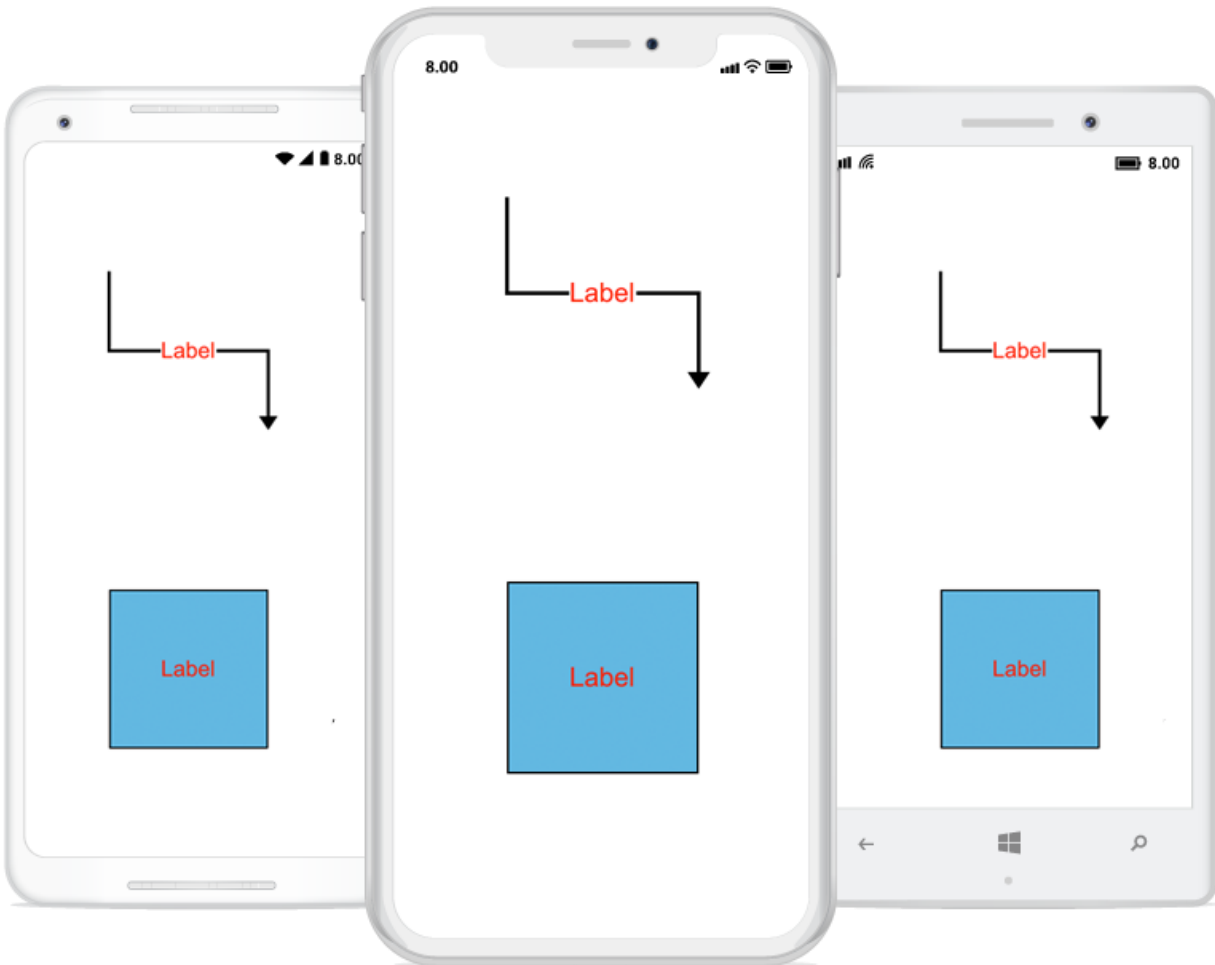
```
// Annotation customization for node
Node node1 = new Node(100, 300, 100, 100);
var label = new Annotation()
{
    Content = "Label",
    FontFamily = "Arial",
    FontSize = 14,
    TextBrush = new SolidBrush(Color.Red)
};
node1.Annotations.Add(label);
diagram.AddNode(node1);
// Annotation customization for connector
Connector Connector1 = new Connector();
Connector1.SourcePoint = new Point(100, 100);
Connector1.TargetPoint = new Point(200, 200);
var label = new Annotation()
{
```

```

Content = "Label",
FontFamily = "Arial",
FontSize = 14,
TextBrush = new SolidColorBrush(Color.Red)
} ;
Connector1.Annotations.Add(label);
diagram.AddConnector(Connector1);

```

The following output is displayed as result of the above code example.



### Alignment

Annotation can be aligned relative to the Node boundaries. It has Horizontal and Vertical Alignment settings. It is quite tricky when all four alignments are used together but gives you more control over alignment.

### XML

```

<!--Annotation alignment for Node-->
<control:SfDiagram x:Name="diagram">
  <control:SfDiagram.Nodes>
    <control:Node OffsetX="100" OffsetY="300" Width="100" Height="100">
      <control:Node.Annotations>

```

```

<control:AnnotationCollection>
<control:Annotation Content="Lable" HorizontalAlignment="Center"
VerticalAlignment="Top"/>
</control:AnnotationCollection>
</control:Node.Annotations>
</control:Node>
</control:SfDiagram.Nodes>
<!--Annotation alignment for connector -->
<control:SfDiagram.Connectors>
<control:Connector SourcePoint="100,100" TargetPoint="200,200"
SegmentType="OrthoSegment">
<control:Connector.Annotations>
<control:AnnotationCollection>
<control:Annotation Content="Lable" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</control:AnnotationCollection>
</control:Connector.Annotations>
</control:Connector>
</control:SfDiagram.Connectors>
</control:SfDiagram>

```

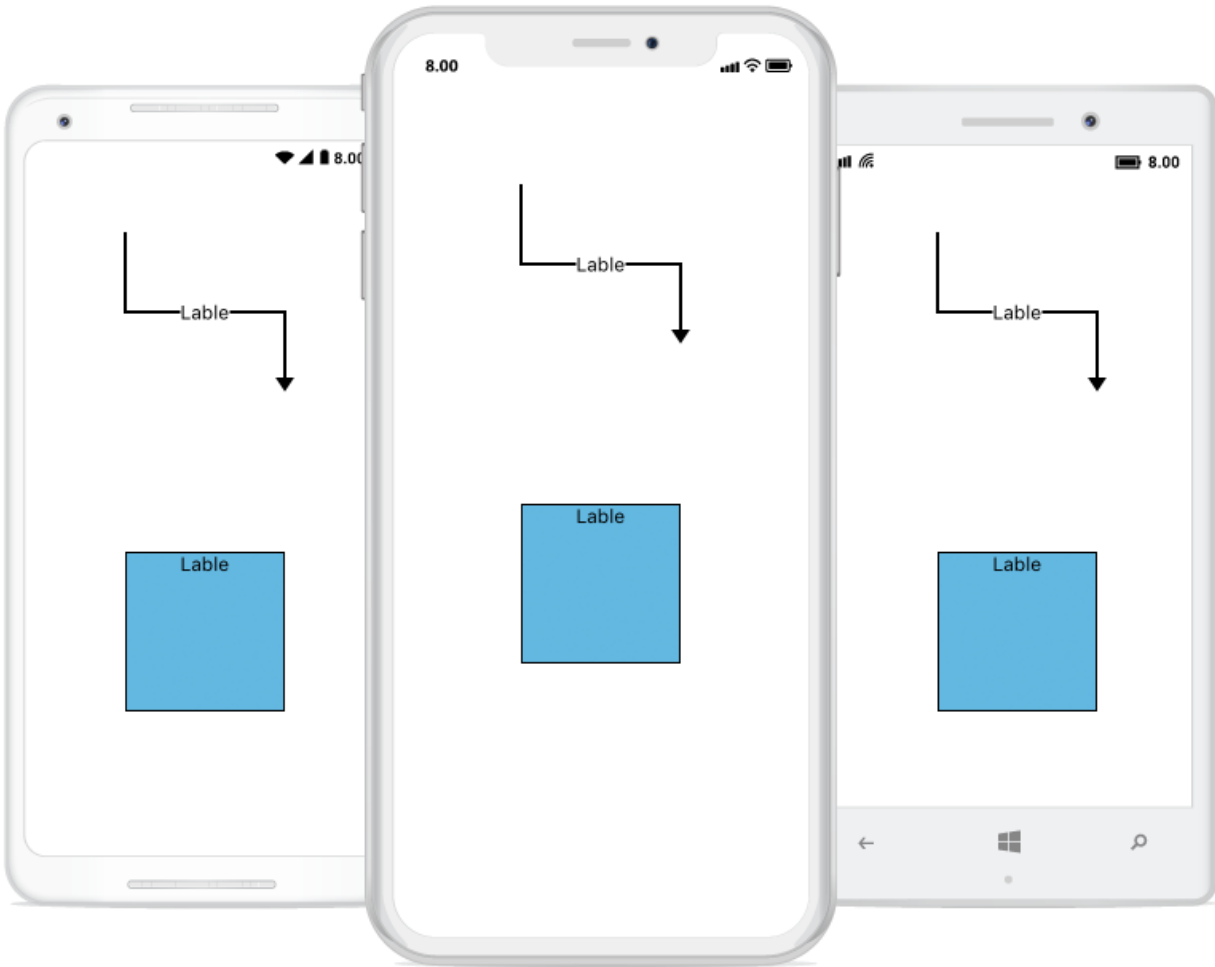
**C#**

```

// Annotation alignment for Node
Node node1 = new Node(100, 300, 100, 100);
Annotation label = new Annotation()
{
    Content = "Label",
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Top
};
node1.Annotations.Add(label);
diagram.AddNode(node1);
//Annotation alignment for Connector
Connector Connector1 = new Connector();
Connector1.SourcePoint = new Point(100, 100);
Connector1.TargetPoint = new Point(200, 200);
Connector1.Annotations.Add(new Annotation() { Content = "Label",
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center
});
diagram.AddConnector(Connector1);

```

The following output is displayed as result of the above code example.



## Port

SfDiagram provides support to define custom ports for making connections. When a Connector is connected between two Nodes, its end points are automatically docked to Node's nearest boundary, Port act as the connection points of node and allows to create connections with only specific points.

### Create ports for a node

To add a port, you need to define the port object and add it to Ports property of Node. The NodeOffsetX and NodeOffsetY property of Port accepts an object of fractions and used to determine the position of Ports. The following code illustrates how to add ports when initializing the Node.

The following code illustrates how to add ports to Node.

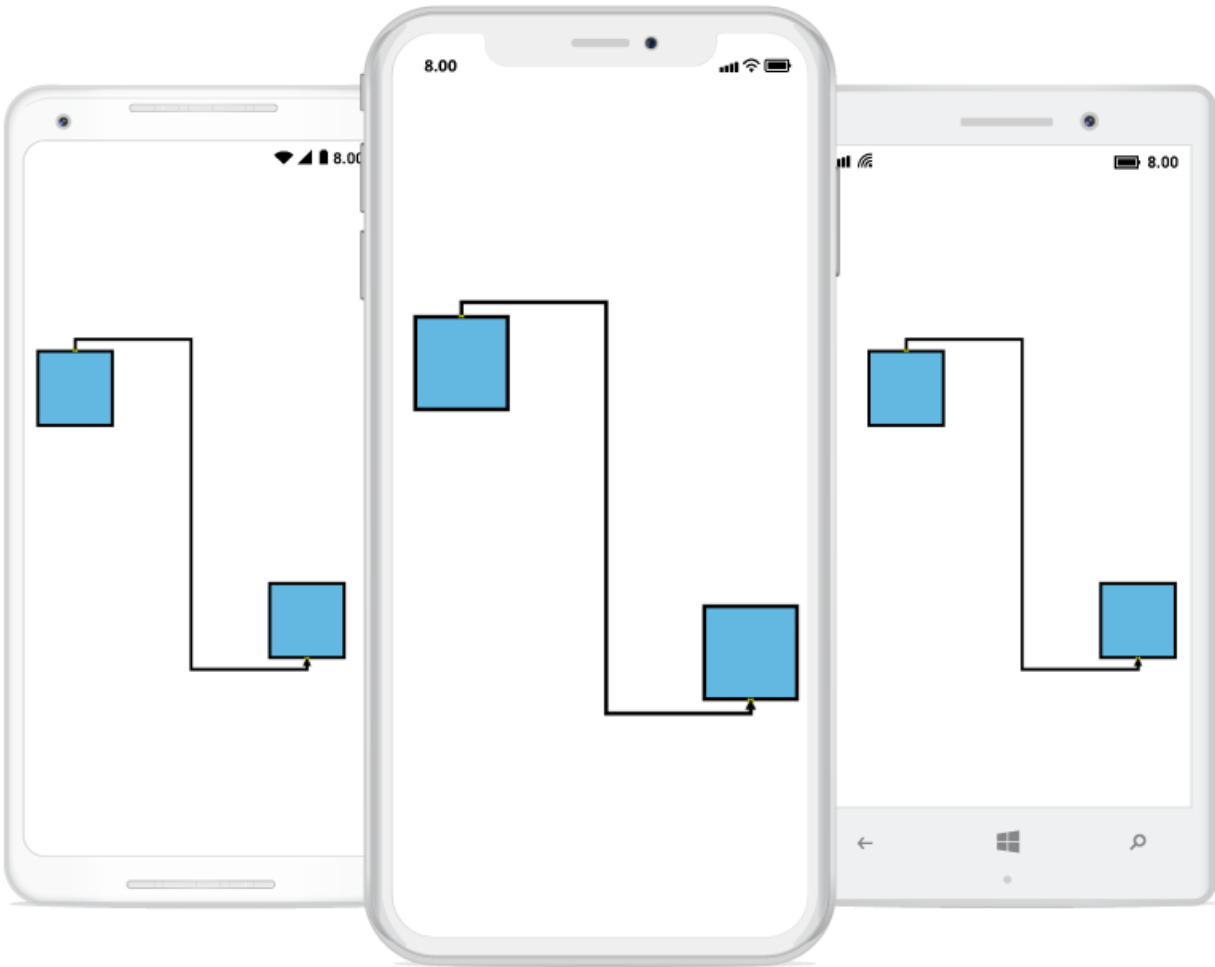
### XML

```
<!--creating node instance-->
<control:Node x:Name="node1" OffsetX="300" OffsetY="300" Width="100"
Height="100" ShapeType="Rectangle" >
  <!--creating port instance-->
  <control:Node.Ports >
    <control:PortCollection>
      <control:Port x:Name="port1" NodeOffsetX="0.5" NodeOffsetY="1"/>
    </control:PortCollection>
  </control:Node.Ports >
</control:Node>
```

```
</control:Node.Ports >  
</control:Node>
```

## C#

```
//Creating node instance  
Node node1 = new Node() { OffsetX = 300, OffsetY = 300, Width = 100, Height  
= 100, ShapeType = ShapeType.Rectangle };  
sfDiagram.AddNode(node1);  
Node node2 = new Node() { OffsetX = 600, OffsetY = 600, Width = 100, Height  
= 100, ShapeType = ShapeType.Rectangle };  
sfDiagram.AddNode(node2);  
// creating port instance  
Port port1 = new Port();  
port1.NodeOffsetX = 0.5;  
port1.NodeOffsetY = 0;  
Port port2 = new Port();  
port2.NodeOffsetX = 0.5;  
port2.NodeOffsetY = 1;  
//adding port to the node instance  
node1.Ports.Add(port1);  
node2.Ports.Add(port2);  
//creating and connecting the ports with connector  
Connector connector = new Connector() { SourceNode = node1, TargetNode =  
node2, SourcePort = port1, TargetPort = port2 };  
sfDiagram.AddConnector(connector);
```



### Accessing a port from the node instance

You can access the port by its index from the Ports property in a node.

The following code illustrates how to access a port from the node instance.

#### **C#**

```
Node node = sfDiagram.Nodes[0];  
//Accessing the port by index from the Port collection in node  
Port port = node.Ports[0];
```

### Remove a port from the node

To remove the port from a node. You can use Remove() method in Ports property in a node.

The following code illustrates how to remove a port from the node.

#### **C#**

```
Node node = sfDiagram.Nodes[0];  
//Accessing the port by index from the Port collection in node  
Port port = node.Ports[0];  
//Removing the port for the node  
node.Ports.Remove(port);
```

## Customization

You can customize a port by resizing it and applying style to it.

The following code illustrates how to customization a port.

### XML

```
<control:Node x:Name="node1" OffsetX="300" OffsetY="300" Width="100"
Height="100" ShapeType="Rectangle" >
  <!--creating port instance-->
  <control:Node.Ports >
    <control:PortCollection>
      <control:Port x:Name="port1" NodeOffsetX="0.5" NodeOffsetY="1" Height="10"
Width="10" ShapeType=" Circle">
        <control:Port.Style>
          <control:Style StrokeWidth="3" />
        </control:Port.Style>
      </control:Port>
    </control:PortCollection>
  </control:Node.Ports >
</control:Node>
```

### C#

```
//Creating a node
Node node1 = new Node() { OffsetX = 300, OffsetY = 300, Width = 100, Height
= 100, ShapeType = ShapeType.Rectangle };
sfDiagram.AddNode(node1);
// creating a port
Port port = new Port();
port.NodeOffsetX = 0.5;
port.NodeOffsetY = 1;
//adding the port to the node instance
node1.Ports.Add(port);
port.Height = 10;
port.Width = 10;
//Setting the shape type for the port
port.ShapeType = ShapeType.Circle;
//Creating style instance for the port
Style style = new Style();
style.StrokeWidth = 3;
style.Brush = new SolidBrush(Color.Black);
port.Style = style;
```

## Stencil

Stencil has a collection of Symbols. Stencil is used to clone the desired symbol by dragging it from the Stencil and dropping it into the SfDiagram. Each symbol can be grouped together by using the SymbolGroup .

### XML

```
<!-- Add namespace in xaml page -->
xmlns:sfDiagram="clr-
namespace:Syncfusion.SfDiagram.XForms;assembly=Syncfusion.SfDiagram.XForms"
```

## Add default shapes into stencil

The following example illustrates how to add the Symbol into a Collection:

### XML

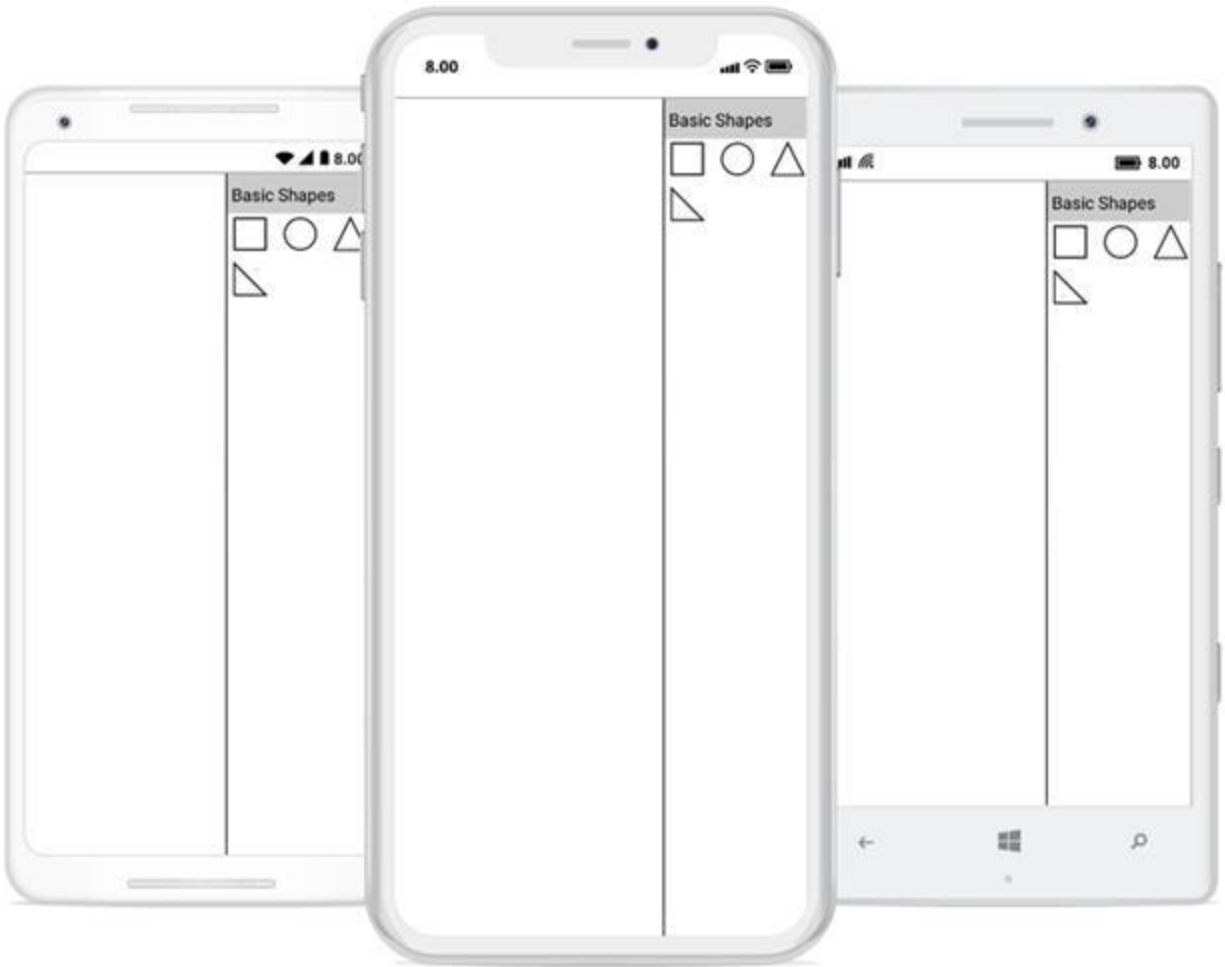
```
<ContentPage.Resources>
  <ResourceDictionary>
    <!-- Add Symbols into SymbolCollection -->
    <control:SymbolCollection x:Key="collection1">
      <control:Node Height="50" Width="50" ShapeType="Rectangle" />
      <control:Node Height="50" Width="50" ShapeType="Ellipse" />
      <control:Node Height="50" Width="50" ShapeType="Triangle" />
      <control:Node Height="50" Width="50" ShapeType="RightAngleTriangle" />
    </control:SymbolCollection>
    <!-- Add collection into SymbolGroup -->
    <control:SymbolGroups x:Key="groups">
      <control:SymbolGroup SymbolSource="{StaticResource collection1}"
        HeaderName="BasicShapes" />
    </control:SymbolGroups>
  </ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
  <Grid x:Name="grid">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="500"/>
      <ColumnDefinition Width="500"/>
    </Grid.ColumnDefinitions>
    <!-- Add SfDiagram and stencil in xaml page -->
    <control:SfDiagram x:Name="diagram">
    </control:SfDiagram>
    <control:Stencil x:Name="stencil" SymbolGroups="{StaticResource groups}" >
    </control:Stencil>
  </Grid>
</ContentPage.Content>
```

### C#

```
//Add Symbols into SymbolCollection
SymbolCollection Collection1 = new SymbolCollection();
Collection1.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.Rectangle});
Collection1.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.Ellipse});
Collection1.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.Triangle});
Collection1.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.RightAngleTriangle});
//Add collection into SymbolGroup
stencil.SymbolGroups.Add(new SymbolGroup() { SymbolSource = coll, HeaderName
= "BasicShapes" });
```

This Collection will be the SymbolSource to the Stencil. Based on the SymbolSource, the Stencil will populate the Symbols.





[Add custom shapes into stencil](#)

The following example illustrates how to add the custom shapes into a Collection:

#### **C#**

```
//Custom shapes
Pen pen = new Pen();
pen.StrokeBrush = new SolidBrush(Color.Red);
pen.StrokeWidth = 2;
SolidBrush brush = new SolidBrush(Color.Yellow);
pen.Brush = brush;
Pen pen1 = new Pen();
pen1.StrokeBrush = new SolidBrush(Color.Gray);
pen1.StrokeWidth = 2;
SolidBrush brush1 = new SolidBrush(Color.Blue);
brush1.FillColor = Color.Blue;
pen1.Brush = brush1;
Node custom = new Node();
SfGraphics graphics = new SfGraphics();
Pen pen2 = new Pen();
pen2.StrokeBrush = new SolidBrush(Color.Blue);
pen2.StrokeWidth = 2;
SolidBrush brush2 = new SolidBrush(Color.FromRgb(99, 184, 225));
```

```
brush2.FillColor = Color.FromRgb(99, 184, 225);
pen2.Brush = brush2;
graphics.DrawRectangle(pen2, new Xamarin.Forms.Rectangle(0, 0, 50, 50));
custom.UpdateSfGraphics(graphics);
Node custom1 = new Node();
SfGraphics grap4 = new SfGraphics();
SfGraphicsPath sfp4 = new SfGraphicsPath();
Pen pen14 = new Pen();
pen14.StrokeBrush = new SolidBrush(Color.Blue);
pen14.StrokeWidth = 2;
SolidBrush brush14 = new SolidBrush(Color.Transparent);
brush14.FillColor = Color.Transparent;
pen14.Brush = brush14;
List<Point> coll4 = new List<Point>();
coll4.Add(new Point(0, 12));
coll4.Add(new Point(12, 12));
coll4.Add(new Point(12, 6));
coll4.Add(new Point(12, 42));
coll4.Add(new Point(12, 30));
coll4.Add(new Point(0, 30));
grap4.DrawLine(pen14, coll4.ToArray());
sfp4.MoveTo(12, 6);
sfp4.CubicTo(12, 6, 38, 20, 12, 36);
grap4.DrawPath(sfp4);
custom1.UpdateSfGraphics(grap4);
Node custom2 = new Node();
SfGraphics grap5 = new SfGraphics();
SfGraphicsPath sfp5 = new SfGraphicsPath();
List<Point> pointscol5 = new List<Point>();
pointscol5.Add(new Point(0, 15));
pointscol5.Add(new Point(15, 15));
pointscol5.Add(new Point(15, 0));
pointscol5.Add(new Point(30, 15));
pointscol5.Add(new Point(15, 30));
pointscol5.Add(new Point(15, 15));
sfp5.AddLines(pointscol5.ToArray());
grap5.DrawPath(sfp5);
custom2.UpdateSfGraphics(grap5);
Node custom3 = new Node();
Pen pen4 = new Pen();
pen4.StrokeBrush = new SolidBrush(Color.Blue);
pen4.StrokeWidth = 2;
SolidBrush brush4 = new SolidBrush(Color.White);
brush4.FillColor = Color.White;
pen4.Brush = brush4;
SfGraphics grap6 = new SfGraphics();
SfGraphicsPath sfp6 = new SfGraphicsPath();
List<Point> pointscol6 = new List<Point>();
pointscol6.Add(new Point(0, 15));
pointscol6.Add(new Point(15, 15));
pointscol6.Add(new Point(15, 0));
pointscol6.Add(new Point(30, 15));
pointscol6.Add(new Point(15, 30));
pointscol6.Add(new Point(15, 15));
sfp6.AddLines(pointscol5.ToArray());
grap6.DrawPath(sfp6);
grap6.DrawEllipse(pen4, new Xamarin.Forms.Rectangle(30, 12, 5, 5));
```

```

custom3.UpdateSfGraphics(grap6);
SymbolCollection CustomShapeCollection = new SymbolCollection();
col4.Add(custom);
col4.Add(custom1);
col4.Add(custom2);
col4.Add(custom3);
//Add custom shapes into group
stencil.SymbolGroups.Add(new SymbolGroup() { SymbolSource =
CustomShapeCollection , HeaderName = "Custom Shapes" });

```



### Add category heading text

We can able to add category of symbol group heading in stencil. The following example illustrates how to add category heading text in stencil.

#### XML

```

<ResourceDictionary>
  <!--SymbolCollection1-->
  <control:SymbolCollection x:Key="collection1">
    <control:Node Height="50" Width="50" ShapeType="Rectangle" />
    <control:Node Height="50" Width="50" ShapeType="Ellipse" />
    <control:Node Height="50" Width="50" ShapeType="Triangle" />
  </control:SymbolCollection>

```

```

<control:Node Height="50" Width="50" ShapeType="RightAngleTriangle" />
</control:SymbolCollection>
<!--SymbolCollection2-->
<control:SymbolCollection x:Key="collection2">
<control:Node Height="50" Width="50" ShapeType="RoundedRectangle" />
<control:Node Height="50" Width="50" ShapeType="Rectangle" />
<control:Node Height="50" Width="50" ShapeType="Diamond" />
<control:Node Height="50" Width="50" ShapeType="Parallelogram" />
</control:SymbolCollection>
<!--SymbolCollection3-->
<control:SymbolCollection x:Key="collection3">
<control:Connector SourcePoint="50,50" TargetPoint="100,100"
SegmentType="OrthoSegment" />
<control:Connector SourcePoint="50,50" TargetPoint="100,100"
SegmentType="StraightSegment" />
</control:SymbolCollection>
<!--Add category of symbolgroup with heading text -->
<control:SymbolGroups x:Key="groups">
<control:SymbolGroup SymbolSource="{StaticResource collection1}"
HeaderName="BasicShapes" />
<control:SymbolGroup SymbolSource="{StaticResource collection2}"
HeaderName="Flow Chart" />
<control:SymbolGroup SymbolSource="{StaticResource collection3}"
HeaderName="Connectors" />
</control:SymbolGroups>
</ResourceDictionary>

```

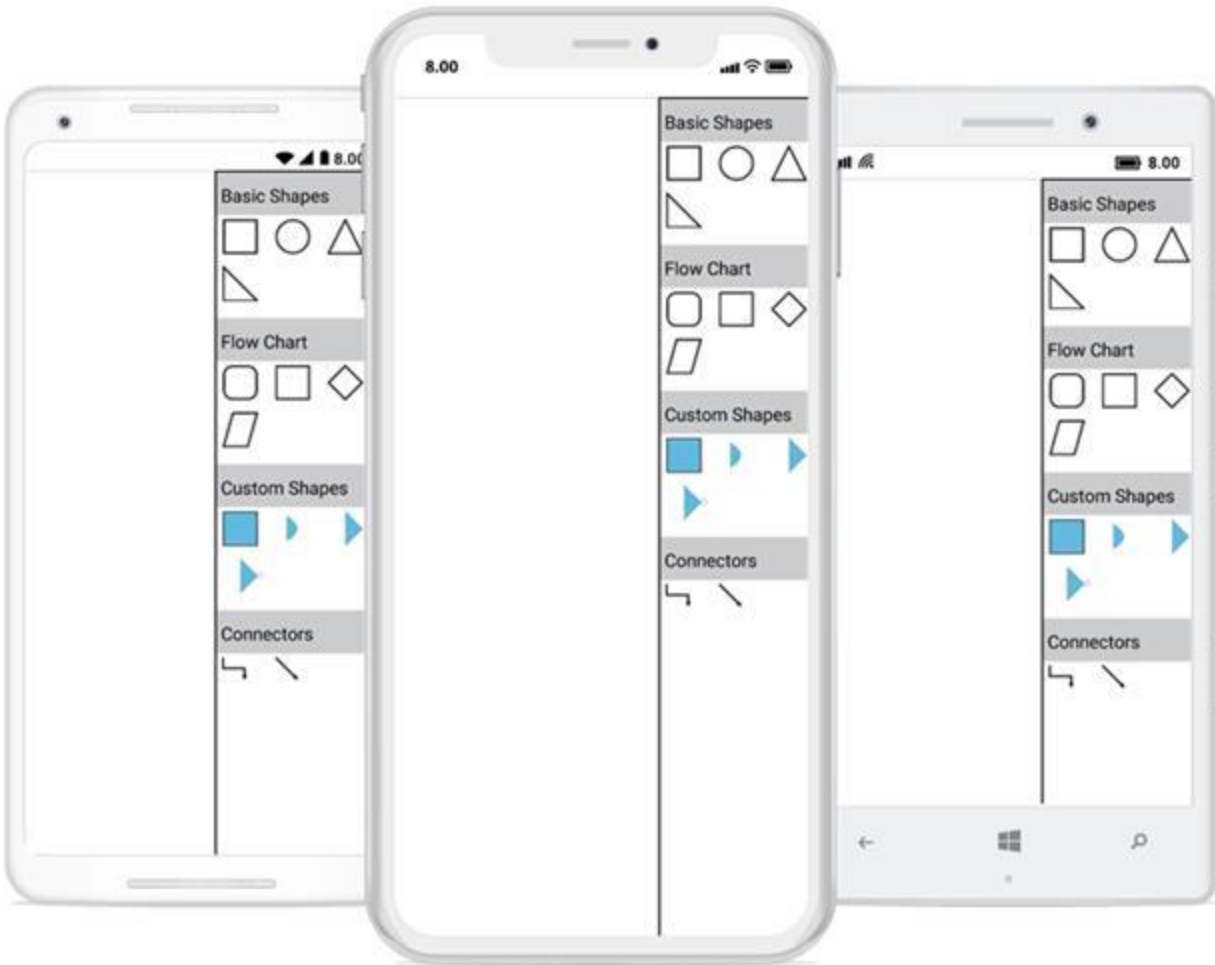
## C#

```

//SymbolCollection1
SymbolCollection coll = new SymbolCollection();
coll.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.Rectangle, Style = new Syncfusion.SfDiagram.XForms.Style() { Brush =
new SolidColorBrush(Color.White), StrokeBrush = new SolidColorBrush(Color.Gray) }
});
coll.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.Ellipse, Style = new Syncfusion.SfDiagram.XForms.Style() { Brush =
new SolidColorBrush(Color.White), StrokeBrush = new SolidColorBrush(Color.Gray) } });
coll.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.Triangle, Style = new Syncfusion.SfDiagram.XForms.Style() { Brush =
new SolidColorBrush(Color.White), StrokeBrush = new SolidColorBrush(Color.Gray) }
});
coll.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.RightAngleTriangle, Style = new
Syncfusion.SfDiagram.XForms.Style() { Brush = new SolidColorBrush(Color.White),
StrokeBrush = new SolidColorBrush(Color.Gray) } });
//SymbolCollection2
SymbolCollection coll1 = new SymbolCollection();
coll1.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.RoundedRectangle, Style = new Syncfusion.SfDiagram.XForms.Style()
{ Brush = new SolidColorBrush(Color.White), StrokeBrush = new
SolidColorBrush(Color.Gray) } });
coll1.Add(new Node() { Width = 50, Height = 50, ShapeType =
ShapeType.Rectangle, Style = new Syncfusion.SfDiagram.XForms.Style() { Brush =
new SolidColorBrush(Color.White), StrokeBrush = new SolidColorBrush(Color.Gray) }
});

```

```
coll1.Add(new Node() { Width = 50, Height = 50, ShapeType =  
ShapeType.Diamond, Style = new Syncfusion.SfDiagram.XForms.Style() { Brush =  
new SolidBrush(Color.White), StrokeBrush = new SolidBrush(Color.Gray) } });  
coll1.Add(new Node() { Width = 50, Height = 50, ShapeType =  
ShapeType.Parallelogram, Style = new Syncfusion.SfDiagram.XForms.Style() {  
Brush = new SolidBrush(Color.White), StrokeBrush = new  
SolidBrush(Color.Gray) } });  
//SymbolCollection3  
SymbolCollection con1 = new SymbolCollection();  
con1.Add(new Connector() { SegmentType = SegmentType.OrthoSegment,  
SourcePoint = new Point(0, 0), TargetPoint = new Point(50, 50) });  
con1.Add(new Connector() { SegmentType = SegmentType.StraightSegment,  
SourcePoint = new Point(0, 0), TargetPoint = new Point(50, 50) });  
//Add category of symbol group with heading text  
stencil.SymbolGroups.Add(new SymbolGroup() { SymbolSource = coll, HeaderName  
= "BasicShapes" });  
stencil.SymbolGroups.Add(new SymbolGroup() { SymbolSource = coll1,  
HeaderName = "Flow Chart" });  
stencil.SymbolGroups.Add(new SymbolGroup() { SymbolSource = coll2,  
HeaderName = "CustomShapes" });  
stencil.SymbolGroups.Add(new SymbolGroup() { SymbolSource = con1, HeaderName  
= "Connectors" });  
diagram.Stencil=stencil;
```



## Layouts

SfDiagram provides support to auto-arrange the nodes in the Diagram area that is referred as **Layout**.

We have explained the Automatic Layout with Employee class and DataSourceSettings. The followings are initial steps for all the Layout.

### Create class for data

Now, you have to create a class, Employee with properties to store the employee's information like Team, Role, ID, reporting person ID, etc. You also have to create a collection that stores a collection of the employees.

### C#

```
//Employee Business Object
public class Employee
{
    public string Team { get; set; }
    public string Role { get; set; }
    public int EmployeeId { get; set; }
}

//Employee Collection
public class Employees : ObservableCollection<Employee>
```

```
{
}
```

Initialize data source settings

#### XML

```
<!--Initializes the DataSourceSettings -->
<control:SfDiagram.DataSourceSettings>
<control:DataSourceSettings x:Key="DataSourceSettings" ParentId="Team"
Id="EmployeeId"
DataSource="{StaticResource Employees}" />
</control:SfDiagram.DataSourceSettings>
```

#### C#

```
// Initializes the DataSourceSettings
DataSourceSettings setting = new DataSourceSettings();
setting.DataSource = employee;
setting.Id = "ID";
setting.ParentId = "ReportingId";
diagram.DataSourceSettings = setting;
```

Organization layout

An organizational chart is a Diagram that displays the structure of an organization and relationships. To create an organizational chart, Type should be set as LayoutType.Organization. The following code example illustrates how to create an organizational chart.

#### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<local:Employees x:Key="Employees">
<!--Employee Collection -->
<local:Employee EmployeeId="0" Role="Project Management" />
<local:Employee EmployeeId= "1" Role= "R and D Team" Team= "0"/>
<local:Employee EmployeeId= "3" Role= "Philosophy" Team= "1"/>
<local:Employee EmployeeId= "4" Role= "Organization" Team= "1"/>
<local:Employee EmployeeId= "5" Role= "Technology" Team= "1"/>
<local:Employee EmployeeId= "7" Role= "Funding" Team= "1"/>
<local:Employee EmployeeId= "8" Role= "Resource Allocation" Team= "1"/>
<local:Employee EmployeeId= "9" Role= "Targeting" Team= "1"/>
<local:Employee EmployeeId= "11" Role= "Evaluation" Team= "1"/>
<local:Employee EmployeeId= "156" Role= "HR Team" Team= "0"/>
<local:Employee EmployeeId= "13" Role= "Recruitment" Team= "156"/>
<local:Employee EmployeeId= "113" Role= "Training" Team= "156"/>
<local:Employee EmployeeId= "112" Role= "Employee Relation" Team= "156"/>
<local:Employee EmployeeId= "14" Role= "Record Keeping" Team= "156"/>
<local:Employee EmployeeId= "17" Role= "Production and Sales Team" Team=
"0"/>
<local:Employee EmployeeId= "119" Role= "Design" Team= "17"/>
<local:Employee EmployeeId= "19" Role= "Operation" Team= "17"/>
<local:Employee EmployeeId= "20" Role= "Support" Team= "17"/>
<local:Employee EmployeeId= "21" Role= "Quality Assurance" Team= "17"/>
<local:Employee EmployeeId= "23" Role= "Customer Interaction" Team= "17"/>
```

```

<local:Employee EmployeeId= "24" Role= "Support and Maintenance" Team=
"17"/>
<local:Employee EmployeeId= "25" Role= "Task Coordination" Team= "17"/>
</local:Employees>
<control:DirectedTreeLayout x:Key="TreeLayout" TreeOrientation="TopToBottom"
Type="Organization"/>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<Grid x:Name="grid">
<control:SfDiagram x:Name="diagram" >
<!--Initializes the DataSourceSettings-->
<control:SfDiagram.DataSourceSettings>
<control:DataSourceSettings x:Key="DataSourceSettings" ParentId="Team"
Id="EmployeeId"
DataSource="{StaticResource Employees}" />
</control:SfDiagram.DataSourceSettings>
<!--Initializes the Layout-->
<control:SfDiagram.LayoutManager>
<control:LayoutManager x:Key="LayoutManager"
Layout="{StaticResource TreeLayout}" />
</control:SfDiagram.LayoutManager> </control:SfDiagram>
</Grid>
</ContentPage.Content>

```

## C#

```

//Employee Collection
ObservableCollection<Employee> employee = new
ObservableCollection<Employee>();
employee.Add(new Employee { EmployeeId = 0, Role = "Project Management" });
employee.Add(new Employee { EmployeeId = 1, Role = "R and D Team", Team =
"0" });
employee.Add(new Employee { EmployeeId = 3, Role = "Philosophy", Team = "1"
});
employee.Add(new Employee { EmployeeId = 4, Role = "Organization", Team =
"1" });
employee.Add(new Employee { EmployeeId = 5, Role = "Technology", Team = "1"
});
employee.Add(new Employee { EmployeeId = 7, Role = "Funding", Team = "1"
});
employee.Add(new Employee { EmployeeId = 8, Role = "Resource Allocation"
, Team = "1" });
employee.Add(new Employee { EmployeeId = 9, Role = "Targeting", Team = "1"
});
employee.Add(new Employee { EmployeeId = 11, Role = "Evaluation", Team =
"1" });
employee.Add(new Employee { EmployeeId = 156, Role = "HR Team", Team = "0"
});
employee.Add(new Employee { EmployeeId = 13, Role = "Recruitment", Team =
"156" });
employee.Add(new Employee { EmployeeId = 113, Role = "Training", Team =
"156" });
employee.Add(new Employee { EmployeeId = 112, Role = "Employee Relation"
, Team = "156" });

```



```

employee.Add(new Employee { EmployeeId = 14 , Role = "Record Keeping", Team
= "156" });
employee.Add(new Employee { EmployeeId = 17 , Role = "Production and Sales
Team" ,Team = "0" });
employee.Add(new Employee { EmployeeId = 119, Role = "Design", Team = "17"
});
employee.Add(new Employee { EmployeeId = 19 , Role = "Operation", Team =
"17" });
employee.Add(new Employee { EmployeeId = 20 , Role = "Support" ,Team = "17"
});
employee.Add(new Employee { EmployeeId = 21 , Role = "Quality Assurance"
,Team = "17" });
employee.Add(new Employee { EmployeeId = 23 , Role = "Customer Interaction"
,Team = "17" });
employee.Add(new Employee { EmployeeId = 24 , Role = "Support and
Maintenance" ,Team = "17" });
employee.Add(new Employee { EmployeeId = 25 , Role = "Task Coordination",
Team = "17" });
//Set parentId and id for DataSourceSettings
DataSourceSettings setting = new DataSourceSettings();
setting.DataSource = employee;
setting.Id = "EmployeeId";
setting.ParentId = "Team";
diagram.DataSourceSettings = setting;
diagram.LayoutManager = new LayoutManager() { Layout = new
DirectedTreeLayout() { TreeOrientation = TreeOrientation.TopToBottom, Type =
LayoutType.Organization } };

```

Organizational chart layout starts parsing from root and iterate through all its child elements.

“BeginNodeLayout” event provides necessary information of a Node’s children and the way to arrange (Orientation, Type etc.) them.

### BeginNodeLayout

User can change ChartType and Orientation by using BeginNodeLayout event of the SfDiagram. This event will fire for each Node added in Layout when the layout is getting updated. Default ChartType is Alternate and default orientation is Vertical. The following code example illustrates how to register an event and how to change ChartType and orientation.

### C#

```

// Registering an event
diagram.BeginNodeLayout += Diagram_BeginNodeLayout;
private void Diagram_BeginNodeLayout(object sender, BeginNodeLayoutEventArgs
args)
{
    if (!args.HasSubTree)
    {
        args.Type = ChartType.Left;
        args.Orientation = Orientation.Vertical;
    }
}

```

### BeginNodeRender

User can change node content using BeginNodeRender event of the SfDiagram. This event will fire for each Node added in Layout when the layout is getting updated.

#### C#

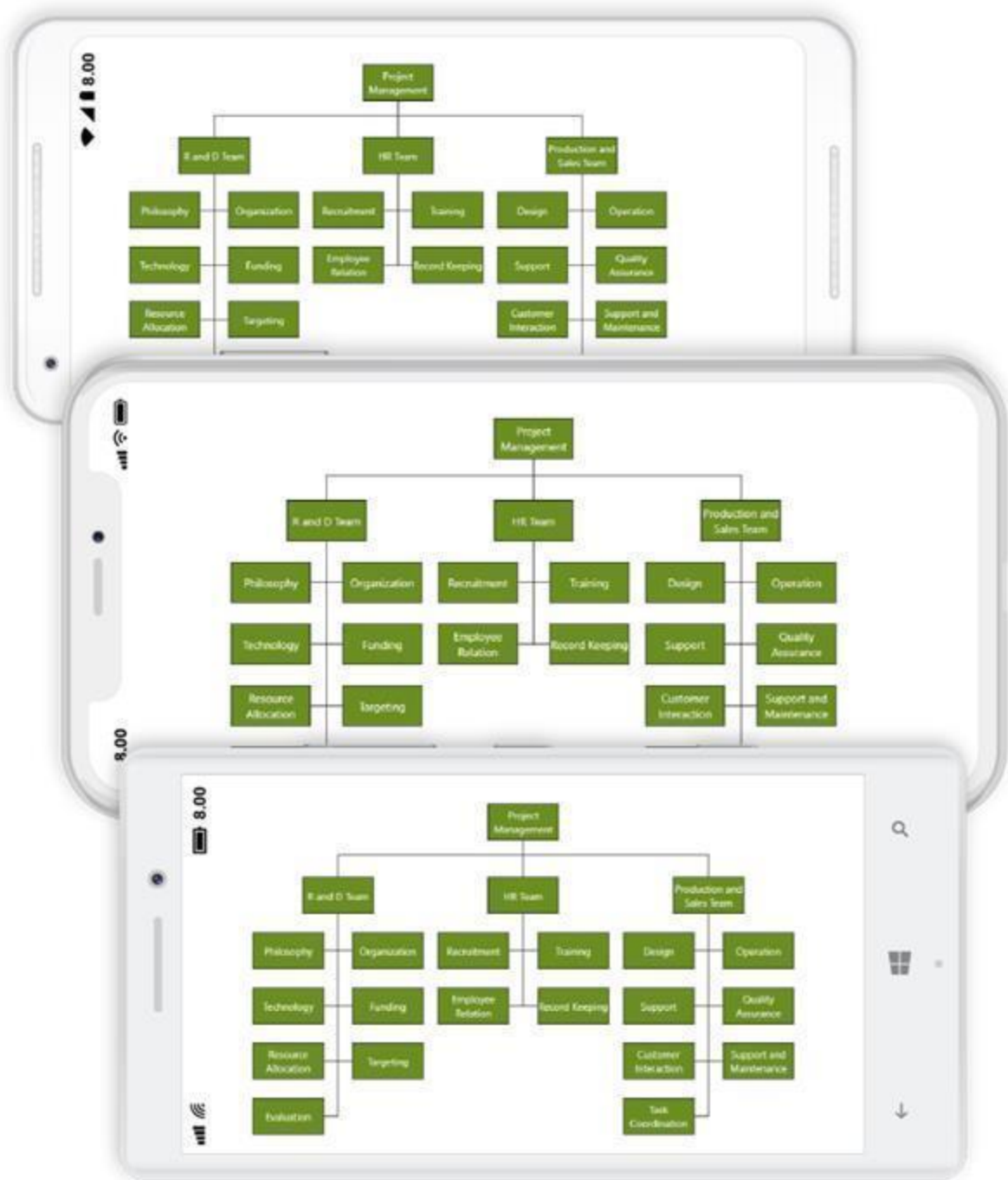
```
// Registering an event
diagram.BeginNodeRender += Diagram_BeginNodeRender;
private void Diagram_BeginNodeRender(object sender, BeginNodeRenderEventArgs args)
{
    Node node = args.Item as Node;
    node.Width = 150;
    node.Height = 60;
    node.ShapeType = ShapeType.Rectangle;
    Syncfusion.SfDiagram.XForms.Style style = new
    Syncfusion.SfDiagram.XForms.Style() {Brush = new SolidColorBrush(Color.OliveDrab)
    };
    node.Style = style;
    AnnotationCollection annotations = new AnnotationCollection();
    Annotation annotation = new Annotation()
    {
        Content = (node.Content as Employee).Role,
        HorizontalAlignment = HorizontalAlignment.Center,
        VerticalAlignment = VerticalAlignment.Center,
        TextBrush = new SolidColorBrush(Color.White)
    };
    annotations.Add(annotation);
    node.Annotations = annotations;
}
```

### Expand and collapse node

User can able to expand and collapse the parent node using NodeClicked event of the SfDiagram. This event will fire when click node in Layout.

#### C#

```
// Registering an event
diagram.BeginNodeLayout += Diagram_BeginNodeLayout;
void Diagram_NodeClicked(object sender, NodeClickedEventArgs args)
{
    if ((args.Item.Content as Employee).HasChild && args.Item.IsExpanded)
    {
        args.Item.IsExpanded = false;
    }
    else if ((args.Item.Content as Employee).HasChild && !args.Item.IsExpanded)
    {
        args.Item.IsExpanded = true;
    }
}
```



**Note:** Diagram supports expand and collapse in Xamarin.Forms.Android and Xamarin.Forms.iOS alone.

#### [Drag-and-drop support for directed tree layout](#)

It is easier to drag a child or parent node to some other node in the directed tree layout. The following code shows how to enable draggable option in layout.

#### C#

```
(diagram.LayoutManager.Layout as DirectedTreeLayout).IsDraggable = true;
```

The following code shows how to add the child of dropped node while dragging the node using the `LayoutNodeDropped` event.

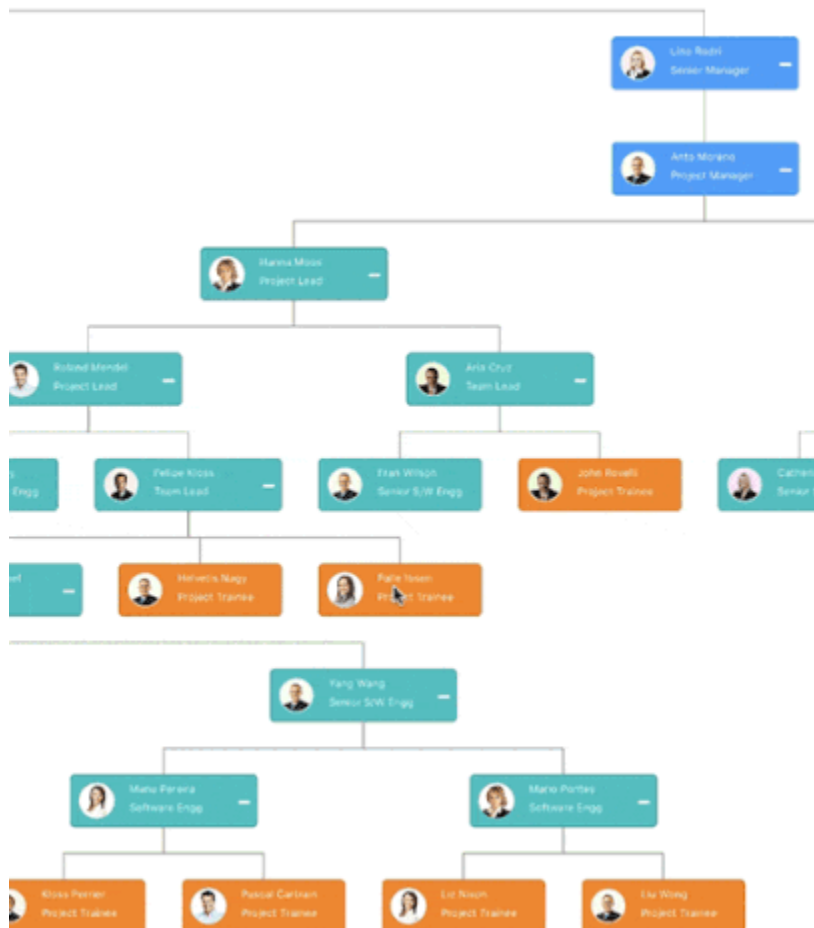
### C#

```
// Registering an event
diagram.LayoutNodeDropped += Diagram_OnLayoutNodeDropped;
//Define the LayoutNodeDropped event
private void Diagram_OnLayoutNodeDropped(object sender,
LayoutNodeDroppedEventArgs args)
{
    Node draggedNode = args.DraggedItem as Node;
    Node droppedNode = args.DroppedItem as Node;
    bool contain = true;
    if (draggedNode != RootNode && draggedNode != droppedNode)
    {
        Node ParentNode = GetParent((droppedNode.Content as
DiagramEmployee).ReportingPerson);
        do
        {
            if (ParentNode != draggedNode)
            {
                contain = false;
            }
            else
            {
                contain = true;
                break;
            }
            ParentNode = GetParent((ParentNode.Content as
DiagramEmployee).ReportingPerson);
        } while (ParentNode != RootNode);
        if (!contain)
        {
            List<Connector> connectors = draggedNode.InConnectors as List<Connector>;
            Connector con; bool hasChild = false;
            for (int i = 0; i < connectors.Count; i++)
            {
                con = connectors[i];
                con.SourceNode = droppedNode;
                hasChild = true;
            }
            if (hasChild)
            {
                Node PrevParentNode = GetParent((draggedNode.Content as
DiagramEmployee).ReportingPerson);
                if (PrevParentNode != null && PrevParentNode.OutConnectors.Count == 0)
                {
                    (PrevParentNode.Content as DiagramEmployee).HasChild = false;
                    UpdateTemplate(PrevParentNode, "-");
                }
                DiagramEmployee ParentEmployee = (droppedNode.Content as DiagramEmployee);
                (draggedNode.Content as DiagramEmployee).ReportingPerson =
                ParentEmployee.Name;
                ParentEmployee.HasChild = true;
            }
        }
    }
}
```

```

UpdateTemplate(droppedNode, "-");
}
droppedNode.IsExpanded = true;
diagram.LayoutManager.Layout.UpdateLayout();
}
}
}
private Node GetParent(string parentId)
{
    foreach (Node node in diagram.Nodes)
    {
        if ((node.Content as DiagramEmployee).Name == parentId)
        {
            return node;
        }
    }
    return RootNode;
}

```



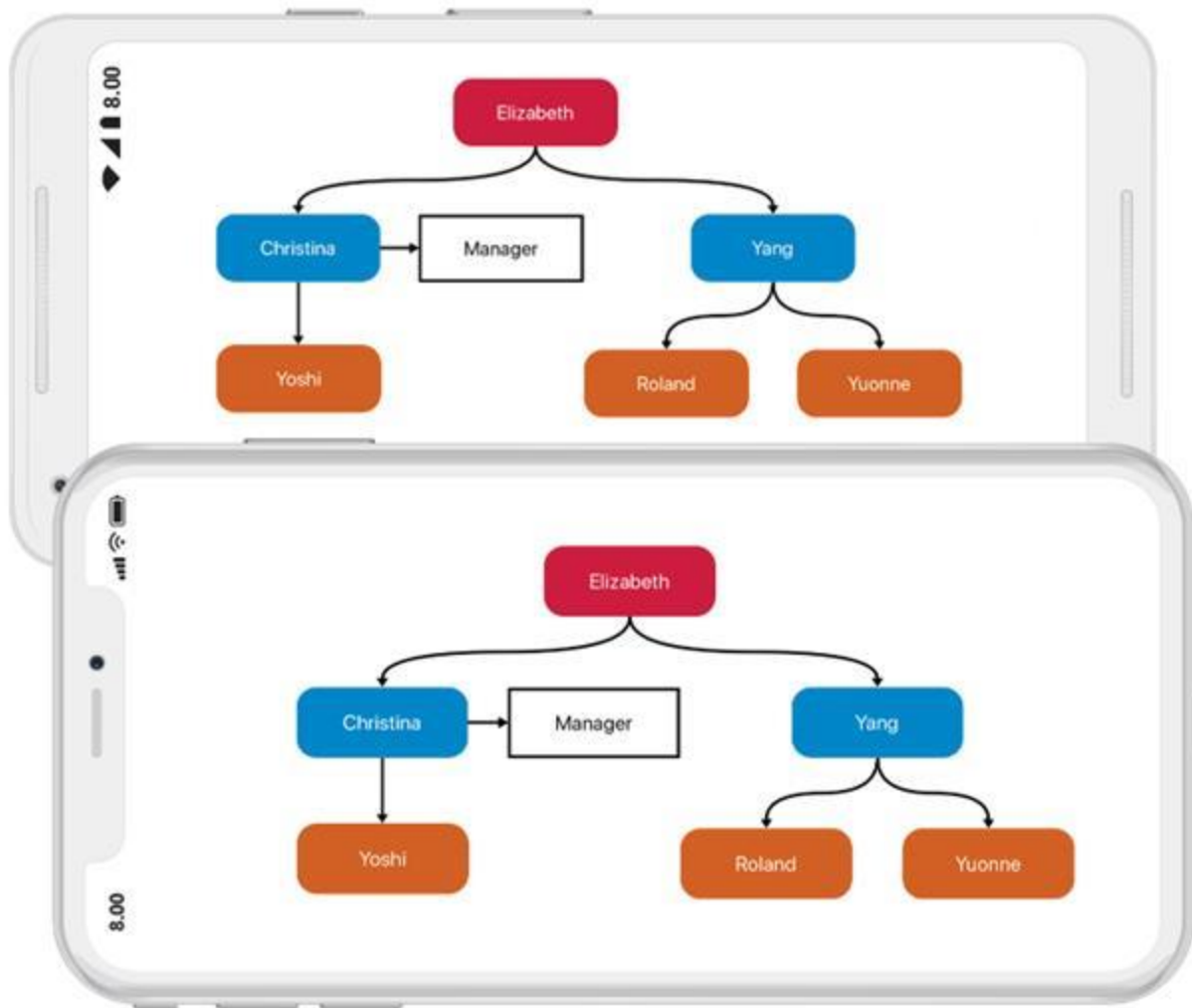
**Note:** Diagram supports drag and drop in Xamarin.Forms.Android and Xamarin.Forms.iOS alone.

### Layout sibling spacing

It is easier to provide spacing between sibling nodes of any branch on the directed tree layout. Nodes can also be excluded from the layout. You can provide space for each node by customizing the “SiblingSpace” property of the node. The following code illustrates how to add space to nodes using sibling spacing class instance.

#### C#

```
//Define the sibling spacing for node
private void Diagram_BeginNodeRender(object sender, BeginNodeRenderEventArgs
args)
{
    Node node = (args.Item as Node);
    node.ShapeType = ShapeType.RoundedRectangle;
    node.Width = 90;
    node.Height = 50;
    SiblingSpace siblingSpacing = new SiblingSpace(100,100);
    node.SiblingSpace = siblingSpacing;
    node.Annotations.Add(new Annotation() { Content = ((args.Item as
Node).Content as Employee).Name });
}
```



**Note:** Diagram supports layout sibling with spacing in Xamarin.Forms.Android and Xamarin.Forms.iOS alone.

### Mind map layout

A mind map is a diagram used to visually organize the information. It is hierarchical and shows relationship of the whole idea. It consists of a central ideas, related ideas, and lines connecting them together. It is used for brainstorming, planning, and information gathering etc. The following code example illustrates how to create and mind map layout:

#### C#

```

List<Color> FColor = new List<Color>();
List<Color> SColor = new List<Color>();
Random random = new Random();
int index;
int width=150;int height=75;
//Alternate color selection for child nodes
SColor.Add(Color.FromHex("#d1afdf"));
  
```

```

SColor.Add(Color.FromHex("#90C8C2"));
SColor.Add(Color.FromHex("#8BC1B7"));
SColor.Add(Color.FromHex("#E2C180"));
SColor.Add(Color.FromHex("#BBBFD6"));
SColor.Add(Color.FromHex("#ACCBAA"));
FColor.Add(Color.FromHex("#e9d4f1"));
FColor.Add(Color.FromHex("#d4efed"));
FColor.Add(Color.FromHex("#c4f2e8"));
FColor.Add(Color.FromHex("#f7e0b3"));
FColor.Add(Color.FromHex("#DEE2FF"));
FColor.Add(Color.FromHex("#E5FEE4"));
//Add Root node
var node = AddNode(300, 400, width, height, "Goals");
AddNodeStyle(node, Color.FromHex("#d0ebff"), Color.FromHex("#81bfea"));
RootNode = node;
diagram.AddNode(node);
//Add child nodes for root node
var branch1 = AddNode(100, 100, width, height, "Financial");
index = random.Next(5);
AddNodeStyle(branch1, FColor[index], SColor[index]);
diagram.AddNode(branch1);
var branch1Child1 = AddNode(100, 100, width, height, "Investment");
AddNodeStyle(branch1Child1, (branch1.Style.Brush as SolidBrush).FillColor,
(branch1.Style.StrokeBrush as SolidBrush).FillColor);
diagram.AddNode(branch1Child1);
var branch2 = AddNode(100, 600, width, height, "Social");
index = random.Next(5);
AddNodeStyle(branch2, FColor[index], SColor[index]);
diagram.AddNode(branch2);
var branch2Child1 = AddNode(100, 100, width, height, "Friends");
AddNodeStyle(branch2Child1, (branch2.Style.Brush as SolidBrush).FillColor,
(branch2.Style.StrokeBrush as SolidBrush).FillColor);
diagram.AddNode(branch2Child1);
var branch2Child2 = AddNode(100, 100, width, height, "Family");
AddNodeStyle(branch2Child2, (branch2.Style.Brush as SolidBrush).FillColor,
(branch2.Style.StrokeBrush as SolidBrush).FillColor);
diagram.AddNode(branch2Child2);
var branch3 = AddNode(500, 100, width, height, "Personal");
index = random.Next(5);
AddNodeStyle(branch3, FColor[index], SColor[index]);
diagram.AddNode(branch3);
var branch3Child1 = AddNode(500, 100, width, height, "Sports");
AddNodeStyle(branch3Child1, (branch3.Style.Brush as SolidBrush).FillColor,
(branch3.Style.StrokeBrush as SolidBrush).FillColor);
diagram.AddNode(branch3Child1);
var branch3Child2 = AddNode(500, 100, width, height, "Food");
AddNodeStyle(branch3Child2, (branch3.Style.Brush as SolidBrush).FillColor,
(branch3.Style.StrokeBrush as SolidBrush).FillColor);
diagram.AddNode(branch3Child2);
var branch4 = AddNode(500, 600, width, height, "Work");
index = random.Next(5);
AddNodeStyle(branch4, FColor[index], SColor[index]);
diagram.AddNode(branch4);
var branch4Child1 = AddNode(500, 100, width, height, "Project");
AddNodeStyle(branch4Child1, (branch4.Style.Brush as SolidBrush).FillColor,
(branch4.Style.StrokeBrush as SolidBrush).FillColor);
diagram.AddNode(branch4Child1);

```



```

var branch4Child2 = AddNode(500, 100, width, height, "Career");
AddNodeStyle(branch4Child2, (branch4.Style.Brush as SolidBrush).FillColor,
(branch4.Style.StrokeBrush as SolidBrush).FillColor);
diagram.AddNode(branch4Child2);
//Add connector
diagram.AddConnector(AddConnector(node, branch1));
diagram.AddConnector(AddConnector(node, branch2));
diagram.AddConnector(AddConnector(node, branch3));
diagram.AddConnector(AddConnector(node, branch4));
diagram.AddConnector(AddConnector(branch1, branch1Child1));
diagram.AddConnector(AddConnector(branch2, branch2Child1));
diagram.AddConnector(AddConnector(branch2, branch2Child2));
diagram.AddConnector(AddConnector(branch3, branch3Child1));
diagram.AddConnector(AddConnector(branch3, branch3Child2));
diagram.AddConnector(AddConnector(branch4, branch4Child1));
diagram.AddConnector(AddConnector(branch4, branch4Child2));
//Add connector method
private Connector AddConnector(Node node, Node BranchNode)
{
var connector = new Connector();
connector.SourceNode = node;
connector.TargetNode = BranchNode;
connector.Style.StrokeBrush = new
SolidBrush((connector.TargetNode.Style.StrokeBrush as
SolidBrush).FillColor);
connector.Style.StrokeStyle = StrokeStyle.Dashed;
connector.Style.StrokeWidth = 3;
if (Device.RuntimePlatform == Device.Android)
connector.TargetDecoratorStyle.Width = connector.TargetDecoratorStyle.Width
* App.factor;
connector.TargetDecoratorStyle.Fill =
(connector.TargetNode.Style.StrokeBrush as SolidBrush).FillColor;
connector.TargetDecoratorStyle.Stroke =
(connector.TargetNode.Style.StrokeBrush as SolidBrush).FillColor;
connector.SegmentType = SegmentType.CurveSegment;
return connector;
}
//Add node style method
private void AddNodeStyle(Node node, Color fill, Color Stroke)
{
node.Style.Brush = new SolidBrush(fill);
node.Style.StrokeBrush = new SolidBrush(Stroke);
}
//Add node method
Node AddNode(int x, int y, int w, int h, string text)
{
var node = new Node(x, y, w, h);
node.ShapeType = ShapeType.RoundedRectangle;
node.Style.StrokeWidth = 3;
if (Device.RuntimePlatform == Device.Android)
node.Annotations.Add(new Annotation() { Content = text, FontSize = 14 *
App.factor, TextBrush = new SolidBrush(Color.Black) });
else if (Device.RuntimePlatform == Device.iOS)
node.Annotations.Add(new Annotation() { Content = text, FontSize = 15,
TextBrush = new SolidBrush(Color.Black) });
return node;
}

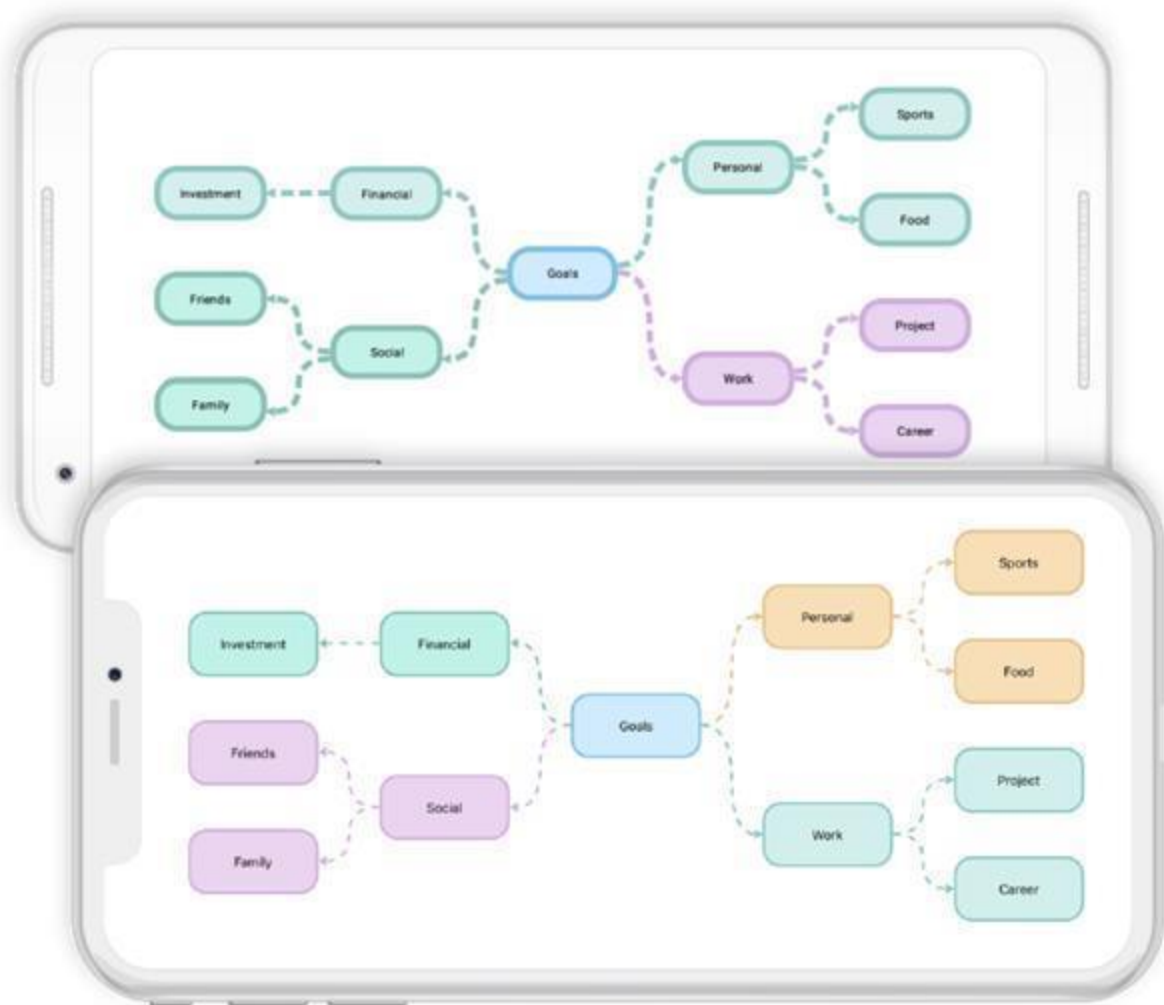
```

### Initializing the mind map layout

The following code illustrates how to initialize the mind map layout in diagram loaded event:

#### C#

```
//Diagram loaded event
diagram.Loaded += Diagram_Loaded;
private void Diagram_Loaded(object sender)
{
    diagram.LayoutManager = new LayoutManager()
    {
        Layout = new MindMapLayout()
        {
            MindMapOrientation = Orientation.Horizontal,
            HorizontalSpacing = 70,
        }
    };
//Updating the layout
    diagram.LayoutManager.Layout.UpdateLayout();
}
```



### Mind map layout style

Mind map styles are defined as a collection of node style with a single connector style. Collection of node style is applied from root node to leaf node either through branch wise or level wise.

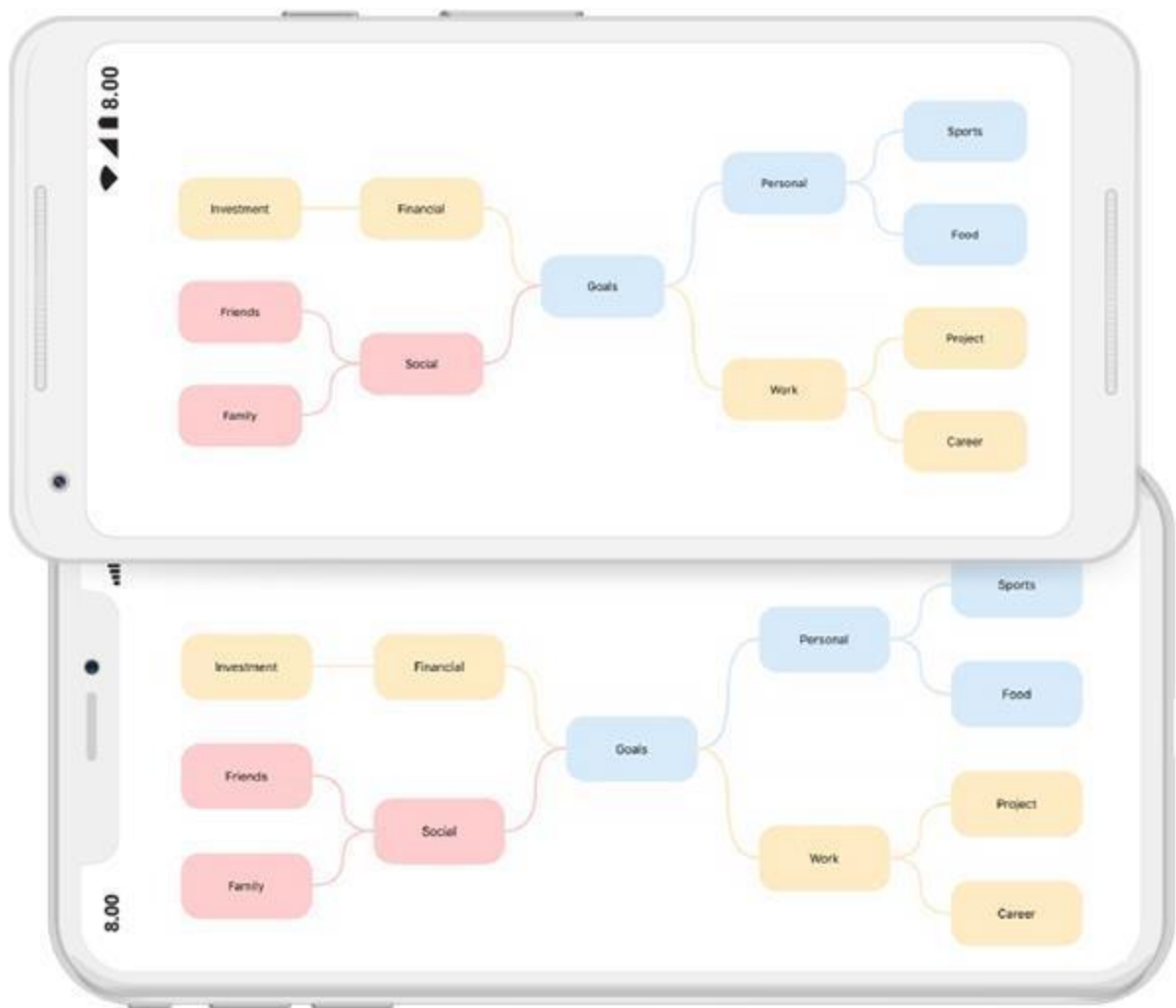
### Branch wise node style

Defined node style collection is applied from branch wise by setting the “ApplyNodeStyleBy” property to branch. The following code example illustrates how to apply style for mind map branch wise.

### C#

```
//Update mind map layout style
private void UpdateTheme()
{
    bool repeat_mode = true;
    nodeStyleCollection.Clear();
    nodeStyleCollection.Add(new NodeStyle(new
    SolidBrush(Color.FromHex("#d7ebfb")), Color.FromHex("#d7ebfb"), objShape1,
    StrokeStyle.Default,
```

```
new Syncfusion.SfDiagram.XForms.TextStyle((int)(14), Color.Black, ".SF UI
Text", HorizontalAlignment.Center, VerticalAlignment.Center));
nodeStyleCollection.Add(new NodeStyle(new
SolidBrush(Color.FromHex("#ffeabc4")), Color.FromHex("#ffeabc4"), objShape2,
StrokeStyle.Default,
new Syncfusion.SfDiagram.XForms.TextStyle((int)(14), Color.Black, ".SF UI
Text", HorizontalAlignment.Center, VerticalAlignment.Center));
nodeStyleCollection.Add(new NodeStyle(new
SolidBrush(Color.FromHex("#ffcdcd")), Color.FromHex("#ffcdcd"), objShape4,
StrokeStyle.Default,
new Syncfusion.SfDiagram.XForms.TextStyle((int)(14), Color.Black, ".SF UI
Text", HorizontalAlignment.Center, VerticalAlignment.Center));
lineStyle = new LineStyle(SegmentType.CurveSegment, StrokeStyle.Default, 3,
ApplyColorFrom.TargetFill, DecoratorType.None, ApplyColorFrom.None,
ApplyColorFrom.None);
(diagram.LayoutManager.Layout as MindMapLayout).UpdateLayoutStyle(new
LayoutStyle(nodeStyleCollection, lineStyle, ApplyNodeStyleBy.Branch,
repeat_mode));
}
private void Diagram_Loaded(object sender)
{
    //hook update theme method in diagram loaded event
    UpdateTheme();
}
```



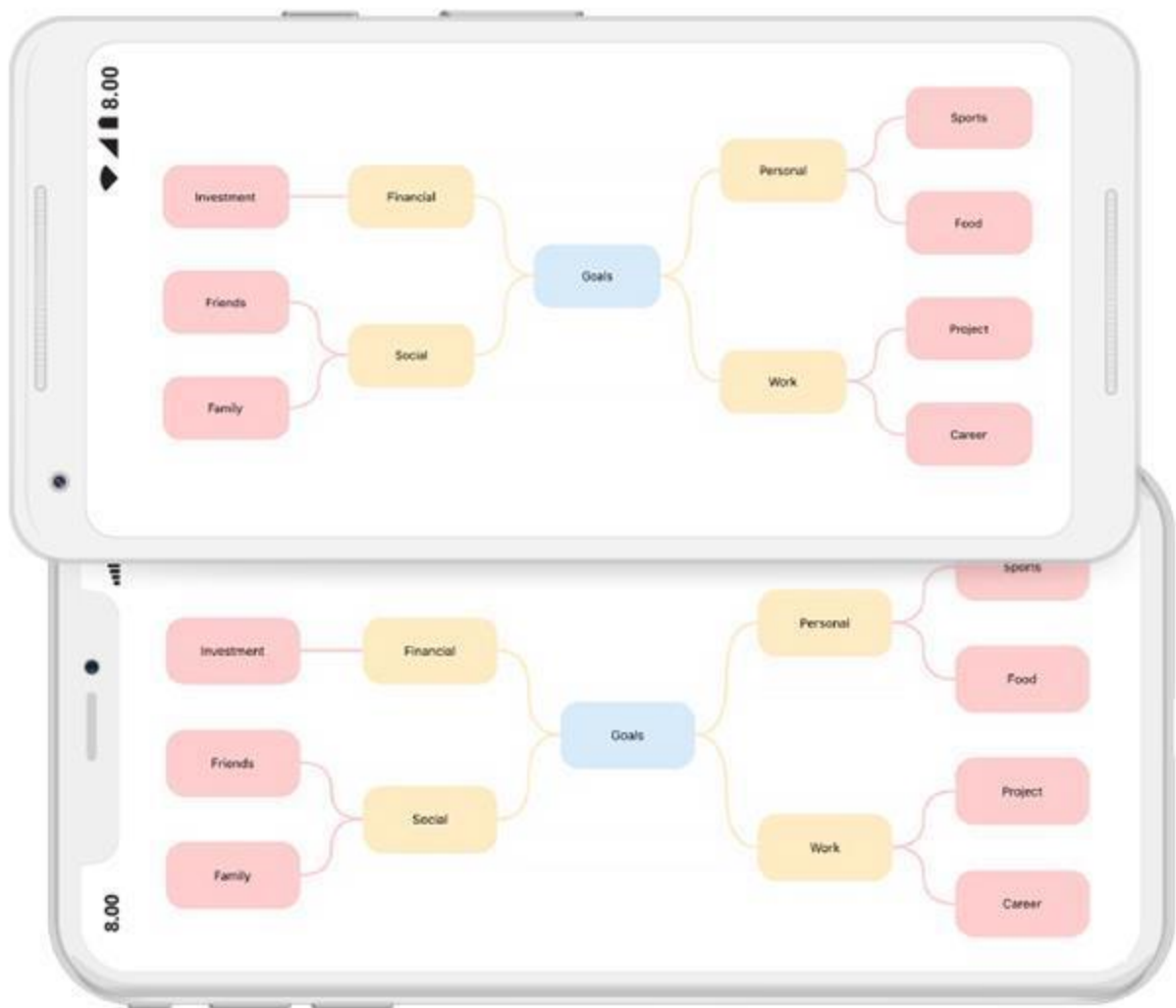
### Level wise node style

Defined node style collection is applied from level wise by setting the “ApplyNodeStyleBy” property to level. The following code example illustrates how to apply style for mind map level wise.

### C#

```
//Update mind map layout style
private void UpdateTheme()
{
    bool repeat_mode = true;
    nodeStyleCollection.Clear();
    nodeStyleCollection.Add(new NodeStyle(new
        SolidBrush(Color.FromHex("#d7ebfb")), Color.FromHex("#d7ebfb"), objShape1,
        StrokeStyle.Default,
        new Syncfusion.SfDiagram.XForms.TextStyle((int)(14), Color.Black, ".SF UI
        Text", HorizontalAlignment.Center, VerticalAlignment.Center)));
}
```

```
nodeStyleCollection.Add(new NodeStyle(new
SolidBrush(Color.FromHex("#ffe4c4")), Color.FromHex("#ffe4c4"), objShape2,
StrokeStyle.Default,
new Syncfusion.SfDiagram.XForms.TextStyle((int)(14), Color.Black, ".SF UI
Text", HorizontalAlignment.Center, VerticalAlignment.Center)));
nodeStyleCollection.Add(new NodeStyle(new
SolidBrush(Color.FromHex("#ffcdcd")), Color.FromHex("#ffcdcd"), objShape4,
StrokeStyle.Default,
new Syncfusion.SfDiagram.XForms.TextStyle((int)(14), Color.Black, ".SF UI
Text", HorizontalAlignment.Center, VerticalAlignment.Center)));
lineStyle = new LineStyle(SegmentType.CurveSegment, StrokeStyle.Default, 3,
ApplyColorFrom.TargetFill, DecoratorType.None, ApplyColorFrom.None,
ApplyColorFrom.None);
(diagram.LayoutManager.Layout as MindMapLayout).UpdateLayoutStyle(new
LayoutStyle(nodeStyleCollection, lineStyle, ApplyNodeStyleBy.Level,
repeat_mode));
}
private void Diagram_Loaded(object sender)
{
    //hook update theme method in diagram loaded event
    UpdateTheme();
}
```



### Repeat mode

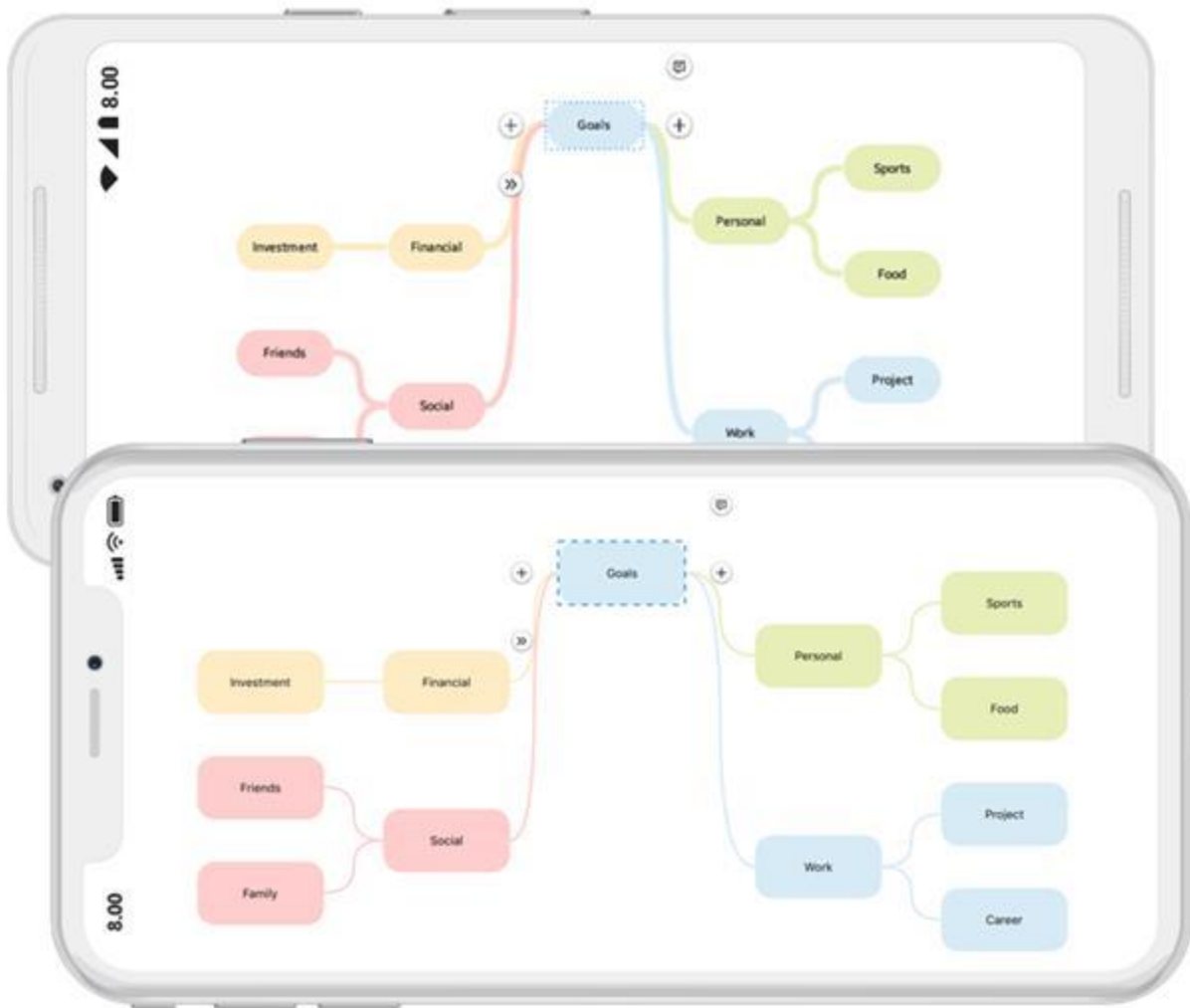
When style collection ends, the next level or branch can have styles. To be cyclic, enable repeat mode or disable it to continue with last style.

### Free form layout

The mind map free form layout provides an option to rearrange nodes in a layout. Mind map layout allows you to enable or disable free form using the “EnableFreeForm” property.

### C#

```
(diagram.LayoutManager.Layout as MindMapLayout).EnableFreeForm = true;
```



**Note:** Diagram supports mind map layout in Xamarin.Forms.Android and Xamarin.Forms.iOS alone.

### Drawing mode

Drawing mode is used to draw continuously on the diagram area for selected mode dynamically.

### Text node

This node has default annotation. TextNode mode will add continuous text node. The following code example illustrates how to enable TextNode mode.

### XML

```
<diagram:SfDiagram x:Name="diagram" DrawingMode="TextNode">
</diagram:SfDiagram>
```

### C#

```
diagram.DrawingMode = DrawingMode.TextNode;
```



## Connector

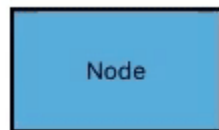
Connector mode add continuous orthogonal connectors on the diagram area. In this mode, you can connect connectors in between Points, Nodes, or Ports. The following code example illustrates how to enable Connector mode.

### XML

```
<diagram:SfDiagram x:Name="diagram" DrawingMode="Connector">
</diagram:SfDiagram>
```

### C#

```
diagram.DrawingMode = DrawingMode.Connector;
```



## Gridlines

Gridlines are the pattern of lines drawn behind the diagram elements. It provides a visual guidance when dragging or arranging the node on the diagram surface.

### Gridlines visibility

The “ShowGrid” property in PageSettings will show or hide the gridlines. The following code example illustrates how to enable grid visibility.

### XML

```
<diagram:SfDiagram x:Name="diagram" >
<diagram:SfDiagram.PageSettings>
<diagram:PageSettings ShowGrid ="true"/>
</diagram:SfDiagram.PageSettings>
</diagram:SfDiagram>
```

**C#**

```
diagram.PageSettings.ShowGrid = true;
```

**Customizing gridlines**

Grid cell size and gridline color can be modified using the “GridSize” and “GridColor” properties respectively. The following code example illustrates how to customize the grid.

**XML**

```
<diagram:SfDiagram x:Name="diagram" >
  <diagram:SfDiagram.PageSettings>
    <diagram:PageSettings GridColor="Pink" GridSize="14"/>
  </diagram:SfDiagram.PageSettings>
</diagram:SfDiagram>
```

**C#**

```
diagram.PageSettings.GridColor = Color.Pink;
diagram.PageSettings.GridSize = 14;
```

**Snapping gridlines**

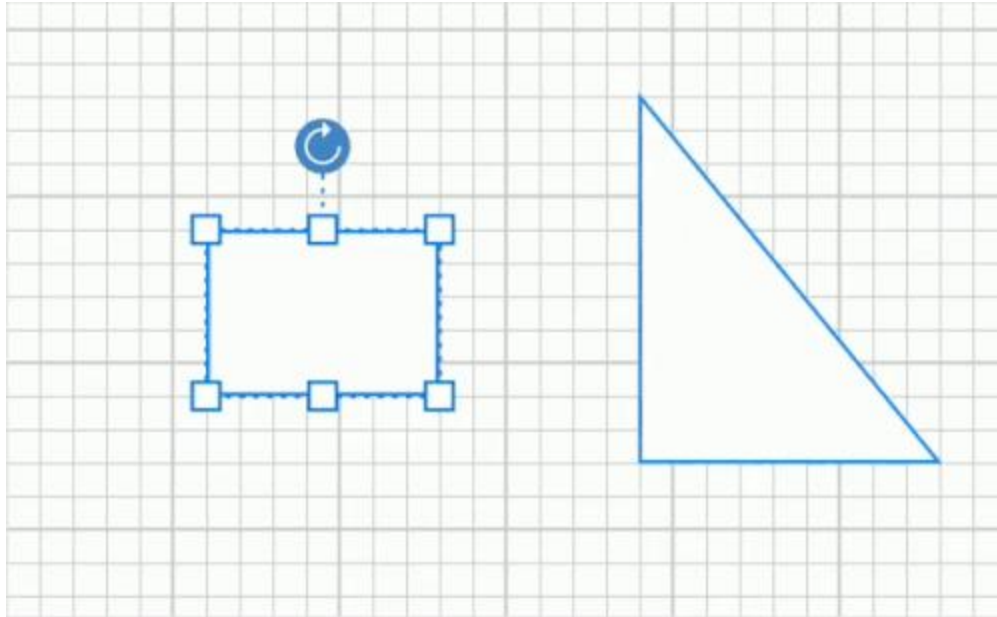
Nodes can be aligned and resized easily using gridlines by enabling the snap to grid. The following code example illustrates how to enable the snap to grid.

**XML**

```
<diagram:SfDiagram x:Name="diagram" >
  <diagram:SfDiagram.PageSettings>
    <diagram:PageSettings SnapToGrid="true"/>
  </diagram:SfDiagram.PageSettings>
</diagram:SfDiagram>
```

**C#**

```
diagram.PageSettings.SnapToGrid = true;
```



## User handles

User handles are customizable handles which can be used to perform custom actions and also default clipboard actions. You can able to customize the user handles using:

- SfGraphicPath
- Template

The following code illustrates how to add custom user handle in diagram:

### C#

```
//Add graphic path into an user handle collection
SfGraphics graph = new SfGraphics();
Pen stroke = new Pen();
stroke.Brush = new SolidBrush(Color.Transparent);
stroke.StrokeWidth = 3;
stroke.StrokeBrush = new SolidBrush(Color.FromRgb(24, 161, 237));
graph.DrawEllipse(stroke, new Rectangle(10, 0, 20, 20));
graph.DrawArc(stroke, 0, 20, 40, 40, 180, 180);
//Add template into an user handles collection
var deleteTemplate = new DataTemplate(() =>
{
    var root = new StackLayout()
    {
        Orientation = StackOrientation.Horizontal,
        BackgroundColor = Color.Transparent
    };
    Image image = new Image();
    image.WidthRequest = 25;
    image.HeightRequest = 25;
    if (Device.RuntimePlatform == Device.iOS)
        image.Source = "Images/delete.png";
    else
        image.Source = "delete.png";
    root.Children.Add(image);
});
```

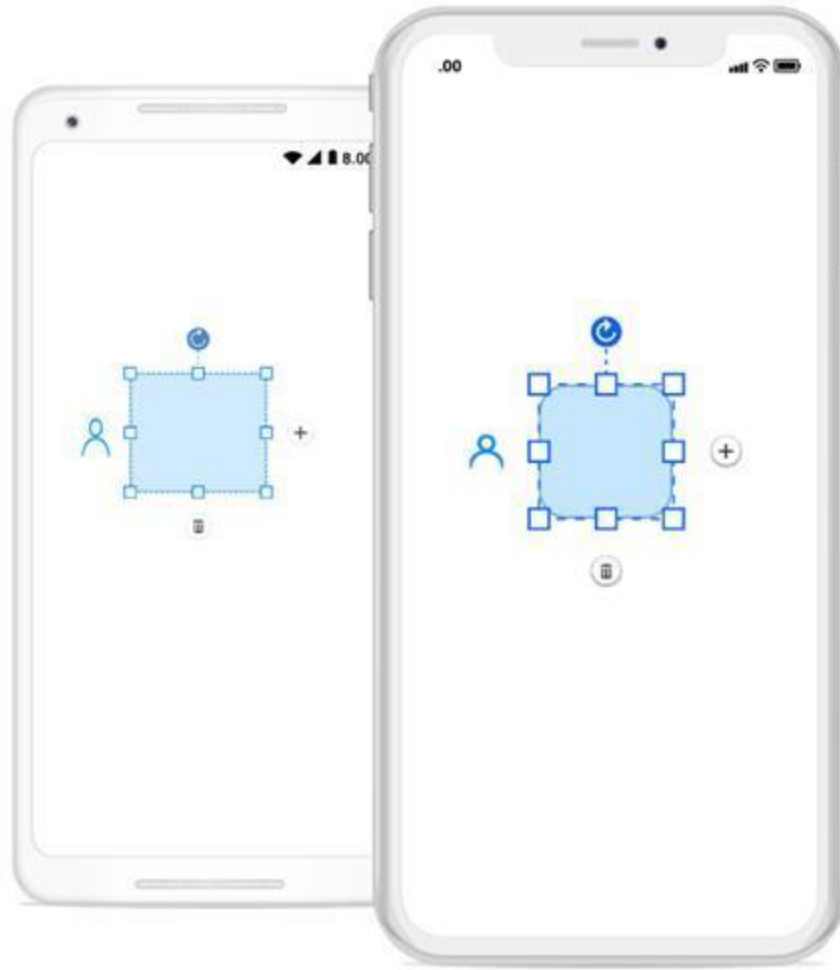
```
return root;
});
var plusTemplate = new DataTemplate(() =>
{
var root = new StackLayout()
{
Orientation = StackOrientation.Horizontal,
BackgroundColor = Color.Transparent
};
Image image = new Image();
image.WidthRequest = 25;
image.HeightRequest = 25;
if (Device.RuntimePlatform == Device.iOS)
image.Source = "Images/plus.png";
else
image.Source = "plus.png";
root.Children.Add(image);
return root;
});
//Add user handle into an user handles collection
diagram.UserHandles.Add(new Syncfusion.SfDiagram.XForms.UserHandle("delete",
UserHandlePosition.Bottom, deleteTemplate) { });
diagram.UserHandles.Add(new Syncfusion.SfDiagram.XForms.UserHandle("graphic
path", UserHandlePosition.Left, graph) { });
diagram.UserHandles.Add(new Syncfusion.SfDiagram.XForms.UserHandle("plus",
UserHandlePosition.Right, plusTemplate) { });
```

### User handles clicked event

The following code illustrate how to define use handles clicked event and its action.

#### C#

```
//User handles clicked event
diagram.UserHandleClicked+=HandleUserHandleClickedEventHandler;
void HandleUserHandleClickedEventHandler(object sender,
UserHandleClickedEventArgs args)
{
//delete user handle click event action
if (args.Item.Name == "delete")
{
diagram.Delete();
}
}
```

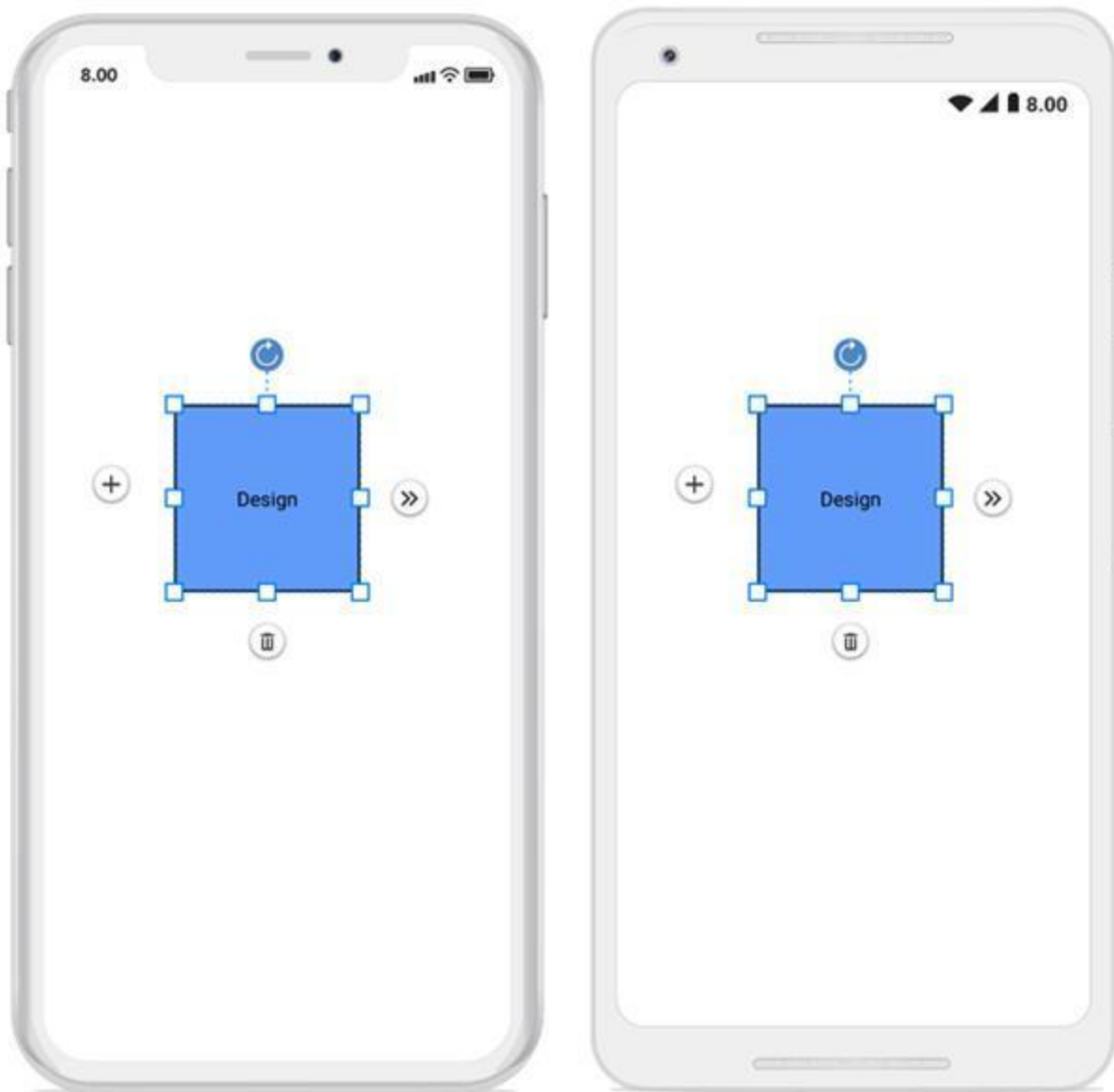


### Customizing user handle position

User handle position can be moved or adjusted from its default position. The following code shows how to adjust the position using the “MoveBy” method.

#### C#

```
//Define the user handle
UserHandleCollection userHandles = new UserHandleCollection();
UserHandle left = new UserHandle("Left", UserHandlePosition.Left,
plusTemplate) { Visible = true };
//Customize the user handle position using move by method
left.MoveBy(-10, -10);
userHandles.Add(left);
userHandles.Add(new UserHandle("Right", UserHandlePosition.Right,
m_expandTemplate) { Visible = true });
userHandles.Add(new UserHandle("Delete", UserHandlePosition.Bottom,
deleteTemplate) { Visible = true });
diagram.UserHandles = userHandles;
```



---

**Note:** Diagram supports user handle in Xamarin.Forms.Android and Xamarin.Forms.iOS alone.

---

### OverviewPanel

The overview panel is used to display the preview (overall mini view) of the entire content of a diagram. This helps you to see the overall picture of a large diagram and easily navigate to a position of the page using view port rectangle.

When working on a large diagram, to navigate, you have to zoom out the entire diagram to find where you are. This solution is not suitable when frequent navigation is required.

The overview control solves this problem by displaying a preview (overall view) of the entire diagram, A rectangle indicates the viewport of the diagram. Navigation is made easy by dragging this rectangle.

### Create overview panel

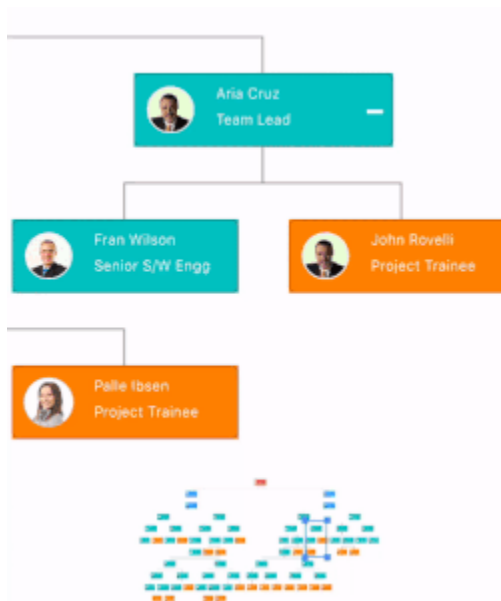
The following code shows how to create overview panel in a diagram.

**XML**

```
<sfdiagram:OverviewPanel x:Name="overview" >
</sfdiagram:OverviewPanel>
```

**C#**

```
//Define over view panel
OverviewPanel overview = new OverviewPanel();
overview.Width = 500;
overview.Height = 500;
overview.IsVisible = true;
diagram.OverviewPanel = overview;
```

*Prevent refresh*

Prevents diagram being updated in overview panel if any interaction takes place in the diagram layer. Prevents the diagram update for the entire actions till the property is set to true. This property is helpful when there is no need for frequent refreshing in the overview panel.

**C#**

```
//Define prevent refresh property
overview.PreventRefresh = true;
```

*Force refresh*

This method force refreshes the diagram in the overview panel. When the overview panel is prevented from updating diagram interaction by enabling PreventRefresh, force refresh is used to update the diagram in the overview panel.

**C#**

```
//Define force refresh method
overview.ForceRefresh();
```

### Customizing view port rect

The “StartX” and “StartY” properties are used to define the start position of the view port rect over the overview panel. The following code shows how to customize the view port rect.

#### C#

```
//Customize the view port rect  
ViewportRect ViewportRect = new ViewportRect(overview);  
ViewportRect.StrokeColor = Color.Black;  
ViewportRect.StartX = 300;  
ViewportRect.StartY = 100;
```

## SfDigitalGauge

### Overview

The digital gauge control is used to display alphanumeric characters in digital (LED display) mode. This control displays a range of values that uses character in combination with numbers.

#### Key features

The digital gauge is composed of several segments which are major UI components. [SfDigitalGauge](#) comprises the following segments to display digital characters:

- 7-segment display
- 14-segment display
- 16-segment display
- 8\*8 dot matrix display

#### *7-segment display*

This type of digital gauge displays digital characters in 7 segments. It is mainly used to display numbers.

#### *14-segment display*

This type of digital gauge displays digital characters in 14 segments. It is mainly used to display both the numbers and alphabet.

#### *16-segment display*

This type of digital gauge displays digital characters in 16 segments. It is also mainly used to display both the numbers and alphabet.

#### *8 \* 8 dot matrix display*

This type of digital gauge displays digital characters in 8\*8 dot matrix segments, in which the characters are spotted by the regular brush and the remaining dots are spotted by the dimmed brush. It is mainly used to display numbers, characters, and special characters.





## Getting Started

This section explains the steps required to configure the [SfDigitalGauge](#) and add basic elements to it using various APIs.

### Adding SfDigitalGauge reference

You can add SfDigitalGauge reference using one of the following methods:

#### Method 1: Adding SfDigitalGauge reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfDigitalGauge to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfGauge](#), and then install it.

![Adding SfDigitalGauge reference from NuGet](Getting-Started\_images/Adding SfDigitalGauge reference.png)

---

**Note:** Install the same version of SfDigitalGauge NuGet in all the projects.

---

#### Method 2: Adding SfDigitalGauge reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfDigitalGauge control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfDigitalGauge assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfGauge.Android.dll Syncfusion.SfGauge.XForms.Android.dll Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfGauge.iOS.dll Syncfusion.SfGauge.XForms.iOS.dll Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfGauge.UWP.dll Syncfusion.SfGauge.XForms.UWP.dll Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

[Launching an application on each platform with SfDigitalGauge.](#)

To use the SfDigitalGauge control inside an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch the SfDigitalGauge in iOS, call the `SfDigitalGaugeRenderer.Init()` in the `FinishedLaunching` overridden method of the AppDelegate class after the Xamarin.Forms Framework has been initialized and before the LoadApplication is called, as demonstrated in the following code example.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
...
global::Xamarin.Forms.Forms.Init ();
Syncfusion.SfGauge.XForms.iOS.SfGaugeRenderer.Init();
LoadApplication (new App ());
...
}
```

### Universal Windows Platform (UWP)

You need to initialize the digital gauge view assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with digital gauge in Release mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
...
rootFrame.NavigationFailed += OnNavigationFailed;
// Add `using System.Reflection;`
List<Assembly> assembliesToInclude = new List<Assembly>();
// Now, add all the assemblies your app uses
assembliesToInclude.Add(typeof(Syncfusion.SfGauge.XForms.UWP.SfGaugeRenderer
).GetTypeInfo().Assembly);
// Replaces Xamarin.Forms.Forms.Init(e);
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
...
}
```

### Android

The Android platform does not require any additional configuration to render the digital gauge.

#### Adding namespace for the assemblies

##### XML

```
xmlns:gauge="clr-
namespace:Syncfusion.SfGauge.XForms;assembly=Syncfusion.SfGauge.XForms"
```

#### C#

```
using Syncfusion.SfGauge.XForms;
```

#### Initialize gauge

You can initialize the [SfDigitalGauge](#) control with a required optimal name using the included namespace.

##### XML

```
<gauge:SfDigitalGauge/>
```

## C#

```
SfDigitalGauge sfDigitalGauge = new SfDigitalGauge();  
this.Content = sfDigitalGauge;
```

### Setting value for digital gauge

The [SfDigitalGauge](#) control provides options to display special characters or values using the [Value](#) property.

## XML

```
<gauge:SfDigitalGauge Value="1 2 3 4"/>
```

## C#

```
SfDigitalGauge sfDigitalGauge = new SfDigitalGauge();  
sfDigitalGauge.Value = "11:59:50 PM";
```

### Setting character type for digital gauge

By using the [CharacterType](#) property, you can set the segments for digital gauge. The digital characters can be drawn in the following four different segments:

- EightCrossEightDotMatrix
- SegmentFourteen
- SegmentSeven
- SegmentSixteen

## XML

```
<gauge:SfDigitalGauge CharacterType="EightCrossEightDotMatrix"/>
```

## C#

```
SfDigitalGauge digital = new SfDigitalGauge();  
digital.CharacterType = CharacterType.EightCrossEightDotMatrix;
```

### Configuring properties

The [CharacterHeight](#), [CharacterWidth](#), and [CharacterStroke](#) properties are used to display characters, which can be customized as shown in the following code snippets:

## XML

```
<gauge:SfDigitalGauge CharacterHeight="60" CharacterWidth="25"  
CharacterStrokeColor="#146CED"/>
```

## C#

```
SfDigitalGauge digital = new SfDigitalGauge();  
digital.CharacterHeight = 60;  
digital.CharacterWidth = 25;  
digital.CharacterStrokeColor = Color.FromRgb(20, 108, 237);
```

The following code example is the complete code of the previous configurations.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GaugeGettingStarted"
x:Class="GaugeGettingStarted.MainPage"
xmlns:gauge="clr-
namespace:Syncfusion.SfGauge.XForms;assembly=Syncfusion.SfGauge.XForms">
<gauge:SfDigitalGauge Value="11:59:50 PM" SegmentStrokeWidth="5"
BackgroundColor="White"
HeightRequest="100" WidthRequest="375"
DisabledSegmentAlpha="25" DisabledSegmentColor="Gray"
CharacterHeight="90" CharacterWidth="25"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="EightCrossEightDotMatrix"
CharacterStrokeColor="#146CED"/>
</ContentPage>
```

### C#

```
using Xamarin.Forms;
using Syncfusion.SfGauge.XForms;
namespace GaugeGettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            //Initialize digital gauge
            SfDigitalGauge digital = new SfDigitalGauge();
            digital.BackgroundColor = Color.White;
            digital.HeightRequest = 100;
            digital.WidthRequest = 375;
            digital.Value = "11:59:50 PM";
            digital.CharacterHeight = 90;
            digital.CharacterWidth = 25;
            digital.HorizontalOptions = LayoutOptions.Center;
            digital.VerticalOptions = LayoutOptions.Center;
            digital.SegmentStrokeWidth = 5;
            digital.CharacterType = CharacterType.EightCrossEightDotMatrix;
            digital.DisabledSegmentAlpha = 25;
            digital.CharacterStrokeColor = Color.FromRgb(20, 108, 237);
            digital.DisabledSegmentColor = Color.Gray;
            this.Content = digital;
        }
    }
}
```

The following screenshot illustrates the result of the previous codes.



You can find the complete getting started sample from this [link](#).

### Customize character segments

The characters of a digital gauge can be customized in terms of [CharacterWidth](#), [CharacterHeight](#), [CharacterSpacing](#), and [CharacterStrokeColor](#).

#### customize character size

The values of digital characters are scaled by altering the height and width of digital characters. It is achieved by setting the [CharacterHeight](#) and [CharacterWidth](#) properties in the digital gauge. Default value of [CharacterHeight](#) and [CharacterWidth](#) is 25.

#### XML

```
<gauge:SfDigitalGauge Value="SYNCFUSION" SegmentStrokeWidth="4"
HeightRequest="120" WidthRequest="330" BackgroundColor="Black"
DisabledSegmentAlpha="30" DisabledSegmentColor="#146CED"
CharacterHeight="117" CharacterWidth="25"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="SegmentSeven"
CharacterStrokeColor="#146CED"/>
```

**C#**

```
SfDigitalGauge digital = new SfDigitalGauge();
digital.HeightRequest = 120;
digital.WidthRequest = 330;
digital.Value = "SYNCFUSION";
digital.CharacterHeight = 117;
digital.CharacterWidth = 25;
digital.HorizontalOptions = LayoutOptions.Center;
digital.VerticalOptions = LayoutOptions.Center;
digital.SegmentStrokeWidth = 4;
digital.CharacterType = CharacterType.SegmentSeven;
digital.DisabledSegmentAlpha = 30;
digital.BackgroundColor = Color.Black;
digital.CharacterStrokeColor = Color.FromRgb(20, 108, 237);
digital.DisabledSegmentColor = Color.FromRgb(20, 108, 237);
```

**Setting character spacing**

The values of digital characters are spaced by altering the space of digital characters. It is achieved by setting the [CharacterSpacing](#) property.

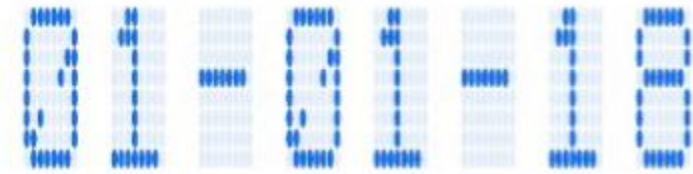
**XML**

```
<gauge:SfDigitalGauge Value="01-01-18" SegmentStrokeWidth="3"
HeightRequest="100" WidthRequest="360" CharacterSpacing="10"
DisabledSegmentAlpha="30" DisabledSegmentColor="#146CED"
CharacterHeight="90" CharacterWidth="25"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="EightCrossEightDotMatrix"
CharacterStrokeColor="#146CED"/>
```

**C#**

```
SfDigitalGauge digital = new SfDigitalGauge();
digital.HeightRequest = 100;
digital.WidthRequest = 360;
digital.Value = "01-01-18";
digital.CharacterHeight = 90;
digital.CharacterWidth = 25;
digital.HorizontalOptions = LayoutOptions.Center;
digital.VerticalOptions = LayoutOptions.Center;
digital.SegmentStrokeWidth = 3;
digital.CharacterSpacing = 10;
```

```
digital.CharacterType = CharacterType.EightCrossEightDotMatrix;
digital.DisabledSegmentAlpha = 30;
digital.CharacterStrokeColor = Color.FromRgb(20, 108, 237);
digital.DisabledSegmentColor = Color.FromRgb(20, 108, 237);
```



### Customize character segment stroke

The values of digital characters color can be customized using the [CharacterStrokeColor](#) property.

### XML

```
<gauge:SfDigitalGauge Value="1 2 3 4 5" SegmentStrokeWidth="3"
HeightRequest="100" WidthRequest="300"
DisabledSegmentAlpha="20" DisabledSegmentColor="#146CED"
CharacterHeight="90" CharacterWidth="25"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="SegmentSeven"
CharacterStrokeColor="Purple"/>
```

### C#

```
SfDigitalGauge digital = new SfDigitalGauge();
digital.HeightRequest = 100;
digital.WidthRequest = 300;
digital.Value = "1 2 3 4 5";
digital.CharacterHeight = 90;
digital.CharacterWidth = 25;
digital.HorizontalOptions = LayoutOptions.Center;
digital.VerticalOptions = LayoutOptions.Center;
digital.SegmentStrokeWidth = 3;
digital.CharacterType = CharacterType.SegmentSeven;
digital.DisabledSegmentAlpha = 20;
digital.CharacterStrokeColor = Color.Purple;
digital.DisabledSegmentColor = Color.FromRgb(20, 108, 237);
```





### Customize disabled segment

You can customize the color and opacity of disabled segments using the [DisabledSegmentColor](#) and [DisabledSegmentAlpha](#) properties. The width of the digital character value can be customized using the [SegmentStrokeWidth](#) property.

#### XML

```
<gauge:SfDigitalGauge Value="1 2 3 4 5" SegmentStrokeWidth="5"
HeightRequest="100" WidthRequest="300"
DisabledSegmentAlpha="25" DisabledSegmentColor="LightSkyBlue"
CharacterHeight="90" CharacterWidth="25"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="SegmentSeven"
CharacterStrokeColor="#146CED"/>
```

#### C#

```
SfDigitalGauge digital = new SfDigitalGauge();
digital.HeightRequest = 100;
digital.WidthRequest = 300;
this.BackgroundColor = Color.White;
digital.Value = "1 2 3 4 5";
digital.CharacterHeight = 90;
digital.CharacterWidth = 25;
digital.HorizontalOptions = LayoutOptions.Center;
digital.VerticalOptions = LayoutOptions.Center;
digital.SegmentStrokeWidth = 5;
digital.CharacterType = CharacterType.SegmentSeven;
digital.DisabledSegmentAlpha = 25;
digital.CharacterStrokeColor = Color.FromRgb(20, 108, 237);
digital.DisabledSegmentColor = Color.LightSkyBlue;
```



### Customize background color of digital gauge

You can customize the background color of the digital gauge using the [BackgroundColor](#) property.

#### XML

```
<gauge:SfDigitalGauge Value="1 2 3 4 5" SegmentStrokeWidth="5"
HeightRequest="100" WidthRequest="300"
DisabledSegmentAlpha="25" DisabledSegmentColor="LightSkyBlue"
CharacterHeight="90" CharacterWidth="25"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="SegmentSeven" BackgroundColor="LightPink"
CharacterStrokeColor="#146CED"/>
```

**C#**

```
SfDigitalGauge digital = new SfDigitalGauge();
digital.HeightRequest = 100;
digital.WidthRequest = 300;
this.BackgroundColor = Color.White;
digital.Value = "1 2 3 4 5";
digital.CharacterHeight = 90;
digital.CharacterWidth = 25;
digital.HorizontalOptions = LayoutOptions.Center;
digital.VerticalOptions = LayoutOptions.Center;
digital.SegmentStrokeWidth = 5;
digital.CharacterType = CharacterType.SegmentSeven;
digital.DisabledSegmentAlpha = 25;
digital.CharacterStrokeColor = Color.FromRgb(20, 108, 237);
digital.DisabledSegmentColor = Color.LightSkyBlue;
digital.BackgroundColor = Color.LightPink;
```

**Character types**

The digital characters can be drawn in the following four different segments:

- Seven
- Fourteen
- Sixteen
- EightCrossEightDotMatrix

**Seven segment**

The seven-segment type is capable of displaying numbers and a few uppercase letters efficiently.

**XML**

```
<gauge:SfDigitalGauge Value="SYNCFUSION" SegmentStrokeWidth="4"
HeightRequest="100" WidthRequest="340"
DisabledSegmentAlpha="25" DisabledSegmentColor="Gray"
CharacterHeight="90" CharacterWidth="25"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="SegmentSeven"
CharacterStrokeColor="#146CED"/>
```

**C#**

```
SfDigitalGauge digital = new SfDigitalGauge();
digital.HeightRequest = 100;
```

```
digital.WidthRequest = 340;  
digital.Value = "SYNCFUSION";  
digital.CharacterHeight = 90;  
digital.CharacterWidth = 25;  
digital.HorizontalOptions = LayoutOptions.Center;  
digital.VerticalOptions = LayoutOptions.Center;  
digital.SegmentStrokeWidth = 4;  
digital.CharacterType = CharacterType.SegmentSeven;  
digital.DisabledSegmentAlpha = 25;  
digital.CharacterStrokeColor = Color.FromRgb(20, 108, 237);  
digital.DisabledSegmentColor = Color.Gray;
```

The image shows the word "SYNCFUSION" rendered in a digital font style using a fourteen-segment character type. The characters are composed of blue segments on a light gray background. The font is clean and modern, with a slight shadow effect behind the characters.

#### Fourteen segment

The fourteen-segment type is capable of displaying numbers and alphabet efficiently.

The image shows the word "SYNCFUSION" rendered in a digital font style using a sixteen-segment character type. The characters are composed of blue segments on a light gray background. The font is clean and modern, with a slight shadow effect behind the characters.

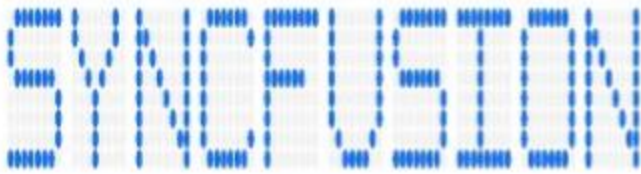
#### Sixteen segment

The sixteen-segment type is capable of displaying numbers and alphabet clearly.

The image shows the word "SYNCFUSION" rendered in a digital font style using an EightCrossEightDotMatrix character type. The characters are composed of blue segments on a light gray background. The font is clean and modern, with a slight shadow effect behind the characters.

#### EightCrossEightDotMatrix segment

The dot matrix segment type is capable of displaying numbers, alphabet, and special characters efficiently.



### Display value types

The digital gauge displays numbers, alphabet, and special characters, which are given in the value property.

#### Setting value to number

Numbers can be displayed in digital gauge in different formats using the `CharacterType` property.

#### XML

```
<gauge:SfDigitalGauge Value="12 34 56" SegmentStrokeWidth="3"
HeightRequest="100" WidthRequest="330"
DisabledSegmentAlpha="25" DisabledSegmentColor="DarkGoldenrod"
CharacterHeight="90" CharacterWidth="30"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="SegmentSeven"
CharacterStrokeColor="DarkGoldenrod"/>
```

#### C#

```
SfDigitalGauge digital = new SfDigitalGauge();
digital.HeightRequest = 100;
digital.WidthRequest = 330;
digital.Value = "12 34 56";
digital.CharacterHeight = 90;
digital.CharacterWidth = 30;
digital.HorizontalOptions = LayoutOptions.Center;
digital.VerticalOptions = LayoutOptions.Center;
digital.SegmentStrokeWidth = 3;
digital.CharacterType = CharacterType.SegmentSeven;
digital.DisabledSegmentAlpha = 25;
digital.CharacterStrokeColor = Color.DarkGoldenrod;
digital.DisabledSegmentColor = Color.DarkGoldenrod;
```



#### Setting value to alphabet

Alphabet can be displayed in digital gauge using any one of the character format types.

**XML**

```
<gauge:SfDigitalGauge Value="SYNCFUSION" SegmentStrokeWidth="3"
HeightRequest="100" WidthRequest="340"
DisabledSegmentAlpha="25" DisabledSegmentColor="Green"
CharacterHeight="90" CharacterWidth="25"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="SegmentSixteen"
CharacterStrokeColor="Green"/>
```

**C#**

```
SfDigitalGauge digital = new SfDigitalGauge();
digital.HeightRequest = 100;
digital.WidthRequest = 340;
digital.Value = "SYNCFUSION";
digital.CharacterHeight = 90;
digital.CharacterWidth = 25;
digital.HorizontalOptions = LayoutOptions.Center;
digital.VerticalOptions = LayoutOptions.Center;
digital.SegmentStrokeWidth = 3;
digital.CharacterType = CharacterType.SegmentSixteen;
digital.DisabledSegmentAlpha = 25;
digital.CharacterStrokeColor = Color.Green;
digital.DisabledSegmentColor = Color.Green;
```



## Setting value to special characters

Special characters can also be displayed in digital gauge using the EightCrossEightDotMatrix character format type.

**XML**

```
<gauge:SfDigitalGauge Value="@ # $ % *" SegmentStrokeWidth="3"
HeightRequest="100" WidthRequest="350"
DisabledSegmentAlpha="25" DisabledSegmentColor="Red"
CharacterHeight="90" CharacterWidth="30"
HorizontalOptions="Center" VerticalOptions="Center"
CharacterType="EightCrossEightDotMatrix"
CharacterStrokeColor="Red"/>
```

**C#**

```
SfDigitalGauge digital = new SfDigitalGauge();
digital.HeightRequest = 100;
digital.WidthRequest = 350;
digital.Value = "@ # $ % *";
```

```
digital.CharacterHeight = 90;  
digital.CharacterWidth = 30;  
digital.HorizontalOptions = LayoutOptions.Center;  
digital.VerticalOptions = LayoutOptions.Center;  
digital.SegmentStrokeWidth = 3;  
digital.CharacterType = CharacterType.EightCrossEightDotMatrix;  
digital.DisabledSegmentAlpha = 25;  
digital.CharacterStrokeColor = Color.Red;  
digital.DisabledSegmentColor = Color.Red;
```



## SfEffectsView

### Overview

The Effects View is a container control that provides modern effects like ripple, selection, scaling, and rotation. Users can render these effects through touch interactions such as touch down, touch up, long press, and also by calling the APIs.



ROTATE TO 180°

### Key features

- Supports highlight and ripple animation.
- Supports selection effect with built-in support for notifications during selection state changes.
- Provides scale down, scale up, and rotation effects.

### Getting Started

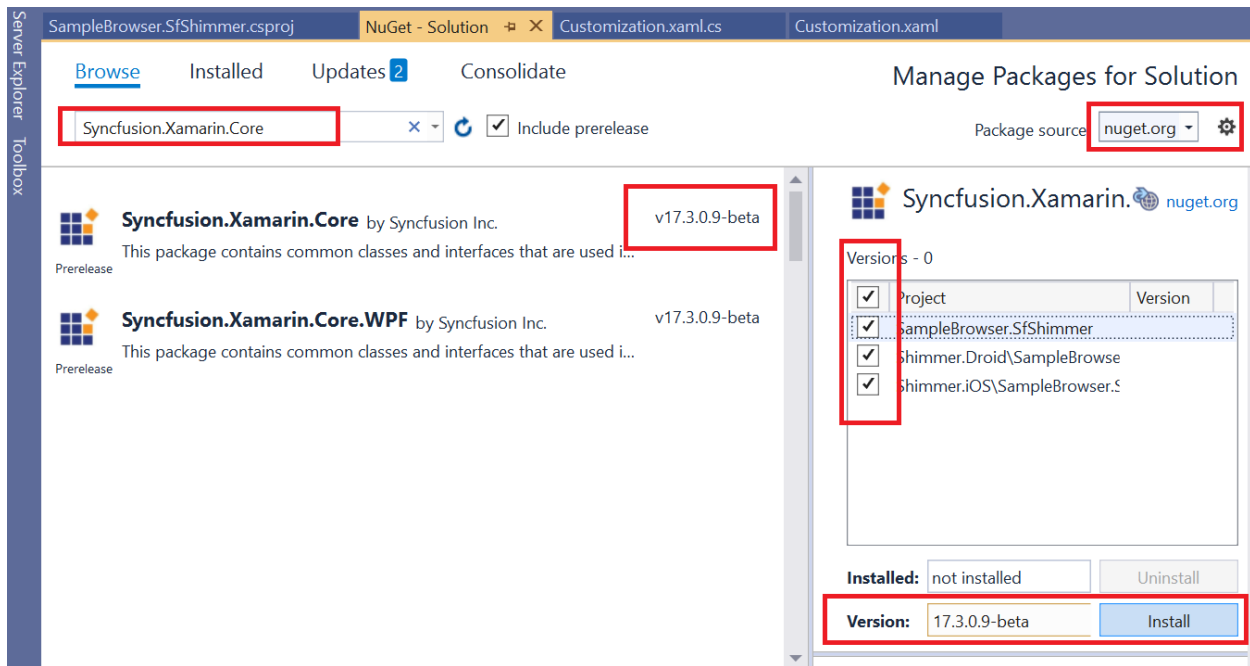
This section explains the steps required to configure the SfEffectsView control.

#### Adding SfEffectsView reference

You can add SfEffectsView reference using one of the following methods:

##### Method 1: Adding SfEffectsView reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Core). To add SfEffectsView to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Core](https://www.nuget.org/packages/Syncfusion.Xamarin.Core), and then install it.



**Note:** SfEffectsView is supported in Android and iOS.

### Method 2: Adding SfEffectsView reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfEffectsView control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfEffectsView assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location : {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching an application on each platform with SfEffectsView

To use the SfEffectsView inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and are discussed in the following sections:

---

**Note:** If you are adding the references from toolbox, the following steps are not needed.

---

#### iOS

To launch the SfEffectsView in iOS, call the `SfEffectsViewRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    global::Xamarin.Forms.Forms.Init();
    SfEffectsViewRenderer.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### Android

Android platform does not require any additional configuration to render the SfEffectsView control.

#### Initializing SfEffectsView

Import the [SfEffectsView](#) control namespace in respective page as demonstrated in the following code sample.

#### XML

```
xmlns:sfEffectsView="clr-
namespace:Syncfusion.XForms.EffectsView;assembly=Syncfusion.Core.XForms"
```

#### C#

```
using Syncfusion.XForms.EffectsView;
```

Then, initialize the SfEffectsView as demonstrated in the following code example.

#### XML

```
<sfEffectsView:SfEffectsView x:Name="EffectsView" Margin="0,5,0,10"
CornerRadius="0,25,0,25"
RippleAnimationDuration="800">
    <Grid>
        <sfBorder:SfBorder BackgroundColor="#2196F3" BorderColor="Transparent"
BorderWidth="0">
            ...
        </sfBorder:SfBorder>
    </Grid>
</sfEffectsView:SfEffectsView>
```



**C#**

```
SfEffectsView effectsView = new SfEffectsView();
effectsView.Margin = new Thickness(0, 5, 0, 10);
effectsView.CornerRadius = new Thickness(0, 25, 0, 25);
effectsView.RippleAnimationDuration = 800;
Grid grid = new Grid();
SfBorder border = new SfBorder();
border.BackgroundColor = Color.FromHex("#2196F3");
border.BorderColor = Color.Transparent;
border.BorderWidth = 0;
...
grid.Children.Add(border);
effectsView.Content = grid;
```

**Effects**

The [SfEffectsView](#) control provides support to ripple effect, highlight effect, and more. This section explains about different effects available in the effects view control.

**Highlight**

**SfEffects.Highlight** is a smooth transition on the background color of the [SfEffectsView](#).

**XML**

```
<sfEffectsView:SfEffectsView TouchDownEffects="Highlight"
HighlightColor="#FF0000">
  <Grid BackgroundColor="White">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="3*" />
      <ColumnDefinition Width="7*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Image Source="Person.png" Margin="7"
Grid.RowSpan="4" />
    <Label Text="Laura Steffi" Grid.Column="1"
FontAttributes="Bold" Grid.Row="0" VerticalTextAlignment="Center"
Margin="15,0,0,0" Font="17" />
    <Label Text="Data Science Analyst" Grid.Column="1" Grid.Row="1"
VerticalTextAlignment="Center" FontAttributes="Bold"
Margin="15,0,0,0" Font="14" />
```

```
<Label Text="laurasteffi@gmail.com" Grid.Column="1" Grid.Row="2"
VerticalTextAlignment="Center" Margin="15,0,0,0" Font="14"/>
<Label Text="011-253-321" Grid.Column="1" Grid.Row="3"
VerticalTextAlignment="Center" Margin="15,0,0,0" Font="14"/>
</Grid>
</sfEffectsView:SfEffectsView>
```

**C#**

```
var effectsView = new SfEffectsView
{
    HighlightColor = Color.FromHex("#FF0000"),
    TouchDownEffects = SfEffects.Highlight,
};
var grid = new Grid
{
    BackgroundColor = Color.White,
    ColumnDefinitions = new ColumnDefinitionCollection()
    {
        new ColumnDefinition{ Width = new GridLength(3, GridUnitType.Star) },
        new ColumnDefinition{ Width = new GridLength(7, GridUnitType.Star) }
    },
};
var image = new Image
{
    Source = "Person.png",
    Margin = 7
};
var name = new Label
{
    Text = "Laura Steffi",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15,0,0,0),
    FontSize = 17
};
var designation = new Label
{
    Text = "Data Science Analyst",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var email = new Label
{
    Text = "laurasteffi@gmail.com",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var phone = new Label
{
    Text = "011-253-321",
    FontAttributes = FontAttributes.Bold,
```

```

VerticalTextAlignment = TextAlignment.Center,
Margin = new Thickness(15, 0, 0, 0),
FontSize = 14
};
grid.Children.Add(image, 0, 0);
Grid.SetRowSpan(image, 4);
grid.Children.Add(name, 1, 0);
grid.Children.Add(designation, 1, 1);
grid.Children.Add(email, 1, 2);
grid.Children.Add(phone, 1, 3);
effectsView.Content = grid;
this.Content = effectsView;

```



### Ripple

The `SfEffects.Ripple` is a growable circle, which is initially placed on the tapped location, and it grows till the whole layout is filled. `SfEffects.Ripple` is rendered based on [InitialRippleFactor](#).

### XML

```

<sfEffectsView:SfEffectsView HorizontalOptions="Center"
TouchDownEffects="Ripple">
<Label BackgroundColor="#D3D2D5" FontAttributes="Bold" FontSize="18"
HeightRequest="50" HorizontalTextAlignment="Center" Text="Ripple"
VerticalOptions="Center" VerticalTextAlignment="Center" WidthRequest="90" />
</sfEffectsView:SfEffectsView>

```

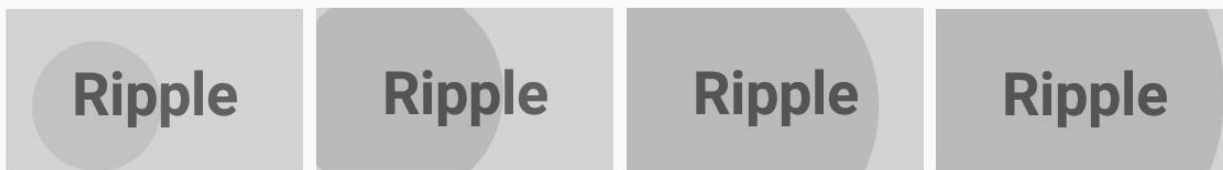
### C#

```

var effectsView = new SfEffectsView()
{
    TouchDownEffects = SfEffects.Ripple,
    HorizontalOptions = LayoutOptions.Center,
    Content = new Label()
    {
        Text = "Ripple",
        HeightRequest = 50,
        WidthRequest = 90,
        BackgroundColor = Color.FromHex("#D3D2D5"),
        FontAttributes = FontAttributes.Bold,
        FontSize = 18,
    }
};

```

```
HorizontalTextAlignment = TextAlignment.Center,
VerticalTextAlignment = TextAlignment.Center,
VerticalOptions = LayoutOptions.Center,
};
this.Content = effectsView;
```



### Scale

**SfEffects.Scale** is a smooth transition from actual size of the object to the size calculated based on [ScaleFactor](#) in pixels.

### XML

```
<sfEffectsView:SfEffectsView TouchDownEffects="None" TouchUpEffects="None"
LongPressEffects="Scale" ScaleFactor="0.85">
<Image Source="Biscuits.png" Aspect="Fill"/>
</sfEffectsView:SfEffectsView>
```

### C#

```
var effectsView = new SfEffectsView
{
    ScaleFactor = 0.85,
    TouchDownEffects = SfEffects.None,
    TouchUpEffects = SfEffects.None,
    LongPressEffects = SfEffects.Scale,
    Content = new Image()
    {
        Source = "Biscuits.png",
        Aspect = Aspect.Fill
    }
};
this.Content = effectsView;
```



### Selection

**SfEffects.Selection** is a smooth color transition to indicate the view state is moved to selected state.

### XML

```
<sfEffectsView:SfEffectsView SelectionColor="#FF0000"  
LongPressEffects="Selection">  
<Image Source="Biscuits.png" Aspect="Fill"/>  
</sfEffectsView:SfEffectsView>
```

### C#

```
var effectsView = new SfEffectsView  
{  
    SelectionColor = Color.FromHex("#FF0000"),  
    LongPressEffects = SfEffects.Selection,  
    Content = new Image()  
    {  
        Source = "Biscuits.png",  
        Aspect = Aspect.Fill  
    }  
};  
this.Content = effectsView;
```



### Rotation

`SfEffects.Rotation` provides a circular movement to the [SfEffectsView](#) around the center of the [SfEffectsView](#) based on the specified [Angle](#).

### XML

```
<sfEffectsView:SfEffectsView Angle="180" TouchDownEffects="Rotation">  
<Image Source="Arrow.png" HeightRequest="70" WidthRequest="70"/>  
</sfEffectsView:SfEffectsView>
```

### C#

```
var effectsView = new SfEffectsView  
{  
    Angle = 180,  
    TouchDownEffects = SfEffects.Rotation,  
    Content = new Image()  
    {  
        Source = "Arrow.png",  
        Aspect = Aspect.Fill  
    }  
};  
this.Content = effectsView;
```

### Combinations

The [SfEffectsView](#) control provides support to apply multiple [SfEffects](#) in combination. The following are some valid combinations of [SfEffects](#):

#### *Highlight and Ripple*

##### **XML**

```
<sfEffectsView:SfEffectsView TouchDownEffects="Highlight,Ripple">
<Label Text="Ripple" FontAttributes="Bold" FontSize="18" HeightRequest="50"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center" />
</sfEffectsView:SfEffectsView>
```

##### **C#**

```
var effectsView = new SfEffectsView()
{
    TouchDownEffects = SfEffects.Highlight | SfEffects.Ripple,
    Content = new Label()
    {
        Text = "Ripple",
        FontAttributes = FontAttributes.Bold,
        FontSize = 18,
        HeightRequest = 50,
        HorizontalTextAlignment = TextAlignment.Center,
        VerticalTextAlignment = TextAlignment.Center,
    }
};
this.Content = effectsView;
```

#### *Highlight and Selection*

##### **XML**

```
<sfEffectsView:SfEffectsView TouchDownEffects="Highlight"
LongPressEffects="Selection">
<Label Text="Sign up" FontAttributes="Bold" FontSize="18" HeightRequest="50"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center" />
</sfEffectsView:SfEffectsView>
```

##### **C#**

```
var effectsView = new SfEffectsView()
{
    TouchDownEffects = SfEffects.Highlight,
    LongPressEffects = SfEffects.Selection,
    Content = new Label()
    {
        Text = "Sign up",
        HeightRequest = 50,
        FontAttributes = FontAttributes.Bold,
        FontSize = 18,
        HorizontalTextAlignment = TextAlignment.Center,
        VerticalTextAlignment = TextAlignment.Center,
    }
};
this.Content = effectsView;
```

*Ripple and Selection***XML**

```
<sfEffectsView:SfEffectsView TouchDownEffects="Ripple"
TouchUpEffects="Selection">
<Label Text="Sign up" FontAttributes="Bold" FontSize="18" HeightRequest="50"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center" />
</sfEffectsView:SfEffectsView>
```

**C#**

```
var effectsView = new SfEffectsView()
{
    TouchDownEffects = SfEffects.Ripple,
    TouchUpEffects = SfEffects.Selection,
    Content = new Label()
    {
        Text = "Sign up",
        HeightRequest = 50,
        FontAttributes = FontAttributes.Bold,
        FontSize = 18,
        HorizontalTextAlignment = TextAlignment.Center,
        VerticalTextAlignment = TextAlignment.Center,
    }
};
this.Content = effectsView;
```

*Highlight, Ripple and Selection***XML**

```
<sfEffectsView:SfEffectsView TouchDownEffects="Highlight,Ripple"
LongPressEffects="Selection">
<Label Text="Sign up" FontAttributes="Bold" FontSize="18" HeightRequest="50"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center" />
</sfEffectsView:SfEffectsView>
```

**C#**

```
var effectsView = new SfEffectsView()
{
    TouchDownEffects = SfEffects.Highlight | SfEffects.Ripple,
    LongPressEffects = SfEffects.Selection,
    Content = new Label()
    {
        Text = "Sign up",
        HeightRequest = 50,
        FontAttributes = FontAttributes.Bold,
        FontSize = 18,
        HorizontalTextAlignment = TextAlignment.Center,
        VerticalTextAlignment = TextAlignment.Center,
    }
};
this.Content = effectsView;
```

*Scale and Selection***XML**

```
<sfEffectsView:SfEffectsView LongPressEffects="Scale,Selection">
  <Label Text="Sign up" FontAttributes="Bold" FontSize="18" HeightRequest="50"
    HorizontalTextAlignment="Center" VerticalTextAlignment="Center" />
</sfEffectsView:SfEffectsView>
```

**C#**

```
var effectsView = new SfEffectsView()
{
    LongPressEffects = SfEffects.Scale | SfEffects.Selection,
    Content = new Label()
    {
        Text = "Sign up",
        HeightRequest = 50,
        FontAttributes = FontAttributes.Bold,
        FontSize = 18,
        HorizontalTextAlignment = TextAlignment.Center,
        VerticalTextAlignment = TextAlignment.Center,
    }
};
this.Content = effectsView;
```

## Features in SfEffectsView

The [SfEffectsView](#) control provides the following additional features to enhance the effects.

## FadeOutRipple

By enabling the [FadeOutRipple](#) property of [SfEffectsView](#), the growable circle will lose its opacity to 0.

**XML**

```
<sfEffectsView:SfEffectsView FadeOutRipple="True"
  RippleAnimationDuration="1000">
  <Grid BackgroundColor="White">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="3*" />
      <ColumnDefinition Width="7*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Image
      Grid.RowSpan="4"
      Margin="7"
      Source="{local:ImageSourceExt SampleBrowser.SfEffectsView.Person.png}" />
    <Label
      Grid.Row="0"
      Grid.Column="1"
      Margin="15,0,0,0"
```



```
Font="17"
FontAttributes="Bold"
Text="Laura Steffi"
VerticalTextAlignment="Center" />
<Label
Grid.Row="1"
Grid.Column="1"
Margin="15,0,0,0"
Font="14"
FontAttributes="Bold"
Text="Data Science Analyst"
VerticalTextAlignment="Center" />
<Label
Grid.Row="2"
Grid.Column="1"
Margin="15,0,0,0"
Font="14"
Text="laurasteffi@gmail.com"
VerticalTextAlignment="Center" />
<Label
Grid.Row="3"
Grid.Column="1"
Margin="15,0,0,0"
Font="14"
Text="011-253-321"
VerticalTextAlignment="Center" />
</Grid>
</sfEffectsView:SfEffectsView>
```

## C#

```
var effectsView = new SfEffectsView
{
    FadeOutRipple = true,
    RippleAnimationDuration = 1000
};
var grid = new Grid
{
    BackgroundColor = Color.White,
    ColumnDefinitions = new ColumnDefinitionCollection()
    {
        new ColumnDefinition{ Width = new GridLength(3, GridUnitType.Star) },
        new ColumnDefinition{ Width = new GridLength(7, GridUnitType.Star) }
    },
};
var image = new Image
{
    Source = "Person.png",
    Margin = 7
};
var name = new Label
{
    Text = "Laura Steffi",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
```

```
FontSize = 17
};
var designation = new Label
{
    Text = "Data Science Analyst",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var email = new Label
{
    Text = "laurasteffi@gmail.com",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var phone = new Label
{
    Text = "011-253-321",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
grid.Children.Add(image, 0, 0);
Grid.SetRowSpan(image, 4);
grid.Children.Add(name, 1, 0);
grid.Children.Add(designation, 1, 1);
grid.Children.Add(email, 1, 2);
grid.Children.Add(phone, 1, 3);
effectsView.Content = grid;
this.Content = effectsView;
```



### IsSelected

Enabling the [IsSelected](#) property of [SfEffectsView](#) sets the view state as selected.

### XML

```
<sfEffectsView:SfEffectsView IsSelected="true" LongPressEffects="Selection">
<Image Source="Biscuits.png" Aspect="Fill"/>
```

```
</sfEffectsView:SfEffectsView>
```

### C#

```
var effectsView = new SfEffectsView
{
    IsSelected = true,
    LongPressEffects = SfEffects.Selection,
    Content = new Image()
    {
        Source = "Biscuits.png",
        Aspect = Aspect.Fill
    }
};
this.Content = effectsView;
```

### ShouldIgnoreTouches

Enabling the [ShouldIgnoreTouches](#) property of [SfEffectsView](#) cancels the direct interaction of the [SfEffectsView](#).

### XML

```
<sfEffectsView:SfEffectsView ShouldIgnoreTouches="true">
<Image Source="Biscuits.png" Aspect="Fill"/>
</sfEffectsView:SfEffectsView>
```

### C#

```
var effectsView = new SfEffectsView
{
    ShouldIgnoreTouches = true,
    Content = new Image()
    {
        Source = "Biscuits.png",
        Aspect = Aspect.Fill
    }
};
this.Content = effectsView;
```

### Interaction in SfEffectsView

You can set the effects on different interactions. This section explains how to set effects on different interactions available in effects view.

### TouchDownEffects

The [TouchDownEffects](#) property of [SfEffectsView](#) is used to render the effects in touch-down interaction.

### XML

```
<sfEffectsView:SfEffectsView TouchDownEffects="Ripple">
<Image Source="Biscuits.png" Aspect="Fill"/>
</sfEffectsView:SfEffectsView>
```

**C#**

```
var effectsView = new SfEffectsView
{
    TouchDownEffects = SfEffects.Ripple,
    Content = new Image()
    {
        Source = "Biscuits.png",
        Aspect = Aspect.Fill
    }
};
this.Content = effectsView;
```

## LongPressEffects

The [LongPressEffects](#) property of [SfEffectsView](#) is used to render the effects in long-press interaction.

**XML**

```
<sfEffectsView:SfEffectsView LongPressEffects="Ripple">
<Image Source="Biscuits.png" Aspect="Fill"/>
</sfEffectsView:SfEffectsView>
```

**C#**

```
var effectsView = new SfEffectsView
{
    LongPressEffects = SfEffects.Ripple,
    Content = new Image()
    {
        Source = "Biscuits.png",
        Aspect = Aspect.Fill
    }
};
this.Content = effectsView;
```

## TouchUpEffects

The [TouchUpEffects](#) property of [SfEffectsView](#) is used to render the effects in touch-up interaction.

**XML**

```
<sfEffectsView:SfEffectsView TouchUpEffects="Ripple">
<Image Source="Biscuits.png" Aspect="Fill"/>
</sfEffectsView:SfEffectsView>
```

**C#**

```
var effectsView = new SfEffectsView
{
    TouchUpEffects = SfEffects.Ripple,
    Content = new Image()
    {
        Source = "Biscuits.png",
        Aspect = Aspect.Fill
    }
};
```

```
this.Content = effectsView;
```

## Methods

You can add or remove effect programmatically using the [ApplyEffects](#) or [Reset](#) method.

### ApplyEffects

The [ApplyEffects](#) method is used to trigger the effects rendering with or without repetition. The following are the optional parameters to be passed:

- **effects** - [SfEffects](#) to be applied. By default, [SfEffects.Ripple](#) will be applied.
- **rippleStartPosition** - [RippleStartPosition](#) can be left, top, right, bottom, or default. By default, ripple starts from the center.
- **rippleStartPoint** - point at which ripple animation starts. The default value is null.
- **repeat** - bool value used to set whether to repeat the applied effect. The default value is false. Only [SfEffects.Ripple](#) and [SfEffects.Highlight](#) can be repeated.

## XML

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<Grid Grid.ColumnSpan="2">
<sfEffectsView:SfEffectsView
x:Name="effectsView"
RippleAnimationDuration="500"
VerticalOptions="Center">
<Image Aspect="Fill" Source="Biscuits.png" />
</sfEffectsView:SfEffectsView>
</Grid>
<Button
x:Name="ApplyScaleEffectButton"
Grid.Row="1"
BackgroundColor="Accent"
Clicked="ApplyScaleEffectButton_Clicked"
HeightRequest="40"
HorizontalOptions="CenterAndExpand"
Text="Apply Effect"
TextColor="White"
VerticalOptions="CenterAndExpand"
WidthRequest="120" />
<Button
x:Name="ScaleResetButton"
Grid.Row="1"
Grid.Column="1"
BackgroundColor="Accent"
Clicked="ScaleResetButton_Clicked"
HeightRequest="40"
```

```
HorizontalOptions="CenterAndExpand"
Text="Reset"
TextColor="White"
VerticalOptions="Center"
WidthRequest="120" />
</Grid>
```

**C#**

```
private void ApplyEffectClicked(object sender, EventArgs e)
{
    effectsView.ApplyEffects(effects: SfEffects.Ripple, repeat: true);
}
```

**Note:** The [SfEffects](#) applied using [ApplyEffects](#) method will be removed only after calling the [Reset](#) method.

**Reset**

The [Reset](#) method is used to reset the `SfEffects.Highlight` and `SfEffects.Ripple` effects, which are applied using the [ApplyEffects](#) method.

**XML**

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Grid Grid.ColumnSpan="2">
    <sfEffectsView:SfEffectsView
      x:Name="effectsView"
      RippleAnimationDuration="500"
      VerticalOptions="Center">
      <Image Aspect="Fill" Source="Biscuits.png" />
    </sfEffectsView:SfEffectsView>
  </Grid>
  <Button
    x:Name="ApplyScaleEffectButton"
    Grid.Row="1"
    BackgroundColor="Accent"
    Clicked="ApplyScaleEffectButton_Clicked"
    HeightRequest="40"
    HorizontalOptions="CenterAndExpand"
    Text="Apply Effect"
    TextColor="White"
    VerticalOptions="CenterAndExpand"
    WidthRequest="120" />
  <Button
    x:Name="ScaleResetButton"
    Grid.Row="1"
    Grid.Column="1"
```

```

BackgroundColor="Accent"
Clicked="ScaleResetButton_Clicked"
HeightRequest="40"
HorizontalOptions="CenterAndExpand"
Text="Reset"
TextColor="White"
VerticalOptions="Center"
WidthRequest="120" />
</Grid>

```

**C#**

```

private void ResetEffectClicked(object sender, EventArgs e)
{
    effectsView.Reset();
}

```



## Customization of SfEffectsView

The [SfEffectsView](#) control provides support to customize the corner radius, animation duration, color, and more. This section explains how to customize the effects view control.

## RippleAnimationDuration

The [RippleAnimationDuration](#) property of [SfEffectsView](#) is used to customize the duration of ripple animation.

**XML**

```

<sfEffectsView:SfEffectsView RippleAnimationDuration="800">
<Image Source="Icon.png" HeightRequest="70" WidthRequest="70"/>
</sfEffectsView:SfEffectsView>

```

**C#**

```

var effectsView = new SfEffectsView()
{
    RippleAnimationDuration = 800,
    Content = new Image()
    {
        Source = "Icon.png",
        HeightRequest = 70,
        WidthRequest = 70
    }
};
this.Content = effectsView;

```

### ScaleAnimationDuration

The [ScaleAnimationDuration](#) property of [SfEffectsView](#) is used to customize the duration of scale animation.

#### XML

```
<sfEffectsView:SfEffectsView ScaleAnimationDuration="800"
LongPressEffects="Scale">
<Image Source="Icon.png" HeightRequest="70" WidthRequest="70"/>
</sfEffectsView:SfEffectsView>
```

#### C#

```
var effectsView = new SfEffectsView()
{
    ScaleAnimationDuration = 800,
    LongPressEffects = SfEffects.Scale,
    Content = new Image()
    {
        Source = "Icon.png",
        HeightRequest = 70,
        WidthRequest = 70
    }
};
this.Content = effectsView;
```

### RotationAnimationDuration

The [RotationAnimationDuration](#) property of [SfEffectsView](#) is used to customize the duration of rotation animation.

#### XML

```
<sfEffectsView:SfEffectsView RotationAnimationDuration="800"
TouchDownEffects="Rotation">
<Image Source="Icon.png" HeightRequest="70" WidthRequest="70"/>
</sfEffectsView:SfEffectsView>
```

#### C#

```
var effectsView = new SfEffectsView()
{
    RotationAnimationDuration = 800,
    LongPressEffects = SfEffects.Rotation,
    Content = new Image()
    {
        Source = "Icon.png",
        HeightRequest = 70,
        WidthRequest = 70
    }
};
this.Content = effectsView;
```

### InitialRippleFactor

The [InitialRippleFactor](#) property of [SfEffectsView](#) is used to customize the initial radius of ripple.



**XML**

```

<sfEffectsView:SfEffectsView InitialRippleFactor="0.1">
<Grid BackgroundColor="#2196F3">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="3*" />
<ColumnDefinition Width="7*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition />
<RowDefinition />
<RowDefinition />
<RowDefinition />
</Grid.RowDefinitions>
<Image Source="Person.png" Margin="7"
Grid.RowSpan="4" />
<Label Text="Laura Steffi" Grid.Column="1" TextColor="White"
FontAttributes="Bold" Grid.Row="0" VerticalTextAlignment="Center"
Margin="15,0,0,0" Font="17" />
<Label Text="Data Science Analyst" Grid.Column="1" Grid.Row="1"
TextColor="WhiteSmoke"
VerticalTextAlignment="Center" FontAttributes="Bold"
Margin="15,0,0,0" Font="14" />
<Label Text="laurasteffi@gmail.com" Grid.Column="1" Grid.Row="2"
TextColor="WhiteSmoke"
VerticalTextAlignment="Center" Margin="15,0,0,0" Font="14" />
<Label Text="011-253-321" Grid.Column="1" Grid.Row="3"
TextColor="WhiteSmoke"
VerticalTextAlignment="Center" Margin="15,0,0,0" Font="14" />
</Grid>
</sfEffectsView:SfEffectsView>

```

**C#**

```

var effectsView = new SfEffectsView
{
    InitialRippleFactor = 0.1
};
var grid = new Grid
{
    BackgroundColor = Color.FromHex("#2196F3"),
    ColumnDefinitions = new ColumnDefinitionCollection()
    {
        new ColumnDefinition{ Width = new GridLength(3, GridUnitType.Star) },
        new ColumnDefinition{ Width = new GridLength(7, GridUnitType.Star) }
    },
};
var image = new Image
{
    Source = "Person.png",
    Margin = 7
};
var name = new Label
{
    Text = "Laura Steffi",
    TextColor = Color.White,

```

```

FontAttributes = FontAttributes.Bold,
VerticalTextAlignment = TextAlignment.Center,
Margin = new Thickness(15, 0, 0, 0),
FontSize = 17
};
var designation = new Label
{
    Text = "Data Science Analyst",
    TextColor = Color.WhiteSmoke,
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var email = new Label
{
    Text = "laurasteffi@gmail.com",
    TextColor = Color.WhiteSmoke,
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var phone = new Label
{
    Text = "011-253-321",
    TextColor = Color.WhiteSmoke,
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
grid.Children.Add(image, 0, 0);
Grid.SetRowSpan(image, 4);
grid.Children.Add(name, 1, 0);
grid.Children.Add(designation, 1, 1);
grid.Children.Add(email, 1, 2);
grid.Children.Add(phone, 1, 3);
effectsView.Content = grid;
this.Content = effectsView;

```



### ScaleFactor

The [ScaleFactor](#) property of [SfEffectsView](#) is used to customize the scale of the view.

### XML

```

<sfEffectsView:SfEffectsView TouchDownEffects="None" TouchUpEffects="None"
LongPressEffects="Scale" ScaleFactor="0.85">
<Image Source="Biscuits.png" Aspect="Fill"/>
</sfEffectsView:SfEffectsView>

```

**C#**

```

var effectsView = new SfEffectsView
{
    ScaleFactor = 0.85,
    TouchDownEffects = SfEffects.None,
    TouchUpEffects = SfEffects.None,
    LongPressEffects = SfEffects.Scale,
    Content = new Image()
    {
        Source = "Biscuits.png",
        Aspect = Aspect.Fill
    }
};
this.Content = effectsView;

```

**HighlightColor**

The [HighlightColor](#) property of [SfEffectsView](#) is used to customize the color of highlight effect.

**XML**

```

<sfEffectsView:SfEffectsView TouchDownEffects="Highlight"
HighlightColor="#2196F3">
  <Grid BackgroundColor="White">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="3*" />
      <ColumnDefinition Width="7*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Image Source="Person.png" Margin="7"
Grid.RowSpan="4" />
    <Label Text="Laura Steffi" Grid.Column="1"
FontAttributes="Bold" Grid.Row="0" VerticalTextAlignment="Center"
Margin="15,0,0,0" Font="17" />
    <Label Text="Data Science Analyst" Grid.Column="1" Grid.Row="1"
VerticalTextAlignment="Center" FontAttributes="Bold"
Margin="15,0,0,0" Font="14" />
    <Label Text="laurasteffi@gmail.com" Grid.Column="1" Grid.Row="2"
VerticalTextAlignment="Center" Margin="15,0,0,0" Font="14" />
    <Label Text="011-253-321" Grid.Column="1" Grid.Row="3"
VerticalTextAlignment="Center" Margin="15,0,0,0" Font="14" />
  </Grid>
</sfEffectsView:SfEffectsView>

```

**C#**

```
var effectsView = new SfEffectsView
{
    HighlightColor = Color.FromHex("#2196F3"),
    TouchDownEffects = SfEffects.Highlight,
};
var grid = new Grid
{
    BackgroundColor = Color.White,
    ColumnDefinitions = new ColumnDefinitionCollection()
    {
        new ColumnDefinition{ Width = new GridLength(3, GridUnitType.Star) },
        new ColumnDefinition{ Width = new GridLength(7, GridUnitType.Star) }
    },
};
var image = new Image
{
    Source = "Person.png",
    Margin = 7
};
var name = new Label
{
    Text = "Laura Steffi",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 17
};
var designation = new Label
{
    Text = "Data Science Analyst",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var email = new Label
{
    Text = "laurasteffi@gmail.com",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var phone = new Label
{
    Text = "011-253-321",
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
grid.Children.Add(image, 0, 0);
Grid.SetRowSpan(image, 4);
grid.Children.Add(name, 1, 0);
grid.Children.Add(designation, 1, 1);
```

```
grid.Children.Add(email, 1, 2);  
grid.Children.Add(phone, 1, 3);  
effectsView.Content = grid;  
this.Content = effectsView;
```



### RippleColor

The [RippleColor](#) property of [SfEffectsView](#) is used to customize the color of ripple.

### XML

```
<sfEffectsView:SfEffectsView RippleColor="#2196F3">  
<Image Source="Biscuits.png" Aspect="Fill"/>  
</sfEffectsView:SfEffectsView>
```

### C#

```
var effectsView = new SfEffectsView  
{  
    RippleColor = Color.FromHex("#2196F3"),  
    Content = new Image()  
    {  
        Source = "Biscuits.png",  
        Aspect = Aspect.Fill  
    }  
};  
this.Content = effectsView;
```



### SelectionColor

The [SelectionColor](#) property of [SfEffectsView](#) is used to customize the color of selection effect.

### XML

```
<sfEffectsView:SfEffectsView SelectionColor="#2196F3"
LongPressEffects="Selection">
<Image Source="Biscuits.png" Aspect="Fill"/>
</sfEffectsView:SfEffectsView>
```

### C#

```
var effectsView = new SfEffectsView
{
    SelectionColor = Color.FromHex("#2196F3"),
    LongPressEffects = SfEffects.Selection,
    Content = new Image()
    {
        Source = "Biscuits.png",
        Aspect = Aspect.Fill
    }
};
this.Content = effectsView;
```



### CornerRadius

The [CornerRadius](#) property of [SfEffectsView](#) is used to customize the corner radius of the [SfEffectsView](#) control.

### XML

```
<sfEffectsView:SfEffectsView CornerRadius="0,25">
  <Grid BackgroundColor="#2196F3">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="3*" />
      <ColumnDefinition Width="7*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Image Source="Person.png" Margin="7"
      Grid.RowSpan="4" />
    <Label Text="Laura Steffi" Grid.Column="1" TextColor="White"
      FontAttributes="Bold" Grid.Row="0" VerticalTextAlignment="Center"
      Margin="15,0,0,0" Font="17" />
    <Label Text="Data Science Analyst" Grid.Column="1" Grid.Row="1"
      TextColor="WhiteSmoke"
      VerticalTextAlignment="Center" FontAttributes="Bold"
      Margin="15,0,0,0" Font="14" />
    <Label Text="laurasteffi@gmail.com" Grid.Column="1" Grid.Row="2"
      TextColor="WhiteSmoke"
      VerticalTextAlignment="Center" Margin="15,0,0,0" Font="14" />
    <Label Text="011-253-321" Grid.Column="1" Grid.Row="3"
      TextColor="WhiteSmoke"
      VerticalTextAlignment="Center" Margin="15,0,0,0" Font="14" />
  </Grid>
</sfEffectsView:SfEffectsView>
```

### C#

```
var effectsView = new SfEffectsView
{
    CornerRadius = new Thickness(0, 25)
};
var grid = new Grid
{
    BackgroundColor = Color.FromHex("#2196F3"),
    ColumnDefinitions = new ColumnDefinitionCollection()
    {
        new ColumnDefinition{ Width = new GridLength(3, GridUnitType.Star) },
        new ColumnDefinition{ Width = new GridLength(7, GridUnitType.Star) }
    },
};
var image = new Image
{
    Source = "Person.png",
    Margin = 7
};
```

```
var name = new Label
{
    Text = "Laura Steffi",
    TextColor = Color.White,
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 17
};
var designation = new Label
{
    Text = "Data Science Analyst",
    TextColor = Color.WhiteSmoke,
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var email = new Label
{
    Text = "laurasteffi@gmail.com",
    TextColor = Color.WhiteSmoke,
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
var phone = new Label
{
    Text = "011-253-321",
    TextColor = Color.WhiteSmoke,
    FontAttributes = FontAttributes.Bold,
    VerticalTextAlignment = TextAlignment.Center,
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 14
};
grid.Children.Add(image, 0, 0);
Grid.SetRowSpan(image, 4);
grid.Children.Add(name, 1, 0);
grid.Children.Add(designation, 1, 1);
grid.Children.Add(email, 1, 2);
grid.Children.Add(phone, 1, 3);
effectsView.Content = grid;
this.Content = effectsView;
```





### Angle

The [Angle](#) property of [SfEffectsView](#) is used to customize the rotation angle.

### XML

```
<sfEffectsView:SfEffectsView Angle="180" TouchDownEffects="Ripple,Rotation">
  <Image Source="Arrow.png" HeightRequest="70" WidthRequest="70"/>
</sfEffectsView:SfEffectsView>
```

### C#

```
var effectsView = new SfEffectsView
{
    Angle = 180,
    TouchDownEffects = SfEffects.Ripple | SfEffects.Rotation,
    Content = new Image()
    {
        Source = "Arrow.png",
        Aspect = Aspect.Fill
    }
};
this.Content = effectsView;
```



ROTATE TO 180°

## SfExpander

### Getting started

The Expander control provides a way to expand and collapse when tapping a header. This section provides a quick overview for working with the `SfExpander` for Xamarin.Forms. This section covers the minimal features that you need to know to get started with the Expander.

### Assembly deployment

After installing Essential Studio for Xamarin, you can find all the required assemblies in the installation folders, {Syncfusion Essential Studio Installed location} \Essential Studio\16.x.x.x\Xamarin\lib

Eg: C:\Program Files (x86) \Syncfusion\Essential Studio\16.1.0.24\Xamarin\lib

---

**Note:** Assemblies can be found in unzipped package location in Mac.

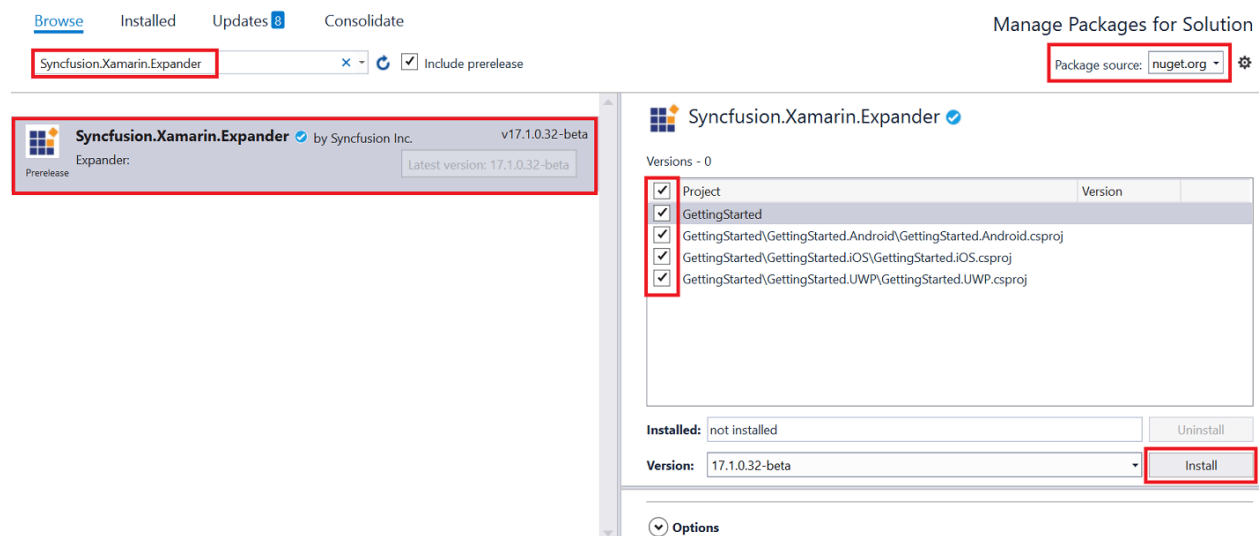
---

### Adding SfExpander reference

You can add SfExpander reference using one of the following methods:

#### Method 1: Adding SfExpander reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Expander). To add SfExpander to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Expander](https://www.nuget.org/packages/Syncfusion.Xamarin.Expander), and then install it.




---

**Note:** Install the same version of Expander NuGet in all the projects.

---

#### Method 2: Adding SfExpander reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfExpander control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfExpander assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Expander.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Expander.XForms.dll Syncfusion.Expander.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Expander.XForms.dll Syncfusion.Expander.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Expander.XForms.dll Syncfusion.Expander.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the expander on each platform

To use the expander in an application, each platform application must initialize the expander renderer. This initialization step varies from platform to platform and is discussed in the following sections:

#### Android

The Android launches the expander without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch the expander in iOS, call the `SfExpanderRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework initialization and before the `LoadApplication` is called, as demonstrated in the following code example:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.XForms.iOS.Expander.SfExpanderRenderer.Init();
}
```

```
LoadApplication (new App ());  
...  
}
```

### Universal Windows Platform (UWP)

The UWP launches the expander without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

### ReleaseMode issue in UWP platform

The known Framework issue in UWP platform is that the custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the expander assemblies in **App.xaml.cs** file in UWP project as in the following code snippet:

### C#

```
// In App.xaml.cs  
protected override void OnLaunched(LaunchActivatedEventArgs e)  
{  
    ...  
    rootFrame.NavigationFailed += OnNavigationFailed;  
    // you'll need to add `using System.Reflection;`  
    List<Assembly> assembliesToInclude = new List<Assembly>();  
    //Now, add all the assemblies your app uses  
    assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.Expander.SfExpanderRender  
er).GetType().Assembly);  
    // replaces Xamarin.Forms.Forms.Init(e);  
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);  
    ...  
}
```

### Creating the Expander

This section explains how to create a simple Xamarin.Forms application with [SfExpander](#). The control should be configured entirely in C# code or by using XAML markup.

- Creating the project.
- Adding expander in Xamarin.Forms.
- Defining expander.

### Creating the project

Create a new blank (.Net Standard) application in Xamarin Studio or Visual Studio for Xamarin.Forms.

### Adding expander in Xamarin.Forms:

To add the expander to your application, follow the steps:

1. Add required assemblies as discussed in assembly deployment section.
2. Import the control namespace as `xmlns:syncfusion="clr-namespace:Syncfusion.XForms.Expander;assembly=Syncfusion.Expander.XForms"` in XAML Page.
3. Create an instance of expander control and add as content for content page.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.Expander;assembly=Syncfusion.Expander.XForms">
<ContentPage.Content>
<syncfusion:SfExpander x:Name="expander"/>
</ContentPage.Content>
</ContentPage>
```

**C#**

```
using Syncfusion.XForms.Expander;
using Xamarin.Forms;
namespace GettingStarted
{
public partial class MainPage : ContentPage
{
SfExpander expander;
public MainPage()
{
InitializeComponent();
expander = new SfExpander();
}
}
}
```

*Defining expander*

**SfExpander** is a layout control comprise of Header and Content. You can load any View in [Header](#) and [Content](#). Content visibility of expander can be set by using **IsExpanded** property of Expander. User can expand or collapse the Content view by tapping Header.

Here, Labels are loaded in Header and Content of expander.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.Expander;assembly=Syncfusion.Expander.XForms">
<ContentPage.Content>
<ScrollView BackgroundColor="#EDF2F5" Grid.Row="1">
<StackLayout>
<syncfusion:SfExpander>
<syncfusion:SfExpander.Header>
<Label TextColor="#495F6E" Text="Veg Pizza" VerticalTextAlignment="Center"
/>
</syncfusion:SfExpander.Header>
<syncfusion:SfExpander.Content>
```

```
<Grid Padding="10,10,10,10" BackgroundColor="#FFFFFF">
<Label TextColor="#303030" Text="Veg pizza is prepared with the items that
meet vegetarian standards by not including any meat or animal tissue
products." HeightRequest="50" VerticalTextAlignment="Center"/>
</Grid>
</syncfusion:SfExpander.Content>
</syncfusion:SfExpander>
<syncfusion:SfExpander>
<syncfusion:SfExpander.Header>
<Label TextColor="#495F6E" Text="Non-veg Pizza"
VerticalTextAlignment="Center" />
</syncfusion:SfExpander.Header>
<syncfusion:SfExpander.Content>
<Grid Padding="10,10,10,10" BackgroundColor="#FFFFFF">
<Label TextColor="#303030" Text="Non-veg pizza is prepared by including the
meat and animal tissue products." HeightRequest="50"
VerticalTextAlignment="Center"/>
</Grid>
</syncfusion:SfExpander.Content>
</syncfusion:SfExpander>
</StackLayout>
</ScrollView>
</ContentPage.Content>
</ContentPage>
```

Now, run the application to render the following output.

### ^ Veg Pizza

Veg pizza is prepared with the items that meet vegetarian standards by not including any meat or animal tissue products.

### ^ Non-veg Pizza

Non-veg pizza is prepared by including the meat and animal tissue products.

You can download expander sample for Xamarin.Forms from here [ExpanderGettingStarted](#).

### Animation duration

**SfExpander** allows to customize the expanding and collapsing duration by using [AnimationDuration](#) property. By default, the animation duration is 150 milliseconds.

#### XML

```
<syncfusion:SfExpander x:Name="expander" AnimationDuration="250"/>
```

#### C#

```
expander.AnimationDuration = 250;
```

### Animation easing

**SfExpander** allows to customize the rate of change of parameter over time or animation style by using [AnimationEasing](#) property. By default, the animation easing is **Linear**.

#### XML

```
<syncfusion:SfExpander x:Name="expander" AnimationEasing="SinOut"/>
```

#### C#

```
expander.AnimationEasing =  
Syncfusion.XForms.Expander.AnimationEasing.SinOut;
```

### Expand and Collapse

**SfExpander** allows to programmatically expand and collapse by using [IsExpanded](#) property of **SfExpander**. Also, expand & collapse interaction by user can be control by handling **Expanding** and **Collapsing** events.

#### XML

```
<syncfusion:SfExpander x:Name="expander" IsExpanded="True"/>
```

#### C#

```
expander.IsExpanded = true;
```

### Appearance

The **Expander** allows customizing appearance of the Icon, and provides various functionalities to the end-user.

### Header icon position

The [SfExpander](#) allows to customize the position of the header icon by using [HeaderIconPosition](#) property. By default, the header icon position is **Start**.

#### XML

```
<syncfusion:SfExpander x:Name="expander" HeaderIconPosition="End" />
```

#### C#



```
expander.HeaderIconPosition = Syncfusion.XForms.Expander.IconPosition.End;
```

### Header background color customization

The [SfExpander](#) allows to customize the background color of the expander header by using [HeaderBackgroundColor](#) property.

#### XML

```
<syncfusion:SfExpander x:Name="expander" HeaderBackgroundColor="Pink"/>
```

#### C#

```
expander.HeaderBackgroundColor = Color.Pink;
```

### Icon color customization

The [SfExpander](#) allows to customize the color of the expander icon by using [IconColor](#) property. By default, [IconColor](#) is black.

#### XML

```
<syncfusion:SfExpander x:Name="expander" IconColor="Accent"/>
```

#### C#

```
expander.IconColor = Color.Accent;
```

### Visual State Manager

The appearance of the [SfExpander](#) can be customized using the following two [VisualStates](#):

- Expanded
- Collapsed

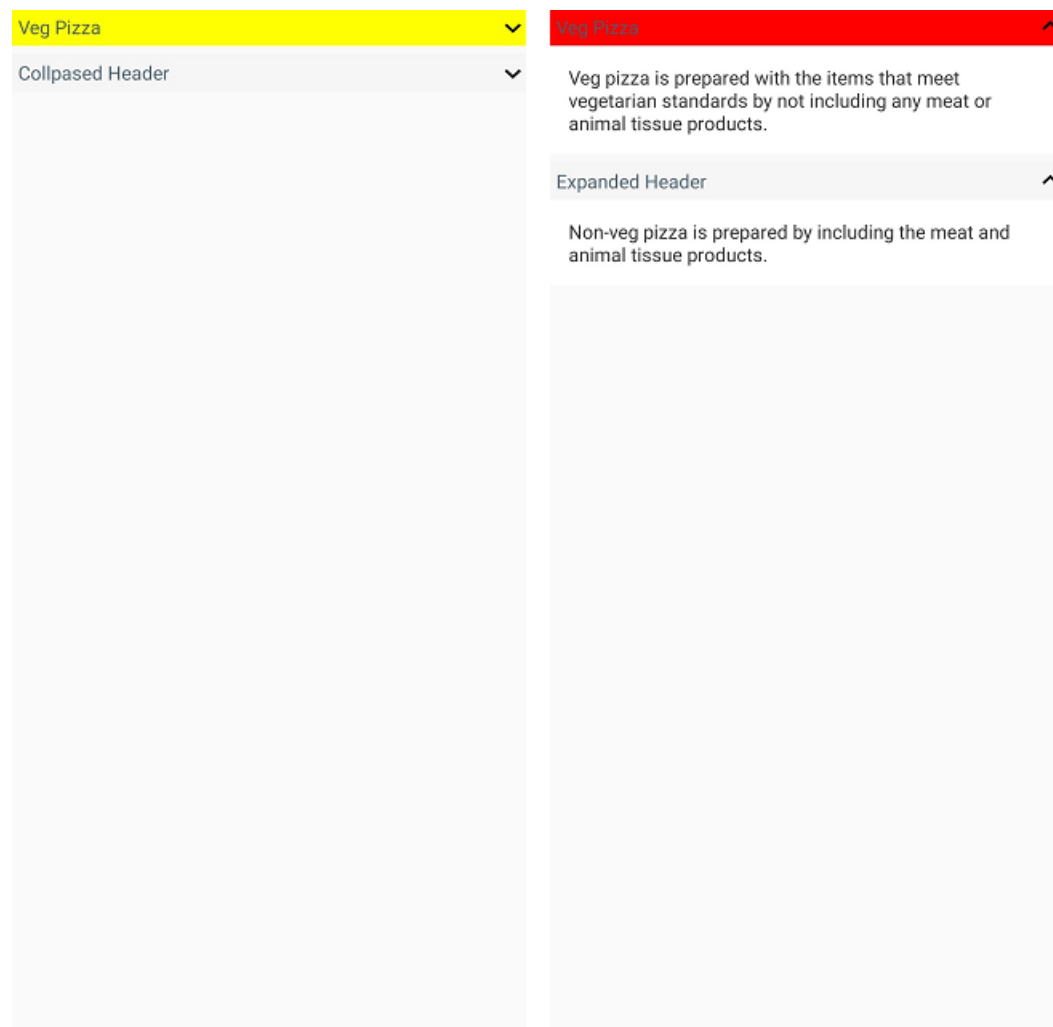
#### XML

```
<syncfusion:SfExpander x:Name="expander">
  <syncfusion:SfExpander.Header>
    <Label Text="Veg Pizza" VerticalTextAlignment="Center"/>
  </syncfusion:SfExpander.Header>
  <syncfusion:SfExpander.Content>
    <Label HeightRequest="50" Text="Veg pizza is prepared with the items that
    meet vegetarian standards by not including any meat or animal tissue
    products." VerticalTextAlignment="Center"/>
  </syncfusion:SfExpander.Content>
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroupList>
      <VisualStateGroup>
        <VisualState Name="Expanded">
          <VisualState.Setters>
            <Setter Property="HeaderBackgroundColor" Value="Red"/>
          </VisualState.Setters>
        </VisualState>
        <VisualState Name="Collapsed">
```

```
<VisualState.Setters>
<Setter Property="HeaderBackgroundColor" Value="Green"/>
</VisualState.Setters>
</VisualState>
</VisualStateGroup>
</VisualStateGroupList>
</VisualStateManager.VisualStateGroups>
</syncfusion:SfExpander>
```

## C#

```
SfExpander expander = new SfExpander();
expander.Header = new Label()
{
    Text="Veg Pizza"
};
expander.Content = new Label()
{
    Text="Veg pizza is prepared with the items that meet vegetarian standards by not including any meat or animal tissue products."
};
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState expanded = new VisualState
{
    Name = "Expanded"
};
expanded.Setters.Add(new Setter { Property =
SfExpander.HeaderBackgroundColorProperty, Value = Color.Red });
expanded.Setters.Add(new Setter { Property =
SfExpander.HeaderBackgroundColorProperty, Value = Color.Red });
VisualState collapsed = new VisualState
{
    Name = "Collapsed"
};
collapsed.Setters.Add(new Setter { Property =
SfExpander.HeaderBackgroundColorProperty, Value = Color.Green });
collapsed.Setters.Add(new Setter { Property =
SfExpander.HeaderBackgroundColorProperty, Value = Color.Green });
commonStateGroup.States.Add(expanded);
commonStateGroup.States.Add(collapsed);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(expander, visualStateGroupList);
this.Content = expander;
```



You can download the entire source of this demo from [here](#).

## SfGradientView

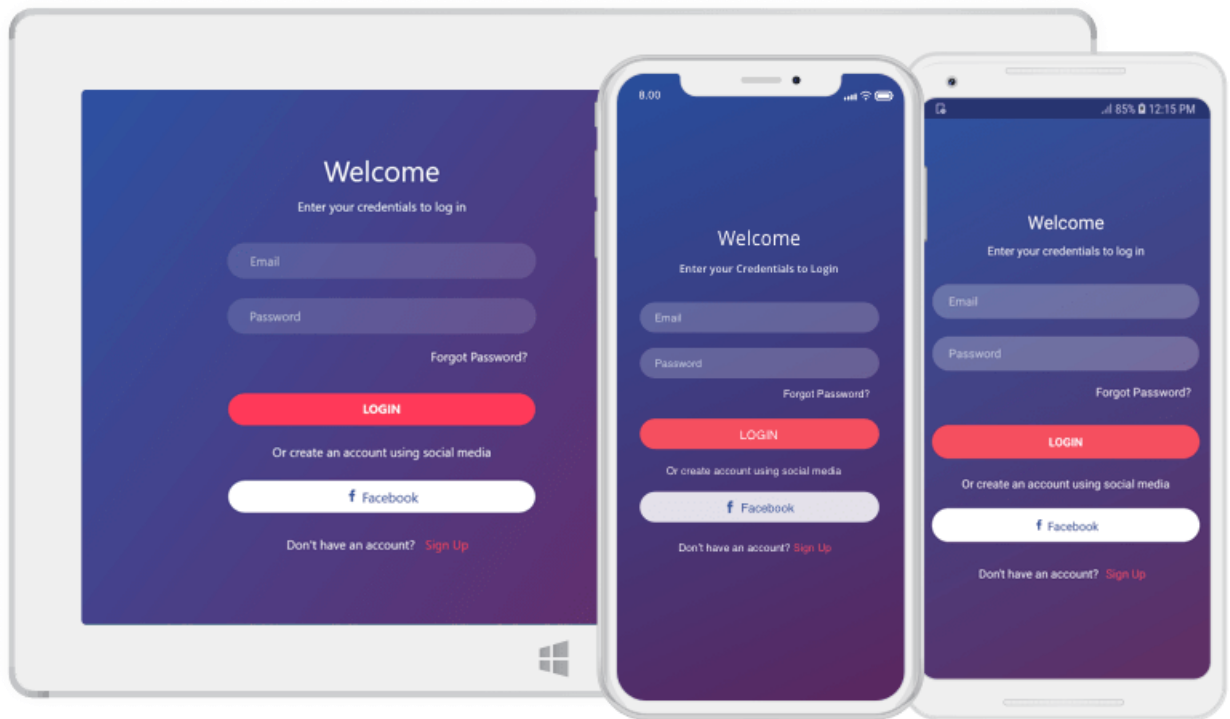
### Overview

The Xamarin.Forms SfGradientView control provides the gradient background to various views of applications. The SfGradientView control supports the following two types of gradient:

- Linear gradient
- Radial gradient

### Key Features

- Customize the background with a linear gradient. To create a linear gradient effect you must define at least two color stops.
- Customize the background with a radial gradient. To create a radial gradient effect you must define at least two color stops.



## Getting Started

This section explains the steps required to work with the gradient view control for Xamarin.Forms.

### Adding SfGradientView reference

You can add SfGradientView reference using one of the following methods:

#### Method 1: Adding SfGradientView reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Core). To add SfGradientView to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Core](https://www.nuget.org/packages/Syncfusion.Xamarin.Core), and then install it.

![Xamarin Forms SfGradientView Nuget reference](images/Adding SfGradientView reference.png)

---

**Note:** Install the same version of Syncfusion.Core.XForms NuGet in all the projects.

---

#### Method 2: Adding SfGradientView reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfGradientView control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfGradientView assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
-----	--

Android	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v17.1.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

Launching an application on each platform with SfGradientView.

To use the SfGradientView control inside an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch the SfGradientView in iOS, call the `SfGradientViewRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms Framework` has been initialized and before the `LoadApplication` is called, as demonstrated in the following code example.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.XForms.iOS.Graphics.SfGradientViewRenderer.Init ();
    LoadApplication (new App ());
    ...
}
```

#### Universal Windows Platform (UWP)

You need to initialize the gradient view assemblies in `App.xaml.cs` in UWP project as demonstrated in the following code samples. This is required to deploy the application with gradient view in Release mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
```

```
{
...
rootFrame.NavigationFailed += OnNavigationFailed;
// Add `using System.Reflection;`
List<Assembly> assembliesToInclude = new List<Assembly>();
// Now, add all the assemblies your app uses
assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.Graphics.SfGradientView
Renderer).GetTypeInfo().Assembly);
// Replaces Xamarin.Forms.Forms.Init(e);
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
...
}
```

### Android

The Android platform does not require any additional configuration to render the gradient view.

### Adding Gradient View

The **SfGradientView** control is configured entirely in C# code or in XAML markup. The **SfGradientView** supports linear gradient and radial gradient. Each gradient is a collection of gradient stops which used to set colors with different offsets. The following steps explain how to create a **SfGradientView** and configure its elements.

### Adding namespace for referred assemblies

#### XML

```
xmlns:gradient="clr-
namespace:Syncfusion.XForms.Graphics;assembly=Syncfusion.Core.XForms"
```

#### C#

```
using Syncfusion.XForms.Graphics;
```

### Adding Linear Gradient Brush

**SfLinearGradientBrush** is used to create linear gradient effects.

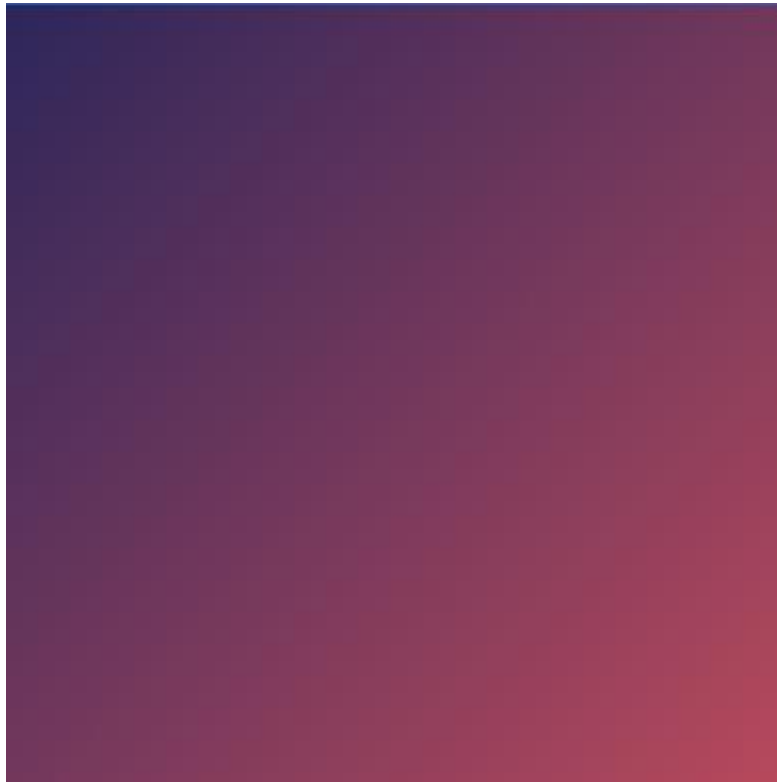
#### XML

```
<Grid>
<gradient:SfGradientView>
<gradient:SfGradientView.BackgroundBrush>
<gradient:SfLinearGradientBrush>
<gradient:SfLinearGradientBrush.GradientStops>
<gradient:SfGradientStop Color="#2d265b" Offset="0.0" />
<gradient:SfGradientStop Color="#b8495c" Offset="1.0" />
</gradient:SfLinearGradientBrush.GradientStops>
</gradient:SfLinearGradientBrush>
</gradient:SfGradientView.BackgroundBrush>
</gradient:SfGradientView>
</Grid>
```

#### C#

```
using Xamarin.Forms;
using Syncfusion.XForms.Graphics;
```

```
using System;
namespace GradientViewGettingStarted
{
    public partial class MainPage : ContentPage
    {
        Public MainPage()
        {
            InitializeComponent();
            Grid grid = new Grid();
            SfGradientView gradientView = new SfGradientView();
            SfLinearGradientBrush linearGradientBrush = new SfLinearGradientBrush();
            linearGradientBrush.GradientStops = new GradientStopCollection()
            {
                new SfGradientStop() { Color = Color.FromHex("#2d265b"), Offset=0.0},
                new SfGradientStop() { Color = Color.FromHex("#b8495c"), Offset=1.0},
            };
            gradientView.BackgroundBrush = linearGradientBrush;
            grid.Children.Add(gradientView);
            this.Content = grid;
        }
    }
}
```



#### Adding Radial Gradient Brush

**SfRadialGradientBrush** is used to create radial gradient effects.

#### XML

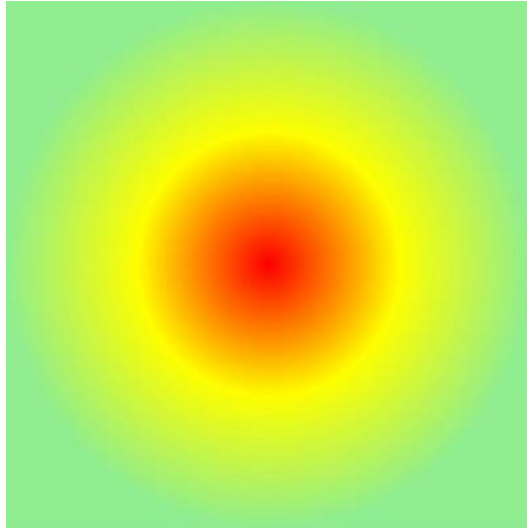
```
<Grid>
    <gradient:SfGradientView>
```

```
<gradient:SfGradientView.BackgroundBrush>
<gradient:SfRadialGradientBrush>
<gradient:SfRadialGradientBrush.GradientStops>
<gradient:SfGradientStop Color="Red" Offset="0.0" />
<gradient:SfGradientStop Color="Yellow" Offset="0.5" />
<gradient:SfGradientStop Color="LightGreen" Offset="1.0" />
</gradient:SfRadialGradientBrush.GradientStops>
</gradient:SfRadialGradientBrush>
</gradient:SfGradientView.BackgroundBrush>
</gradient:SfGradientView>
</Grid>
```

## C#

```
using Xamarin.Forms;
using Syncfusion.XForms.Graphics;
using System;
namespace GradientViewGettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            Grid grid = new Grid();
            SfGradientView gradientView = new SfGradientView();
            SfRadialGradientBrush radialGradientBrush = new SfRadialGradientBrush();
            radialGradientBrush.GradientStops = new GradientStopCollection()
            {
                new SfGradientStop() { Color = Color.Red, Offset=0.0},
                new SfGradientStop() { Color = Color.Yellow, Offset=0.5},
                new SfGradientStop() { Color = Color.LightGreen, Offset=1.0},
            };
            gradientView.BackgroundBrush = radialGradientBrush;
            grid.Children.Add(gradientView);
            this.Content = grid;
        }
    }
}
```





You can find the complete getting started sample from [this](#) link.

### Customization

The SfGradientView control supports customizing the following properties:

- The **GradientStops** for both linear and radial gradients.
- The **StartPoint** and **EndPoint** for linear gradient brush.
- The **Center** and **Radius** for radial gradient brush.

### GradientStops

**GradientStops** is a collection of **SfGradientStop** for both linear gradient and radial gradient; it is used for smooth color transition by specifying the **Color** and **Offset** properties of **SfGradientStop**. An **Offset** ranges from 0 to 1.

### SfLinearGradientBrush

You can customize the linear gradient using the **StartPoint** and **EndPoint** properties of **SfLinearGradientBrush**. By default, the value of **StartPoint** is (0,0) and the value of **EndPoint** is (1,1).

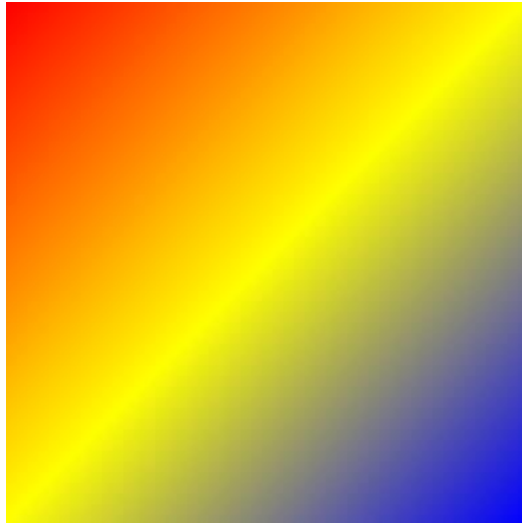
### XML

```
<gradient:SfGradientView>
  <gradient:SfGradientView.BackgroundBrush>
    <gradient:SfLinearGradientBrush>
      <gradient:SfLinearGradientBrush.GradientStops>
        <gradient:SfGradientStop Color="Red" Offset="0.0" />
        <gradient:SfGradientStop Color="Yellow" Offset="0.5" />
        <gradient:SfGradientStop Color="Blue" Offset="1.0" />
      </gradient:SfLinearGradientBrush.GradientStops>
    </gradient:SfLinearGradientBrush>
  </gradient:SfGradientView.BackgroundBrush>
</gradient:SfGradientView>
```

### C#

```
SfGradientView gradientView = new SfGradientView();
SfLinearGradientBrush linearGradient = new SfLinearGradientBrush();
```

```
linearGradient.GradientStops = new GradientStopCollection()
{
    new SfGradientStop() { Color = Color.Red, Offset = 0.0 },
    new SfGradientStop() { Color = Color.Yellow, Offset = 0.5 },
    new SfGradientStop() { Color = Color.Blue, Offset = 1.0 }
};
gradientView.BackgroundBrush = linearGradient
```



*Horizontal linear gradient*

You can use the values of `StartPoint` and `EndPoint` properties of `SfLinearGradientBrush` as (0, 0.5) and (1, 0.5), respectively to align the gradient of background horizontally.

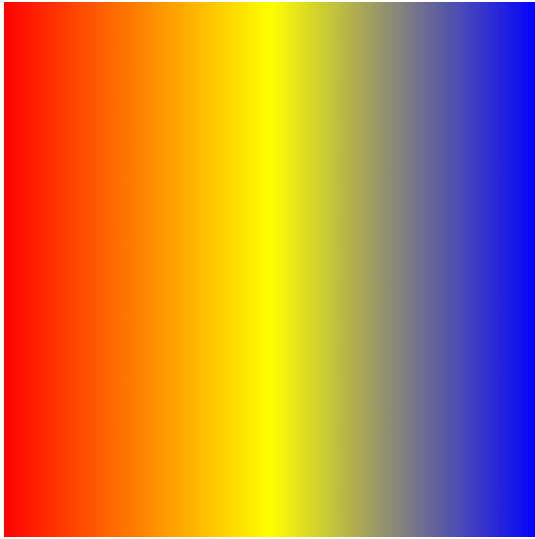
#### XML

```
<gradient:SfGradientView>
<gradient:SfGradientView.BackgroundBrush>
<gradient:SfLinearGradientBrush StartPoint="0, 0.5" EndPoint="1, 0.5">
<gradient:SfLinearGradientBrush.GradientStops>
<gradient:SfGradientStop Color="Red" Offset="0.0" />
<gradient:SfGradientStop Color="Yellow" Offset="0.5" />
<gradient:SfGradientStop Color="Blue" Offset="1.0" />
</gradient:SfLinearGradientBrush.GradientStops>
</gradient:SfLinearGradientBrush>
</gradient:SfGradientView.BackgroundBrush>
</gradient:SfGradientView>
```

#### C#

```
SfGradientView gradientView = new SfGradientView();
SfLinearGradientBrush linearGradient = new SfLinearGradientBrush();
linearGradient.StartPoint = new Point(0, 0.5);
linearGradient.EndPoint = new Point(1, 0.5);
linearGradient.GradientStops = new GradientStopCollection()
{
    new SfGradientStop() { Color = Color.Red, Offset = 0.0 },
    new SfGradientStop() { Color = Color.Yellow, Offset = 0.5 },
    new SfGradientStop() { Color = Color.Blue, Offset = 1.0 }
};
```

```
gradientView.BackgroundBrush = linearGradient;
```



#### *Vertical linear gradient*

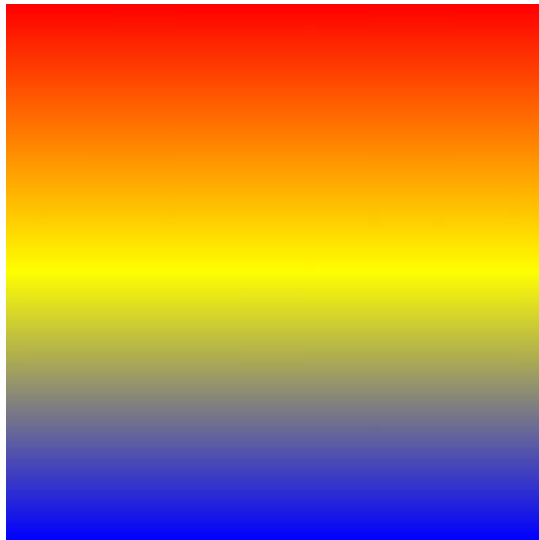
You can use the values of `StartPoint` and `EndPoint` properties of `SfLinearGradientBrush` as (0.5, 0) and (0.5, 1), respectively to align the gradient of background vertically.

#### **XML**

```
<gradient:SfGradientView>
<gradient:SfGradientView.BackgroundBrush>
<gradient:SfLinearGradientBrush StartPoint="0.5, 0" EndPoint="0.5, 1">
<gradient:SfLinearGradientBrush.GradientStops>
<gradient:SfGradientStop Color="Red" Offset="0.0" />
<gradient:SfGradientStop Color="Yellow" Offset="0.5" />
<gradient:SfGradientStop Color="Blue" Offset="1.0" />
</gradient:SfLinearGradientBrush.GradientStops>
</gradient:SfLinearGradientBrush>
</gradient:SfGradientView.BackgroundBrush>
</gradient:SfGradientView>
```

#### **C#**

```
SfLinearGradientBrush linearGradient = new SfLinearGradientBrush();
linearGradient.StartPoint = new Point(0.5, 0);
linearGradient.EndPoint = new Point(0.5, 1);
linearGradient.GradientStops = new GradientStopCollection()
{
    new SfGradientStop() { Color = Color.Red, Offset = 0.0 },
    new SfGradientStop() { Color = Color.Yellow, Offset = 0.5 },
    new SfGradientStop() { Color = Color.Blue, Offset = 1.0 }
};
gradientView.BackgroundBrush = linearGradient;
```



### SfRadialGradientBrush

You can customize the following properties of **SfRadialGradientBrush**:

- Center
- Radius

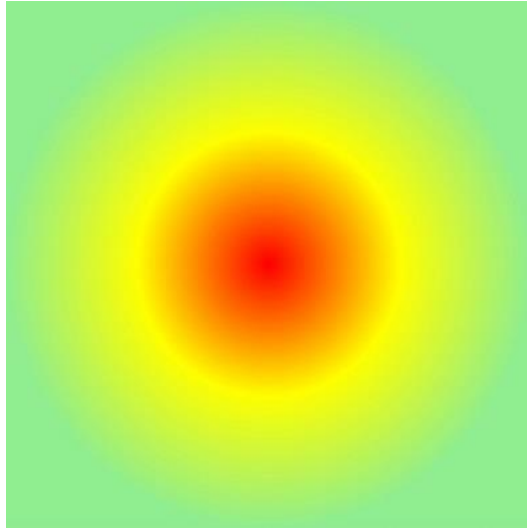
By default, the value of center is (0.5, 0.5) and value of radius is 0.5.

### XML

```
<gradient:SfGradientView>
<gradient:SfGradientView.BackgroundBrush>
<gradient:SfRadialGradientBrush>
<gradient:SfRadialGradientBrush.GradientStops>
<gradient:SfGradientStop Color="Red" Offset="0.0" />
<gradient:SfGradientStop Color="Yellow" Offset="0.5" />
<gradient:SfGradientStop Color="LightGreen" Offset="1.0" />
</gradient:SfRadialGradientBrush.GradientStops>
</gradient:SfRadialGradientBrush>
</gradient:SfGradientView.BackgroundBrush>
</gradient:SfGradientView>
```

### C#

```
SfGradientView gradientView = new SfGradientView();
SfRadialGradientBrush radialGradient = new SfRadialGradientBrush();
radialGradient.GradientStops = new GradientStopCollection()
{
    new SfGradientStop() { Color = Color.Red, Offset = 0.0 },
    new SfGradientStop() { Color = Color.Yellow, Offset = 0.5 },
    new SfGradientStop() { Color = Color.LightGreen, Offset = 1.0 }
};
gradientView.BackgroundBrush = radialGradient;
```



#### *Customize the center*

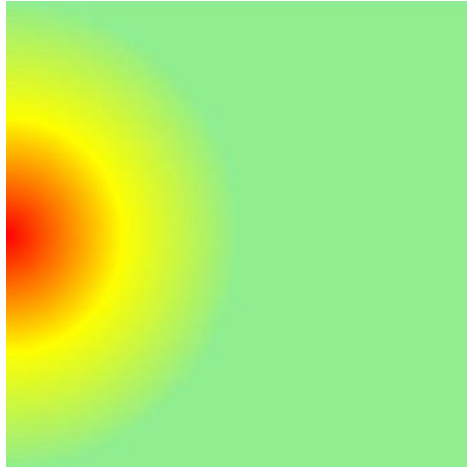
You can customize the center point of radial gradient brush using the `Center` property of `SfRadialGradientBrush`.

#### XML

```
<gradient:SfGradientView>
<gradient:SfGradientView.BackgroundBrush>
<gradient:SfRadialGradientBrush Center="0, 0.5">
<gradient:SfRadialGradientBrush.GradientStops>
<gradient:SfGradientStop Color="Red" Offset="0.0" />
<gradient:SfGradientStop Color="Yellow" Offset="0.5" />
<gradient:SfGradientStop Color="LightGreen" Offset="1.0" />
</gradient:SfRadialGradientBrush.GradientStops>
</gradient:SfRadialGradientBrush>
</gradient:SfGradientView.BackgroundBrush>
</gradient:SfGradientView>
```

#### C#

```
SfGradientView gradientView = new SfGradientView();
SfRadialGradientBrush radialGradient = new SfRadialGradientBrush();
radialGradient.Center = new Point(0, 0.5);
radialGradient.GradientStops = new GradientStopCollection()
{
    new SfGradientStop() { Color = Color.Red, Offset = 0.0 },
    new SfGradientStop() { Color = Color.Yellow, Offset = 0.5 },
    new SfGradientStop() { Color = Color.LightGreen, Offset = 1.0 }
};
gradientView.BackgroundBrush = radialGradient;
```



### *Customize the radius*

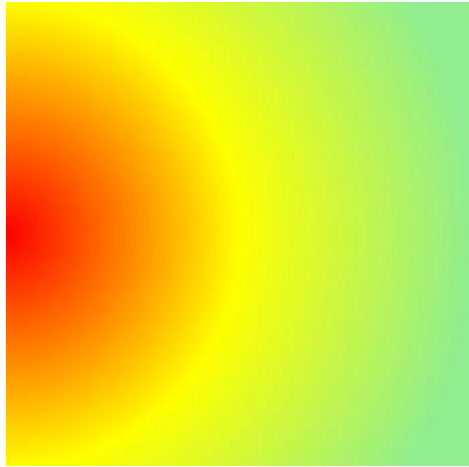
You can customize the radius of the radial gradient brush using `Radius` property of `SfRadialGradientBrush`. The `Radius` property ranges from 0 to 1.

### **XML**

```
<gradient:SfGradientView>
<gradient:SfGradientView.BackgroundBrush>
<gradient:SfRadialGradientBrush Center="0, 0.5" Radius="1">
<gradient:SfRadialGradientBrush.GradientStops>
<gradient:SfGradientStop Color="Red" Offset="0.0" />
<gradient:SfGradientStop Color="Yellow" Offset="0.5" />
<gradient:SfGradientStop Color="LightGreen" Offset="1.0" />
</gradient:SfRadialGradientBrush.GradientStops>
</gradient:SfRadialGradientBrush>
</gradient:SfGradientView.BackgroundBrush>
</gradient:SfGradientView>
```

### **C#**

```
SfGradientView gradientView = new SfGradientView();
SfRadialGradientBrush radialGradient = new SfRadialGradientBrush();
radialGradient.Center = new Point(0, 0.5);
radialGradient.Radius = 1;
radialGradient.GradientStops = new GradientStopCollection()
{
    new SfGradientStop() { Color = Color.Red, Offset = 0.0 },
    new SfGradientStop() { Color = Color.Yellow, Offset = 0.5 },
    new SfGradientStop() { Color = Color.LightGreen, Offset = 1.0 }
};
gradientView.BackgroundBrush = radialGradient;
```



## SfImageEditor

### Overview

The image editor control for Xamarin.Forms is a very handy tool, which is used to edit an image by annotating with free hand drawing paths, text, and built-in shapes; it also allows to crop and flip a image. This control has a built-in toolbar that helps to perform editing operations.

### Key features

- Crop, rotate, and flip operations.
- Adds text, path, and shapes such as rectangle, circle, and arrow.
- Undo and redo operations.
- Zoom and pan operations.
- Localization.
- Custom view annotations.
- Customizing toolbar.
- Z-order.
- Saving images to device.
- Built-in toolbar.

### Getting Started

This section explains the steps required to load an image to the image editor control.

To get start quickly with Xamarin Image Editor control, you can check on this video:

```
<style>#xamarinImageEditorVideoTutorial{width : 90% !important; height: 300px !important }</style>
```

```
<iframe id='xamarinImageEditorVideoTutorial'  
src='https://www.youtube.com/embed/SW1fsk7YbeA'></iframe>
```

### Adding SfImageEditor reference

You can add SfImageEditor reference using one of the following methods:

#### Method 1: Adding SfImageEditor reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.syncfusion.com/nuget-packages). To add SfImageEditor to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfImageEditor](https://www.syncfusion.com/nuget-packages), and then install it.

![[Adding SfImageEditor reference from NuGet]](ImageEditor\_images/Adding SfImageEditor reference.png)

---

**Note:** Install the same version of SfImageEditor NuGet in all the projects.

---

### Method 2: Adding SfImageEditor reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfImageEditor control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfImageEditor assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfImageEditor.XForms.dll Syncfusion.Core.XForms.dll
Android	Syncfusion.SfImageEditor.XForms.Android.dll Syncfusion.SfImageEditor.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll
iOS	Syncfusion.SfImageEditor.XForms.iOS.dll Syncfusion.SfImageEditor.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll
UWP	Syncfusion.SfImageEditor.UWP.dll Syncfusion.SfImageEditor.XForms.UWP.dll Syncfusion.SfImageEditor.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

### Launching the application in iOS

To launch the image editor in iOS, call the `SfImageEditorRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as shown in the following code sample.

### C#



```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
...
global::Xamarin.Forms.Forms.Init();
Syncfusion.SfImageEditor.XForms.iOS.SfImageEditorRenderer.Init();
LoadApplication(new App());
...
}
```

### Universal Windows Platform (UWP)

You need to initialize the image editor view assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with image editor in Release mode in UWP platform.

### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
...
rootFrame.NavigationFailed += OnNavigationFailed;
// Add `using System.Reflection;`
List<Assembly> assembliesToInclude = new List<Assembly>();
// Now, add all the assemblies your app uses
assembliesToInclude.Add(typeof(Syncfusion.SfImageEditor.XForms.UWP.SfImageEd
itorRenderer).GetTypeInfo().Assembly);
// Replaces Xamarin.Forms.Forms.Init(e);
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
...
}
```

### Android

The Android platform does not require any additional configuration to render the image editor.

### Initializing image editor

1. Import SfImageEditor control namespace as `xmlns:syncfusion="clr-namespace:Syncfusion.SfImageEditor.XForms;assembly=Syncfusion.SfImageEditor.XForms" in XAML page.`
2. Set the SfImageEditor control as content to the ContentPage.

### XML

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="ImageEditor_
GettingStarted.ImageEditor_GettingStartedPage"
xmlns:imageeditor="clr-
namespace:Syncfusion.SfImageEditor.XForms;assembly=Syncfusion.SfImageEditor.
XForms">
<ContentPage.Content>
<imageeditor:SfImageEditor />
</ContentPage.Content>
```

---

</ContentPage>

---

**C#**

```
using Syncfusion.SfImageEditor.XForms;
using Xamarin.Forms;
public class App : Application
{
    public App()
    {
        MainPage = new ImageEditor_GettingStartedPage();
    }
}
Public class ImageEditor_GettingStartedPage : ContentPage
{
    public ImageEditor_GettingStartedPage()
    {
        InitializeComponent();
        SfImageEditor editor = new SfImageEditor();
        this.Content = editor;
    }
}
```

- If image is not set to the **Source** property, the appearance of the image will be shown as white canvas. You can perform editing action using built-in toolbar.

## Loading an image to image editor

Refer to the following steps to add an image to the pcl project:

1. Right-click your pcl project.
2. Select **Add Files** submenu from **Add** menu, and a dialog box will appear.
3. Choose and import the desired image to the pcl project.
4. After the image has been imported, ensure whether the image Build Action has been set to **EmbeddedResource**.

---

**Note:** Image formats such as JPEG and PNG can be loaded to the image editor.

---

The following code shows adding an image to the image editor control with the format as "JPEG" and name as "image".

**XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="ImageEditor_
GettingStarted.ImageEditor_GettingStartedPage"
xmlns:imageeditor="clr-
namespace:Syncfusion.SfImageEditor.XForms;assembly=Syncfusion.SfImageEditor.
XForms">
<ContentPage.Content>
<imageeditor:SfImageEditor Source="{Binding Image}" />
</ContentPage.Content>
```

---

</ContentPage>

---

## C#

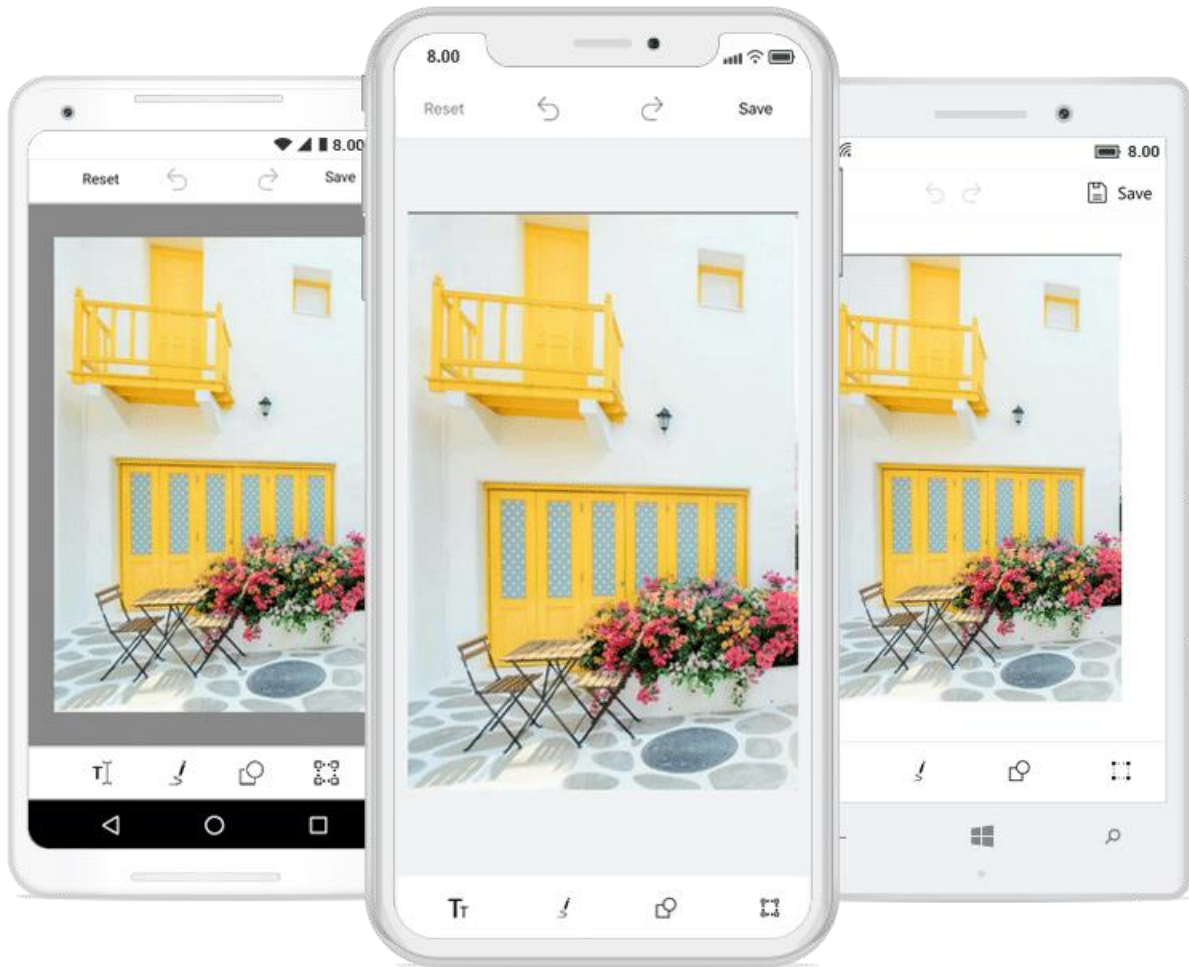
```
using Syncfusion.SfImageEditor.XForms;
using Xamarin.Forms;
public class App : Application
{
    public App()
    {
        MainPage = new ImageEditor_GettingStartedPage();
    }
}
Public class ImageEditor_GettingStartedPage : ContentPage
{
    public ImageEditor_GettingStartedPage()
    {
        InitializeComponent();
        BindingContext = new ImageModel();
        SfImageEditor editor = new SfImageEditor();
        this.Content = editor;
    }
}
class ImageModel
{
    public ImageSource Image { get; set; }
    public ImageModel()
    {
        Image = ImageSource.FromResource("ImageEditor_GettingStarted.Image.jpg");
    }
}
```

---

**Note:** Refer to this [link](#) to know more about loading an image to the image editor source property in different formats.

---

- The following screenshot depicts loading an image to the image editor. You can edit the image using built-in toolbar.



You can find the complete getting started sample from this [link](#).

### Text

You can annotate the desired text elements to an image using the `AddText` method with customization options.

### C#

```
editor.AddText("New Text");
```

### Customize text with TextSettings

You can customize the appearance of the text using the `TextSettings` property.

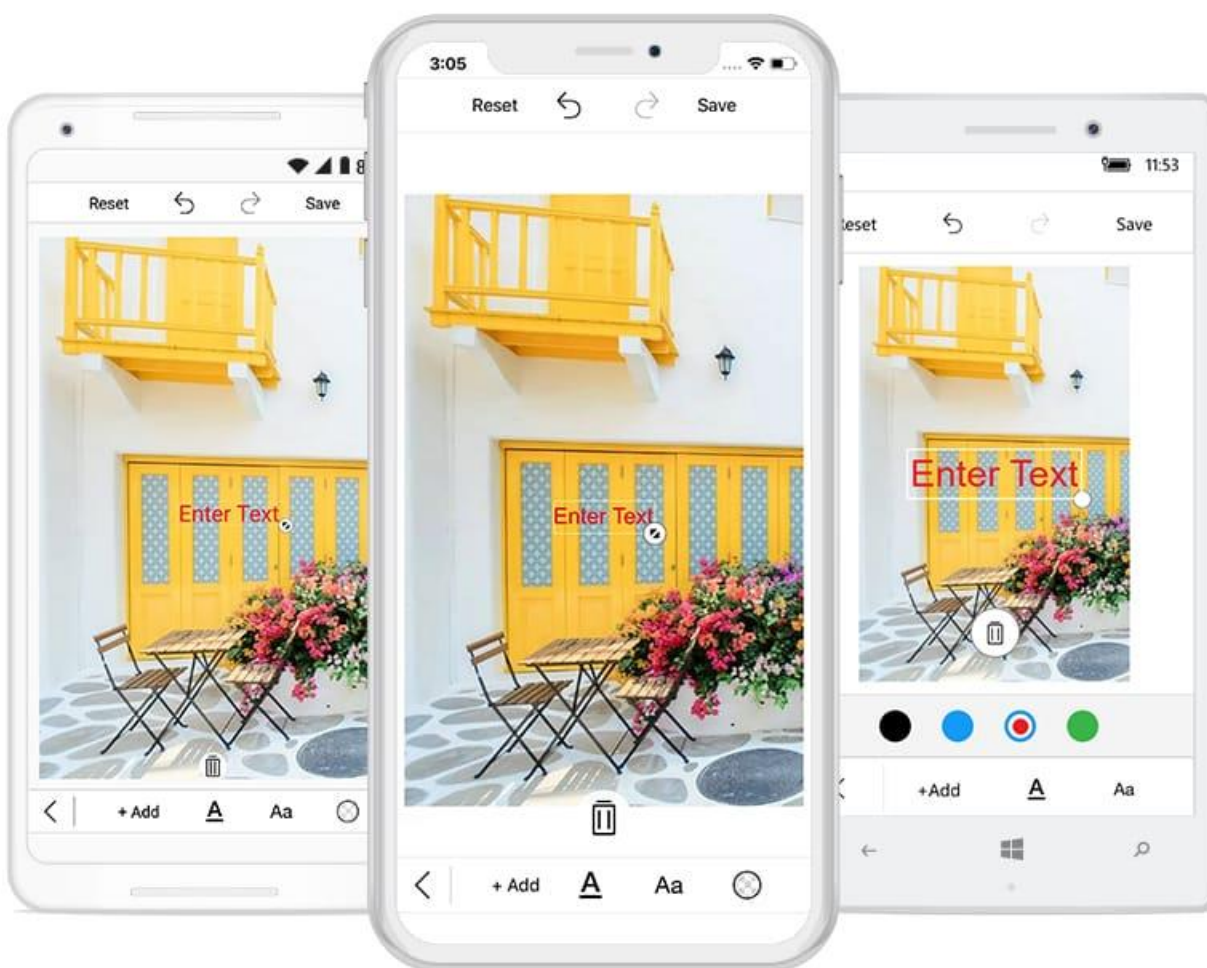
The `TextSettings` property consists of the following properties:

- [Color](#): Defines the color of the desired text.
- [FontSize](#): Specifies the desired font size of the text under text settings.
- [FontFamily](#): Specifies the desired font family for text. Six types of font families are available in toolbar: `Arial`, `Noteworthy`, `Marker Felt`, `SignPainter`, `Bradley Hand`, `Snell Round hand`.

- [Bounds](#): Allows to set frame for the newly added **Text**. You can position the text wherever you want on the image. In percentage, the value of the text frame should fall between 0 and 100.
- [Opacity](#): Changes the opacity of text.
- [Angle](#): Changes the angle of text.
- [TextEffects](#): Changes the effects of the text such as **Bold**, **Italic** and **Underline**.

## C#

```
editor.AddText("New Text", new TextSettings() { Color = Color.Black, FontSize = 16d, FontFamily="Arial", Bounds = new Rectangle(20, 20, 35, 35), Opacity=0.5f, Angle=45, TextEffects = TextEffects.Bold | TextEffects.Italic | TextEffects.Underline});
```



## Custom font family

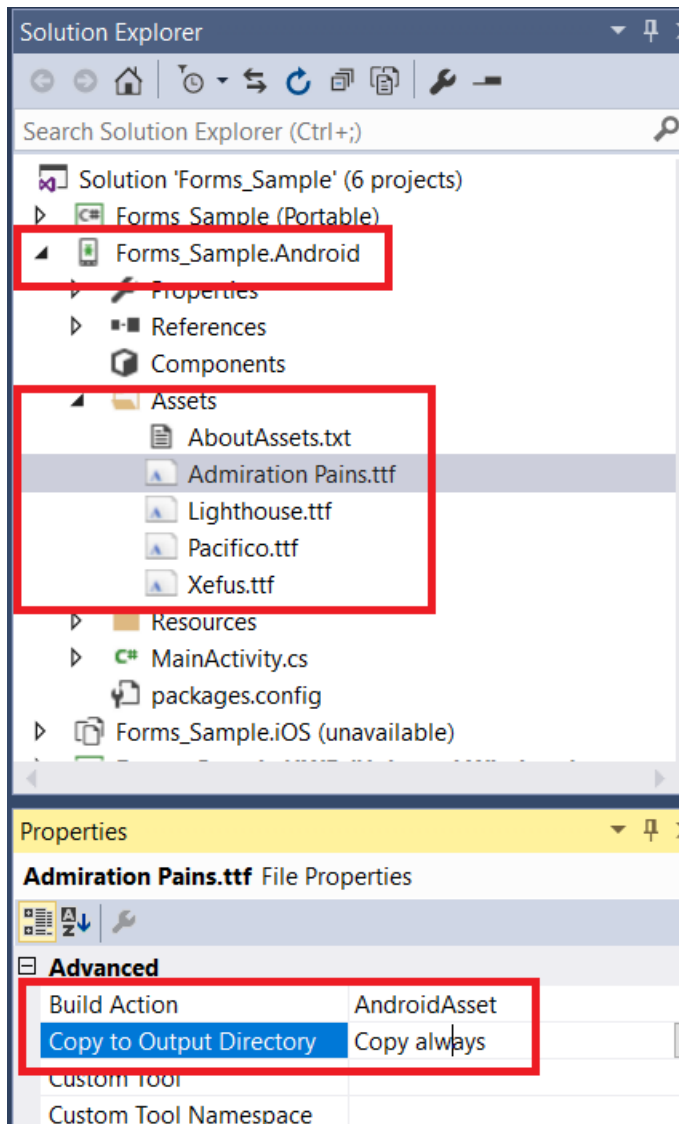
Using a font that is not in the built-in typeface requires some platform-specific coding. The steps required for each platform are shown as follows.

Download the custom fonts in ttf file format, and add these fonts to required folder in particular project file.

### Android

Add the custom fonts to Assets folder in sample.Droid project.

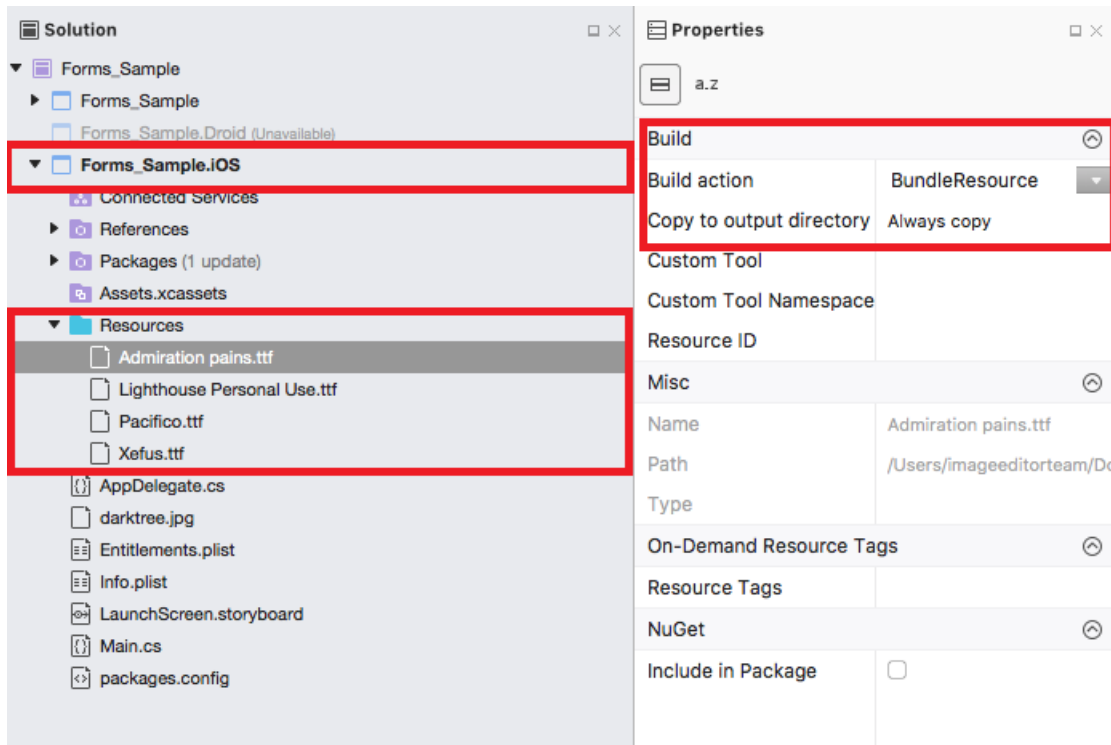
Right-click the font file, and open properties. In properties, change the "Build Action" property of every font file to **AndroidAsset** and "Copy to output directory" to **Copy Always**.



### iOS

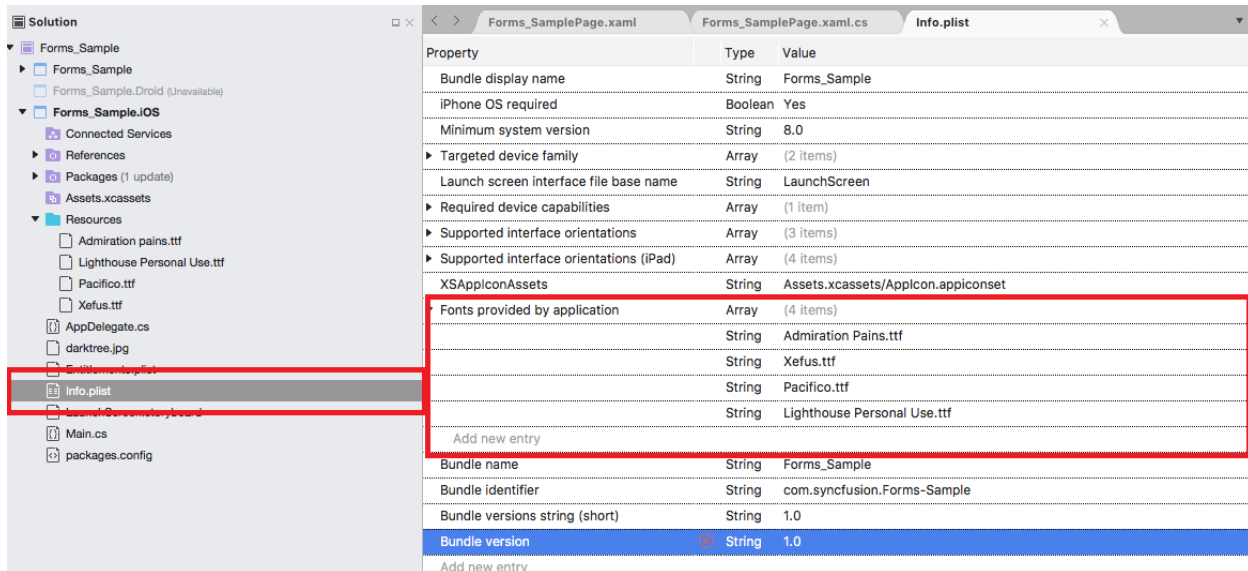
Add custom fonts to resource file in sample.iOS project.

Change the "Build Action" property of every font file to **BundleResource** and "Copy to output directory" to **Copy Always**.



Open the `info.plist` file, and select "Source" at the bottom of the file.

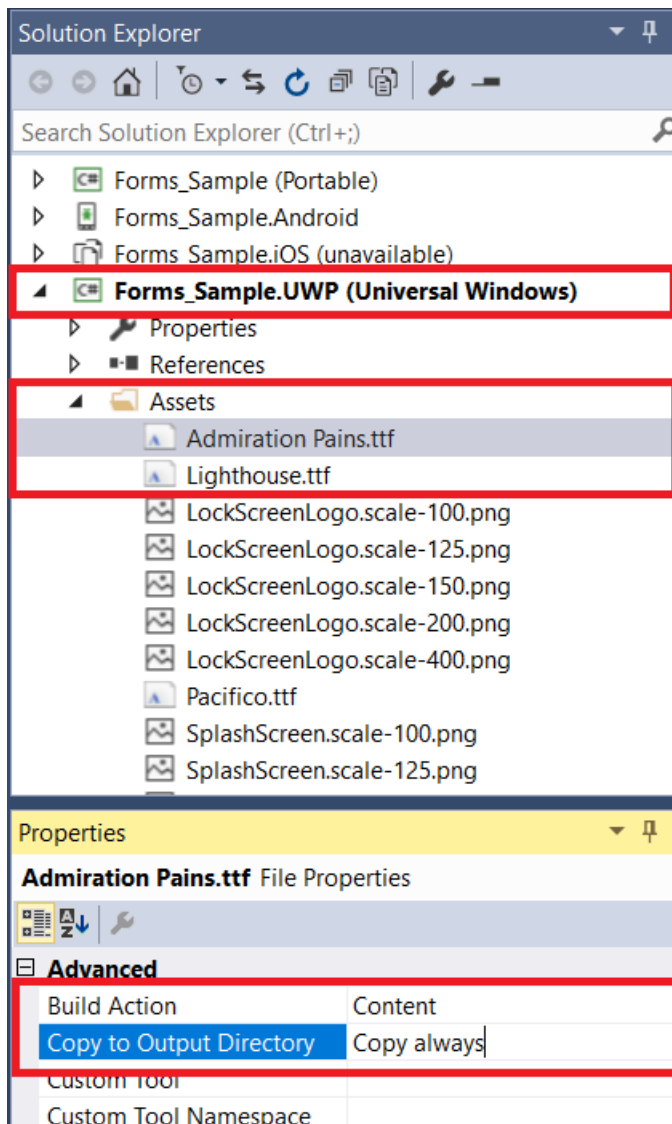
After opened the source file, add "Fonts provided by application" to source file, and add the downloaded custom fonts name with .ttf extension.



UWP

Add custom fonts to Assets folder in sample.UWP project.

Right-click the font file, and open properties. In properties, change the "Build Action" property of every font file to `Content` and "Copy to output directory" to `Copy Always`.

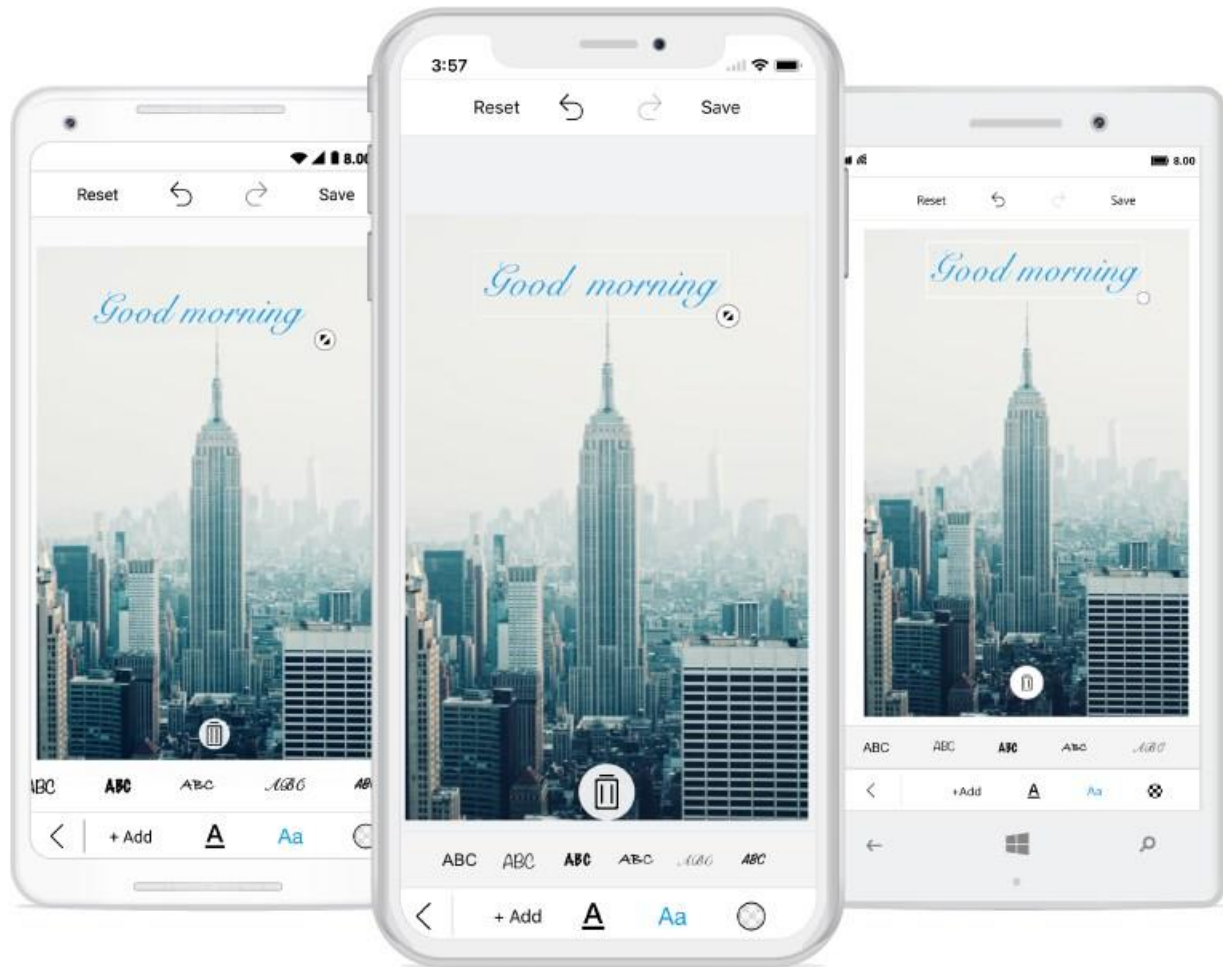


The following code snippet shows applying custom font family. In forms, Android, and iOS, give the font family name, but in UWP, you should mention font file name with .ttf extension and "#" symbol with font family name.

### C#

```
if ((Device.OS == TargetPlatform.Android) || (Device.OS == TargetPlatform.iOS))
    editor.AddText("New Text", new TextSettings() { FontFamily="Pacifico" });
else
    editor.AddText("New Text", new
TextSettings() { FontFamily="Assets/Pacifico.ttf#Pacifico" });
```





## Multiline text and text alignment

### Multiline text

You can annotate multiple line text over an image with the help of text preview window.

### Text alignment

**TextAlignment** is an enum type and text can be aligned with the help of text alignment enum values such as left, right and center.

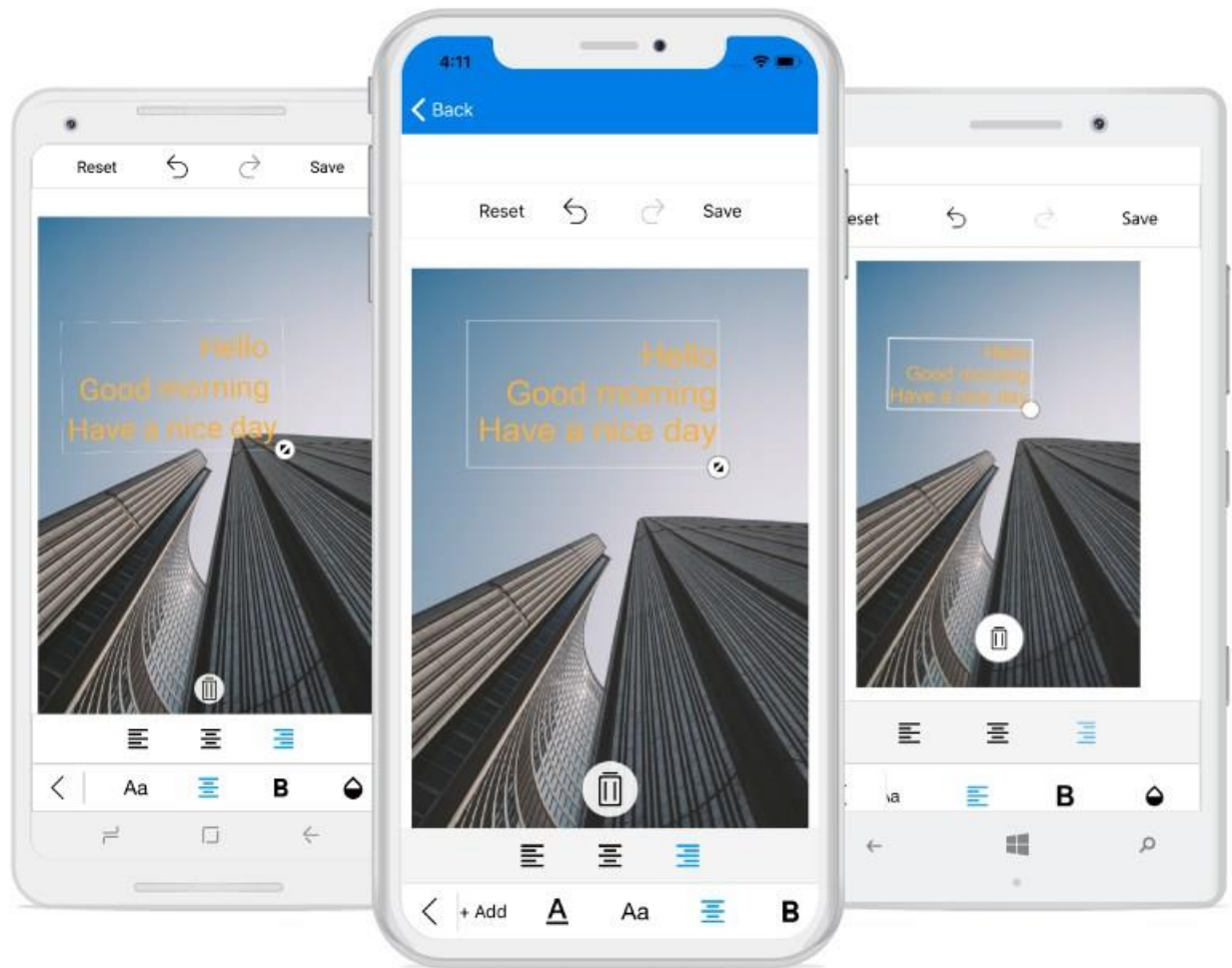
---

**Note:** The default text alignment is **Left** and text alignment is not applicable for single line text.

---

### C#

```
editor.AddText("Hello\nGood morning\nHave a nice day", new TextSettings()  
{TextAlignment = TextAlignment.Right });
```



### Text Rotation

You can rotate and resize the text by enabling the `RotatableElements` property of image editor. `ImageEditorElements` is an enum type with values `Text`, `CustomView` and `None` as shown in the following code snippet.

#### C#

```
editor.RotatableElements = ImageEditorElements.Text;
```

---

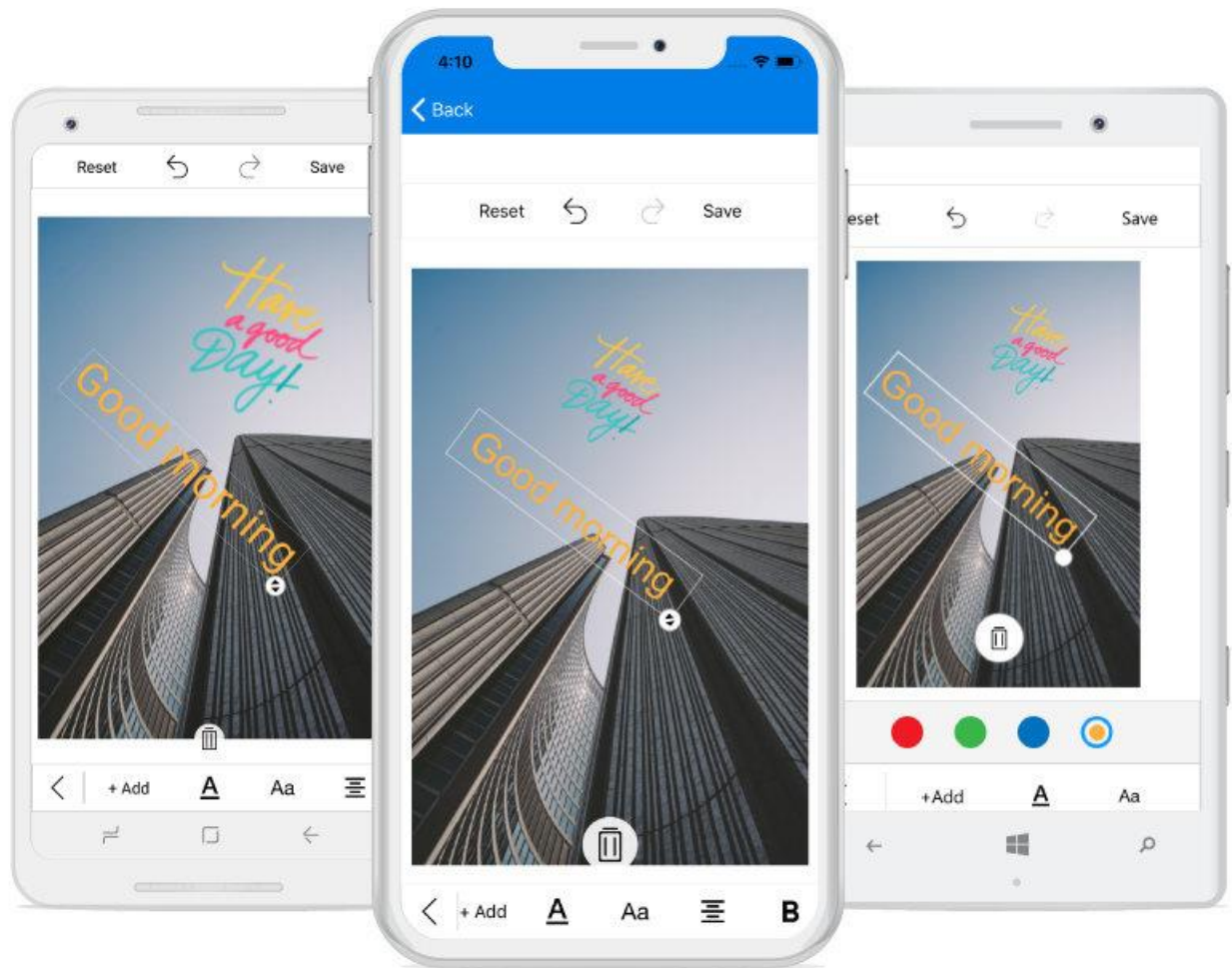
**Note:** The default value for `RotatableElements` is `None`.

---

You can rotate the text based on a particular angle using `Angle` property in `TextSettings` as shown in the following code snippet.

#### C#

```
editor.AddText("Good morning", new TextSettings() {Angle = 45});
```



### Restricting the edit text box pop-up window

You can restrict the edit text box pop-up window using the `IsEditable` property. By Default, the value of the `IsEditable` property is true, so you can edit the text in edit text box pop-up window. When setting the `IsEditable` property to false, the edit text box pop-up window will not be displayed, and you are restricted to edit the text in the edit text box.

#### C#

```
editor.AddText("text", new TextSettings { IsEditable=false });
```

### Shapes

You can annotate any shapes over an image using the `AddShape` method. The following shapes are available in image editor:

- Circle
- Rectangle
- Arrow
- Path

### Selecting a shape type

The `ShapeType` is an enum property with values `Rectangle`, `Circle`, `Arrow`, and `Path`. You can give the desired shape type as an argument to the `AddShape` method.

#### C#

```
editor.AddShape(ShapeType.Circle);
```

### Customizing a shape with pen settings

You can customize the appearance of each shape using the `PenSettings` property:

#### PenSettings

The `PenSettings` property consists of the following properties:

- [Color](#): Specifies the desired stroke color to a shape.
- [FillColor](#): Specifies the desired fill color to a shape.
- [StrokeWidth](#): Allows to denote the stroke width for the desired shape.
- [Mode](#): Determines whether the shape color mode is `Fill` or `Stroke`. It is an enum value.
- [Opacity](#): Denotes opacity for the desired shapes.
- [Bounds](#): Allows to set frame for the newly added shapes (rectangle and circle). You can position the shapes wherever you want on the image. In percentage, the value of the shape frame should fall between 0 and 100.

---

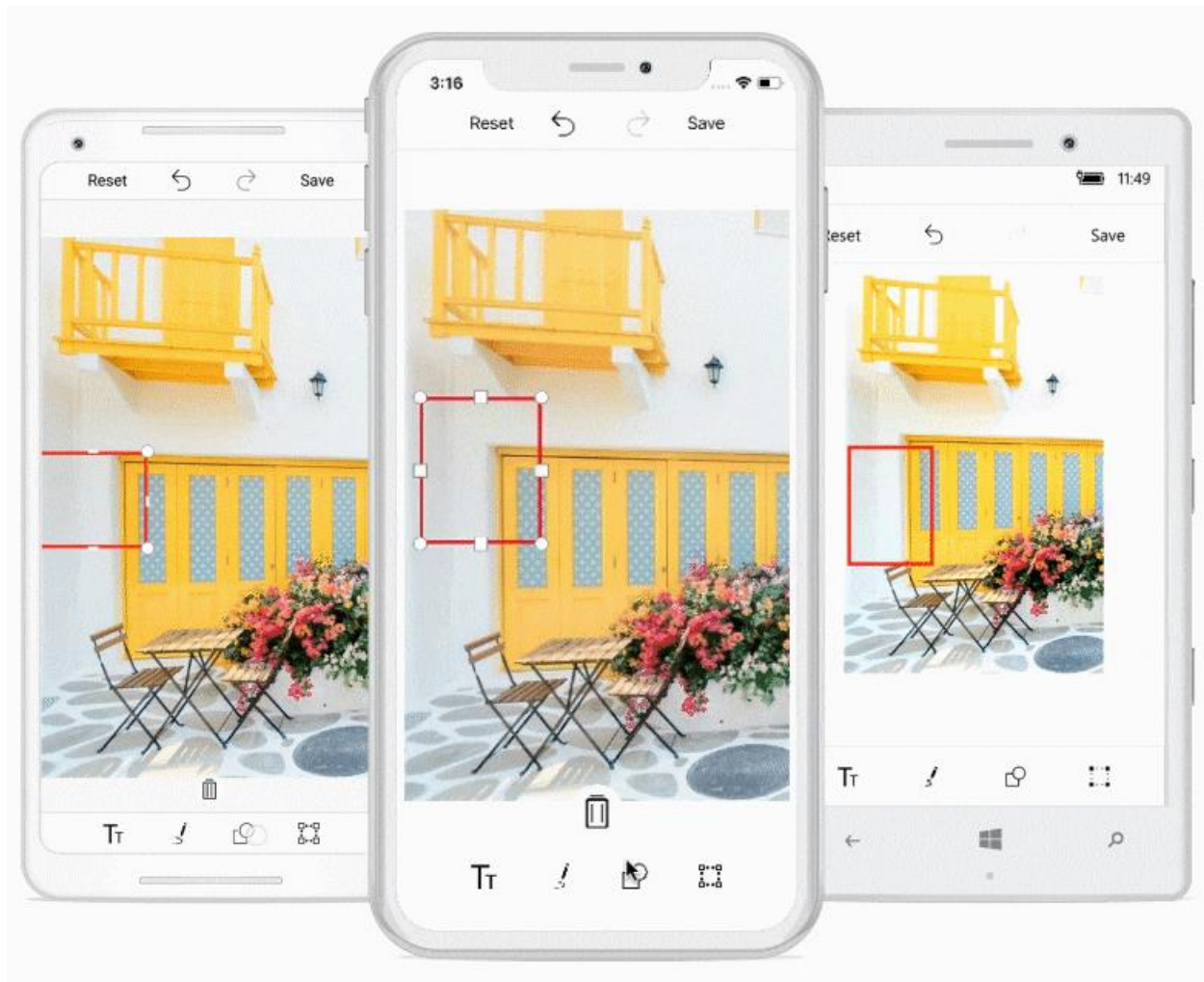
**Note:** The `FillColor` property is applicable only if the `ShapeType` is `Rectangle` or `Circle`.

---

- To add a rectangle, circle, or arrow over an image, specify the `ShapeType` and the desired `PenSettings` as shown in the following code snippet.

#### C#

```
editor.AddShape(ShapeType.Circle, new PenSettings() {Color = Color.Red,  
Mode= Mode.Stroke, Opacity=1f, Bounds = new Rectangle(20,20,35,35)});
```

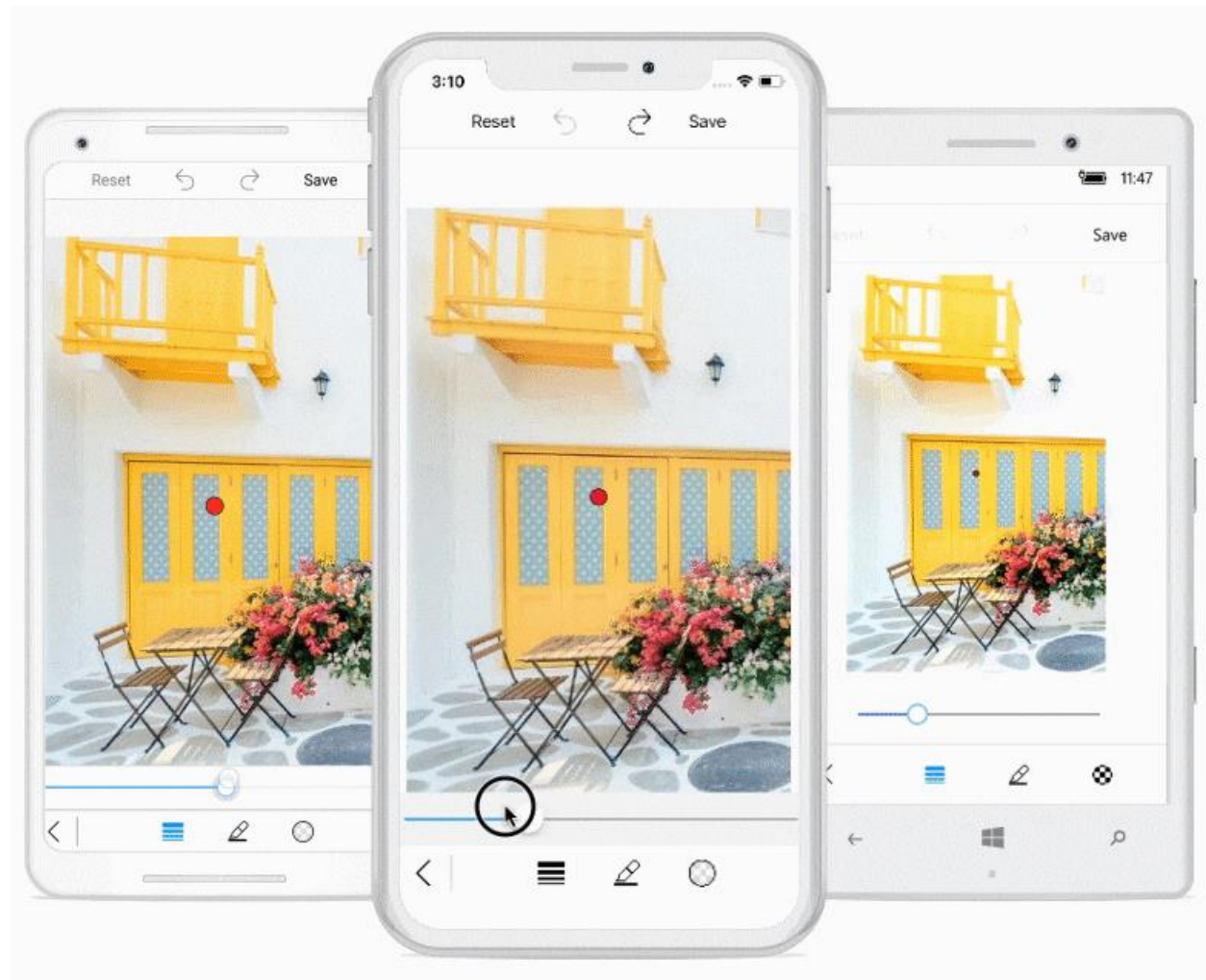


- You can annotate any path on an image using free hand drawing as shown in the following code snippet.

**C#**

```
editor.AddShape(ShapeType.Path, new PenSettings() { StrokeWidth = 10 });
```





### Deleting a shape or text from view

You can delete the selected shape by using the `Delete` method as shown in the following code snippet.

#### C#

```
editor.Delete();
```

---

**Note:** You cannot delete the path.

---

### Transformation

The image editor control is capable of performing the image transformations such as `rotation` and `flip`.

#### Rotation

You can use the `Rotate` method of the image editor to rotate a image. For each rotation, image will be rotated to 90 degrees towards clockwise direction.

---

**Note:** Angle cannot be specified in code to alter the rotation angle of the image.

---

#### C#

```
editor.Rotate();
```

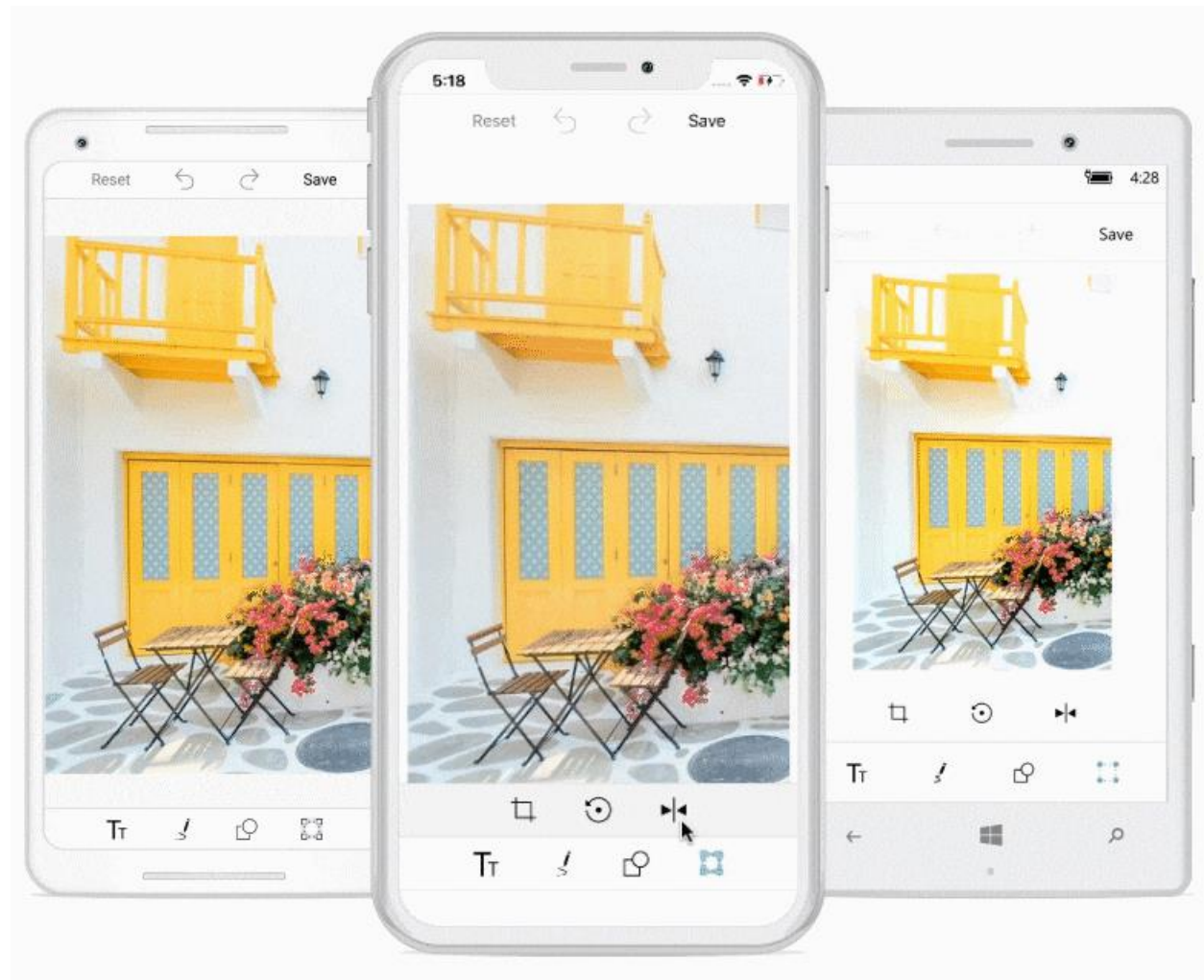
## Flip

The image editor control is capable of showing the mirror image. The `Flip` method flips the image horizontally or vertically based on the `FlipDirection` specified as argument of the flip method.

**Note:** The default flip direction is horizontal.

### C#

```
editor.Flip(FlipDirection.Horizontal);
```



## Crop

You can crop the desired portion of an image using the cropping tool.

### Image cropping ratio

You can crop the image with various aspect ratios. The following cropping ratios are available in built-in toolbar: "Free, Original, Square, 3:1, 1:3, 3:2, 2:3, 4:3, 3:4, 5:4, 4:5, 16:9, 9:16".

Cropping operation can be done in the following two ways:

- Enabling cropping and selecting the crop region visually.
- Entering the cropping area manually.

### *Handling the cropping tool*

The `ToggleCropping` method in the image editor control allows users to enable or disable the cropping region placed over the image to visually choose the area for cropping.

- The following code shows cropping the image to any desired size.

#### **C#**

```
// For free hand cropping.  
editor.ToggleCropping();
```

- The following code shows cropping an image based on its original width and height.

#### **C#**

```
// For cropping a image with its original width and height.  
editor.ToggleCropping(float.NaN, float.NaN);
```

- The following code shows cropping an image based on specific ratio.

#### **C#**

```
// For cropping the image with ratio, x value as 9, and y value as 17.  
editor.ToggleCropping(9, 17);
```

- To position the cropping window with custom location, pass the desired rectangle in `ToggleCropping` method. Each value in the rectangle should be in offset value(0 to 100).

#### **C#**

```
Rectangle rect = new Rectangle(20, 20, 50, 50);  
editor.ToggleCropping(rect);
```

After the cropping area has been selected, the `Crop` method is called, which in turn crops the selected region and displays the cropped image on the image editor.

#### **C#**

```
editor.Crop();
```

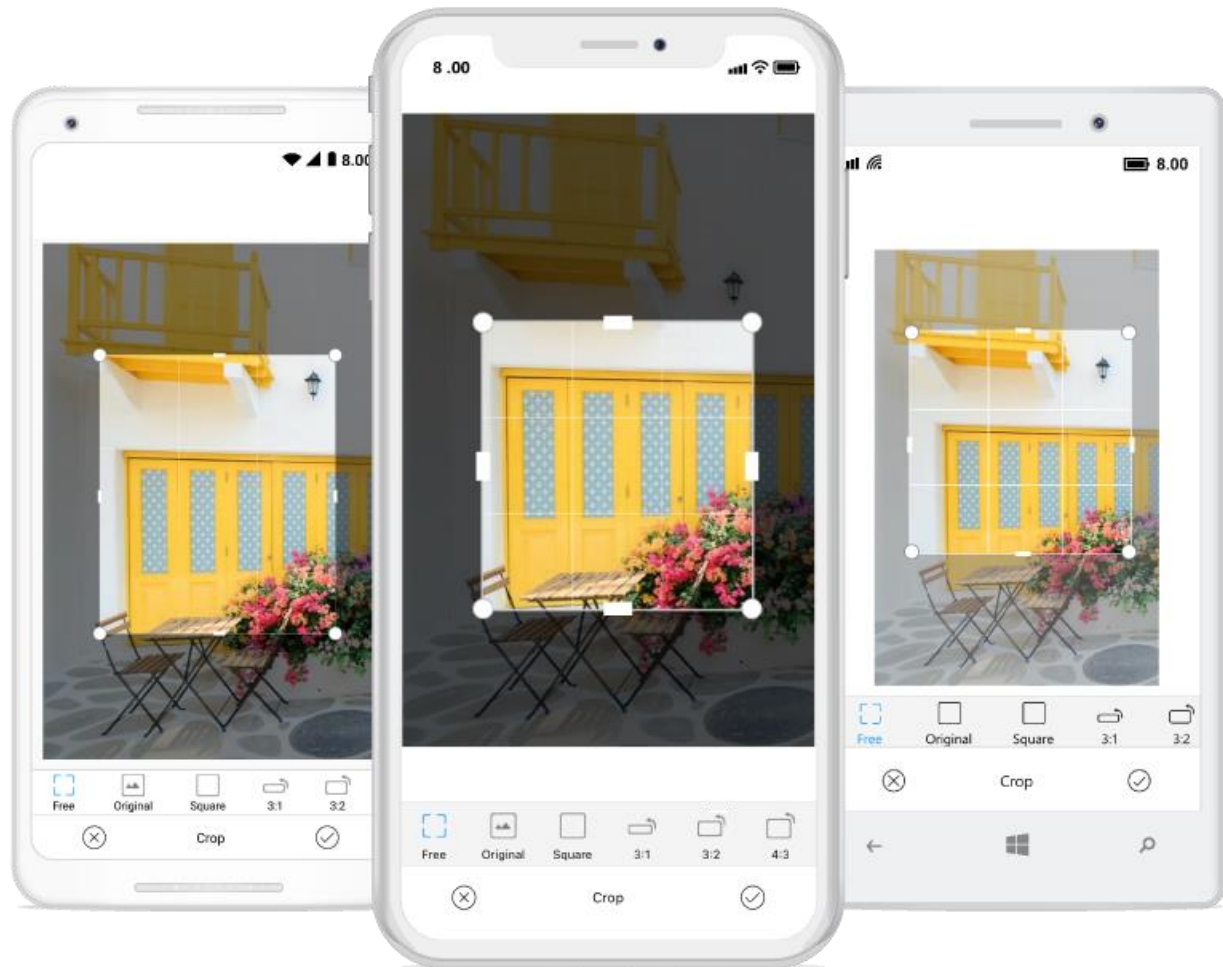
### *Entering the cropping area manually*

To manually enter the cropping area without enabling the cropping functionality, use the overloaded `Crop(Rectangle rect)` method. It can be done by defining a rectangle and passing it to the `Crop(rect)` method.

#### **C#**

```
editor.Crop(new Rectangle(100, 100, 150, 200));
```





### Save

You can save the image along with the current edits to the device using the **Save** method.

The saved image will be added to device for each platform in the following locations:

UWP :

The saved image will be added in default pictures library “C:\Users\your name\Pictures\Saved Pictures”.

Android:

The saved image will be added in default pictures library “Internal storage/Pictures/”.

### C#

```
editor.Save();
```

### Save events

The SfImageEditor has events when performing save operation namely **ImageSaving** and **ImageSaved**.

#### ImageSaving

This event occurs before saving the image. You can control the save functionality using the **Cancel** argument.

- Cancel:

It restricts saving image to the default location when set `Cancel` value to `true`.

#### **C#**

```
public MainPage()
{
    . . .
    editor.ImageSaving += editor_ImageSaving;
    . . .
}
private void editor_ImageSaving(object sender, ImageSavingEventArgs args)
{
    args.Cancel = true;
}
```

- Stream

You can get current image edits as stream using this argument.

#### **C#**

```
private void editor_ImageSaving(object sender, ImageSavingEventArgs args)
{
    var stream = args.Stream;
}
```

#### *ImageSaved*

This event occurs after the image has been saved. To get the location of the saved image, use the location argument as shown in the following code.

#### **C#**

```
public MainPage()
{
    . . .
    editor.ImageSaved += editor_ImageSaved;
    . . .
}
private void editor_ImageSaved(object sender, ImageSavedEventArgs args)
{
    string savedLocation = args.Location; // You can get the saved image
location with the help of this argument
}
```

#### *Saving Image with Custom Size and Format*

The `Save` method in the `SfImageEditor` control allows user to save an image in different format such as `png`, `jpg` and `bmp`.

To choose the format while Saving the image.

#### **C#**

```
editor.Save(".png");
```

To choose the format and size while Saving the image as like below,

### C#

```
editor.Save(".png", new Size(913, 764));
```

**Note:** Supported formats are .png, .jpg and .bmp.

### Reset

The **Reset** method resets the complete set of changes made in image and resets the image to original loaded image.

### C#

```
editor.Reset();
```

### Reset events

The SfImageEditor has events when performing reset operation namely **BeginReset** and **EndReset**.

#### BeginReset

This event occurs before resetting the changes made in an image. You can control the reset functionality using the Cancel argument.

### C#

```
public MainPage()
{
    . . .
    editor.BeginReset += editor_BeginReset;
    . . .
}
private void editor_BeginReset(object sender, BeginResetEventArgs args)
{
    args.Cancel = true; //It restricts resetting image to initial loaded image.
}
```

#### EndReset

This event occurs when reset has been completed.

### C#

```
public MainPage()
{
    . . .
    editor.EndReset += editor_EndReset;
    . . .
}
private void editor_EndReset(object sender, EndResetEventArgs args)
{
    Navigation.PushModalAsync(new NewImageEditorPage()); //Navigates to new page after completing the reset action.
}
```

### ImageLoaded Event

This event will be triggered after the image has been loaded. By using this event, you can add any shapes or text over an image or crop an image while initially loading the image.

#### C#

```
public MainPage()
{
    . . .
    editor.ImageLoaded += Editor_ImageLoaded;
    . . .
}
private void Editor_ImageLoaded(object sender, ImageLoadedEventArgs args)
{
    editor.AddShape(ShapeType.Circle, new PenSettings() {Color =
    Color.Green,Mode = Mode.Stroke });
}
```

### ItemSelected Event

This event will be triggered whenever you tap the selected shapes (rectangle, circle, and arrow) and text. You can get the settings of each selected shapes and text using the ItemSelected argument. You can also change the settings that will affect the selected shape.

#### C#

```
public MainPage()
{
    . . .
    editor.ImageLoaded += Editor_ImageLoaded;
    editor.ItemSelected += Editor_ItemSelected;
    . . .
}
private void Editor_ImageLoaded(object sender, ImageLoadedEventArgs args)
{
    editor.AddShape(ShapeType.Circle, new PenSettings() {Color =
    Color.Green,Mode = Mode.Stroke });
}
private void Editor_ItemSelected(object sender, ItemSelectedEventArgs args)
{
    var Settings = args.Settings;
    if (Settings is PenSettings)
    {
        (Settings as PenSettings).Color = color;
    }
    else
    {
        (Settings as TextSettings).Color = color;
    }
}
```

### ItemUnselected Event

This event will be triggered whenever you change the shape selections from one shape to another shape (rectangle, circle, arrow and text). You can get the settings of previous selected shape and text using the ItemUnselected event. You can also change the settings of previous selected shape.

#### C#

```
public MainPage()
{
    editor.ItemUnselected += Editor_ItemUnselected;
}
private void Editor_ItemUnselected(object sender, ItemUnselectedEventArgs e)
{
    var Settings = e.Settings;
    if (Settings is PenSettings)
    {
        (Settings as PenSettings).Color = Color.Green;
    }
    else
    {
        (Settings as TextSettings).Color = Color.Green;
    }
}
```

### ImageEdited Event

This event occurs whenever you start to edit an image. You can know whether the current image is edited or not by using the IsImageEdited argument.

#### C#

```
public MainPage()
{
    . . .
    editor.ImageEdited += ImageEditor_ImageEdited;
    . . .
}
private void ImageEditor_ImageEdited(object sender, ImageEditedEventArgs e)
{
    If (args.IsImageEdited)
    {
    }
}
```

### Undo and Redo

One of the important features of the image editor control is to perform **Undo** and **Redo** operations for adding shapes, text, and drawing paths.

#### Undo

The **Undo** method is used to revert the changes done previously over an image.

Undo can be performed for the following operations:

- Add/delete shapes or text.
- Change positions.

- Color/fill changes.
- Path drawings.

### C#

```
editor.Undo();
```

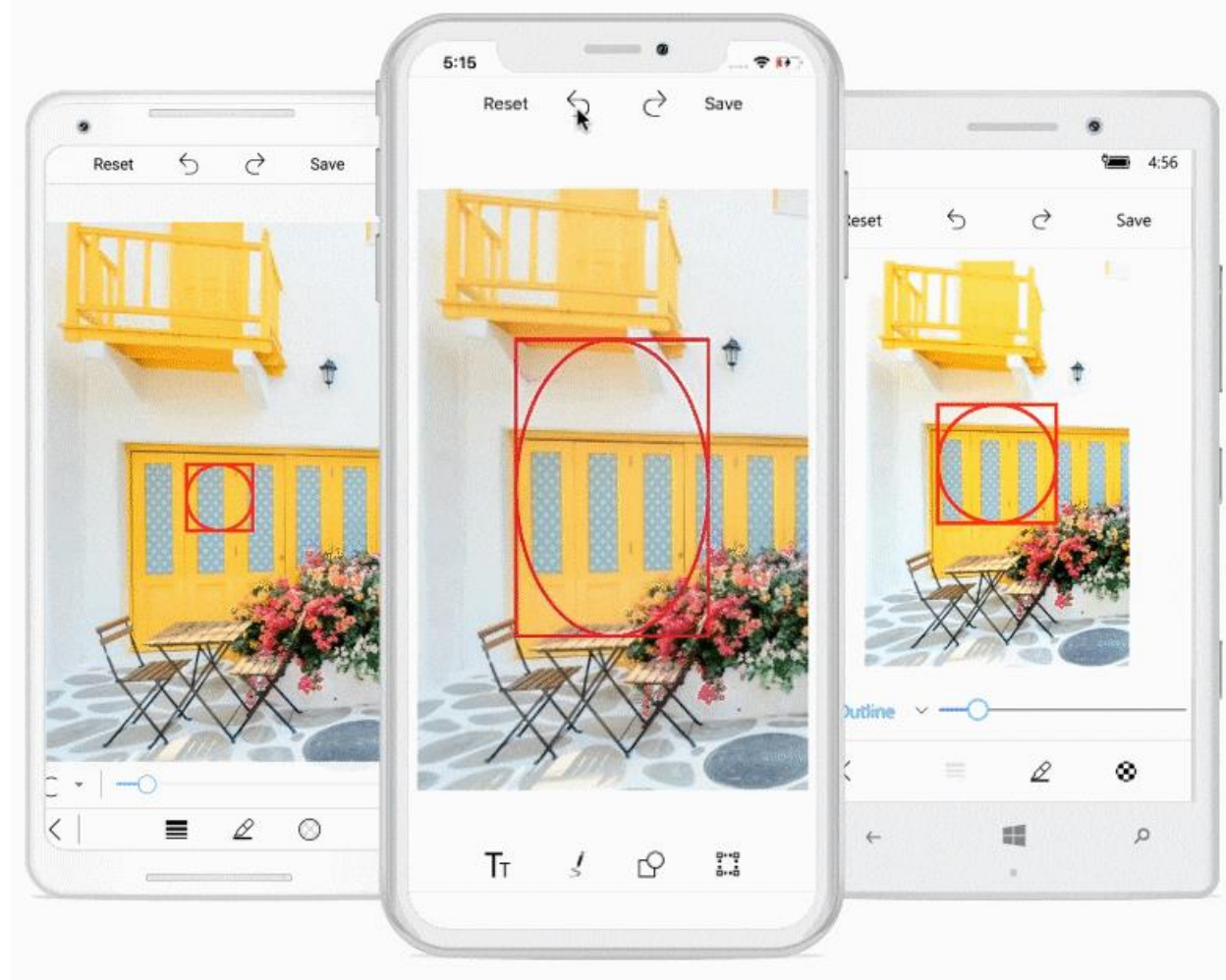
### Redo

The **Redo** method is used to redo the changes that are reverted by undo operation.

### C#

```
editor.Redo();
```

**Note:** Undo and redo cannot be applied for cropping and transformations.



### Toolbar customization

You can customize the color palette, visibility, and appearance of each toolbar item.

### Customize toolbar items

The image editor control provides support to customize and configure the appearance of toolbar menu. You can customize the toolbar by adding respective `FooterToolBarItem` and `HeaderToolBarItem`.

#### Toolbar item

You can customize each toolbar item using the `Text` and `Icon` properties.

#### Name

You can get or set the names of built-in toolbar and dynamically added toolbar items using the `Name` property.

#### C#

```
public MainPage()
{
    InitializeComponent();
    SfImageEditor editor = new SfImageEditor();
    var itemName = editor.ToolbarSettings.ToolbarItems[2].Name;
}
```

**Note:** The following built-in toolbar item names are available in image editor: Back, Text, Add, TextColor, FontFamily, Arial, Noteworthy, Marker Felt, Bradley Hand, SignPainter, Opacity, Path, StrokeThickness, Colors, Opacity, Shape, Rectangle, StrokeThickness, Circle, Arrow, Transform, Crop, free, original, square, 3:1, 3:2, 4:3, 5:4, 16:9, Rotate, Flip, Reset, Undo, Redo, and Save.

**Note:** You cannot modify the names of existing built-in toolbar items and cannot create toolbar item with these list.

The toolbar menu contains a set of header and footer menu items that help to perform editing actions. This can be categorized into the following types:

1. `HeaderToolBarItem`
2. `FooterToolBarItem`
3. SubItems

#### Adding HeaderToolBarItem

The `HeaderToolBarItem` is placed on the top of the image editor, and you can customize the header toolbar item using the `Icon` and `Text` properties:

#### C#

```
editor.ToolbarSettings.ToolbarItems.Add(new HeaderToolBarItem() { Icon =  
    ImageSource.FromResource("ImageEditor.share.png"), Text="Share" });
```

#### Adding FooterToolBarItem

The `FooterToolBarItem` is placed on the bottom of the image editor, and you can customize the footer toolbar item using the `Icon` and `Text` properties.

Refer to the following code snippet to customize footer toolbar item.

#### C#

```
editor.ToolbarSettings.ToolbarItems.Add(new FooterToolbarItem() { Icon =
ImageSource.FromResource("ImageEditor.delete.png"), Text="Delete" });
editor.ToolbarSettings.ToolbarItems.Add(new FooterToolbarItem() { Icon =
ImageSource.FromResource("ImageEditor.more.png"), Text="More" });
```

### Adding SubItems to the FooterToolbarItem

The **SubItems** is only applicable for **FooterToolbarItem**, and it represents grouped action of respective footer toolbar item. The SubItems will be placed above the footer toolbar item layout, and you can also customize the appearance of sub items as main toolbar items.

Refer to the following code snippet to customize sub items of footer toolbar item.

#### C#

```
editor.ToolbarSettings.ToolbarItems.Add(new FooterToolbarItem()
{
    Text = "More",
    Icon = ImageSource.FromResource("ImageEditor.Image.more.png"),
    SubItems = new ObservableCollection<ToolbarItem>()
    {
        new ToolbarItem() {
            Icon = ImageSource.FromResource("ImageEditor.Image.download.png")
        },
        new ToolbarItem() {
            Icon = ImageSource.FromResource("ImageEditor.Image.share.png")
        }
    }
});
```

**Note:** You can remove the existing toolbar items [names](#) from image editor toolbarItems collection based on the index value and change the icon and text values dynamically for any of the already added toolbar item based on the index as shown in the following code snippet.

#### C#

```
editor.ToolbarSettings.ToolbarItems[5].Text = "new item";
editor.ToolbarSettings.ToolbarItems[3].Icon =
ImageSource.FromResource("ImageEditor.Image.jpg");
```

### ToolbarItemSelected event

Whenever you tap the toolbar menu item, the **ToolbarItemSelected** event will be triggered, and you can get the respective tapped toolbar item as an argument as shown in the following code snippet.

#### C#

```
public MainPage()
{
    editor.ToolbarSettings.ToolbarItemSelected +=
    ToolbarSettings_ToolbarItemSelected;
}
private void ToolbarSettings_ToolbarItemSelected(object sender,
ToolbarItemSelectedEventArgs e)
{
    DisplayAlert("Selected ToolbarItem is " + e.ToolbarItem.Text, "Ok",
"Cancel");
}
```



```
}

```

### *MoveSubItemsToFooterToolbar*

The `MoveSubItemsToFooterToolbar` is boolean property of the `ToolbarItemSelected` event argument; it decides the placement of each sub items of respective footer toolbar item.

If you set the value to `true`, the respective sub items of footer item will be placed on footer toolbar layout. If you set `false`, then the sub items will be placed above the footer toolbar layout.

### **C#**

```
public MainPage()
{
    . . .
    SfImageEditor edit = new SfImageEditor();
    edit.ToolbarSettings.ToolbarItems.Add(new FooterToolbarItem()
    {
        Text = "NewFooterItem",
        SubItems = new ObservableCollection<ToolbarItem>()
        {
            new ToolbarItem() { Text= "Subitem1"},
            new ToolbarItem() { Text= "Subitem2"},
            new ToolbarItem() { Text= "Subitem3"},
        }
    });
    edit.ToolbarSettings.ToolbarItemSelected +=
    ToolbarSettings_ToolbarItemSelected;
    . . .
}

private void ToolbarSettings_ToolbarItemSelected(object sender,
ToolbarItemSelectedEventArgs e)
{
    if(e.ToolbarItem != null && e.ToolbarItem is FooterToolbarItem)
    {
        if(e.ToolbarItem.Text == "NewFooterItem")
        {
            e.MoveSubItemsToFooterToolbar = false;
        }
    }
}
```

---

**Note:** This is not applicable for built-in footer toolbar items.

---

### To hide/show toolbar

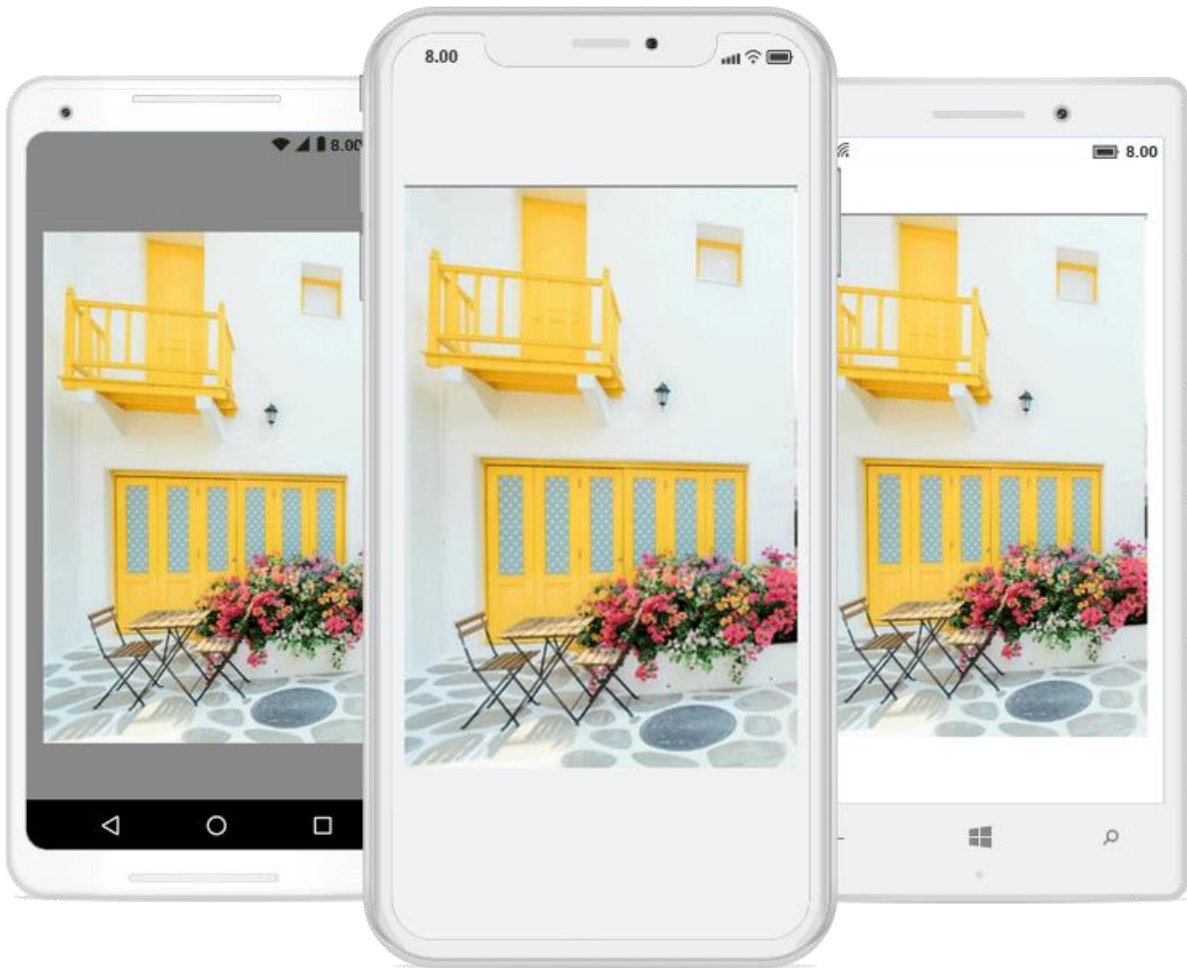
To show or hide the toolbar, set the `IsVisible` property of toolbar to either true or false. By default, the `IsVisible` property is set to true.

### **XML**

```
<imageeditor:SfImageEditor.ToolbarSettings>
<imageeditor:ToolbarSettings IsVisible="false" />
</imageeditor:SfImageEditor.ToolbarSettings>
```

### **C#**

```
editor.ToolbarSettings.IsVisible = false;
```



#### To hide/show the toolbar item

You can hide or show the toolbar items by specifying their icon names and setting the boolean values to true or false.

**Note:** You can customize an icon by specifying its [names](#).

#### C#

```
editor.SetToolbarItemVisibility("text,save", false);
```



### To customize the ColorPalette

You can change the default colors of the ColorPalette in toolbar.

#### XML

```
<imageeditor:SfImageEditor.ColorPalette>
  <Color>Yellow</Color>
  <Color>Pink</Color>
  <Color>Violet</Color>
</imageeditor:SfImageEditor.ColorPalette>
```

#### C#

```
ObservableCollection<Color> CustomColorPalette = new
ObservableCollection<Color>()
{
  Color.Yellow,
  Color.Pink,
  Color.Violet
};
editor.ColorPalette = CustomColorPalette;
```

### Default Color Selected Index

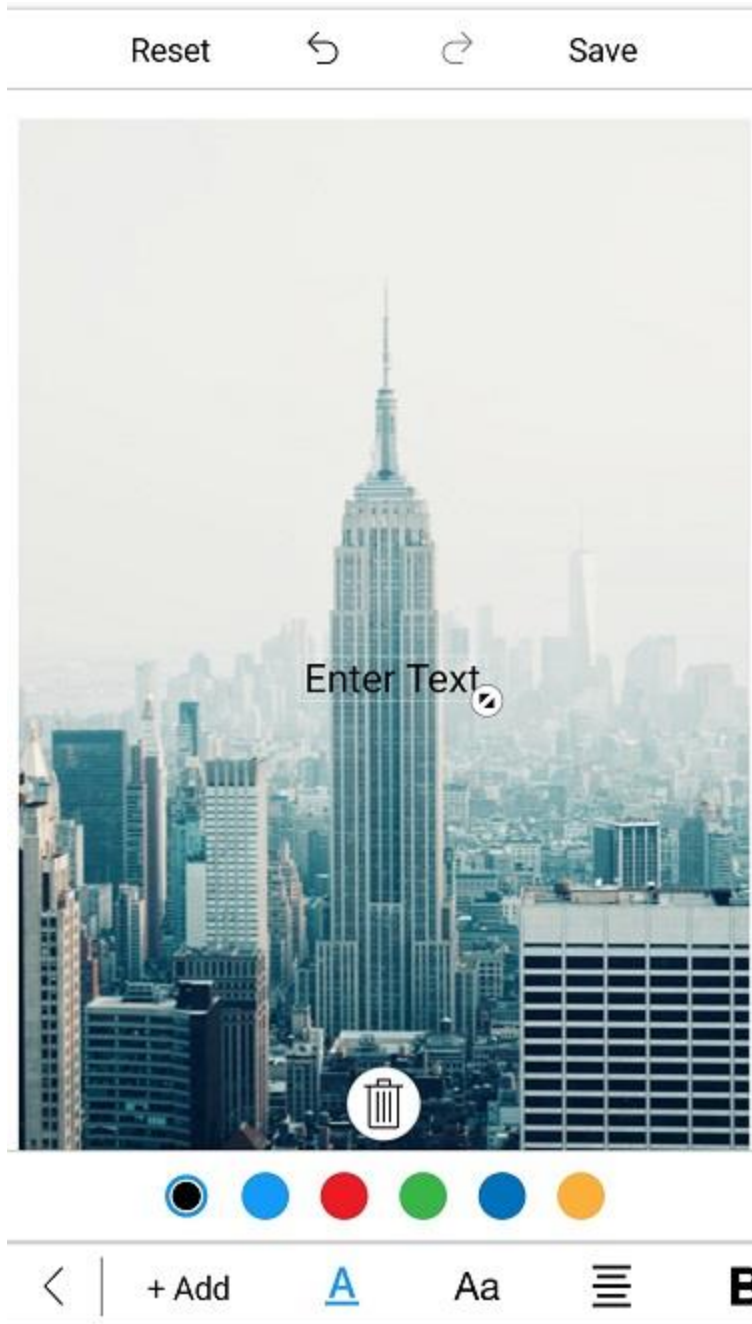
You can change the default index of the color palette in toolbar. By default, color palette index value is 2.

#### XML

```
<imageeditor:SfImageEditor x:Name="editor" DefaultSelectedColorIndex="0"/>
```

#### C#

```
editor.DefaultSelectedColorIndex = 0;
```



### ToolbarHeight Customization

You can customize **height of the toolbar** and toolbar items **icon** and **text**.

#### *Customize toolbar height*

The image editor control supports to customize the default height of the **Header**, **Footer**, and **Sub item** using the following properties:

1. HeaderToolbarHeight
2. FooterToolbarHeight
3. SubItemToolbarHeight

The toolbar items will be resized based on the height. To change the height of the toolbar, refer to the following code snippet.

#### XML

```
<imageeditor:SfImageEditor.ToolbarSettings>  
<imageeditor:ToolbarSettings  
HeaderToolbarHeight="70"  
FooterToolbarHeight="70"  
SubItemToolbarHeight="70"/>  
</imageeditor:SfImageEditor.ToolbarSettings>
```

#### C#

```
editor.ToolbarSettings.HeaderToolbarHeight = 70;  
editor.ToolbarSettings.FooterToolbarHeight = 70;  
editor.ToolbarSettings.SubItemToolbarHeight = 70;
```



#### Individual toolbar item height customization

You can arrange the toolbar items based on the toolbar height using the following properties:

1. TextHeight
2. IconHeight

To change the toolbar item Text and Icon height, refer to the following code snippet.

#### C#

```
FooterToolbarItem footerItem = new FooterToolbarItem()  
{  
    IconHeight=40,  
    TextHeight=20,  
    Icon = ImageSource.FromResource("ImageEditor.share.png"),  
    Text = "Share"  
};  
editor.ToolbarSettings.ToolbarItems.Add(footerItem);
```

## Zooming

The image editor control provides support to zoom and pan actions over an image.

The following properties are used for zooming feature of the image editor control:

- EnableZooming
- Maximum ZoomLevel
- PanningMode

### Enable zooming

You can enable or disable the zooming functionality by setting the **EnableZooming** value to true or false. By default, the **EnableZooming** value is set to true.

#### XML

```
<imageeditor:SfImageEditor EnableZooming="false"/>
```

#### C#

```
editor.EnableZooming = false;
```

### Maximum zoom level

You can set the maximum zoom level to image using the **MaximumZoomLevel** property.

#### XML

```
<imageeditor:SfImageEditor EnableZooming="true" MaximumZoomLevel="8"/>
```

#### C#

```
editor.EnableZooming = true;  
editor.MaximumZoomLevel = 8;
```

### Panning mode

The image editor control provides support for panning and allows to pan the image with two fingers or single finger by setting the **PanningMode**.

The following properties are used for panning:

- **SingleFinger**: Zooms or pans the image with single finger, but shapes and text selection cannot be performed with this mode.
- **TwoFinger**: Zooms or pans the image with two finger. The shapes and text selection can be performed with this mode.

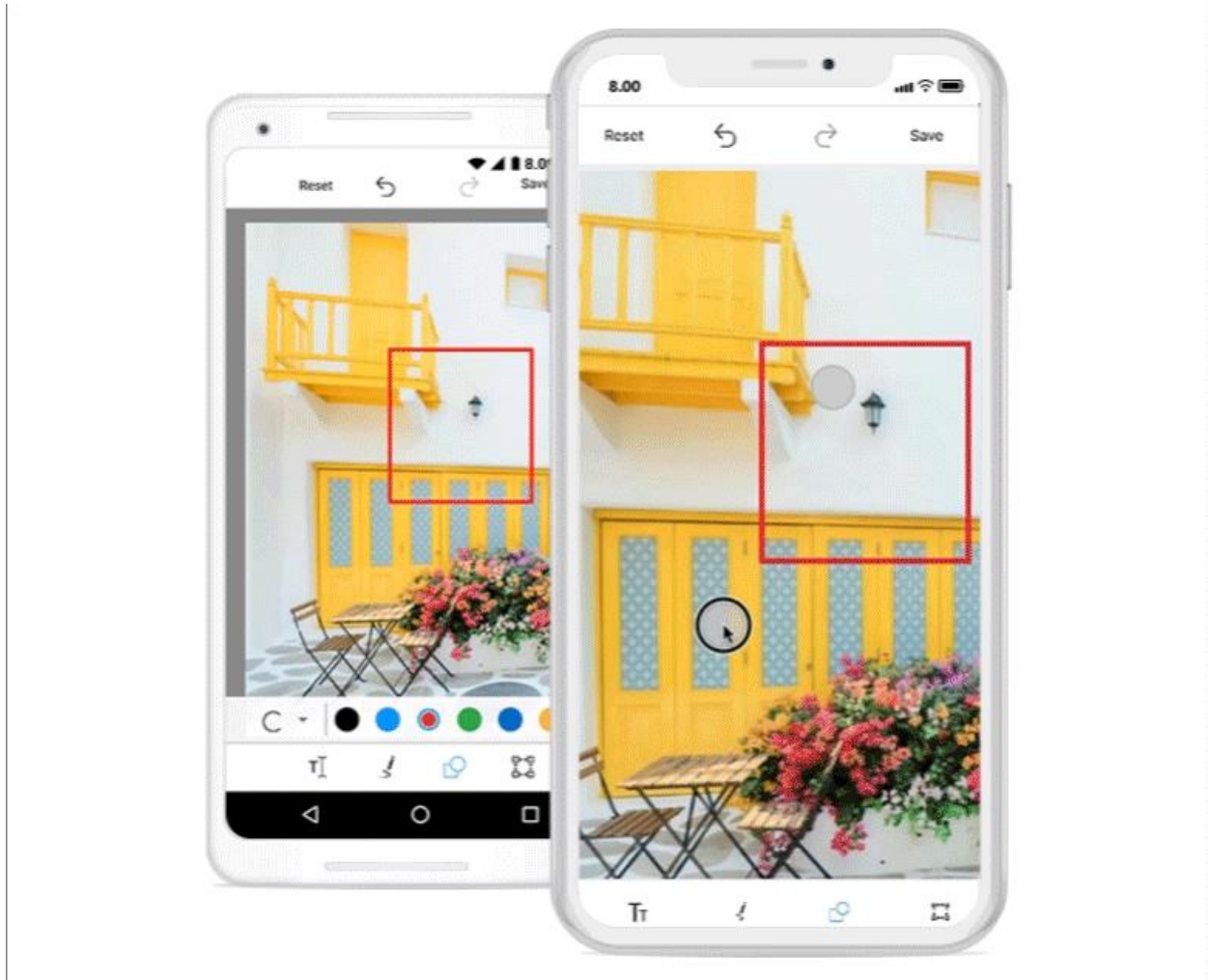
By default, the **PanningMode** value is set to **TwoFinger**.

#### XML

```
<imageeditor:SfImageEditor EnableZooming="true" PanningMode="TwoFinger"/>
```

**C#**

```
editor.PanningMode = PanningMode.TwoFinger;
```

**Serialization and Deserialization**

The image editor control provides support to serialize and deserialize the shapes (circle, arrow, and rectangle), free hand drawing, text and toolbar settings. You can save the current state of the image editor and load it back when it is needed.

**Serialization**

The SaveEdits() method is used to serialize the current edits of shapes. The serialized object will be returned in the form of JSON stream.

**C#**

```
SfImageEditor editor;  
public MainPage()  
{  
    InitializeComponent();  
    Grid grid = new Grid();  
    editor = new SfImageEditor();  
    Button saveEdits = new Button();
```

```
saveEdits.Text = "Serialize";
saveEdits.Clicked += SaveEdits_Clicked;
grid.RowDefinitions.Add(new RowDefinition() { Height = 100 });
grid.RowDefinitions.Add(new RowDefinition() { Height = GridLength.Star });
grid.Children.Add(saveEdits, 0, 0);
grid.Children.Add(editor, 0, 1);
this.Content = grid;
}
void SaveEdits_Clicked(object sender, EventArgs e)
{
    Stream stream = editor.SaveEdits();
}
```

You can save stream into .txt format file. If you save stream as .txt format file to deserialize the shapes, then set the Build action to **Embedded resource** in project.

Sample text file: [ImageEditor.txt](#).

### Deserialization

The LoadEdits() method is used to deserialize the edits over an image.

### C#

```
SfImageEditor editor;
public MainPage()
{
    InitializeComponent();
    Grid grid = new Grid();
    editor = new SfImageEditor();
    Button LoadEdits = new Button();
    LoadEdits.Text = "Deserialize";
    LoadEdits.Clicked += LoadEdits_Clicked;
    grid.RowDefinitions.Add(new RowDefinition() { Height = 100 });
    grid.RowDefinitions.Add(new RowDefinition() { Height = GridLength.Star });
    grid.Children.Add(LoadEdits, 0, 0);
    grid.Children.Add(editor, 0, 1);
    this.Content = grid;
}
void LoadEdits_Clicked(object sender, EventArgs e)
{
    var assembly = typeof(App).GetTypeInfo().Assembly;
    Stream stream = new MemoryStream();
    var fileName = "namespace_name.Chart.txt";
    stream = assembly.GetManifestResourceStream(fileName);
    if (stream != null)
        editor.LoadEdits(stream);
}
```





### Moving shapes to front and back

The image editor control allows to change the position of shapes/edits that are arranged over the editor. This can be achieved using the following methods:

1. BringToFront
2. SendToBack
3. BringForward
4. SendBackward

#### BringToFront

The BringToFront method is used to bring the selected shapes/text to the front of a group of elements over an image.

#### C#

```
editor.BringToFront();
```

### SendToBack

The SendToBack method is used to send the selected shapes/text to the back of a group of elements over an image.

#### **C#**

```
editor.SendToBack();
```

### BringForward

The BringForward method is used to bring the selected shapes/text to one step front of a group of elements over an image.

#### **C#**

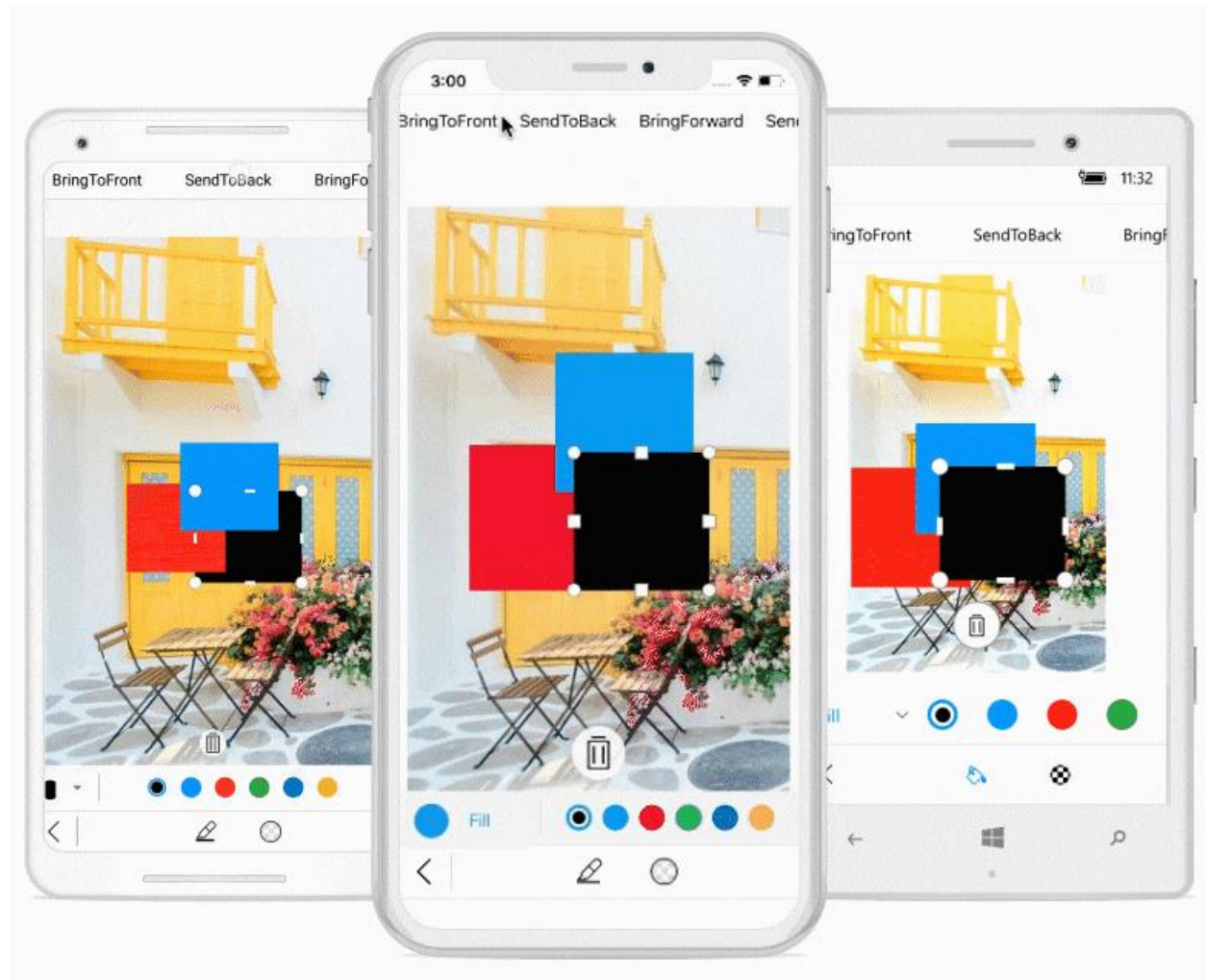
```
editor.BringForward();
```

### SendBackward

The SendBackward method is used to send the selected shapes/text to one step backward of a group of elements over an image.

#### **C#**

```
editor.SendBackward();
```



## Localization

The image editor control supports localization. You can localize the contents of image editor by adding equivalent localized strings.

### Change default language

Based on resource strings in the project, the contents are localized. By default, the image editor control is available in English.

You can localize image editor contents in the following two ways:

- Using Resx file from PCL.
- From platform-specific projects.

### Using Resx file from PCL

You can localize the text from PCL by adding equivalent localized strings in the resource file. Add the required resx files with **Build Action** -> **EmbeddedResource** (File name should contains culture code) under the **Resources** folder.

E.g., For Japanese, filename should be **Syncfusion.SfImageEditor.XForms.ja-JP.resx**.

Initializes a new instance of the `ResourceManager` class that looks up resources with the specified root name in the given assembly.

### C#

```
ImageEditorResourceManager.Manager = new ResourceManager (ResXPath,  
Assembly);
```

Here,

ResXPath => Full path of the resx file

Assembly => Application assembly (PCL)

The `CurrentCulture` must be set in the platform projects.

Convert the platform-specific format to a .NET format, and set it to `Thread.CurrentThread.CurrentCulture`.

You can use the value set to this static property from our source to read the values in PCL project's `Syncfusion.SfImageEditor.XForms.ja-JP.resx` files.

### C#

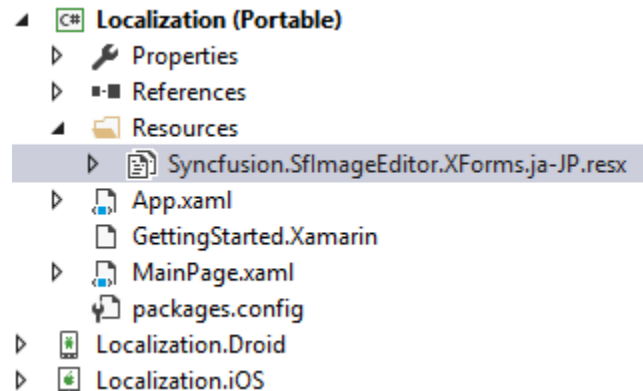
```
public void SetLocale(CultureInfo culture)  
{  
    Thread.CurrentThread.CurrentCulture = culture;  
    Thread.CurrentThread.CurrentUICulture = culture;  
}  
public CultureInfo GetCurrentCultureInfo()  
{  
    var netLanguage = "en";  
    var androidLocale = Java.Util.Locale.Default;  
    netLanguage =  
        AndroidToLanguage(androidLocale.ToString().Replace("_", "-"));  
    CultureInfo culture = null;  
    try  
    {  
        culture = new CultureInfo(netLanguage);  
    }  
    catch  
    {  
        try  
        {  
            var fallback = ToFallbackLanguage(new  
                PlatformCulture(netLanguage));  
            culture = new CultureInfo(fallback);  
        }  
        catch  
        {  
            culture = new CultureInfo("en");  
        }  
    }  
    return culture;  
}
```

The following code snippet shows converting Japanese language to `CultureInfo` equivalent.

**C#**

```
var netLanguage = platCulture.LanguageCode;
case "ja":
netLanguage = "ja-JP"; //equivalent to Japanese for this app
break;
```

Change the language preference in device.



The following screenshot shows localizing the text to Japanese language.

Name	Value
Add	追加する
Cancel	キャンセル
Crop	作物
Done	完了
EditText	テキストの編集
EnterText	テキストを入力
Fill	塗りつぶす
Free	無料
OK	完了
Original	元の
Outline	アウトライン
Reset	リセット
Save	セーブ
Saving	保存する
Square	平方

#### From platform-specific projects

You can localize the text from different platforms by adding equivalent localized strings in resource file. For localizing text, configure it to each platform separately.

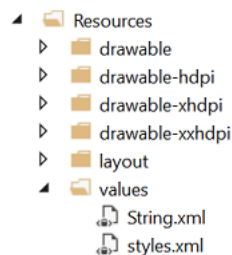
- Localizing the text in Android renderer.
- Localizing the text in iOS renderer.
- Localizing the text in UWP renderer.

### Android

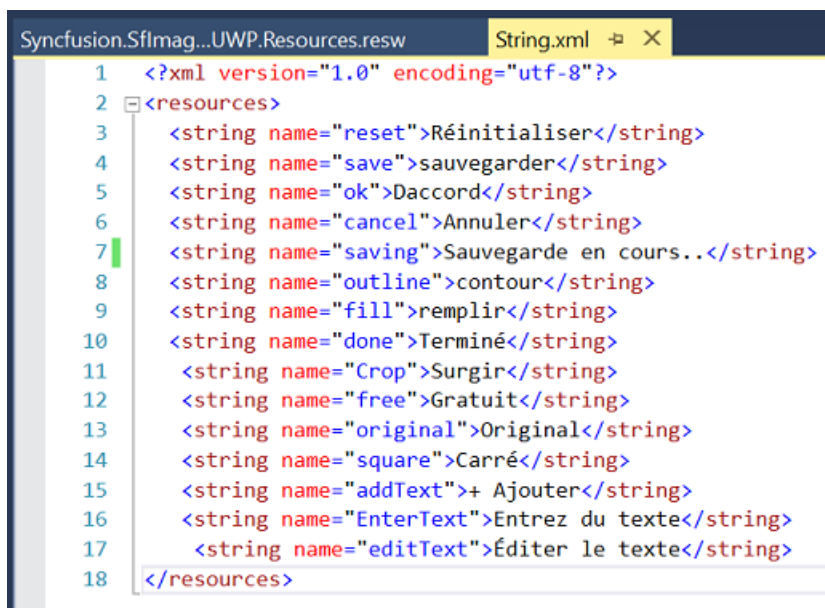
You can localize the text available in the control by adding equivalent localized strings in resource file.

Create String.xml in resource file in Android

Location- ProjectName.Android/Resources/values/Strings.Xml



The following screenshot shows localizing the text to French language.

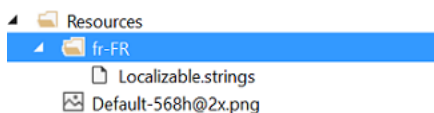


### iOS

You can localize custom text available in the control by adding equivalent localized strings in resource file.

Create Localizable.strings in resource file in iOS.

Location- ProjectName.iOS/Resources/Localizable.strings



The following screenshot shows localizing the text to French language.

```

Localizable.strings
1  "Reset"="Réinitialiser";
2  "Save"="sauvegarder";
3  "Ok"="D'accord";
4  "Cancel"="Annuler";
5  "Outline"="contour";
6  "Fill"="remplir";
7  "EnterText"="Entrez du texte";
8  "Crop"="Surgir";
9  "Free"="Gratuit";
10 "Original"="Original";
11 "Square"="Carré";
12 "Add" = "+ Ajouter";

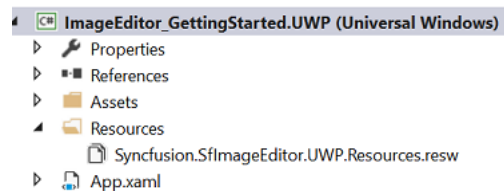
```

### UWP

You can localize custom text available in the control by adding equivalent localized strings in resource file.

Create Syncfusion.SfImageEditor.UWP.Resources file in UWP.

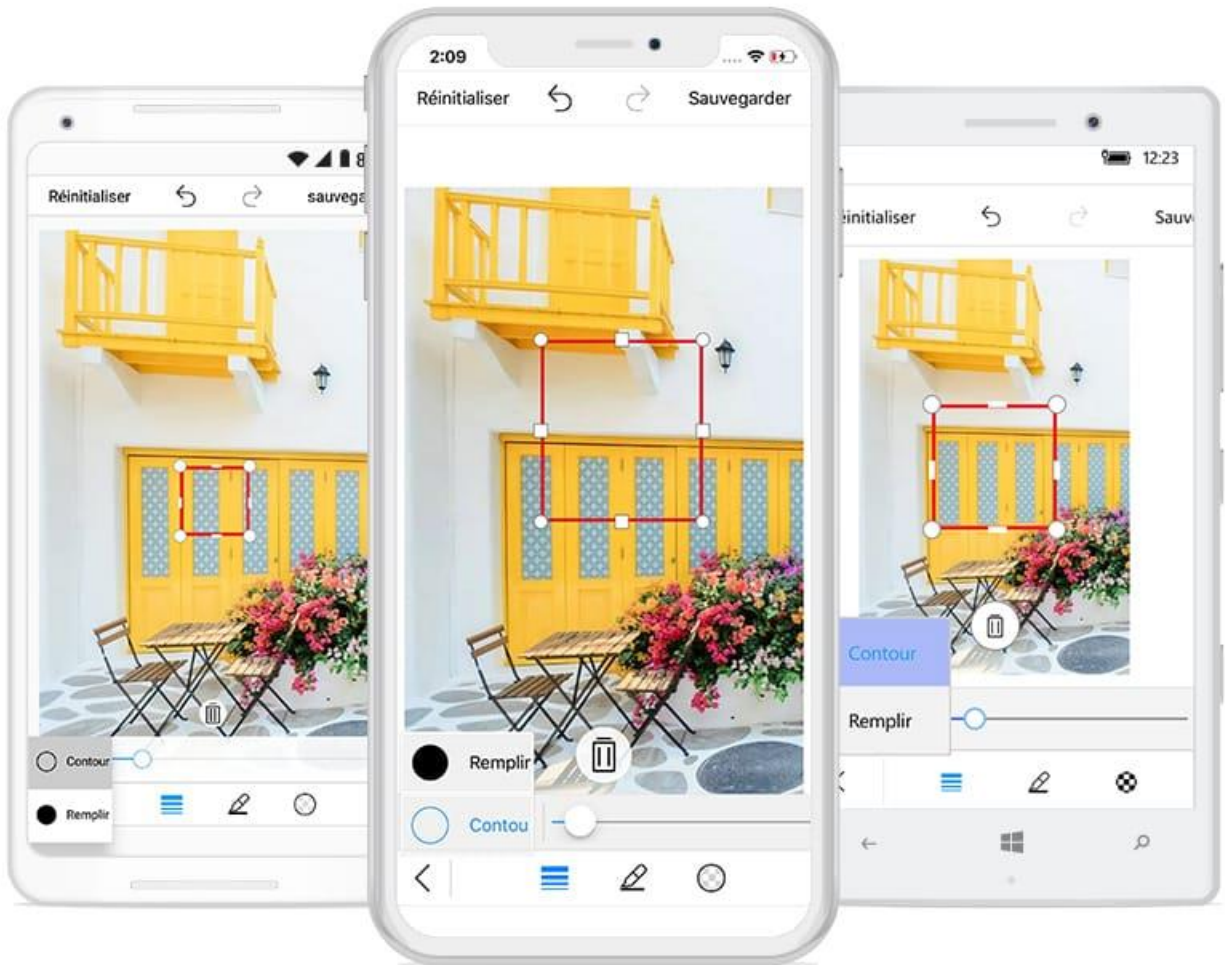
Location- ProjectName.UWP/Resources/Syncfusion.SfImageEditor.UWP.Resources



The following screenshot shows localizing the text to French language.

Syncfusion.SfImage...UWP.Resources.resw String.xml		
Add Resource Remove Resource		
	Name	Value
▶	Add	Ajouter
	Cancel	Annuler
	CropText	Surgir
	Enter Text	Entrez du texte
	EnterText	Entrez du texte
	Fill	Remplir
	Free	Free
	Ok	D'accord
	Original	Original
	Outline	Contour
	Reset	Réinitialiser
	Save	Sauvegarder
	Square	Carré
*		

The following screenshot shows localizing the text to French language in image editor.



### CustomView

You can add any custom shapes or views to an image using the `AddCustomView` method in the image editor control. To add a custom view, specify the view and its desired `CustomViewSettings` as shown in the following code snippet.

#### C#

```
Image customImage = new Image() { HeightRequest = 200, WidthRequest = 200 };
Assembly assembly = Assembly.GetAssembly(typeof(Sample));
customImage.Source =
    ImageSource.FromResource("sample_namespace.CustomImage.png", assembly);
imageEditor.AddCustomView(customImage, new CustomViewSettings());
```

### CustomViewSettings

The `CustomViewSettings` is defined to set the values for `CanMaintainAspectRatio`, `Bounds` and `Angle`.

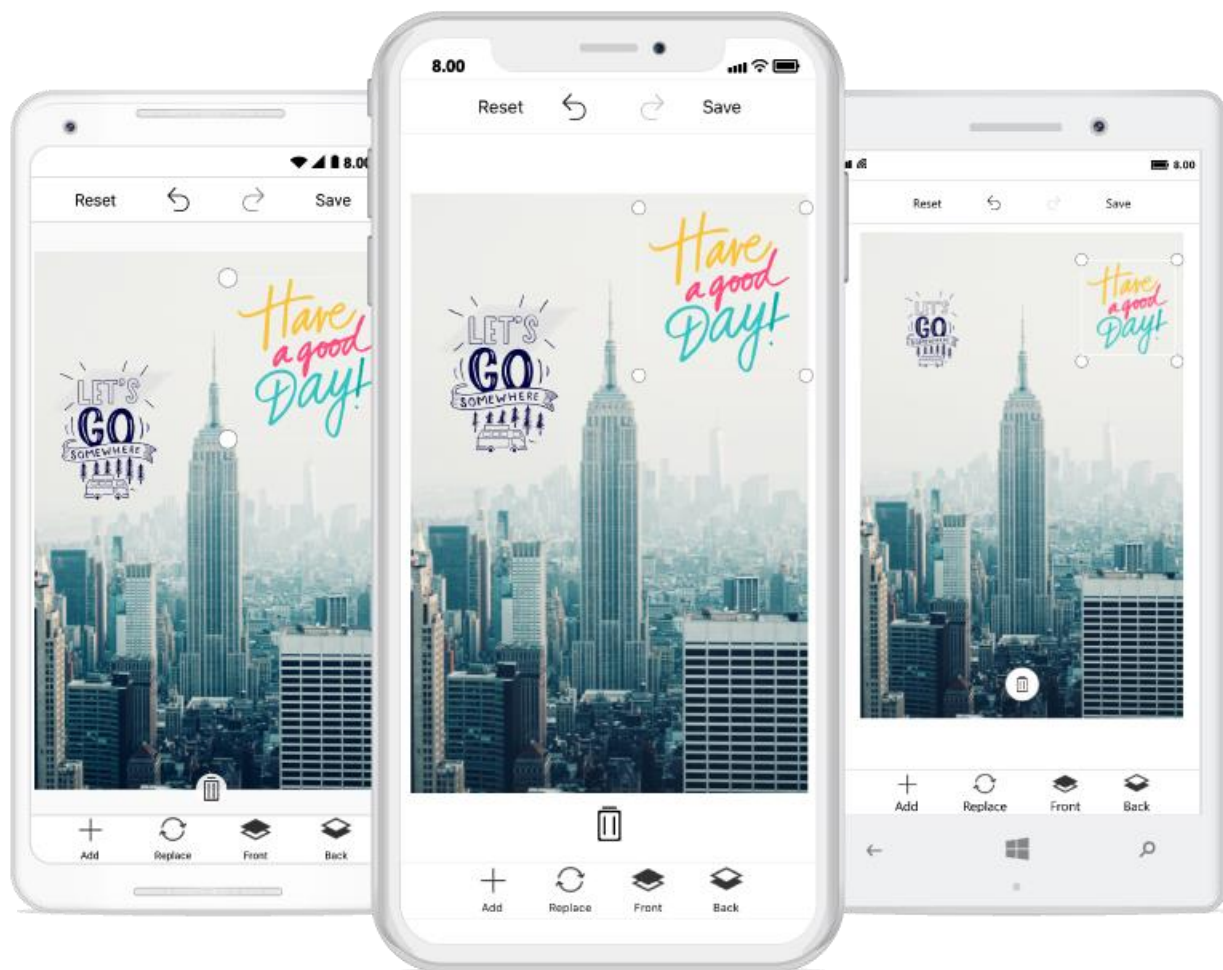
- The `CanMaintainAspectRatio` property is used to decide whether the aspect ratio value needs to be maintained when resizing the custom view.



- Bounds property is used to set the bounds of the custom view. Using this property, you can position the custom view wherever you want on the image. In percentage, the value should fall between 0 and 100.
- Angle property is used to set the angle of the custom view. Using this property, you can rotate the custom view at desired angle.

**C#**

```
CustomViewSettings customViewSettings = new CustomViewSettings()  
{  
    CanMaintainAspectRatio = false,  
    Bounds = new Rectangle(0, 0, 100, 100),  
    Angle=45  
};
```



### CustomView Rotation

You can rotate and resize the custom view by enabling the `RotatableElements` property of image editor. `ImageEditorElements` is an enum type with values `Text`, `CustomView` and `None` as shown in the following code snippet.

**C#**

```
editor.RotatableElements = ImageEditorElements.CustomView;
```

**Note:** The default value for RotatableElements is **None**.

You can rotate the custom view based on a particular angle using **Angle** property in **CustomViewSettings** as shown in the following code snippet.

**C#**

```
imageEditor.AddCustomView(customImage, new CustomViewSettings() {Angle = 45});
```

![SfImageEditor](ImageEditor\_images/rotation.png)

### Image Filter

Using the image editor control, you can add effects such as Hue, Saturation, Brightness, Contrast, Blur, and Sharpen to the image. These effects can be applied from toolbar or using the **ApplyImageEffect** method. The **ApplyImageEffect** method contains two arguments: **ImageEffect** and **EffectValue**. The **ImageEffect** is an Enum, which contains the following effects:

- Hue
- Saturation
- Brightness
- Contrast
- Blur
- Sharpen
- None

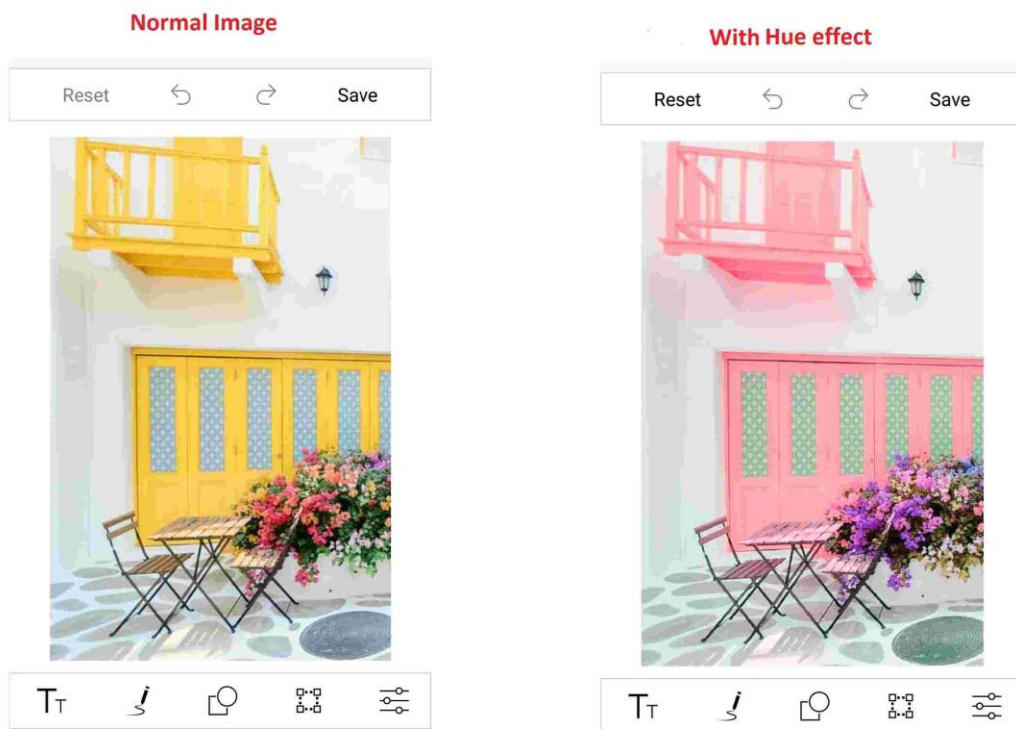
The **EffectValue** are the corresponding **ImageEffect** values, which varies for each effect, and they are explained as follows.

### Hue

The Hue represents the dominant wavelength of the color. The value of hue effect ranges from -180 to 180.

**C#**

```
public MainPage()  
{  
    . . .  
    editor.ApplyImageEffect(ImageEffect.Hue, 120);  
    . . .  
}
```

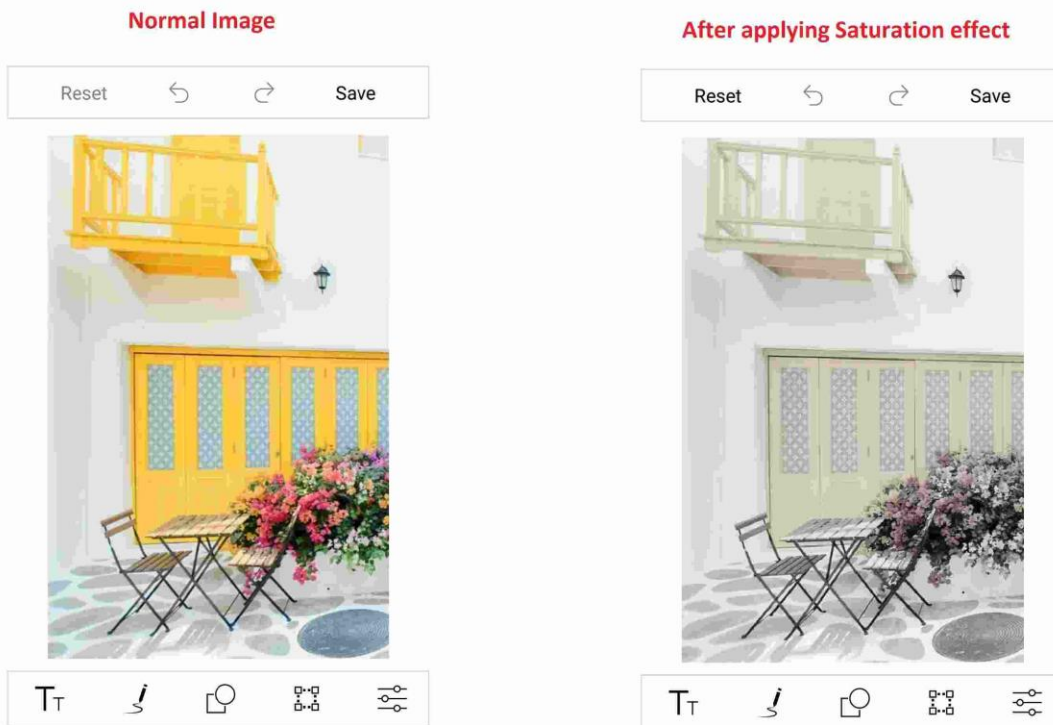


### Saturation

The Saturation represents the intensity of the color. The value of the saturation effect ranges from -100 to 100.

### C#

```
public MainPage()
{
    . . .
    editor.ApplyImageEffect(ImageEffect.Saturation, 20);
    . . .
}
```

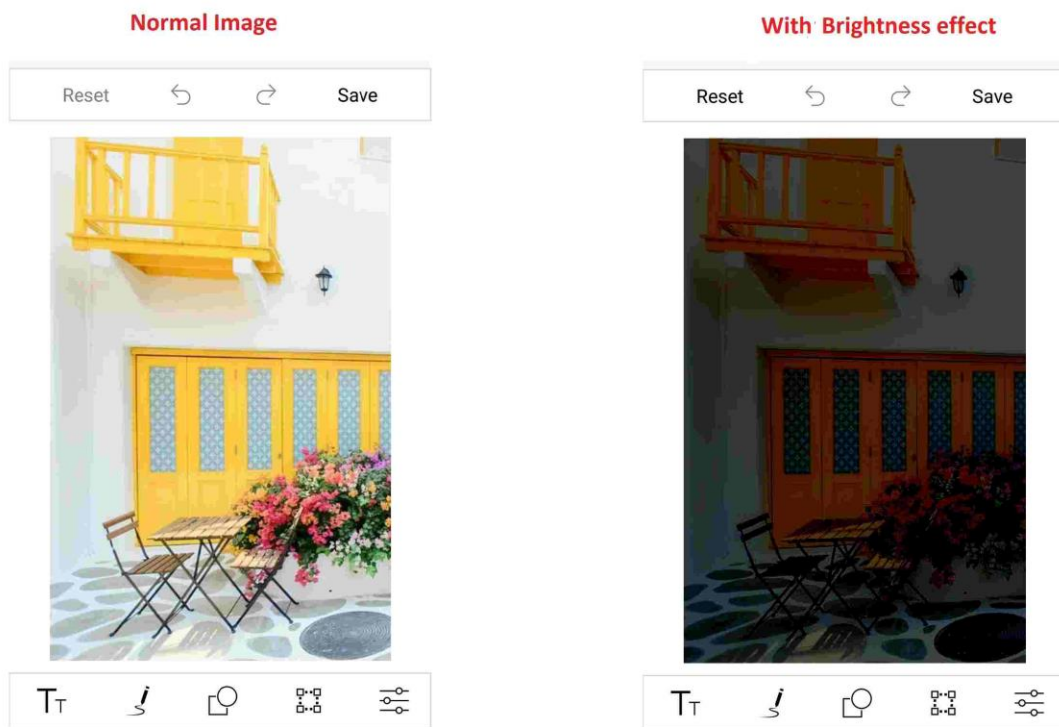


## Brightness

The brightness represents how bright the color is. The value of brightness effect ranges from -100 to 100.

### C#

```
public MainPage()  
{  
    . . .  
    editor.ApplyImageEffect(ImageEffect.Brightness, 80);  
    . . .  
}
```

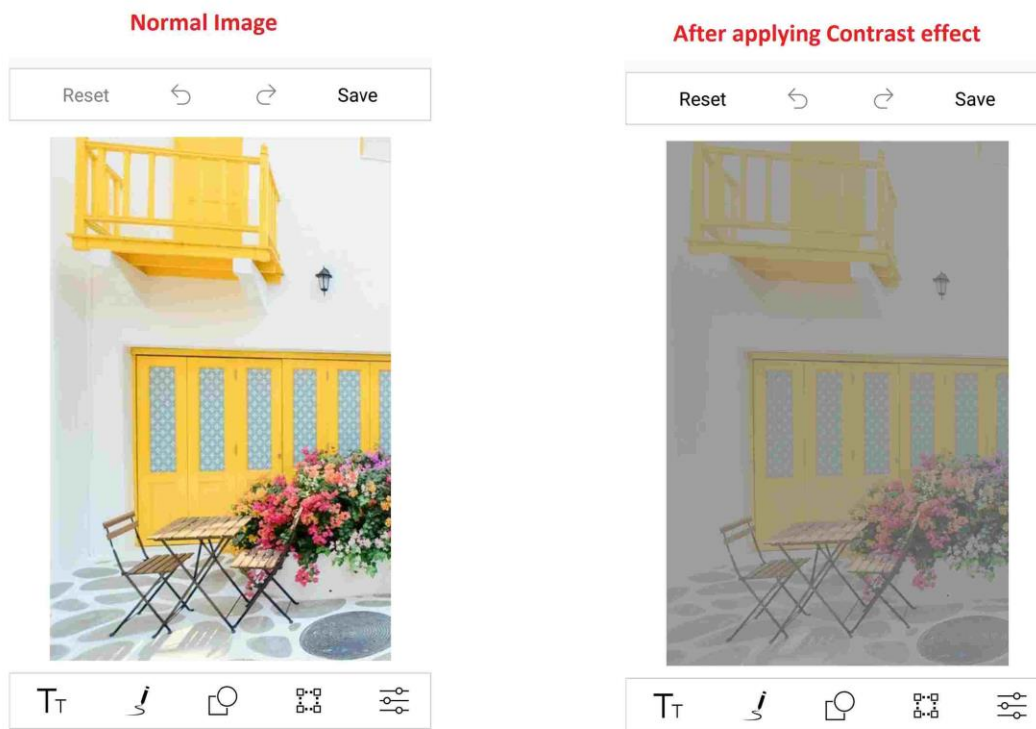


### Contrast

The contrast represents the color contrast of an image. The value of contrast effect ranges from -100 to 100.

### C#

```
public MainPage()
{
    . . .
    editor.ApplyImageEffect(ImageEffect. Contrast, 30);
    . . .
}
```

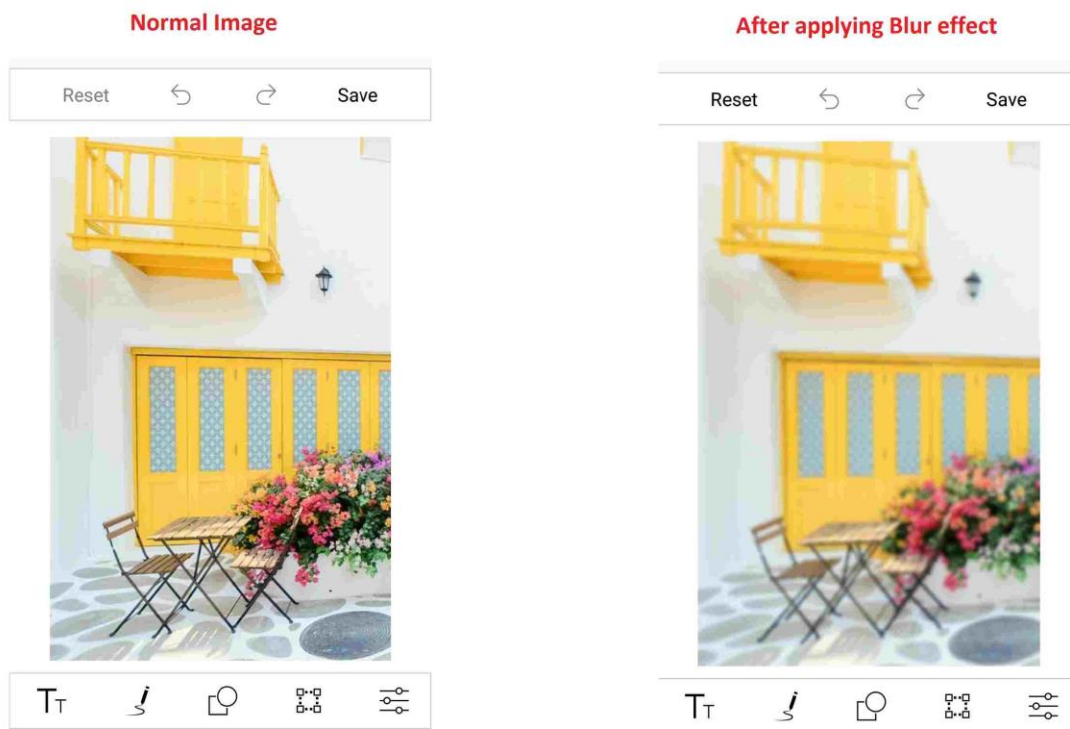


## Blur

The Blur represents the clearness of the image. The value of blur effect ranges from 0 to 6.

### C#

```
public MainPage()
{
    . . .
    editor.ApplyImageEffect(ImageEffect.Blur, 4);
    . . .
}
```



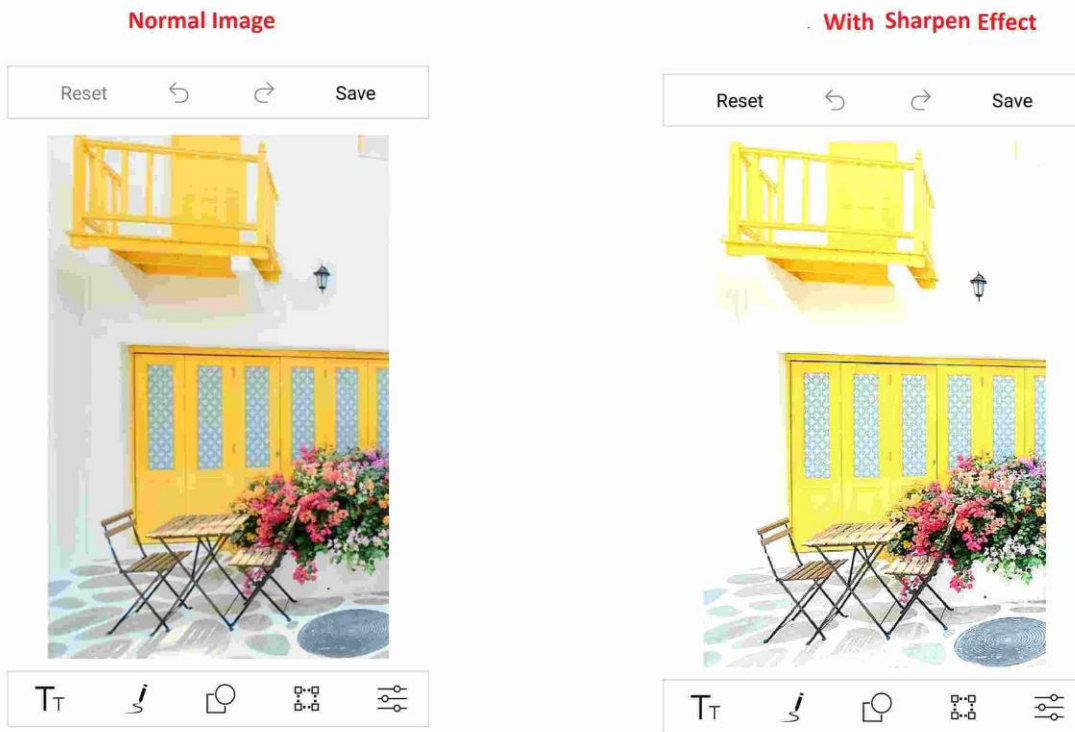
### Sharpen

Sharpen is used to highlight edges and fine details in an image. The value of sharpen effect ranges from 0 to 6.

#### C#

```
public MainPage()  
{  
    . . .  
    editor.ApplyImageEffect(ImageEffect.Sharpen, 2);  
    . . .  
}
```



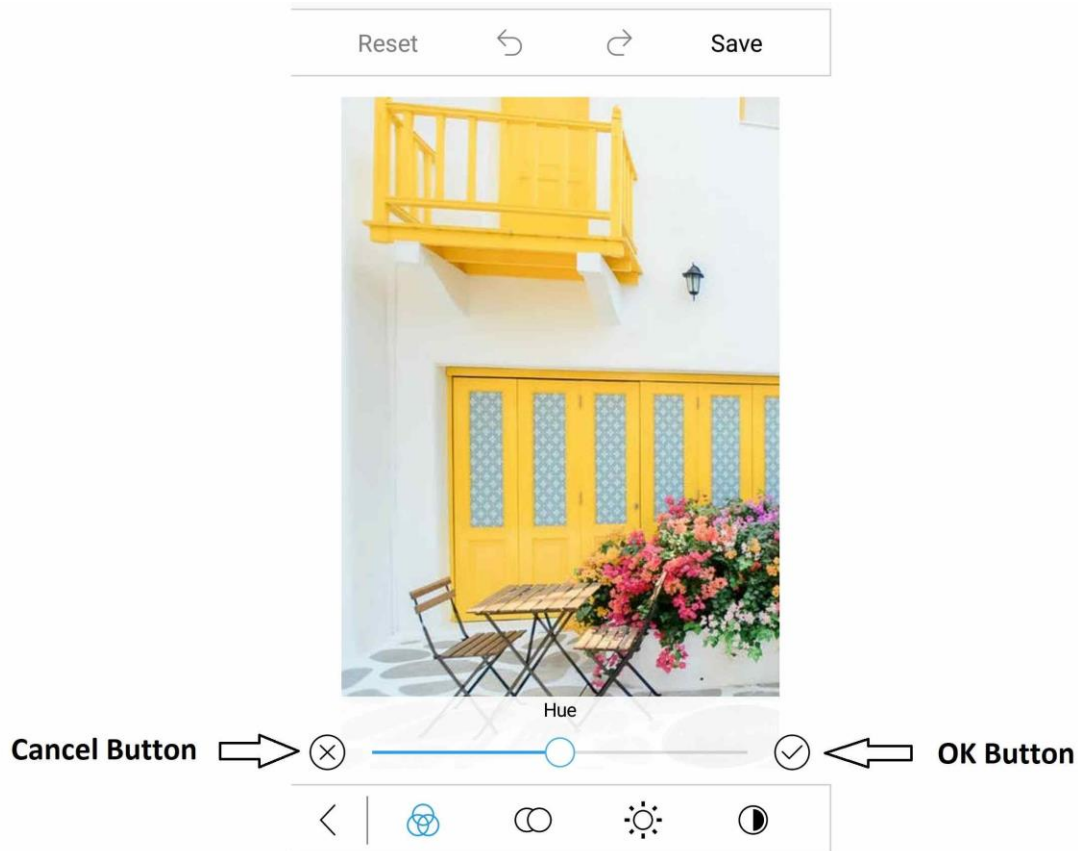


---

**Note:** The Image Effect enum also contains “None” option, which removes all the previously applied effects, which are not saved displays the original image. When applying effect through the Apply Image effect method, the effects will be saved automatically. But, if you apply effect from toolbar, each effect will be saved only when clicking the OK button, else all the applied effects will not be saved.

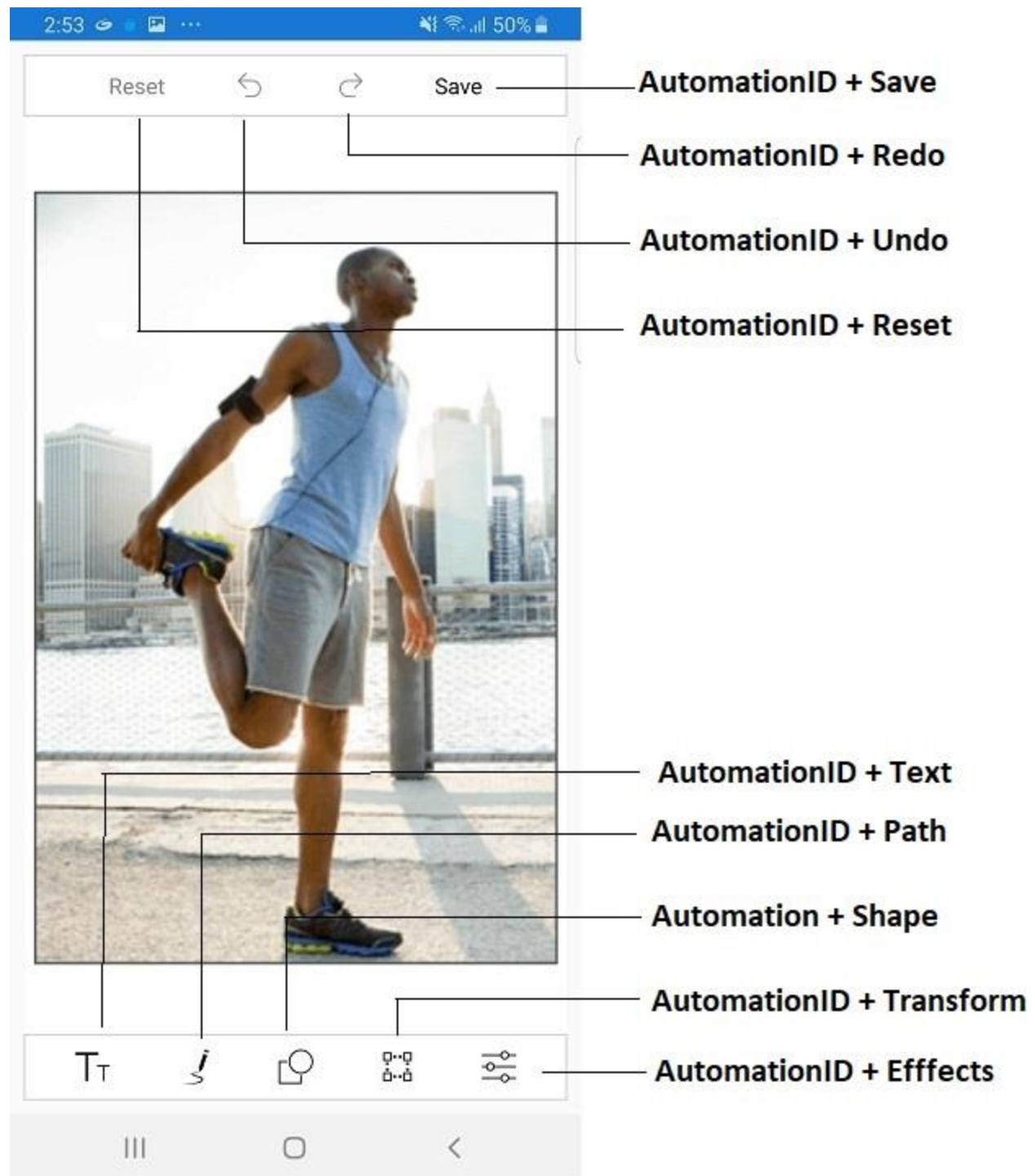
---

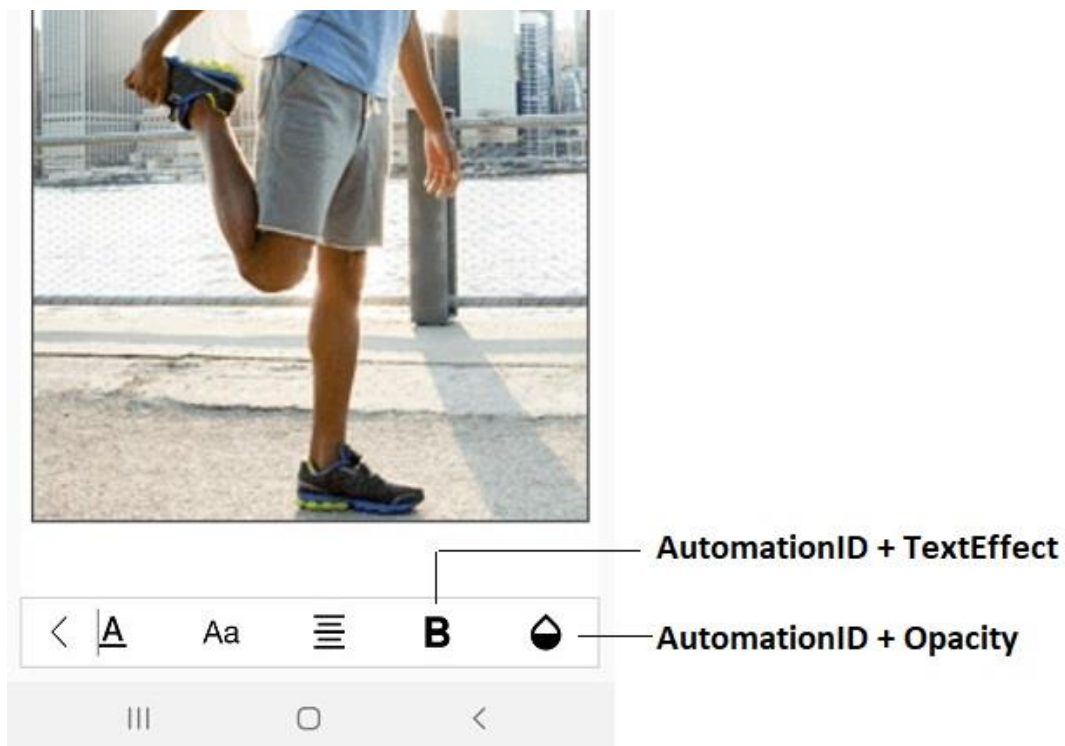


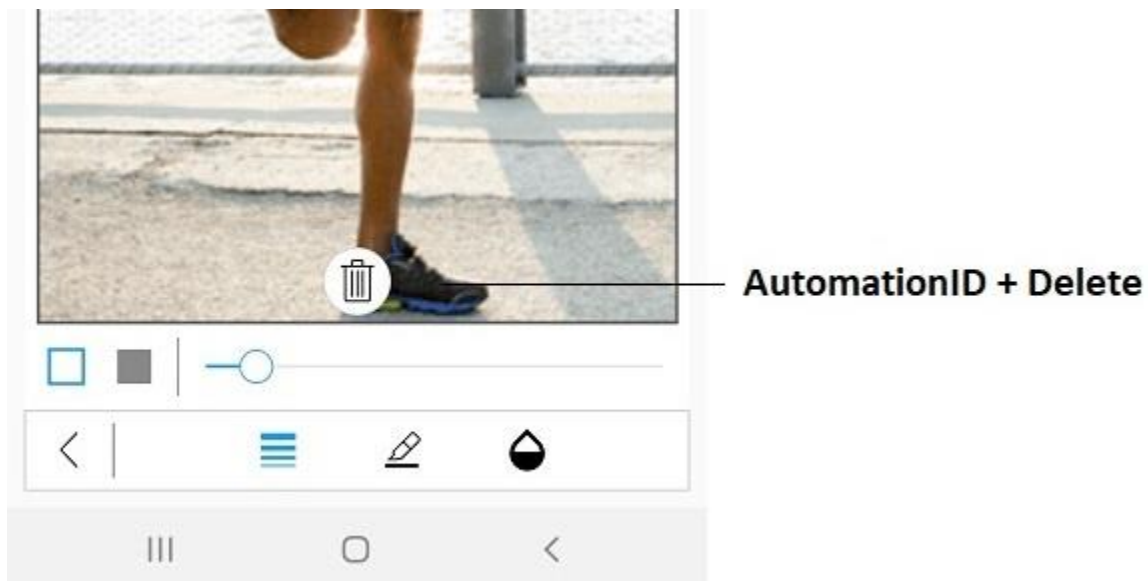
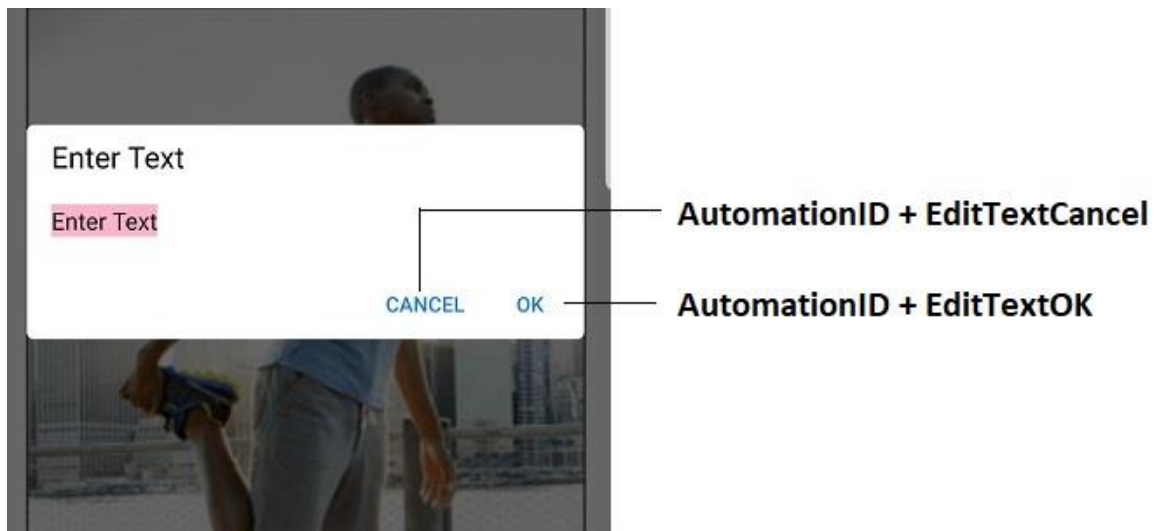


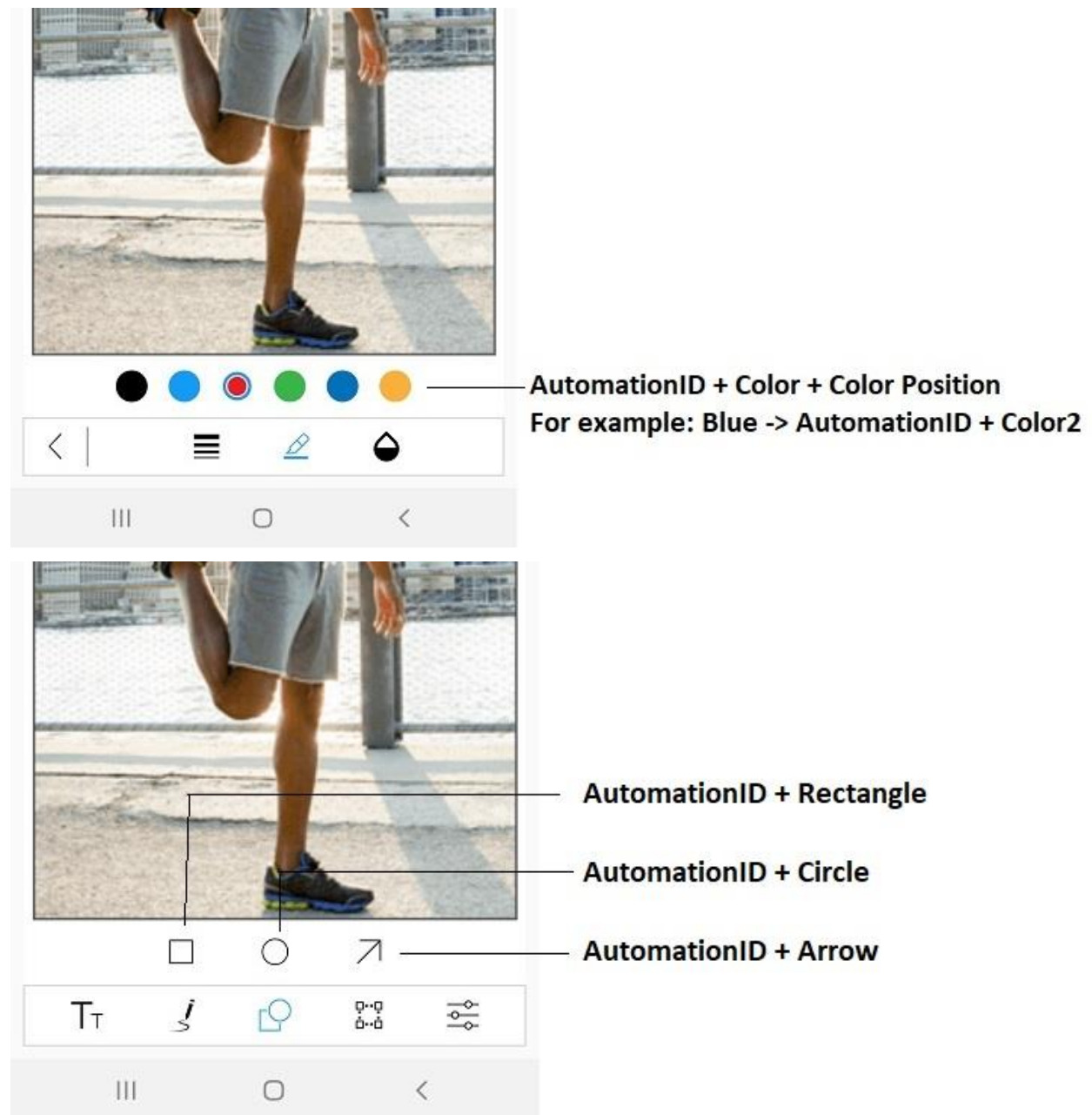
### Automation ID

The SfImageEditor control has built-in automation Id for inner elements. The `AutomationId` API allows the automation framework to find and interact with the inner elements of the SfImageEditor control. To keep unique AutomationId, these inner elements' AutomationIds are updated based on the control's `AutomationId`. For example, if you set automationId as `ImageEditor`, then the automation framework will interact with the Undo button as `ImageEditorUndo`. The following screenshot illustrates the AutomationIds for toolbar elements.

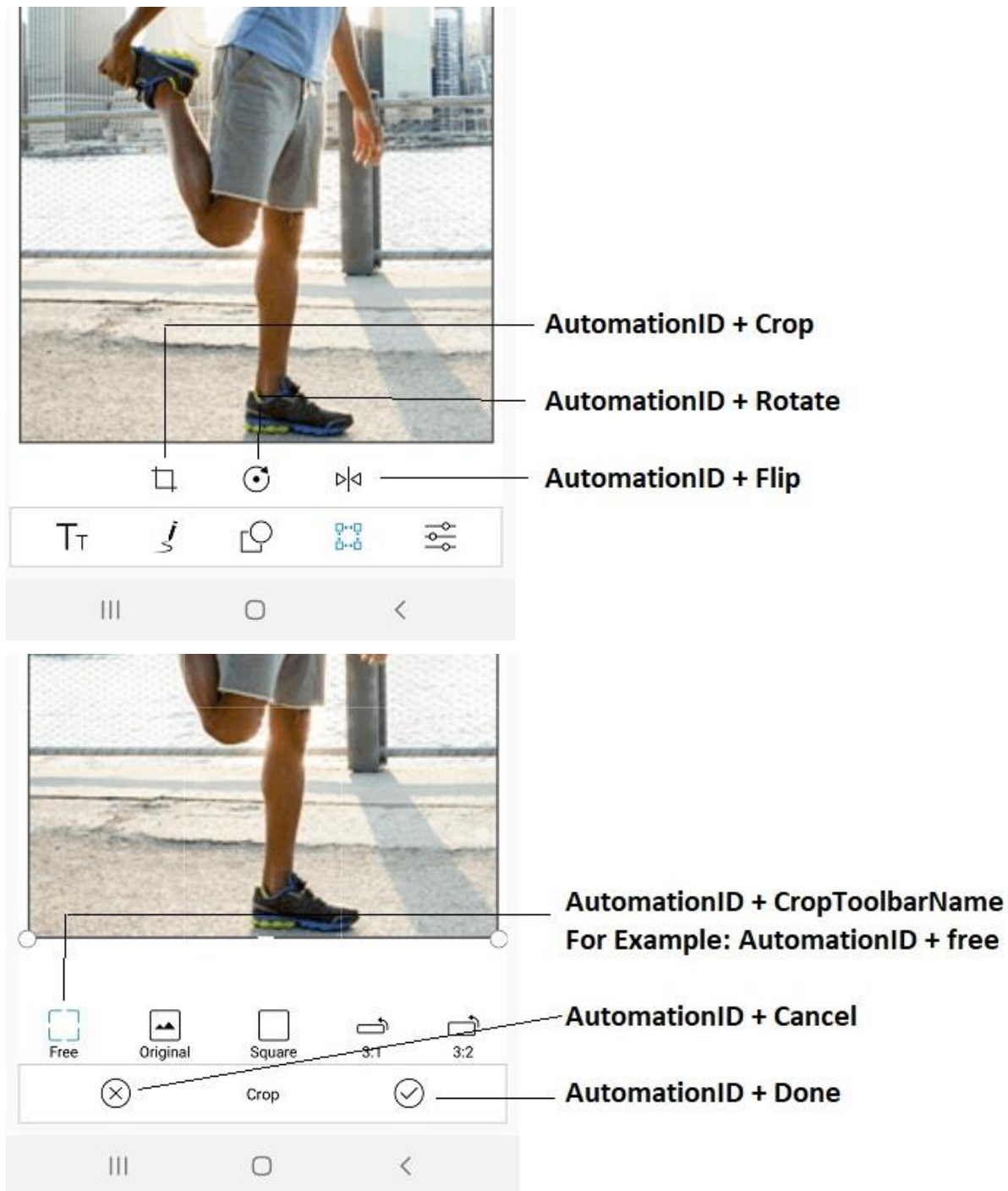












## SfKanban

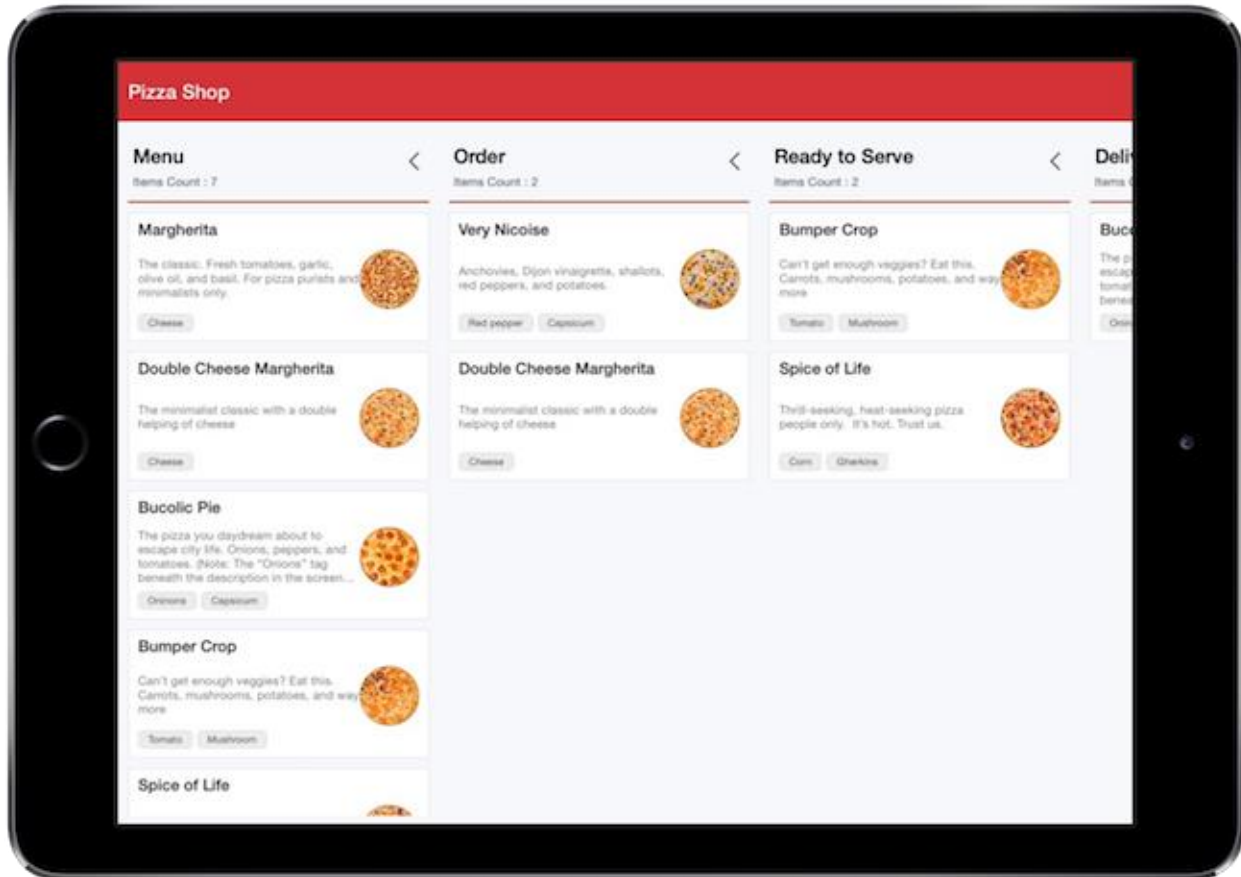
### Overview

The Kanban control is an efficient way to visualize a workflow at each stage of completion.

### Key features

- Visualize the workflow of any process.

- Limit a work in progress (WIP).
- Manage workflow transitions.
- Customize at a high level.
- Transition smoothly within processes.



## Getting Started

This section provides a quick overview for working with Essential Kanban for Xamarin.Forms. It is an efficient way to visualize the workflow at each stage along its path to completion.

### Adding SfKanban reference

You can add SfKanban reference in one of the following methods:

#### Method 1: Adding SfKanban reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfKanban). To add kanban to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfKanban](https://www.nuget.org/packages/Syncfusion.Xamarin.SfKanban), and then install it.

![Adding SfKanban reference from NuGet](SfKanban\_images/Adding SfKanban reference.png)

---

**Note:** Install the same version of kanban NuGet in all the projects.

---

#### Method 2: Adding SfKanban reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfKanban control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfKanban assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location : {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfKanban.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfKanban.Android.dll Syncfusion.SfKanban.XForms.Android.dll Syncfusion.SfKanban.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfKanban.iOS.dll Syncfusion.SfKanban.XForms.iOS.dll Syncfusion.SfKanban.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfKanban.UWP.dll Syncfusion.SfKanban.XForms.UWP.dll Syncfusion.SfKanban.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching an application on each platform with kanban

To use the kanban inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and are discussed in the following sections:

**Note:** If you are adding the references from toolbox, below steps are not needed.



*iOS*

To launch the kanban in iOS, call the `SfKanbanRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

**C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    SfKanbanRenderer.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

*Universal Windows Platform (UWP)*

To deploy the kanban in `Release` mode, initialize the kanban assemblies in the `App.xaml.cs` file in the UWP project as demonstrated in the following code samples.

**C#**

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    if (rootFrame == null)
    {
        List<Assembly> assembliesToInclude = new List<Assembly>();
        assembliesToInclude.Add(typeof(SfKanbanRenderer).GetTypeInfo().Assembly);
        Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    }
    ...
}
```

*Android*

Android platform does not require any additional configuration to render the kanban control.

## Initialize Kanban

Import '[SfKanban](#)' namespace as shown below in your respective page,

**XML**

```
xmlns:kanban="clr-namespace:Syncfusion.SfKanban.XForms;assembly=Syncfusion.SfKanban.XForms"
```

**C#**

```
using Syncfusion.SfKanban.XForms;
```

Create an instance of '[SfKanban](#)' control and set to Content property of a Page.

**XML**

```
<kanban:SfKanban>
```

```
</kanban:SfKanban>
```

### C#

```
SfKanban kanban = new SfKanban();  
this.Content = kanban;
```

### Initialize view model

Create a ViewModel class with a collection property to hold a collection of [KanbanModel](#) instances as shown below. Each [KanbanModel](#) instance represent a card in Kanban control.

### C#

```
public class ViewModel  
{  
    public ObservableCollection<KanbanModel> Cards { get; set; }  
    public ViewModel()  
    {  
        Cards = new ObservableCollection<KanbanModel>();  
        Cards.Add(new KanbanModel()  
        {  
            ID = 1,  
            Title = "iOS - 1002",  
            ImageURL = "Image1.png",  
            Category = "Open",  
            Description = "Analyze customer requirements",  
            ColorKey = "Red",  
            Tags = new string[] { "Incident", "Customer" }  
        });  
        Cards.Add(new KanbanModel()  
        {  
            ID = 6,  
            Title = "Xamarin - 4576",  
            ImageURL = "Image2.png",  
            Category = "Open",  
            Description = "Show the retrieved data from the server in grid control",  
            ColorKey = "Green",  
            Tags = new string[] { "Story", "Customer" }  
        });  
        Cards.Add(new KanbanModel()  
        {  
            ID = 13,  
            Title = "UWP - 13",  
            ImageURL = "Image4.png",  
            Category = "In Progress",  
            Description = "Add responsive support to application",  
            ColorKey = "Brown",  
            Tags = new string[] { "Story", "Customer" }  
        });  
        Cards.Add(new KanbanModel()  
        {  
            ID = 2543,  
            Title = "Xamarin_iOS - 2543",  
            Category = "Code Review",  
            ImageURL = "Image3.png",
```

```

Description = "Check login page validation",
ColorKey = "Brown",
Tags = new string[] { "Story", "Customer" }
});
}
}

```

Set the `ViewModel` instance as the `BindingContext` of your Page; this is done to bind properties of `ViewModel` to [SfKanban](#).

**Note:** Add namespace of `ViewModel` class in your XAML page if you prefer to set `BindingContext` in XAML.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="GettingStartedKanban.MainPage"
xmlns:kanban="clr-
namespace:Syncfusion.SfKanban.XForms;assembly=Syncfusion.SfKanban.XForms"
xmlns:local="clr-namespace:GettingStartedKanban">
<ContentPage.BindingContext>
<local:ViewModel>
</local:ViewModel>
</ContentPage.BindingContext>
</ContentPage>

```

### C#

```
this.BindingContext = new ViewModel();
```

Binding data to SfKanban

Bind the above data to [SfKanban](#) using [ItemsSource](#) property.

### XML

```

<kanban:SfKanban ItemsSource="{Binding Cards}">
</kanban:SfKanban>

```

### C#

```
kanban.SetBinding(SfKanban.ItemsSourceProperty, "Cards");
```

Defining columns

The columns are generated automatically based on the distinct values of '[KanbanModel.Category](#)' from '[ItemsSource](#)'. But, you can also define the columns by setting '[AutoGenerateColumns](#)' property to false and adding '[KanbanColumn](#)' instance to '[Columns](#)' property of '[SfKanban](#)'.

### XML

```

<kanban:SfKanban x:Name="kanban" AutoGenerateColumns="False">
<kanban:SfKanban.Columns>
<kanban:KanbanColumn x:Name="openColumn" Title="To Do" >
</kanban:KanbanColumn>

```

```
<kanban:KanbanColumn x:Name="progressColumn" Title="In Progress">
</kanban:KanbanColumn>
<kanban:KanbanColumn x:Name="codeColumn" Title="Code Review" >
</kanban:KanbanColumn>
<kanban:KanbanColumn x:Name="doneColumn" Title="Done" >
</kanban:KanbanColumn>
</kanban:SfKanban.Columns>
</kanban:SfKanban>
```

## C#

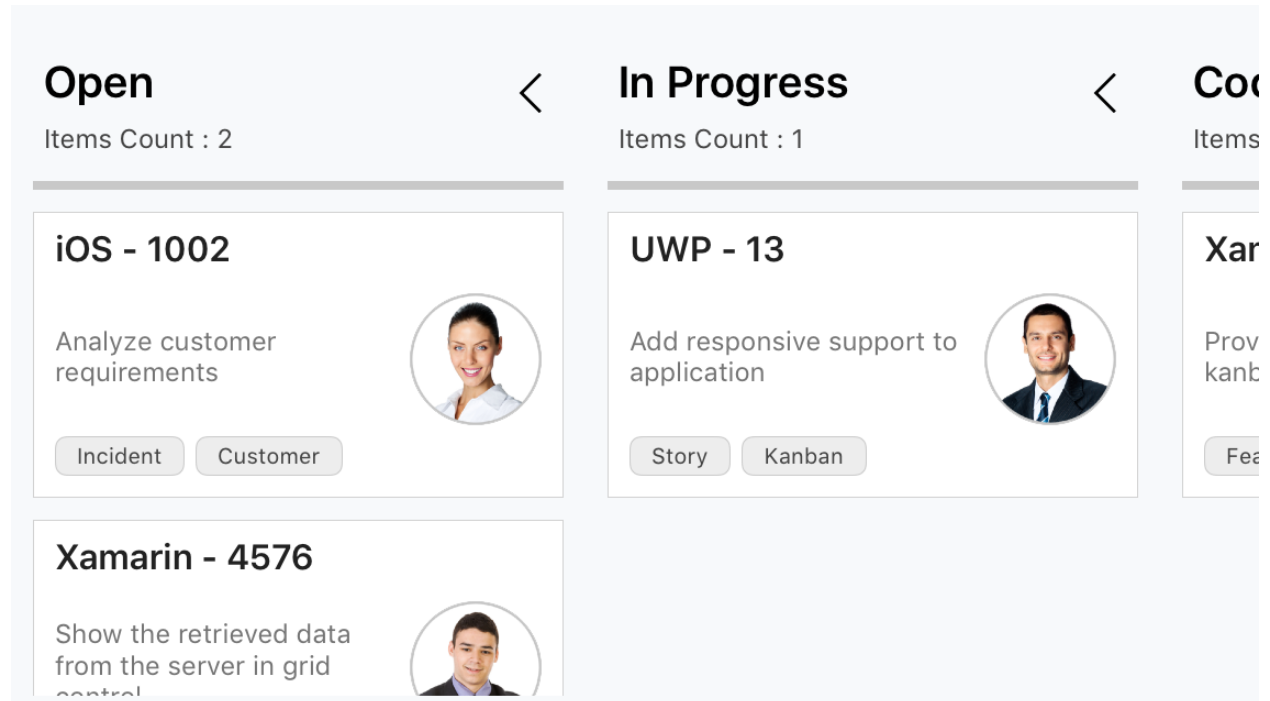
```
kanban.AutoGenerateColumns = false;
KanbanColumn openColumn = new KanbanColumn();
openColumn.Title = "To Do";
kanban.Columns.Add(openColumn);
KanbanColumn progressColumn = new KanbanColumn();
progressColumn.Title = "In Progress";
kanban.Columns.Add(progressColumn);
KanbanColumn codeColumn = new KanbanColumn();
codeColumn.Title = "Code Review";
kanban.Columns.Add(codeColumn);
KanbanColumn doneColumn = new KanbanColumn();
doneColumn.Title = "Done";
kanban.Columns.Add(doneColumn);
```

Define the categories of column using [Categories](#) property of [KanbanColumn](#) and cards will be added to the respective columns.

## C#

```
openColumn.Categories = new List<object>() { "Open" };
progressColumn.Categories = new List<object>() { "In Progress" };
codeColumn.Categories = new List<object>() { "Code Review" };
doneColumn.Categories = new List<object>() { "Done" };
```

This is how the final output will look like on iOS, Android and Windows devices. You can download the entire source code of this demo from [here](#).



## Column

### Customizing Column Size

By default, columns are sized smartly to arrange the default elements of the cards with better readability. However, you can define the minimum and maximum width for the columns in [SfKanban](#) using [SfKanban.MinimumColumnWidth](#) and [SfKanban.MaximumColumnWidth](#) properties respectively.

#### XML

```
<kanban:SfKanban MinimumColumnWidth ="300" MaximumColumnWidth ="340">
</kanban:SfKanban>
```

#### C#

```
kanban.MinimumColumnWidth = 300;
kanban.MaximumColumnWidth = 340;
```

You can also define the exact column width using [SfKanban.ColumnWidth](#) property.

#### XML

```
<kanban:SfKanban ColumnWidth ="250">
</kanban:SfKanban>
```

#### C#

```
kanban.ColumnWidth = 250;
```

### Categorizing Columns

If [ItemsSource](#) contains custom objects, the path of the property which can be used to categorize the card should be explicitly defined using [ColumnMappingPath](#) property. By default, [SfKanban](#) will automatically categorize the items using [KanbanModel.Category](#) property.

#### XML

```
<kanban:SfKanban ColumnMappingPath="Group">
</kanban:SfKanban>
```

#### C#

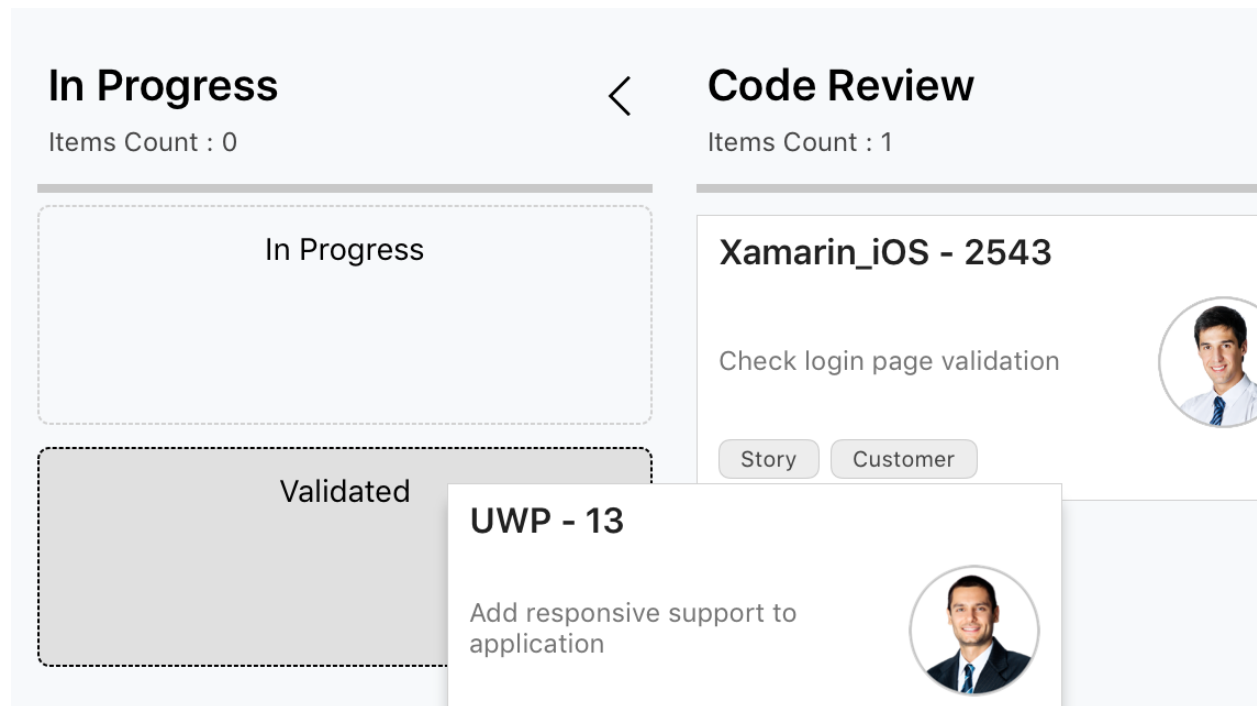
```
kanban.ColumnMappingPath = "Group";
```

### Multiple category for a column

More than one category can be mapped to a column by assigning multiple values to Categories collection of [KanbanColumn](#). For e.g., you can map “In progress, Validate types under “In progress” column.

#### C#

```
progressColumn.Categories = new List<object>() { "In Progress", "Validated"
};
```



### Headers

Header shows the category [Title](#), items count, min and max informations of a column. The UI of the header can be replaced entirely using [SfKanban.HeaderTemplate](#) property. The following code snippet and screenshot illustrates this.

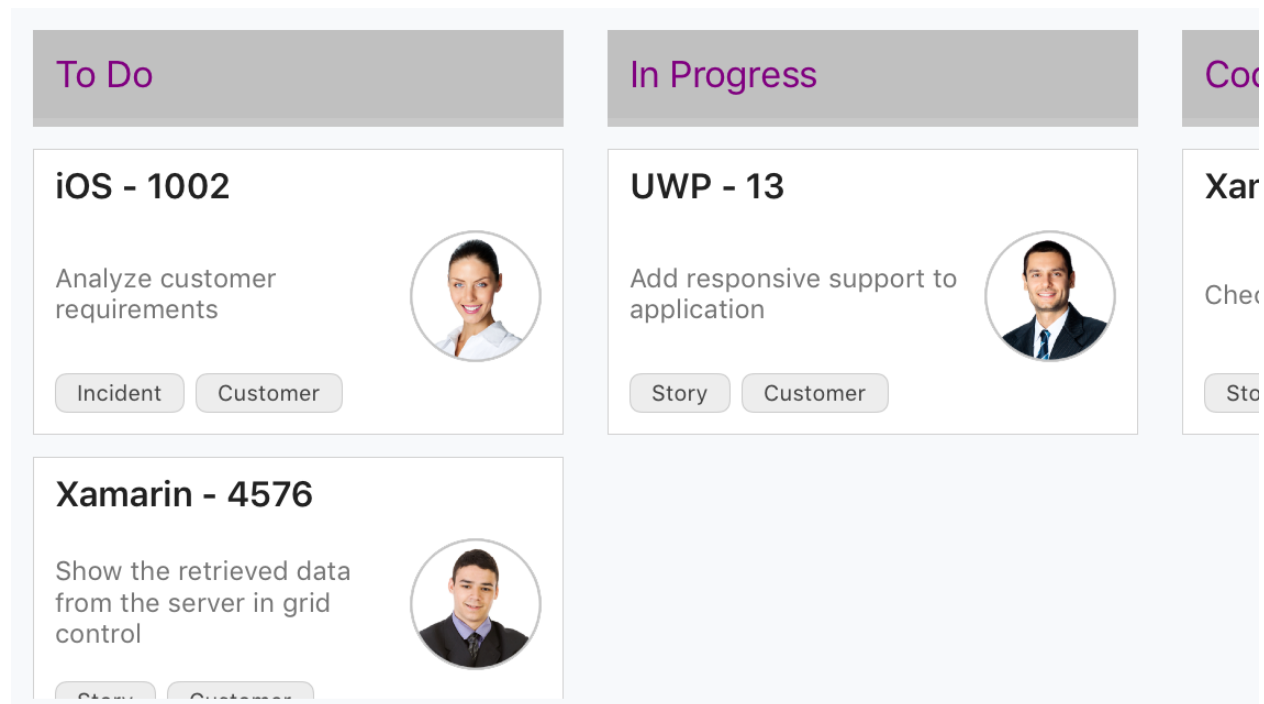
#### XML

```
<kanban:SfKanban.HeaderTemplate >
<DataTemplate>
<StackLayout WidthRequest="300" HeightRequest="40"
BackgroundColor="Silver">
<Label Margin="10" Text="{Binding Path=Title}" TextColor="Purple"
HorizontalOptions="Start" />
</StackLayout>
</DataTemplate>
</kanban:SfKanban.HeaderTemplate>
```

**C#**

```
var headerTemplate = new DataTemplate(() => {
    StackLayout root = new StackLayout() {
        WidthRequest = 300,
        HeightRequest = 40,
        BackgroundColor = Color.Silver
    };
    Label label = new Label();
    label.Margin = new Thickness(10);
    label.SetBinding(Label.TextProperty, new Binding("Title") );
    label.TextColor = Color.Purple;
    label.HorizontalOptions = LayoutOptions.Start;
    root.Children.Add(label);
    return root;
});
kanban.HeaderTemplate = headerTemplate;
```

The following output is displayed as a result of the above code example.



### Expand/Collapse Column

Columns can be expanded/collapsed by tapping the toggle button which is placed at top right corner of the Kanban header. [KanbanColumn.IsExpanded](#) ' property is used to programmatically expand/collapse the Kanban column. The following code example describes the above behavior.

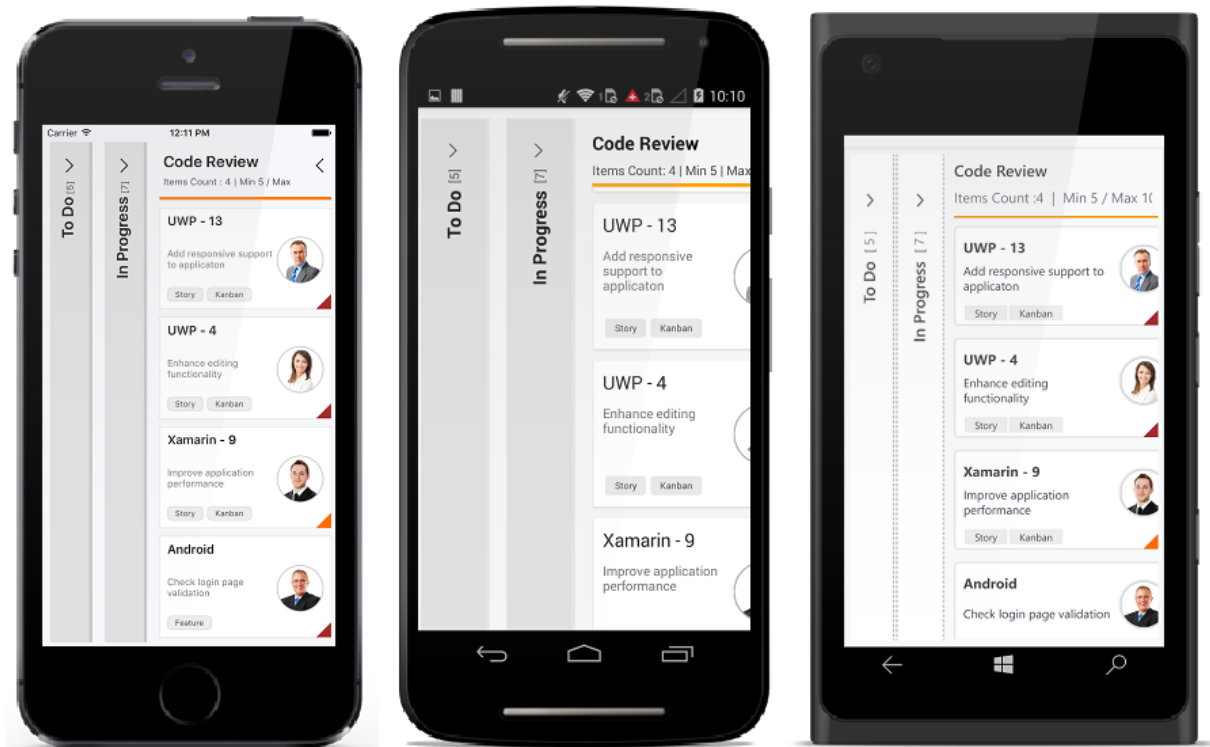
#### XML

```
<kanban:SfKanban.Columns>
<kanban:KanbanColumn x:Name="column1" Title="To Do" IsExpanded="false" />
<kanban:KanbanColumn x:Name="column2" Title="In Progress" IsExpanded="false" />
</kanban:SfKanban.Columns>
```

#### C#

```
KanbanColumn column1 = new KanbanColumn();
column1.IsExpanded = false;
KanbanColumn column2 = new KanbanColumn();
column2.IsExpanded = false;
kanban.Columns.Add(column1);
kanban.Columns.Add(column2);
```

The following output is displayed as a result of the above code example.



### Enable/Disable Drag & Drop

You can enable and disable the drag and drop operation of the cards for particular column using [KanbanColumn.AllowDrag](#) and [KanbanColumn.AllowDrop](#) properties.



The following code is used to disable the drag operation from progress column.

#### XML

```
<kanban:SfKanban.Columns>
<kanban:KanbanColumn AllowDrag="false"/>
</kanban:SfKanban.Columns>
```

#### C#

```
KanbanColumn progressColumn = new KanbanColumn();
progressColumn.AllowDrag = false;
```

The following code is used to disable the drop operation of the cards into the progress column.

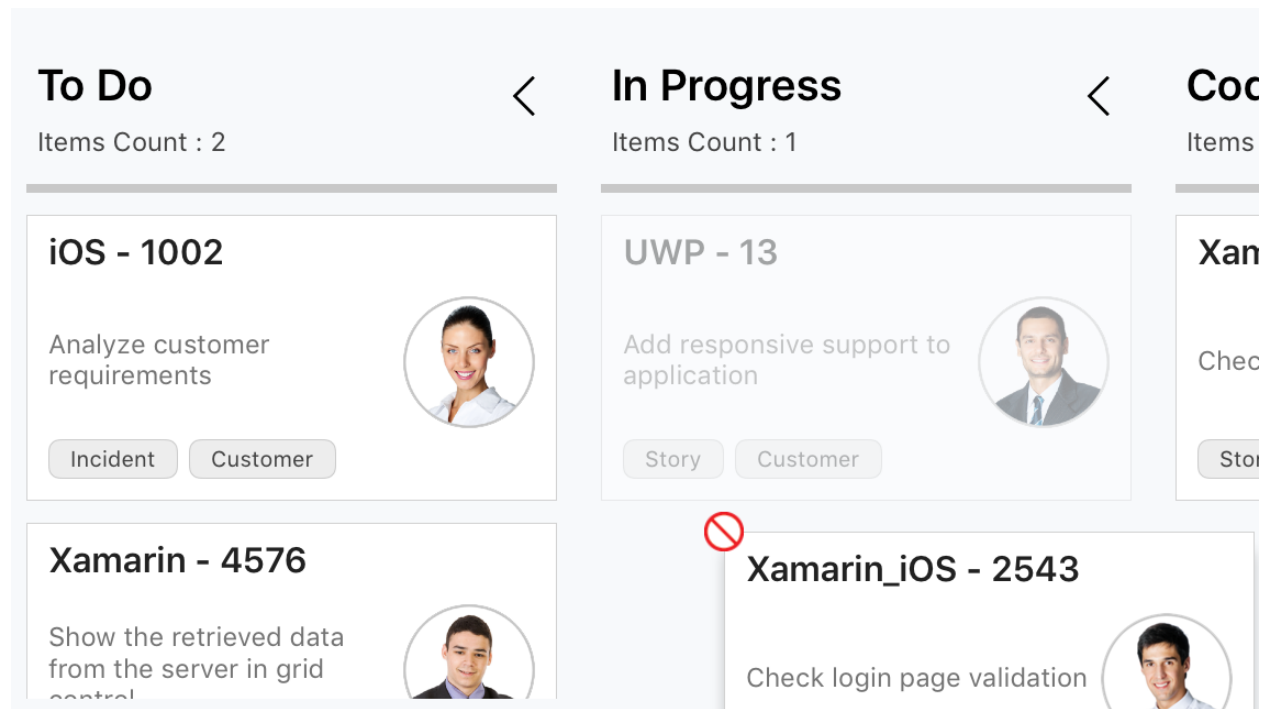
#### XML

```
<kanban:SfKanban.Columns>
<kanban:KanbanColumn AllowDrop="false"/>
</kanban:SfKanban.Columns>
```

#### C#

```
KanbanColumn progressColumn = new KanbanColumn();
progressColumn.AllowDrop = false;
```

The following output demonstrates the above example code.



Items Count

[ItemsCount](#) property is used to get the total cards count in each column.

**C#**

```
int count = todoColumn.ItemsCount;
```

## Work In-Progress Limit

[MinimumLimit](#) and [MaximumLimit](#) properties are used to define the minimum and maximum number of items in a column. If the actual items count is exceeded or lesser than the specified limits, the error bars are used to indicate this violation. Following properties of [ErrorbarSettings](#) are used to customize the appearance of error bar.

- [Color](#) - used to change the default color of the error bar.
- [MaxValidationColor](#) - used to change the maximum validation color of the error bar.
- [MinValidationColor](#) - used to change the minimum validation color of the error bar.
- [Height](#) - used to change the height of the error bar.

**XML**

```
<kanban:KanbanColumn x:Name="todoColumn" Title="To Do" MinimumLimit="2"
MaximumLimit="1">
</kanban:KanbanColumn>
```

**C#**

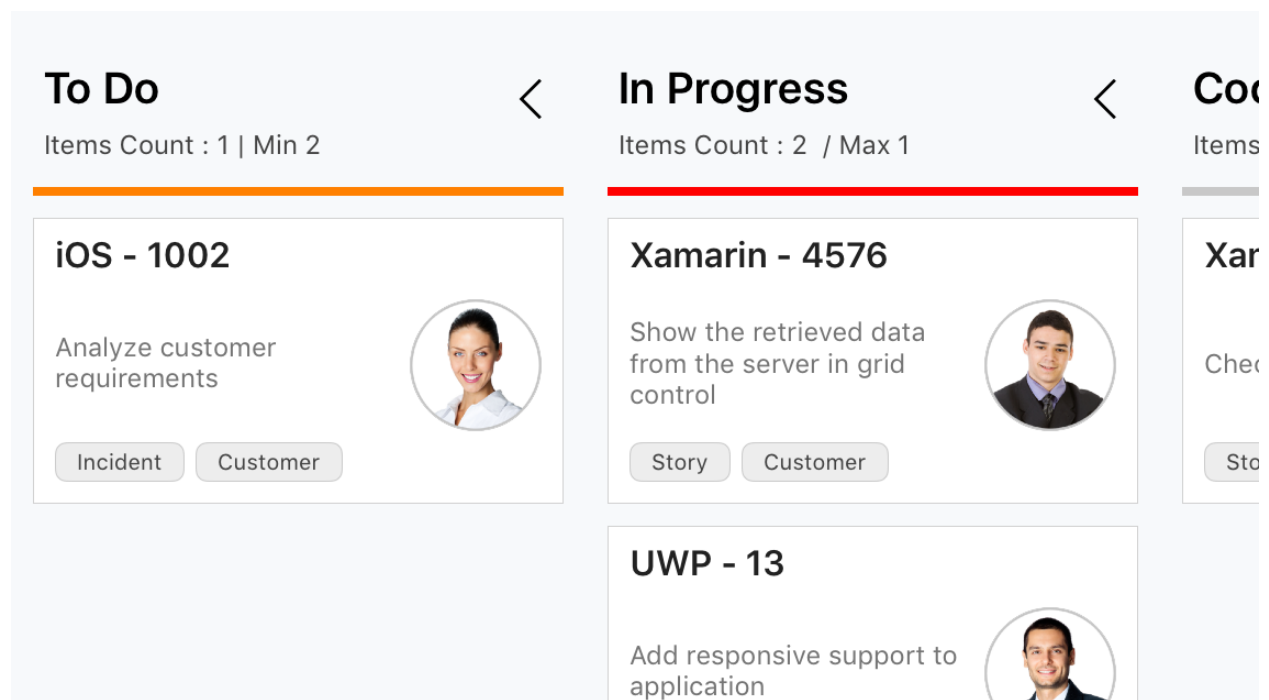
```
todoColumn.MinimumLimit = 2;
todoColumn.MaximumLimit = 1;
```

**XML**

```
<kanban:KanbanColumn x:Name="todoColumn" Title="To Do" MinimumLimit="3"
MaximumLimit="5">
<kanban:KanbanColumn.ErrorbarSettings>
<kanban:KanbanErrorBarSettings Color="Green" MinValidationColor="Orange"
MaxValidationColor="Red" Height="4"/>
</kanban:KanbanColumn.ErrorbarSettings>
</kanban:KanbanColumn>
```

**C#**

```
todoColumn.ErrorbarSettings.Color = Color.Green;
todoColumn.ErrorbarSettings.MinValidationColor = Color.Orange;
todoColumn.ErrorbarSettings.MaxValidationColor = Color.Red;
todoColumn.ErrorbarSettings.Height = 4;
```



## Workflows

This feature is used to define the flow of the card transitions from one state to another state. You need to create an instance of [KanbanWorkflow](#) class and add it to [SfKanban.Workflows](#) property to define the workflow for each column. The [KanbanWorkflow](#) contains the following properties to define the source category and target categories.

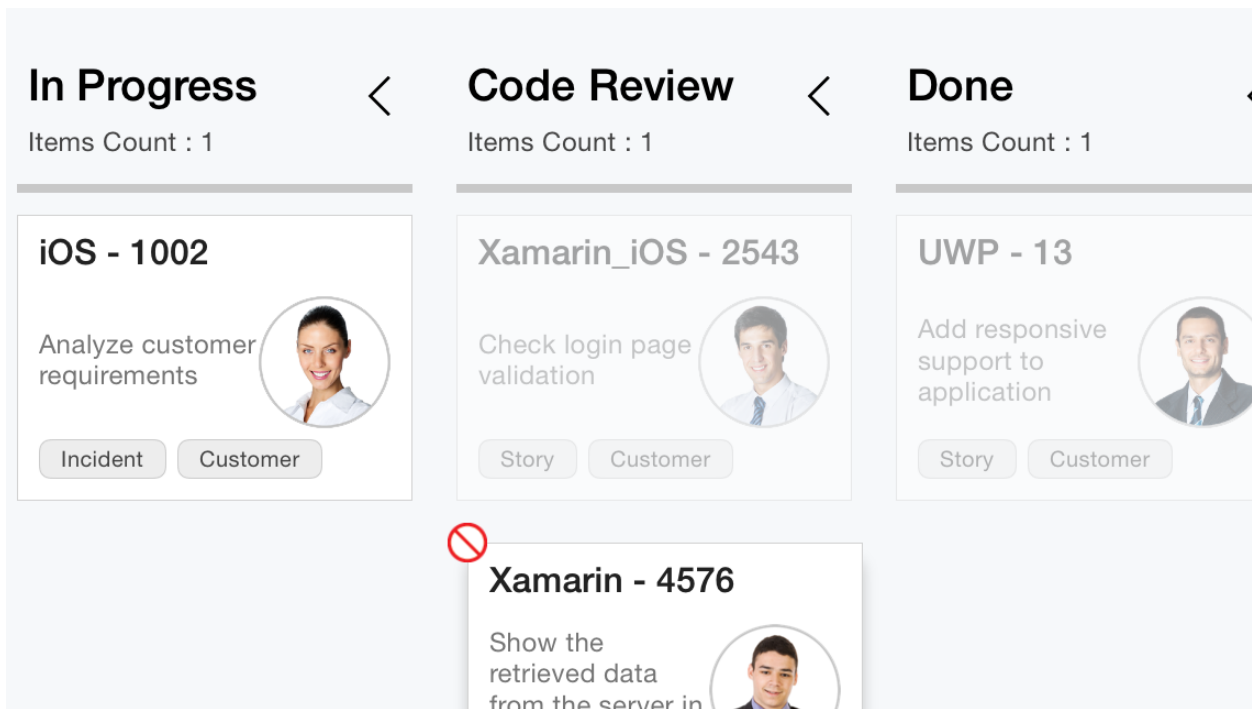
- [Category](#) - Used to define the source category/state.
- [AllowedTransitions](#) - Used to define the list of categories/states, the card with the state specified in [KanbanWorkflow.Category](#) is allowed to transit.

The following code example describes the workflow functionality.

### C#

```
var workflows = new List<KanbanWorkflow>();
var openWorkflow = new KanbanWorkflow();
openWorkflow.Category = "Open";
//Define the list of categories which accepts the cards from Open state.
openWorkflow.AllowedTransitions = new List<object> { "In Progress" };
workflows.Add(openWorkflow);
var progressWorkflow = new KanbanWorkflow();
progressWorkflow.Category = "In Progress";
//Define the list of categories which accepts the cards from "In Progress"
state.
progressWorkflow.AllowedTransitions = new List<object> { "Open", "Code
Review", "Closed-No Code Changes" };
workflows.Add(progressWorkflow);
kanban.Workflows = workflows;
```

In the below output, you can see the card which was picked from Open state is not allowed to drop on "Code Review" and "Done" state, because we have defined to move the card from Open to "In Progress" state only and not to any other states.



## Cards

The default elements of a card can be customized using the below properties of [KanbanModel](#).

- [Title](#) - Used to set the title of a card.
- [ImageURL](#) - Used to set the image URL of a card. The image will be displayed at right side in default card template.
- [Category](#) - Used to set the category of a card. Based on the category the cards will be added to the respective columns.
- [Description](#) - Used to set the description text of a card.
- [ColorKey](#) - Used to specify the indicator color key. The [Color](#) value of the corresponding [Key](#) should be added in [ColorModel](#) collection of [SfKanban](#).
- [Tags](#) - Used to specify the tags of a card. The tags will be displayed at bottom in default card template.
- [ID](#) - Used to set the ID of a card.

## C#

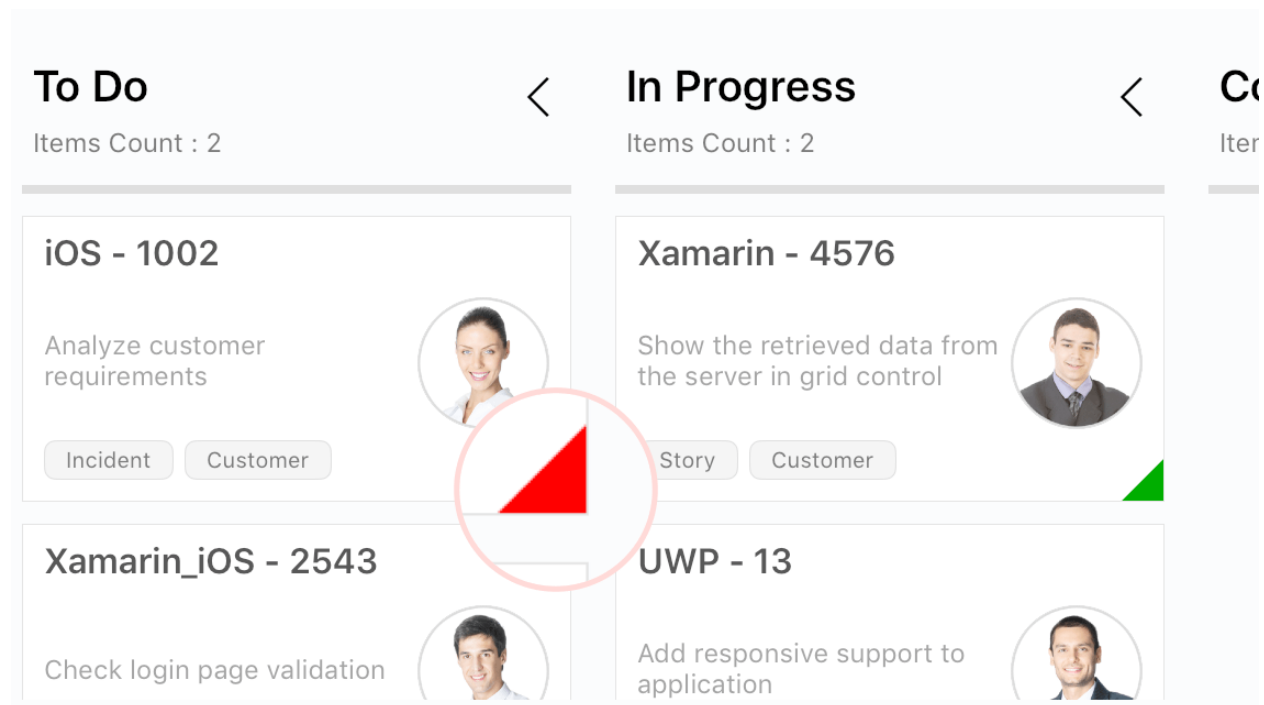
```
new KanbanModel ()
{
    ID = 1,
    Title = "iOS - 1002",
    ImageURL = "Image1.png",
    Category = "Open",
    Description = "Analyze customer requirements",
    ColorKey = "Red",
    Tags = new string[] { "Incident", "Customer" }
```

```
});
```

Following code snippet is used to define the colors for each key.

### C#

```
List<KanbanColorMapping> colorModels = new List<KanbanColorMapping>();
colorModels.Add(new KanbanColorMapping("Green", Color.Green));
colorModels.Add(new KanbanColorMapping("Red", Color.Red));
colorModels.Add(new KanbanColorMapping("Aqua", Color.Aqua));
colorModels.Add(new KanbanColorMapping("Blue", Color.Blue));
kanban.ColorModel = colorModels;
```



### Template

You can replace the entire card template with your own design using [SfKanban.CardTemplate](#) property. The following code snippet and screenshot illustrates this.

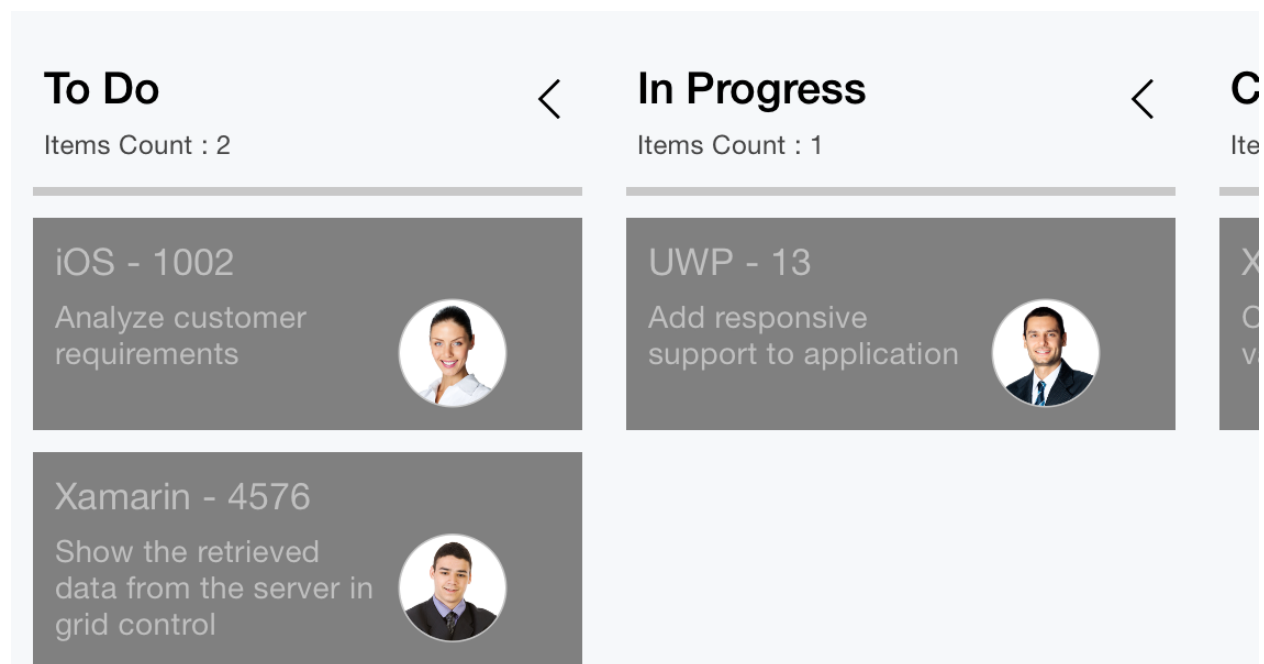
### XML

```
<kanban:SfKanban.CardTemplate >
<DataTemplate>
<StackLayout WidthRequest="250" Orientation="Vertical"
BackgroundColor="Gray" Padding="10,10,10,10">
<StackLayout Orientation="Horizontal">
<Label Text="{Binding Path=Title}" TextColor="Silver"
HorizontalOptions="StartAndExpand" >
</Label>
</StackLayout>
<StackLayout Orientation="Horizontal">
<Label Text="{Binding Description}" WidthRequest="150" FontSize="14"
TextColor="Silver" LineBreakMode="WordWrap" ></Label>
```

```
<Image Source="{Binding ImageURL}" HeightRequest="50" WidthRequest="50"
></Image>
</StackLayout>
</StackLayout>
</DataTemplate>
</kanban:SfKanban.CardTemplate>
```

## C#

```
var cardTemplate = new DataTemplate(() =>
{
    StackLayout root = new StackLayout()
    {
        WidthRequest = 250,
        Orientation = StackOrientation.Vertical,
        Padding = new Thickness(10),
        BackgroundColor = Color.Gray
    };
    StackLayout titleLayout = new StackLayout();
    Label title = new Label()
    {
        TextColor = Color.Silver,
        HorizontalOptions = LayoutOptions.StartAndExpand
    };
    title.SetBinding(Label.TextProperty, new Binding("Title"));
    titleLayout.Children.Add(title);
    StackLayout contentLayout = new StackLayout()
    {
        Orientation = StackOrientation.Horizontal
    };
    Label desc = new Label()
    {
        WidthRequest = 150,
        FontSize = 14,
        TextColor = Color.Silver,
        LineBreakMode = LineBreakMode.WordWrap
    };
    desc.SetBinding(Label.TextProperty, new Binding("Description"));
    Image image = new Image()
    {
        HeightRequest = 50,
        WidthRequest = 50
    };
    image.SetBinding(Image.SourceProperty, new Binding("ImageURL"));
    contentLayout.Children.Add(desc);
    contentLayout.Children.Add(image);
    root.Children.Add(titleLayout);
    root.Children.Add(contentLayout);
    return root;
});
kanban.CardTemplate = cardTemplate;
```



### Data template selector

You can customize the appearance of each card with different templates based on specific constraints using [DataTemplateSelector](#).

#### Create a data template selector

Create a custom class by inheriting `DataTemplateSelector`, and override the `OnSelectTemplate` method to return the `DataTemplate` for that item. At runtime, the SfKanban invokes the `OnSelectTemplate` method for each item and passes the data object as parameter.

#### C#

```
public class KanbanTemplateSelector : DataTemplateSelector
{
    private readonly DataTemplate menuTemplate;
    private readonly DataTemplate orderTemplate;
    private readonly DataTemplate readyToServeTemplate;
    private readonly DataTemplate deliveryTemplate;
    public KanbanTemplateSelector()
    {
        menuTemplate = new DataTemplate(typeof(MenuTemplate));
        orderTemplate = new DataTemplate(typeof(OrderTemplate));
        readyToServeTemplate = new DataTemplate(typeof(ReadyToServeTemplate));
        deliveryTemplate = new DataTemplate(typeof(DeliveryTemplate));
    }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject container)
    {
        var data = item as CustomKanbanModel;
        if (data == null)
            return null;
        string category = data.Category?.ToString();
        return category.Equals("Menu") ? menuTemplate :
```

```
category.Equals("Dining") || category.Equals("Delivery") ? orderTemplate :
category.Equals("Ready to Serve") ? readyToServeTemplate : deliveryTemplate;
}
}
```

### Applying the data template selector

Assign custom **DataTemplateSelector** to the [CardTemplate](#) of the SfKanban in either XAML or C#.

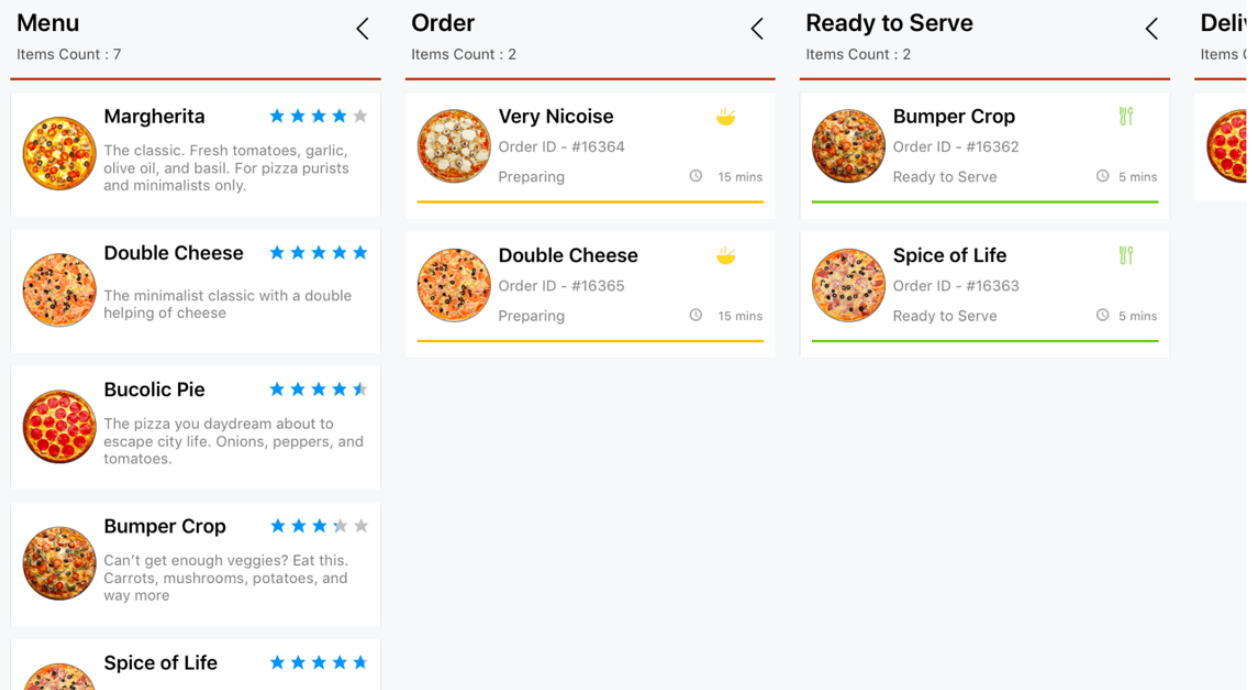
#### **XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="SimpleSample.MainPage"
xmlns:kanban="clr-
namespace:Syncfusion.SfKanban.XForms;assembly=Syncfusion.SfKanban.XForms"
xmlns:local="clr-namespace:SimpleSample;assembly=SimpleSample">
<ContentPage.Resources>
<ResourceDictionary>
<local:KanbanTemplateSelector x:Key="kanbanTemplateSelector" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.BindingContext>
<local:KanbanCustomViewModel />
</ContentPage.BindingContext>
<kanban:SfKanban x:Name="kanban" HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" ItemsSource="{Binding Cards}"
CardTemplate="{StaticResource kanbanTemplateSelector}" >
...
</kanban:SfKanban>
</ContentPage>
```

#### **C#**

```
SfKanban kanban = new SfKanban();
kanban.ItemsSource = viewModel.Cards;
kanban.CardTemplate = new KanbanTemplateSelector();
```





## Placeholder

The placeholder is to denote a card's new position in the [KanbanColumn](#). It will appear while dragging a card over the column.

### Placeholder style

[PlaceholderStyle](#) property is used to customize the placeholder. Following [KanbanPlaceholderStyle](#) properties are used to customize its appearance.

- [BackgroundColor](#) - This property is used to change the background color of the placeholder.
- [BorderColor](#) - This property is used to change the border color of the placeholder.
- [BorderThickness](#) - This property is used to change the border width of the placeholder.
- [StrokeDashArray](#) - This property is used to change the dashes of the placeholder border.
- [FontSize](#) - This is used to change the text size of the placeholder.
- [TextColor](#) - This property is used to change the text color of the placeholder.

Following properties are used to customize the selected category when you have more than one category in a column.

- [SelectedBackgroundColor](#) - This property is used to change the background color of the selected placeholder.
- [SelectedBorderColor](#) - This property is used to change the border color of the selected placeholder.
- [SelectedBorderThickness](#) - This property is used to change the border width of the selected placeholder.
- [SelectedStrokeDashArray](#) - This property is used to change the dashes of the selected placeholder.
- [SelectedFontSize](#) - This is used to change the font size of the text in selected placeholder.
- [SelectedTextColor](#) - This property is used to change the color of the text in selected placeholder.

The following code example describes the above behavior.

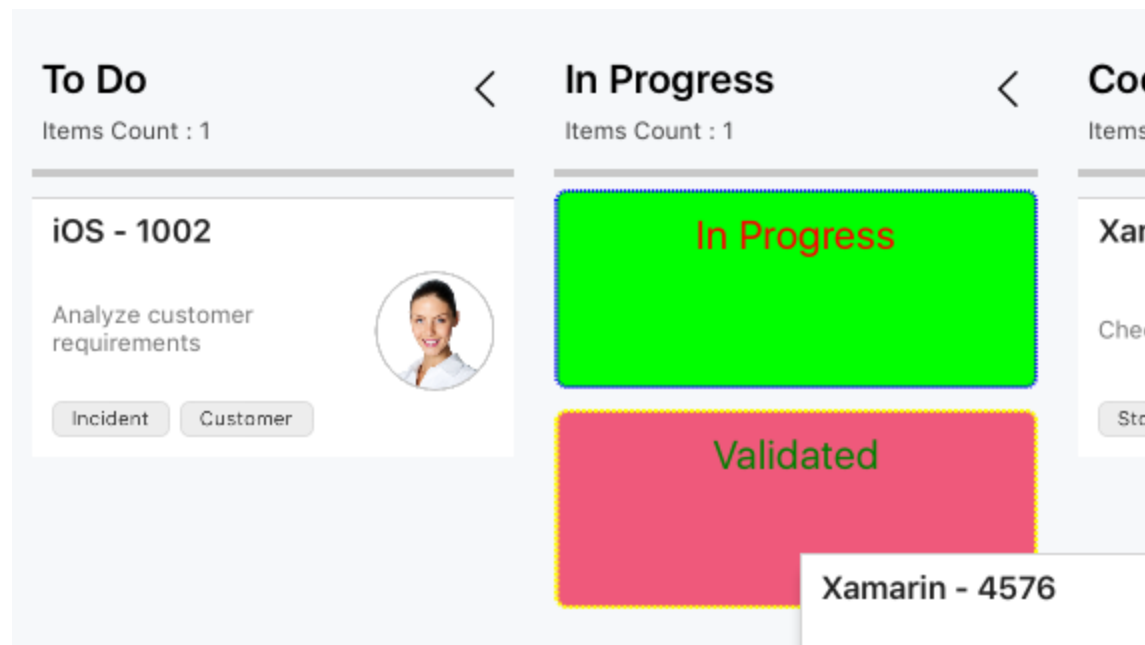
### XML

```
<kanban:SfKanban.PlaceholderStyle >  
<kanban:KanbanPlaceholderStyle FontSize="16"  
TextColor="Green"  
BackgroundColor="Fuchsia"  
BorderColor="Fuchsia"  
BorderThickness="2"  
SelectedFontSize="16"  
SelectedTextColor="Red"  
SelectedBorderColor="Yellow"  
SelectedBorderThickness="2"  
SelectedBackgroundColor="Green">  
</kanban:KanbanPlaceholderStyle>  
</kanban:SfKanban.PlaceholderStyle>
```

### C#

```
KanbanPlaceholderStyle style = new KanbanPlaceholderStyle();  
style.FontSize = 20;  
style.TextColor = Color.Green;  
style.BackgroundColor = Color.FromRgb(239, 89, 123);  
style.BorderColor = Color.Blue;  
style.BorderThickness = 2;  
style.StrokeDashArray = new double[] { 1, 1 };  
style.SelectedFontSize = 20;  
style.SelectedTextColor = Color.Red;  
style.SelectedBorderColor = Color.Yellow;  
style.SelectedBackgroundColor = Color.FromRgb(0, 255, 0);  
style.SelectedBorderThickness = 2;  
style.SelectedStrokeDashArray = new double[] { 2, 1 };  
kanban.PlaceholderStyle = style;
```

The following output demonstrates the above code example.



## Localization

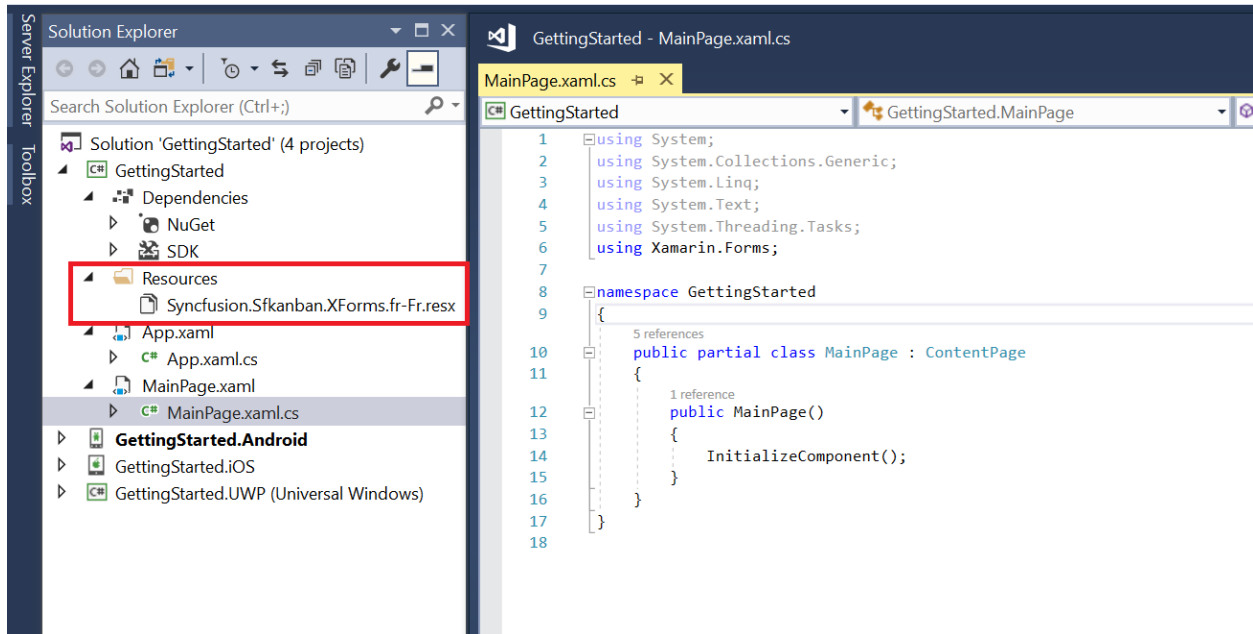
You can localize [SfKanban](#) in all the platforms by adding a .resx file in a .NET Standard project alone. The following steps describe how to localize SfKanban in a project and you can download the complete sample from this [link](#).

---

**Note:** Here, the resources have been already created for some cultures and shared them on [Syncfusion GitHub](#) for your convenience.

---

1. Add a new folder in the .NET Standard project named Resources.
2. Add resource files for the languages you wish to support, and set their Build Action to EmbeddedResource. The name of the resource file should be \$name of the Syncfusion component\$+\$language code\$.resx. For example, if you add a resource file for the French culture, add the Syncfusion.SfKanban.XForms.fr-FR.resx file to Resources folder as illustrated in the following screenshot.



Provide the French values for each key in the respective .resx files. Here, “ItemCount” and “Max” are the keys, and “Fritems” and “frmax” are their respective French values.

### XML

```
<data name="ItemCount" xml:space="preserve">
  <value>Fritems</value>
</data>
<data name="Max" xml:space="preserve">
  <value>frmax</value>
</data>
```

4. Set the resource manager to [‘KanbanResourceManager.Manager’](#) as demonstrated in the following code to get the resource manager from the users. For more details, refer [Localization](#).

### C#

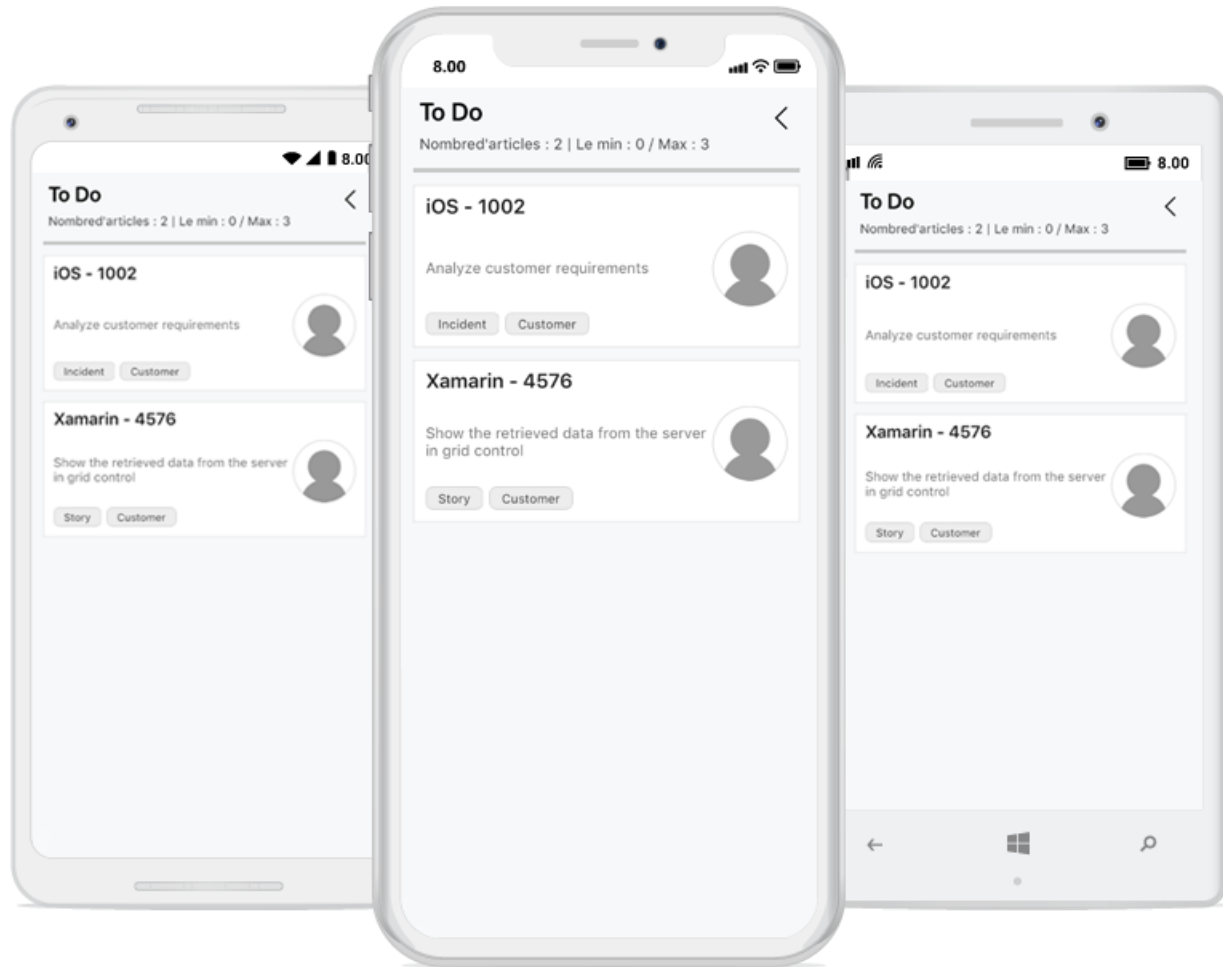
```
KanbanResourceManager.Manager = new
ResourceManager("GettingStarted.Resources.Syncfusion.SfKanban.XForms",
Application.Current.GetType().Assembly);
```

### Localize at application level

You can also localize the text at application-level regardless of the language selected on the device. The following platform-specific codes are needed to localize the text at application-level. Use the [DependencyServices](#) to set this from .NET Standard project.

### C#

```
//For Android and iOS,
Thread.CurrentThread.CurrentUICulture = new CultureInfo("fr-FR");
//For UWP,
CultureInfo.CurrentUICulture = new CultureInfo("fr-FR");
```



## Events

### ItemTapped

[ItemTapped](#) event is triggered when you tap on any card. The argument contains the following information.

- [Column](#) - Used to get the column of the card.
- [Data](#) - Used to get the underlying model of the card.
- [Index](#) - Used to get the index of the card in a column.

### Command

The `CardTappedCommand` property is used to associate a command with an instance of SfKanban. This property is most often set with MVVM pattern to bind callbacks back into the ViewModel.

### CommandParameter

The `CardTappedCommandParameter` property is used to set the parameter reference, based on which the event argument is shown.

---

## NOTE

The default value of the `CardTappedCommandParameter` is `null`.

## XML

```
<kanban:SfKanban CardTappedCommand="{Binding CardTappedCommand}"
CardTappedCommandParameter="1">
  <!--Initialize the column-->
</kanban:SfKanban >
```

## C#

```
public class ViewModel
{
    public ViewModel()
    {
        CardTappedCommand = new Command<object>(CardTappedEvent);
        public ICommand CardTappedCommand { get; set; }
    }
    private void CardTappedEvent(object args)
    {
        // handle event action.
    }
}
```

### DragStart

[DragStart](#) event is triggered when you start to drag a card. The argument contains the following information.

- [Cancel](#) - Used to cancel the drag action.
- [Data](#) - Used to get the underlying model of the card.
- [KeepItem](#) - Determines whether to keep the dragged card in the source location itself, until it is dropped in a new location. When it is true, the preview of the card will be created for dragging.
- [SourceColumn](#) - Used to get the source column of card.
- [SourceIndex](#) - Used to get the index of the card in source column.

### DragEnd

[DragEnd](#) event is triggered whenever the card is dropped or dragging action is canceled. The argument contains the following information.

- [Cancel](#) - Used to cancel the drag action.
- [Data](#) - Used to get the underlying model of the card.
- [SourceColumn](#) - Used to get the source column of the card.
- [SourceIndex](#) - Used to get the index of the card in source column.
- [TargetCategory](#) - Used to get the category of the column where the card is going to be dropped.
- [TargetColumn](#) - Used to get the current column which is the drop target for the card.
- [TargetIndex](#) - Used to get the index of the card in target column.

### DragEnter

[DragEnter](#) event is triggered when a card enters into a column while dragging. The argument contains the following information.

- [Cancel](#) - Used to cancel the drag action.

- [Data](#) - Used to get the underlying model of the card.
- [IsAboveMaximumLimit](#) - Used to know whether the total cards count of the target column will be above the maximum limit if you drop the card in target column. You can define the maximum limit of the cards using `KanbanColumn.MaximumLimit`.
- [SourceColumn](#) - Used to get the source column of the card.
- [SourceIndex](#) - Used to get the index of the card in source column.
- [TargetColumn](#) - Used to get the column upon which the card enters.
- [TargetIndex](#) - Used to get the index of the card in target column.

### DragLeave

[DragLeave](#) event is triggered when a card leaves a column while dragging. The argument contains the following information.

- [Data](#) - Used to get the underlying model of the card.
- [IsBelowMinimumLimit](#) - Used to know whether the total cards count of the target column will be below the minimum limit if you remove the card from target column. You can define the minimum limit of the cards using `KanbanColumn.MinimumLimit`.
- [SourceColumn](#) - Used to get the source column of the card.
- [SourceIndex](#) - Used to get the index of the card in source column.
- [TargetColumn](#) - Used to get the column from which the card leaves.

### DragOver

[DragOver](#) event is triggered when a card is dragged to a new index within a column. The argument contains the following information.

- [Cancel](#) - Used to cancel the drag action.
- [Data](#) - Used to get the underlying model of the card.
- [SourceColumn](#) - Used to get the source column of the card.
- [SourceIndex](#) - Used to get the index of the card in source column.
- [TargetColumn](#) - Used to get the current column which is the drop target for the card.
- [TargetIndex](#) - Used to get the new index of the card in target column.

### ColumnsGenerated

[ColumnsGenerated](#) event will be fired after the columns are generated automatically. You can access the auto-generated columns using [SfKanban.ActualColumns](#) property.

## SfLinearGauge

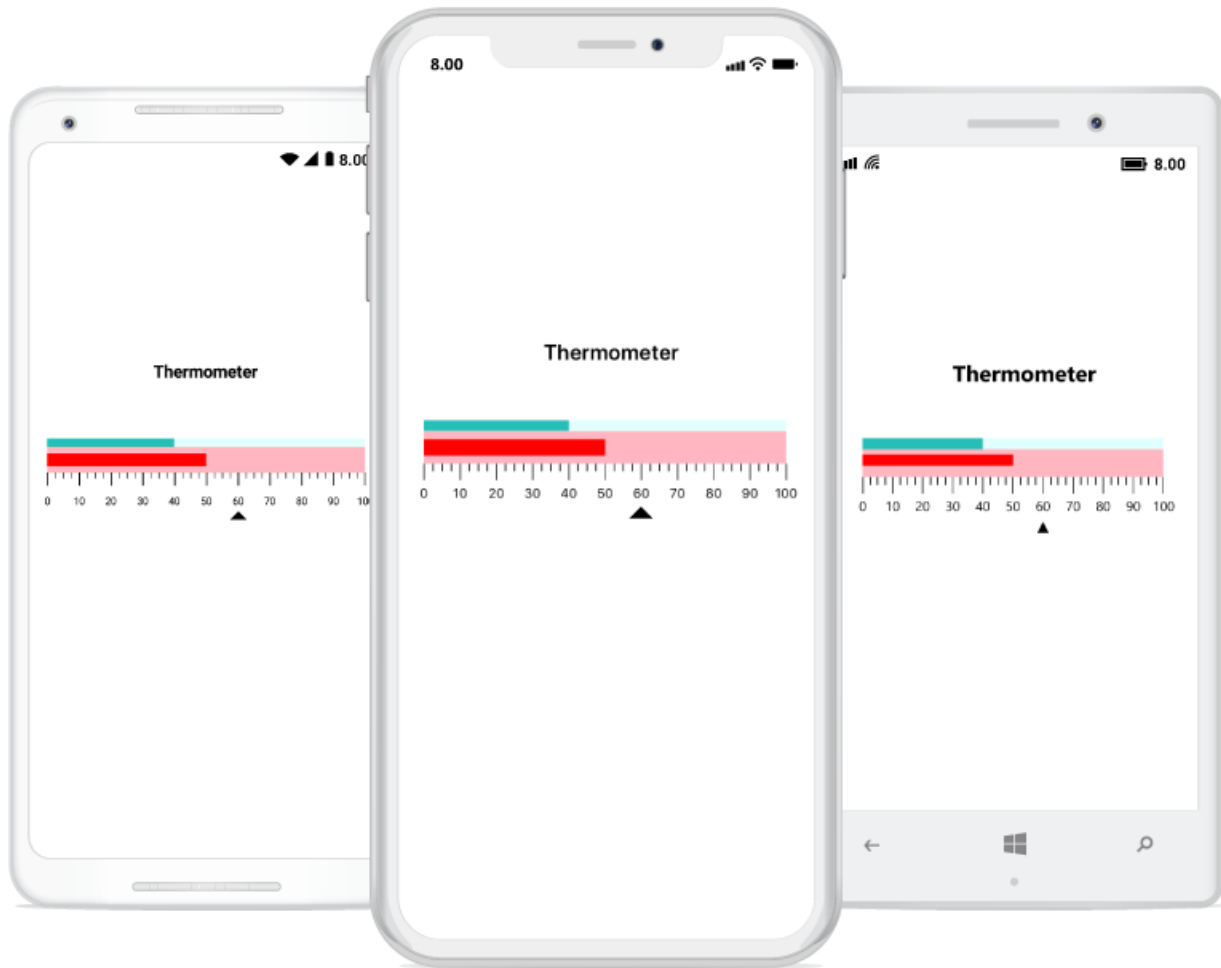
### Overview

[SfLinearGauge](#) displays a range of values graphically along the linear scale, which is considered as the linear form of the linear gauge. It measures the values of the scale and it is present in the horizontal, vertical sliding, or meter.

[SfLinearGauge](#) is used to visualize the numerical values of a scale in linear manner. By using the [SfLinearGauge](#) control, it is possible to render the thermometer.

## Key Features

- [Scales](#): Supports to adding multiple scales on linear gauge by horizontal and vertical orientations.
- [Pointers](#): Supports to add multiple pointers (bar and symbol) on the linear scale.
- [Range](#): Highlights the desired range of values in the gauge scale.
- [Annotations](#): Annotations are used to mark the specific area of interest in the gauge area with texts, shapes, or images. You can add any number of annotations to the gauge.



## Getting Started

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

### Adding SfLinearGauge reference

You can add SfLinearGauge reference using one of the following methods:

#### Method 1: Adding SfLinearGauge reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfLinearGauge to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfGauge](#), and then install it.



![[Adding SfLinearGauge reference from NuGet]](getting-started\_images/Adding SfLinearGauge reference.png)

---

**Note:** Install the same version of SfLinearGauge NuGet in all the projects.

---

### Method 2: Adding SfLinearGauge reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfLinearGauge control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfLinearGauge assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfGauge.Android.dll Syncfusion.SfGauge.XForms.Android.dll Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfGauge.iOS.dll Syncfusion.SfGauge.XForms.iOS.dll Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfGauge.UWP.dll Syncfusion.SfGauge.XForms.UWP.dll Syncfusion.SfGauge.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Launching an application on each platform with SfLinearGauge.

To use the SfLinearGauge control inside an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

#### iOS

To launch the SfLinearGauge in iOS, call the `SfLinearGaugeRenderer.Init()` in the `FinishedLaunching` overridden method of the AppDelegate class after the Xamarin.Forms Framework has been initialized and before the LoadApplication is called, as demonstrated in the following code example.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.SfGauge.XForms.iOS.SfGaugeRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

#### Universal Windows Platform (UWP)

You need to initialize the linear gauge view assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with linear gauge in Release mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    // Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.SfGauge.XForms.UWP.SfGaugeRenderer)
        .GetTypeInfo().Assembly);
    // Replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

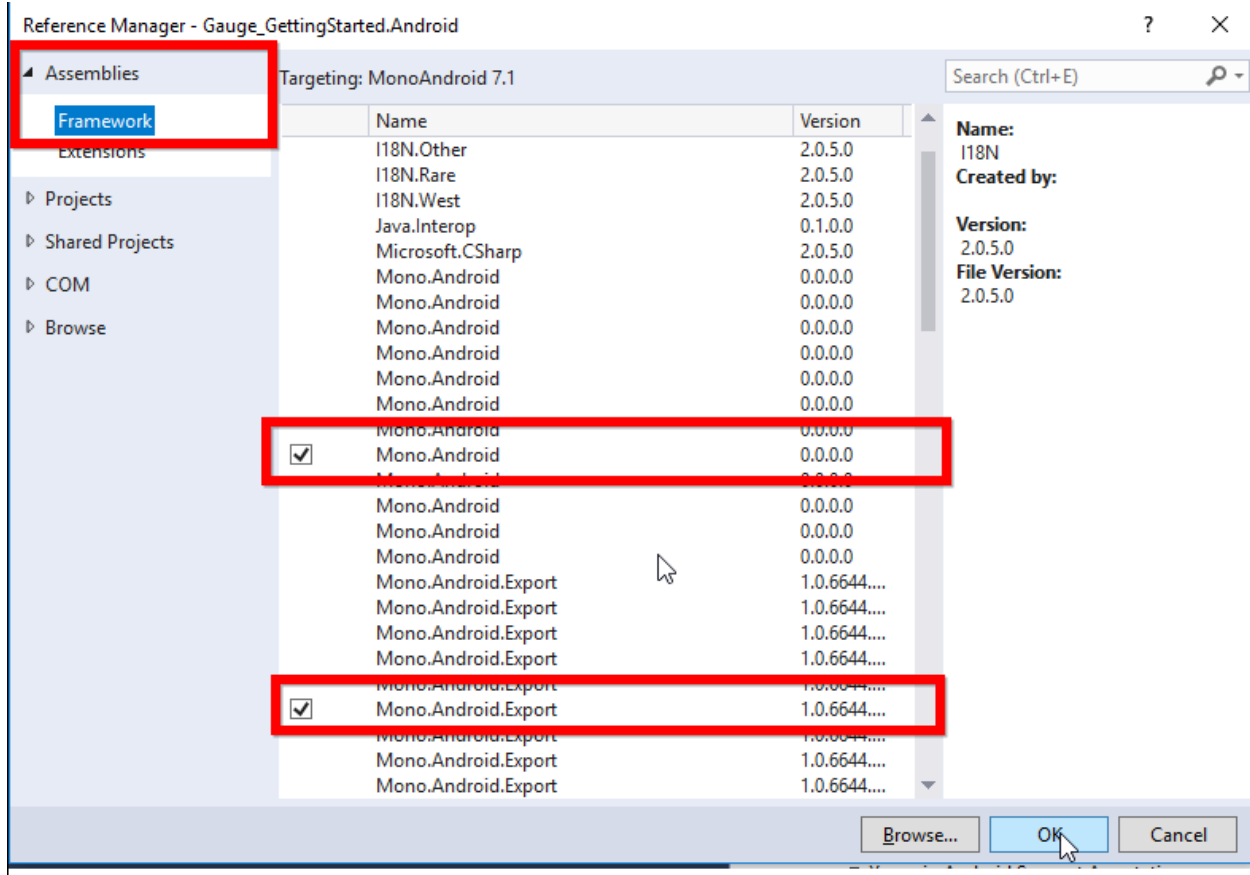
#### Android

The Android platform does not require any additional configuration to render the linear gauge.

#### Reference Mono.Android.Export

1. In the Solution Explorer in the Android project, right-click on References and choose Add Reference.
2. In the Add Reference window, select the Assemblies tab and choose the Framework.

3. In the Framework tab, ensure Mono.Android and Mono.Android.Export is checked and click ok.



## Adding namespace for the added assemblies

## XML

```
xmlns:linear="clr-
namespace:Syncfusion.SfGauge.XForms;assembly=Syncfusion.SfGauge.XForms"
```

## C#

```
using Syncfusion.SfGauge.XForms;
```

Initialize gauge

You can initialize the [SfLinearGauge](#) control with a required optimal name by using the included namespace.

**XML**

<gauge:SfLinearGauge/>

## C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
this.Content = linearGauge;
```

### Adding header

You can assign a unique header to [SfLinearGauge](#) by using the [LinearHeader](#) property and position it wherever as you desired by using the [Offset](#) property.

#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Header>
    <gauge:LinearHeader Text="Thermometer" TextSize="20" FontAttributes="Bold"
      Offset="0.35,0.35"/>
  </gauge:SfLinearGauge.Header>
</gauge:SfLinearGauge>
```

#### C#

```
SfLinearGauge linearGauge=new SfLinearGauge();
LinearHeader linearHeader = new LinearHeader();
linearHeader.Text = "Thermometer";
linearHeader.TextSize = 20;
linearHeader.FontAttributes = FontAttributes.Bold;
linearHeader.Offset = new Point(0.35, 0.35);
linearGauge.Header = linearHeader;
```

### Configuring scales

Scales is a collection of [LinearScale](#), which is used to indicate the numeric values. Scale bar, ticks, labels, ranges, and pointers are the sub elements of a scale.

The [MinimumValue](#) and [MaximumValue](#) properties allow you to set the scale range.

#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

#### C#

```
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
```

```
linearScale.MinorTickSettings.Thickness = 1;  
linearGauge.Scales.Add(linearScale);  
this.Content = linearGauge;
```

Adding a symbol pointer

[SymbolPointer](#) is a shape that can be placed to mark the pointer value in gauge.

#### XML

```
<gauge:LinearScale.Pointers>  
<gauge:SymbolPointer Value="60" Offset="45" Color="#757575"/>  
</gauge:LinearScale.Pointers>
```

#### C#

```
SymbolPointer symbolPointer = new SymbolPointer();  
symbolPointer.Value = 60;  
symbolPointer.Offset = 45;  
symbolPointer.Color = Color.FromHex("#757575");  
linearScale.Pointers.Add(symbolPointer);
```

Adding a bar pointer

[BarPointer](#) is used to mark the scale values. It starts at the beginning of gauge and ends at the pointer value.

#### XML

```
<gauge:LinearScale.Pointers>  
<gauge:BarPointer Value="50" Color="#757575" />  
</gauge:LinearScale.Pointers>
```

#### C#

```
BarPointer barPointer = new BarPointer();  
barPointer.Value = 50;  
barPointer.Color = Color.FromHex("#757575");  
linearScale.Pointers.Add(barPointer);
```

Adding ranges

You can categorize the scale values using the start and end values properties in [LinearRange](#). You can add multiple ranges for a scale using the `ranges` property.

#### XML

```
<gauge:LinearScale.Ranges>  
<gauge:LinearRange StartValue="0" EndValue="40" Color="#27beb7" Offset="-20"/>  
<gauge:LinearRange StartValue="40" EndValue="100" Color="LightCyan" Offset="-20"/>  
</gauge:LinearScale.Ranges>
```

#### C#

```

LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 40;
linearRange.Color = Color.FromHex("#27beb7");
linearRange.Offset = -20;
linearScale.Ranges.Add(linearRange);
LinearRange linearRange1 = new LinearRange();
linearRange1.StartValue = 40;
linearRange1.EndValue = 100;
linearRange1.Color = Color.LightCyan;
linearRange1.Offset = -20;
linearScale.Ranges.Add(linearRange1);
linearGauge.Scales.Add(linearScale);

```

The following code example gives you the complete code of above configurations.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:gauge="clr-
namespace:Syncfusion.SfGauge.XForms;assembly=Syncfusion.SfGauge.XForms"
xmlns:local="clr-namespace:Gauge_GettingStarted"
x:Class="Gauge_GettingStarted.MainPage">
<gauge:SfLinearGauge>
<gauge:SfLinearGauge.Header>
<gauge:LinearHeader Text="Thermometer" TextSize="20" FontAttributes="Bold"
Offset="0.35,0.35"/>
</gauge:SfLinearGauge.Header>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
<gauge:LinearScale.Pointers>
<gauge:SymbolPointer Value="60" Offset="45" Color="#757575"/>
<gauge:BarPointer Value="50" Color="#757575" />
</gauge:LinearScale.Pointers>
<gauge:LinearScale.Ranges>
<gauge:LinearRange StartValue="0" EndValue="40" Color="#27beb7" >
<gauge:LinearRange.Offset>
<OnPlatform x:TypeArguments="x:Double" iOS="-20" Android="-20" WinPhone="-
40" />
</gauge:LinearRange.Offset>
</gauge:LinearRange>
<gauge:LinearRange StartValue="40" EndValue="100" Color="LightCyan">
<gauge:LinearRange.Offset>
<OnPlatform x:TypeArguments="x:Double" iOS="-20" Android="-20" WinPhone="-
40" />
</gauge:LinearRange.Offset>
</gauge:LinearRange>
</gauge:LinearScale.Ranges>

```

```

</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
</ContentPage>

```

## C#

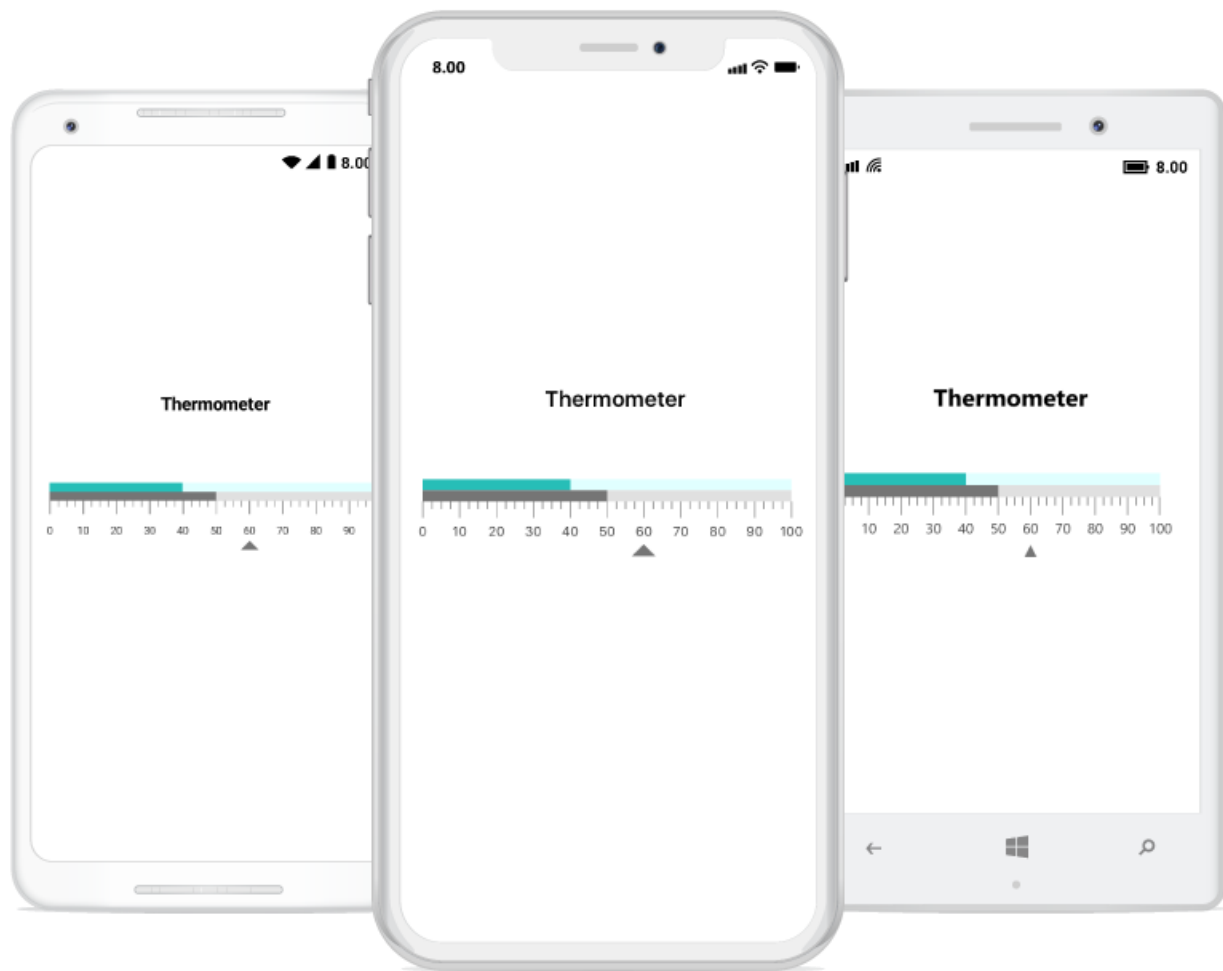
```

using Syncfusion.SfGauge.XForms;
using Xamarin.Forms;
namespace Gauge_GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            //Initializing linear gauge
            SfLinearGauge linearGauge = new SfLinearGauge();
            //Adding header
            LinearHeader linearHeader = new LinearHeader();
            linearHeader.Text = "Thermometer";
            linearHeader.TextSize = 20;
            linearHeader.FontAttributes = FontAttributes.Bold;
            linearHeader.Offset = new Point(0.35, 0.35);
            linearGauge.Header = linearHeader;
            //Initializing scale for linear gauge
            LinearScale linearScale = new LinearScale();
            linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
            linearScale.LabelColor = Color.FromHex("#424242");
            linearScale.MajorTickSettings.Thickness = 1;
            linearScale.MajorTickSettings.Length = 15;
            linearScale.MajorTickSettings.Color = Color.Gray;
            linearScale.MinorTickSettings.Color = Color.Gray;
            linearScale.MinorTickSettings.Length = 7;
            linearScale.MinorTickSettings.Thickness = 1;
            linearGauge.Scales.Add(linearScale);
            //Adding symbol pointer
            SymbolPointer symbolPointer = new SymbolPointer();
            symbolPointer.Value = 60;
            symbolPointer.Offset = 45;
            symbolPointer.Color = Color.FromHex("#757575");
            linearScale.Pointers.Add(symbolPointer);
            //Adding bar pointer
            BarPointer barPointer = new BarPointer();
            barPointer.Value = 50;
            barPointer.Color = Color.FromHex("#757575");
            linearScale.Pointers.Add(barPointer);
            //Adding linear ranges
            LinearRange linearRange = new LinearRange();
            linearRange.StartValue = 0;
            linearRange.EndValue = 40;
            linearRange.Color = Color.FromHex("#27beb7");
            linearRange.Offset = Device.RuntimePlatform == Device.iOS ? -20 :
            Device.RuntimePlatform == Device.Android ? -20 : -40;
            linearScale.Ranges.Add(linearRange);
            LinearRange linearRange1 = new LinearRange();

```

```
linearRange1.StartValue = 40;  
linearRange1.EndValue = 100;  
linearRange1.Color = Color.LightCyan;  
linearRange1.Offset = Device.RuntimePlatform == Device.iOS ? -20 :  
Device.RuntimePlatform == Device.Android ? -20 : -40;  
linearScale.Ranges.Add(linearRange1);  
linearGauge.Scales.Add(linearScale);  
this.Content = linearGauge;  
}  
}  
}
```

The following screenshot illustrates the result of the above codes.



You can find the complete getting started sample from this [link](#).

## Scales

Scales is a collection of [LinearScale](#), which integrates labels, tick marks, ranges, and pointers to customize the basic look and feel of the [SfLinearGauge](#).

### Linear scale

[LinearScale](#) contains sub elements such as ticks, labels, [Ranges](#), and [Pointers](#).

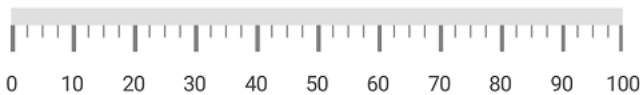


**XML**

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242">
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

**C#**

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearGauge.Scales.Add(linearScale);
```



## Setting minimum and maximum values for scale

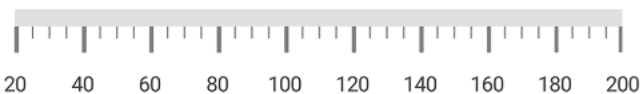
To change minimum and maximum values of linear scale, use the [MinimumValue](#) and [MaximumValue](#) properties as shown in the following code snippet.

**XML**

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale MinimumValue="20" MaximumValue="200"
      ScaleBarColor="#e0e0e0" LabelColor="#424242">
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

**C#**

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MinimumValue = 20;
linearScale.MaximumValue = 200;
linearGauge.Scales.Add(linearScale);
```



### Setting interval for scale

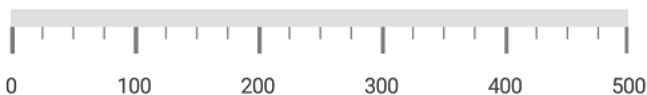
The [Interval](#) property allows you to set the intervals for scale. The default [Interval](#) property of scale is auto interval. Auto interval defines the count of the scale labels as 3 for 100 pixels.

#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale MinimumValue="0" MaximumValue="500" Interval="100"
      ScaleBarColor="#e0e0e0" LabelColor="#424242">
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

#### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MinimumValue = 0;
linearScale.MaximumValue = 500;
linearScale.Interval = 100;
linearGauge.Scales.Add(linearScale);
```



### Setting maximum labels

The [MaximumLabels](#) property defines the count of the scale labels in the 100 pixels. By default, the maximum labels for 100 pixels is 3.

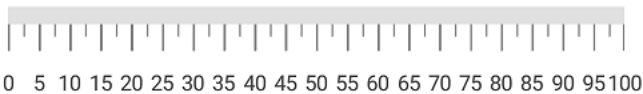
#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale MaximumLabels="4" ScaleBarColor="#e0e0e0"
      LabelColor="#424242" MinorTicksPerInterval="1">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

#### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
```

```
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.MinorTicksPerInterval = 1;
linearScale.MaximumLabels = 4;
linearGauge.Scales.Add(linearScale);
```



### Scale customization

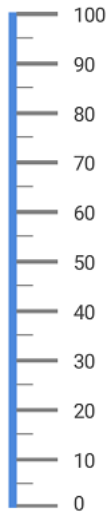
You can customize the color, length, size, and position of the [LinearScale](#) by using the [ScaleBarColor](#), [ScaleBarLength](#), [ScaleBarSize](#), and [Offset](#) properties, respectively.

### XML

```
<gauge:SfLinearGauge Orientation="OrientationVertical">
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#4c88dc" LabelOffset="-5"
      MinorTicksPerInterval="1" Offset="10" ScaleBarLength="300" ScaleBarSize="5"
      LabelColor="#424242">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Length="25" Color="Gray"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Length="10" Color="Gray"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
linearGauge.Orientation = Orientation.OrientationVertical;
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromRgb(76, 136, 220);
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MinorTicksPerInterval = 1;
linearScale.MajorTickSettings.Length = 25;
linearScale.MinorTickSettings.Length = 10;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.LabelOffset = -5;
linearScale.Offset = 10;
linearScale.ScaleBarLength = 300;
linearScale.ScaleBarSize = 5;
linearGauge.Scales.Add(linearScale);
```



### Scale Offset

The space between the control and linear scale can be customized using the `ScaleOffset` property.

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleOffset="40" ScaleBarSize="20"
      CornerRadiusType="Start" CornerRadius="10" ScaleBarColor="#e0e0e0"
      LabelColor="#424242">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
public MainPage()
{
    SfLinearGauge linearGauge = new SfLinearGauge();
    LinearScale linearScale = new LinearScale();
    linearScale.ScaleBarSize = 20;
    linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
    linearScale.CornerRadiusType = CornerRadiusType.Start;
    linearScale.CornerRadius = 10;
    linearScale.ScaleOffset = 40;
    linearScale.MajorTickSettings.Thickness = 1;
    linearScale.MajorTickSettings.Length = 15;
    linearScale.MajorTickSettings.Color = Color.Gray;
    linearScale.MinorTickSettings.Color = Color.Gray;
    linearScale.MinorTickSettings.Length = 7;
    linearScale.MinorTickSettings.Thickness = 1;
    linearScale.LabelColor = Color.FromHex("#424242");
}
```

```
linearGauge.Scales.Add(linearScale);
}
}
```

### Setting opposite position

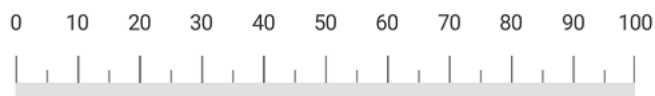
To place the scale at opposite to its original position, set the [OpposedPosition](#) property to true in the scale.

#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale OpposedPosition="True" ScaleBarColor="#e0e0e0"
      LabelColor="#424242" MinorTicksPerInterval="1">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

#### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.MinorTicksPerInterval = 1;
linearScale.OpposedPosition = true;
linearGauge.Scales.Add(linearScale);
```



### Setting scale direction

You can set the scale position to its forward and backward using the [ScalePosition](#) property.

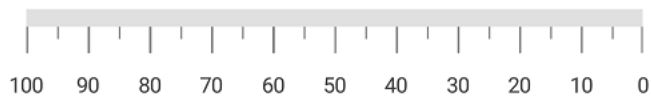
#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      MinorTicksPerInterval="1" ScalePosition="BackWard">
      <gauge:LinearScale.MajorTickSettings>
```

```
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

**C#**

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.MinorTicksPerInterval = 1;
linearScale.ScalePosition = ScalePosition.BackWard;
linearGauge.Scales.Add(linearScale);
```

**Setting corner radius type for scale**

Corners of the [LinearScale](#) can be customized by setting the value to the [CornerRadiusType](#) property. All corners of linear scale can be customized using the **Start**, **End**, **Both**, and **None** options.

[CornerRadius](#) property used to reduce the radius of the corners.

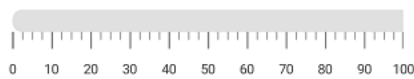
**XML**

```
<gauge:SfLinearGauge>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarSize="20" CornerRadiusType="Start"
CornerRadius="10" ScaleBarColor="#e0e0e0" LabelColor="#424242">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

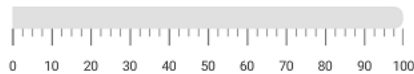
**C#**

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarSize = 20;
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.CornerRadiusType = CornerRadiusType.Start;
linearScale.CornerRadius = 10;
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.LabelColor = Color.FromHex("#424242");
linearGauge.Scales.Add(linearScale);
```

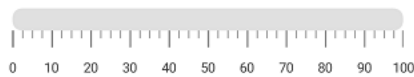
Start



End



Both



## Multiple scales

It helps you to add multiple scales to the same linear gauge and customize all the scales in a [Scales](#) collection.

## XML

```
<gauge:SfLinearGauge Orientation="OrientationVertical">
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarLength="500" MinimumValue="0" Interval="10"
      MaximumValue="100" Offset="9"
      ScaleBarColor="Gray" ScaleBarSize="5" LabelColor="Gray"
      MinorTicksPerInterval="1"
      LabelPostfix="°F" LabelOffset="-6">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Color="Gray" Thickness="1"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Color="Gray"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
    <gauge:LinearScale MinimumValue="0" ScaleBarLength="500" Interval="10"
      MaximumValue="100">
```

```

ShowTicks="False" ShowLabels="False" ScaleBarSize="5"
ScaleBarColor="Transparent"
CornerRadiusType="Both" LabelColor="Black" >
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Color="Black"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Color="Black"/>
</gauge:LinearScale.MinorTickSettings>
<gauge:LinearScale.Pointers>
<gauge:BarPointer Value="45" Thickness="7" CornerRadiusType="End"
CornerRadius="3"
Color="#f95c85"/>
</gauge:LinearScale.Pointers>
</gauge:LinearScale>
<gauge:LinearScale MinimumValue="0" ScaleBarLength="500"
MinorTicksPerInterval="1"
MaximumValue="38" Interval="2" ScaleBarColor="Gray" ScaleBarSize="5"
LabelColor="Gray" OpposedPosition="True" Offset="-9" LabelPostfix="°C">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Color="Gray" Thickness="1"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Color="Gray"/>
</gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

```

**C#**

```

SfLinearGauge linearGauge = new SfLinearGauge();
linearGauge.Orientation = Orientation.OrientationVertical;
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarLength = 500;
linearScale.MinimumValue = 0;
linearScale.Interval = 10;
linearScale.MaximumValue = 100;
linearScale.Offset = 9;
linearScale.ScaleBarColor = Color.Gray;
linearScale.ScaleBarSize = 5;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MajorTickSettings.Thickness = 1;
linearScale.LabelColor = Color.Gray;
linearScale.MinorTicksPerInterval = 1;
linearScale.LabelPostfix = "°F";
linearScale.LabelOffset = -6;
linearGauge.Scales.Add(linearScale);
LinearScale scale = new LinearScale();
scale.MinimumValue = 0;
scale.ScaleBarLength = 500;
scale.Interval = 10;
scale.MaximumValue = 100;
scale.ShowTicks = false;
scale.ShowLabels = false;

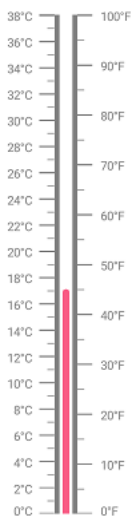
```



```

scale.ScaleBarSize = 5;
scale.ScaleBarColor = Color.Transparent;
scale.CornerRadiusType = CornerRadiusType.Both;
scale.MajorTickSettings.Color = Color.Black;
scale.MinorTickSettings.Color = Color.Black;
scale.LabelColor = Color.Black;
BarPointer barPointer = new BarPointer();
barPointer.Value = 45;
barPointer.Thickness = 7;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.CornerRadius = 3;
barPointer.Color = Color.FromRgb(249, 92, 133);
scale.Pointers.Add(barPointer);
linearGauge.Scales.Add(scale);
LinearScale linearScale1 = new LinearScale();
linearScale1.MinimumValue = 0;
linearScale1.ScaleBarLength = 500;
linearScale1.MinorTicksPerInterval = 1;
linearScale1.MaximumValue = 38;
linearScale1.Interval = 2;
linearScale1.ScaleBarColor = Color.Gray;
linearScale1.ScaleBarSize = 5;
linearScale1.MajorTickSettings.Color = Color.Gray;
linearScale1.MinorTickSettings.Color = Color.Gray;
linearScale1.LabelColor = Color.Gray;
linearScale1.OpposedPosition = true;
linearScale1.Offset = -9;
linearScale1.MajorTickSettings.Thickness = 1;
linearScale1.LabelOffset = -5;
linearScale1.LabelPostfix = "°C";
linearGauge.Scales.Add(linearScale1);

```



### Setting gradient color for scale

You can give smooth color transition to scale to specifying the different colors based on scale value by using [GradientStops](#) property.

### XML

```

<gauge:SfLinearGauge>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarSize="15" MinorTicksPerInterval="1"
LabelColor="Black">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Black" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Black" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
<gauge:LinearScale.GradientStops>
<gauge:GaugeGradientStop Value="10" Color="#51c9e1"/>
<gauge:GaugeGradientStop Value="40" Color="#93e9e1"/>
<gauge:GaugeGradientStop Value="50" Color="#c5e692"/>
<gauge:GaugeGradientStop Value="60" Color="#fedd2b"/>
<gauge:GaugeGradientStop Value="100" Color="#e87813"/>
</gauge:LinearScale.GradientStops>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

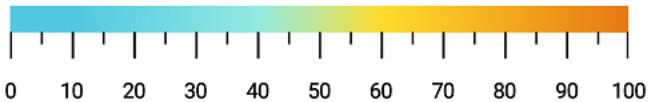
```

**C#**

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
GaugeGradientStop gaugeGradientStop = new GaugeGradientStop();
gaugeGradientStop.Value = 10;
gaugeGradientStop.Color = Color.FromRgb(81, 201, 225);
linearScale.GradientStops.Add(gaugeGradientStop);
GaugeGradientStop gaugeGradientStop1 = new GaugeGradientStop();
gaugeGradientStop1.Value = 40;
gaugeGradientStop1.Color = Color.FromRgb(147, 233, 225);
linearScale.GradientStops.Add(gaugeGradientStop1);
GaugeGradientStop gaugeGradientStop2 = new GaugeGradientStop();
gaugeGradientStop2.Value = 50;
gaugeGradientStop2.Color = Color.FromRgb(197, 230, 146);
linearScale.GradientStops.Add(gaugeGradientStop2);
GaugeGradientStop gaugeGradientStop3 = new GaugeGradientStop();
gaugeGradientStop3.Value = 60;
gaugeGradientStop3.Color = Color.FromRgb(254, 221, 43);
linearScale.GradientStops.Add(gaugeGradientStop3);
GaugeGradientStop gaugeGradientStop4 = new GaugeGradientStop();
gaugeGradientStop4.Value = 100;
gaugeGradientStop4.Color = Color.FromRgb(232, 120, 19);
linearScale.GradientStops.Add(gaugeGradientStop4);
linearScale.ScaleBarSize = 15;
linearScale.MajorTickSettings.Color = Color.Black;
linearScale.MinorTickSettings.Color = Color.Black;
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.MinorTicksPerInterval = 1;
linearScale.LabelColor = Color.Black;
linearGauge.Scales.Add(linearScale);

```



### Tick Setting

The [TickSetting](#) property is used to identify the gauge's data value by marking the gauge scale in regular increments.

### Ticks visibility

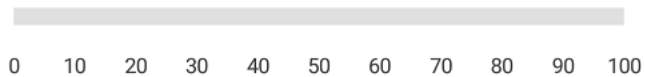
Ticks visibility can be customized using the [ShowTicks](#) property of linear scale.

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242" LabelOffset
    ="-10" ShowTicks ="False">
  </gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.LabelOffset = -10;
linearScale.ShowTicks = false;
linearGauge.Scales.Add(linearScale);
```



### Tick customization

You can customize the color and thickness of ticks by using the [Color](#) and [Thickness](#) properties. The ticks length also can be customized using the [Length](#) property as demonstrated below.

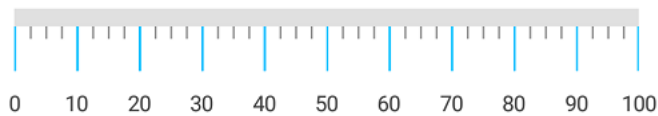
### Major tick customization

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Length ="25" Color = "DeepSkyBlue"/>
      </gauge:LinearScale.MajorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

**C#**

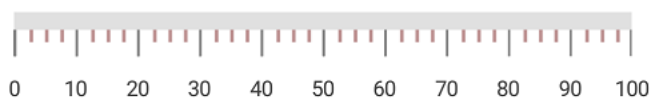
```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
LinearTickSettings majorTickSettings = new LinearTickSettings();
majorTickSettings.Thickness = 1;
majorTickSettings.Length = 25;
majorTickSettings.Color = Color.DeepSkyBlue;
linearScale.MajorTickSettings = majorTickSettings;
linearGauge.Scales.Add(linearScale);
```

*Minor tick customization***XML**

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      MinorTicksPerInterval="3">
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="2" Length="7" Color="RosyBrown"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

**C#**

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MinorTicksPerInterval = 3;
LinearTickSettings minorTickSettings = new LinearTickSettings();
minorTickSettings.Thickness = 2;
minorTickSettings.Length = 7;
minorTickSettings.Color = Color.RosyBrown;
linearScale.MinorTickSettings = minorTickSettings;
linearGauge.Scales.Add(linearScale);
```



### Setting minor ticks per interval

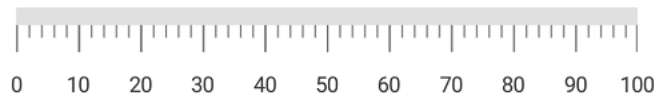
The [Interval](#) property is used to calculate the tick counts for a scale. Similar to ticks, minor ticks are also calculated by using the [MinorTicksPerInterval](#) property.

#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      MinorTicksPerInterval = "4">
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

#### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MinorTicksPerInterval = 4;
linearGauge.Scales.Add(linearScale);
```



### Setting position for ticks

The major and minor ticks can be positioned far away from the scale by using the [Offset](#) property.

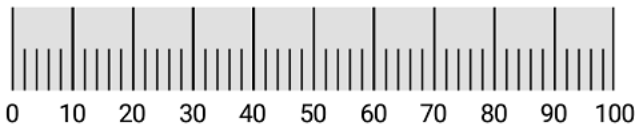
#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="Black" ScaleBarSize
      = "40" LabelOffset = "-9">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color = "Black" Length = "40" Offset
          = "-40" />
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Color = "Black" Length = "20" Offset = "-20" />
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

#### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarSize = 40;
linearScale.ScaleBarColor= Color.FromHex("#e0e0e0");
```

```
linearScale.MajorTickSettings.Color = Color.Black;
linearScale.MinorTickSettings.Color = Color.Black;
linearScale.MajorTickSettings.Length = 40;
linearScale.MinorTickSettings.Offset = -20;
linearScale.MajorTickSettings.Offset = -40;
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MinorTickSettings.Length = 20;
linearScale.LabelOffset = -9;
linearScale.MinorTicksPerInterval = 4;
linearScale.LabelColor = Color.Black;
linearGauge.Scales.Add(linearScale);
```



## Labels

[LinearScale](#) labels associate a numeric value with major scale tick marks.

### Label color customization

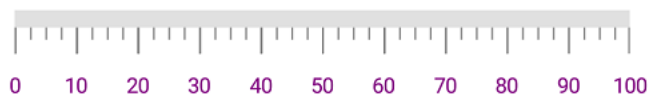
The label color can be changed using the [LabelColor](#) property.

## XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="Purple">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

## C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.LabelColor = Color.Purple;
linearGauge.Scales.Add(linearScale);
```



### Label font customization

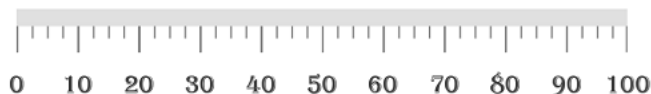
The label font can be customized by using the [LabelFontSize](#), [FontAttribute](#), and [FontFamily](#) properties.

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      LabelFontSize="15" FontAttributes="Bold">
      <gauge:LinearScale.FontFamily>
        <OnPlatform x:TypeArguments="x:String" iOS="Chalkduster"
          Android="algerian.ttf" WinPhone="Chiller" />
      </gauge:LinearScale.FontFamily>
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.LabelFontSize = 15;
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.FontAttributes = FontAttributes.Bold;
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.FontFamily = Device.RuntimePlatform == Device.iOS ?
"Chalkduster" : Device.RuntimePlatform == Device.Android ? "algerian.ttf" :
"Chiller";
linearGauge.Scales.Add(linearScale);
```



### Setting position for labels

The labels can be positioned far away from the ticks by using the [LabelOffset](#) property in pixel.

#### XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242" LabelOffset
    ="5">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

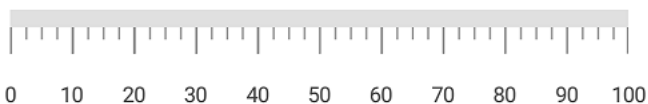
```

#### C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.LabelOffset = 5;
linearGauge.Scales.Add(linearScale);

```



### Setting postfix and prefix for labels

You can postfix/prefix values to the scale labels using the [LabelPostfix](#) and [LabelPrefix](#) properties, respectively.

#### Setting label postfix

The [LabelPostfix](#) property allows you to postfix the values to scale labels.

#### XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale MinimumValue = "0" MaximumValue = "1000" Interval = "200"
    LabelPostfix = "K" ScaleBarColor="#e0e0e0" LabelColor="#424242">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

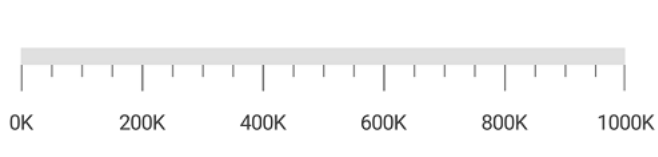
```



```
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

**C#**

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.MinimumValue = 0;
linearScale.MaximumValue = 1000;
linearScale.Interval = 200;
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.LabelPostfix = "K";
linearGauge.Scales.Add(linearScale);
```

*Setting label prefix*

The [LabelPrefix](#) property allows you to prefix the values to scale labels.

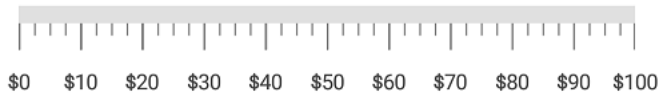
**XML**

```
<gauge:SfLinearGauge>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242" LabelPrefix
="$">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

**C#**

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
```

```
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.LabelPrefix = "$";
linearGauge.Scales.Add(linearScale);
```



### Custom labels

Linear scale supports custom label format using the [CustomLabels](#) property.

You can give labels in an array that you want to place in scale.

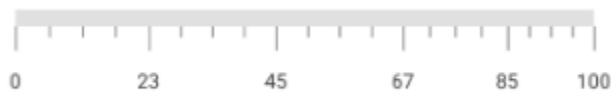
### XML

```
Namespace:
xmlns:sys="clr-namespace:System;assembly=mscorlib"
...
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242" >
      <gauge:LinearScale.CustomLabels>
        <x:Array Type="{x:Type x:Double}">
          <sys:Double>0</sys:Double>
          <sys:Double>23</sys:Double>
          <sys:Double>45</sys:Double>
          <sys:Double>67</sys:Double>
          <sys:Double>85</sys:Double>
          <sys:Double>100</sys:Double>
        </x:Array>
      </gauge:LinearScale.CustomLabels>
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.MajorTickSettings.Thickness = 1;
```

```
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.CustomLabels = new double[] { 0, 23, 45, 67, 85, 100 };
linearGauge.Scales.Add(linearScale);
Content = linearGauge;
```



### Labels visibility

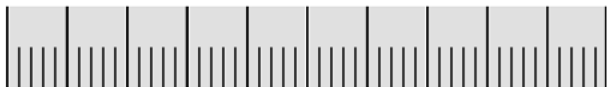
Labels visibility can be customized using the [ShowLabels](#) property of linear scale.

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" ShowLabels = "False" ScaleBarSize
    ="40">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color ="Black" Length ="40" Offset
        ="-40" />
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Color ="Black" Length = "20" Offset ="-20" />
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarSize = 40;
linearScale.ScaleBarColor= Color.FromHex("#e0e0e0");
linearScale.MajorTickSettings.Color = Color.Black;
linearScale.MinorTickSettings.Color = Color.Black;
linearScale.MajorTickSettings.Length = 40;
linearScale.MinorTickSettings.Offset = -20;
linearScale.MajorTickSettings.Offset = -40;
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MinorTickSettings.Length = 20;
linearScale.MinorTicksPerInterval = 4;
linearScale.ShowLabels = false;
linearGauge.Scales.Add(linearScale);
```



## Ranges

Range is a visual element, which begins and ends at specified values within a scale. You can add any number of range for a scale by using the array of range objects.

### Setting start and end values for range

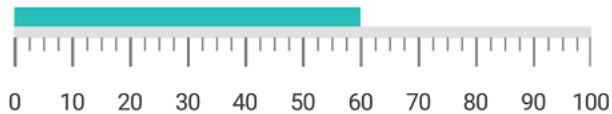
Start and end values of ranges are set by using the [StartValue](#) and [EndValue](#) properties.

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartValue="0" EndValue="60" Color="#27beb7" Offset = "-20"/>
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 60;
linearRange.Color = Color.FromHex("#27beb7");
linearRange.Offset = -20;
linearScale.Ranges.Add(linearRange);
linearGauge.Scales.Add(linearScale);
```



### Range customization

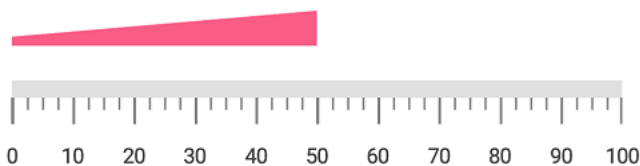
To change the range's background color, use the [Color](#) property of linear range. The thickness of the range can be changed using the [StartWidth](#) and [EndWidth](#) properties.

### XML

```
<gauge:SfLinearGauge>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
<gauge:LinearScale.Ranges>
<gauge:LinearRange StartValue="0" EndValue="50" Color="#f95c85" Offset = "-30" StartWidth = "-5" EndWidth = "-20"/>
</gauge:LinearScale.Ranges>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearGauge.Scales.Add(linearScale);
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.Color = Color.FromRgb(249, 92, 133);
linearRange.EndValue = 50;
linearRange.StartWidth = -5;
linearRange.EndWidth = -20;
linearRange.Offset = -30;
linearScale.Ranges.Add(linearRange);
linearGauge.Scales.Add(linearScale);
```



### Setting position for range

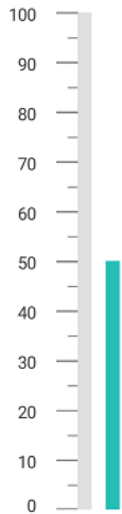
The range can be adjusted above or below the scale by using the [Offset](#) value in pixels.

### XML

```
<gauge:SfLinearGauge Orientation = "OrientationVertical">
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      OpposedPosition = "True" Interval = "10" ScaleBarLength="350"
      MinorTicksPerInterval = "1">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartValue="0" EndValue="50" Color="#27beb7" Offset =
          "30"/>
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
linearGauge.Orientation = Orientation.OrientationVertical;
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.MinorTicksPerInterval = 1;
linearScale.OpposedPosition = true;
linearScale.Interval = 10;
linearScale.ScaleBarLength = 350;
linearScale.LabelColor = Color.FromHex("#424242");
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 50;
linearRange.Color = Color.FromHex("#27beb7");
linearRange.Offset = 30;
linearScale.Ranges.Add(linearRange);
linearGauge.Scales.Add(linearScale);
```



### Setting multiple ranges

You can add n number of ranges to a scale by using the [LinearRange](#) property of range as demonstrated below.

#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="Transparent" ShowTicks="False"
      LabelColor="#424242" Interval="25" LabelFontSize="14" LabelOffset="-10"
      MinorTicksPerInterval="0">
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartValue="0" EndValue="25" Color="#6de500" StartWidth
          ="-10" EndWidth="-15"/>
        <gauge:LinearRange StartValue="25" EndValue="50" Color="#53ad00" StartWidth
          ="-15" EndWidth="-20"/>
        <gauge:LinearRange StartValue="50" EndValue="75" Color="#009148" StartWidth
          ="-20" EndWidth="-25"/>
        <gauge:LinearRange StartValue="75" EndValue="100" Color="#026623" StartWidth
          ="-25" EndWidth="-30"/>
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

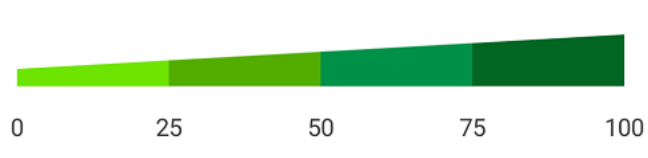
#### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.Transparent;
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.LabelFontSize = 14;
linearScale.LabelOffset = -10;
linearScale.Interval = 25;
linearScale.MinorTicksPerInterval = 0;
linearScale.ShowTicks = false;
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
```

```

linearRange.EndValue = 25;
linearRange.Color = Color.FromHex("#6de500");
linearRange.StartWidth = -10;
linearRange.EndWidth = -15;
linearScale.Ranges.Add(linearRange);
LinearRange linearRange1 = new LinearRange();
linearRange1.StartValue = 25;
linearRange1.EndValue = 50;
linearRange1.Color = Color.FromHex("#53ad00");
linearRange1.StartWidth = -15;
linearRange1.EndWidth = -20;
linearScale.Ranges.Add(linearRange1);
LinearRange linearRange2 = new LinearRange();
linearRange2.StartValue = 50;
linearRange2.EndValue = 75;
linearRange2.Color = Color.FromHex("#009148");
linearRange2.StartWidth = -20;
linearRange2.EndWidth = -25;
linearScale.Ranges.Add(linearRange2);
LinearRange linearRange3 = new LinearRange();
linearRange3.StartValue = 75;
linearRange3.EndValue = 100;
linearRange3.Color = Color.FromHex("#026623");
linearRange3.StartWidth = -25;
linearRange3.EndWidth = -30;
linearScale.Ranges.Add(linearRange3);
linearGauge.Scales.Add(linearScale);

```



### Setting gradient color for range

You can give smooth color transition to range to specifying the different colors based on range value by using [GradientStops](#) property.

### XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="Transparent" ScaleBarSize ="20"
      ShowTicks="False" LabelColor="#424242" Interval="25" LabelFontSize ="14"
      LabelOffset="10" MinorTicksPerInterval ="0">
      <gauge:LinearScale.Ranges>
        <gauge:LinearRange StartWidth="20" EndWidth="20" StartValue="0"
          EndValue="100">
          <gauge:LinearRange.GradientStops>
            <gauge:GaugeGradientStop Value="0" Color="#FFF9C2C3"/>
            <gauge:GaugeGradientStop Value="100" Color="#FFD91D71"/>
          </gauge:LinearRange.GradientStops>
        </gauge:LinearRange>
      </gauge:LinearScale.Ranges>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>

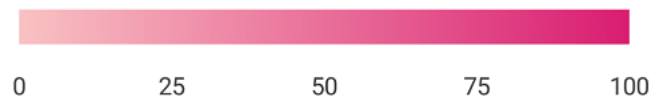
```



```
</gauge:SfLinearGauge>
```

**C#**

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.Transparent;
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.LabelFontSize = 14;
linearScale.LabelOffset = 10;
linearScale.Interval = 25;
linearScale.MinorTicksPerInterval = 0;
linearScale.ShowTicks = false;
linearScale.ScaleBarSize = 20;
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.EndValue = 100;
linearRange.StartWidth = 20;
linearRange.EndWidth = 20;
ObservableCollection<GaugeGradientStop> gradientColor = new
ObservableCollection<GaugeGradientStop>()
{
    new GaugeGradientStop() {Value = 0, Color = Color.FromHex("#FFF9C2C3") },
    new GaugeGradientStop() {Value = 100, Color = Color.FromHex("#FFD91D71") }
};
linearRange.GradientStops = gradientColor;
linearScale.Ranges.Add(linearRange);
linearGauge.Scales.Add(linearScale);
```

**Pointers**

[SfLinearGauge](#) provides support to mark the values using [BarPointer](#) and [SymbolPointer](#).

Adding bar pointer to scale

[BarPointer](#) is used to mark the scale values. It starts at the beginning of gauge and ends at the pointer value.

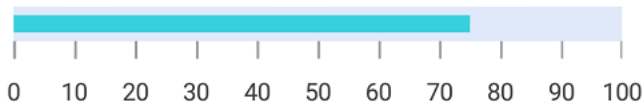
**XML**

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e9f9" LabelColor="#424242" ScaleBarSize
    = "20" LabelFontSize="14" MinorTicksPerInterval="0">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="#9E9E9E" Length="10"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.Pointers>
        <gauge:BarPointer Value="75" EnableAnimation="false" Color="#36d1dc" />
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
```

```
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e9f9");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.ScaleBarSize = 20;
linearScale.LabelFontSize = 14;
linearScale.MinorTicksPerInterval = 0;
linearScale.MajorTickSettings.Color = Color.FromHex("#9E9E9E");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 10;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.EnableAnimation = false;
barPointer.Color = Color.FromHex("#36d1dc");
linearScale.Pointers.Add(barPointer);
linearGauge.Scales.Add(linearScale);
```



### Bar pointer customization

The bar pointer's UI is customized by using the [Color](#) and [Thickness](#) properties.

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e9f9" LabelColor="#424242" ScaleBarSize
      = "40" LabelFontSize="14" MinorTicksPerInterval="0">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="#9E9E9E" Length="10"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.Pointers>
        <gauge:BarPointer Value="75" EnableAnimation="false" Color="#36d1dc"
          Thickness="20"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

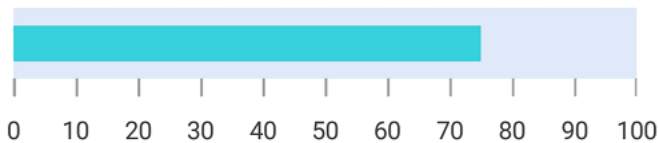
### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e9f9");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.ScaleBarSize = 40;
```

```

linearScale.LabelFontSize = 14;
linearScale.MinorTicksPerInterval = 0;
linearScale.MajorTickSettings.Color = Color.FromHex("#9E9E9E");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 10;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.Thickness = 20;
barPointer.EnableAnimation = false;
barPointer.Color = Color.FromHex("#36d1dc");
linearScale.Pointers.Add(barPointer);
linearGauge.Scales.Add(linearScale);

```



#### Setting corner radius type for bar pointer

Corners of the [BarPointer](#) can be customized by setting the value to the [CornerRadiusType](#) property. All corners of bar pointer can be customized using the [Start](#), [End](#), [Both](#), and [None](#) options.

[CornerRadius](#) property used to reduce the radius of the corners.

#### XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e9f9" LabelColor="#424242" ScaleBarSize
    = "40" LabelFontSize="14" MinorTicksPerInterval="0">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="#9E9E9E" Length="10"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.Pointers>
        <gauge:BarPointer Value="75" EnableAnimation="false" Color="#36d1dc"
        Thickness="20" CornerRadiusType="Start" CornerRadius="10"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

```

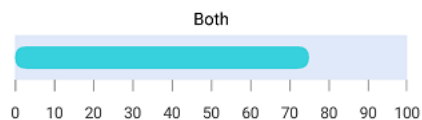
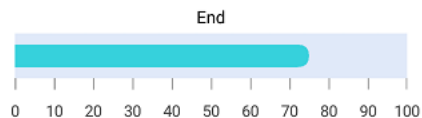
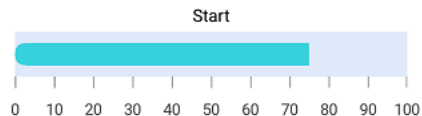
#### C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e9f9");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.ScaleBarSize = 40;
linearScale.LabelFontSize = 14;
linearScale.MinorTicksPerInterval = 0;
linearScale.MajorTickSettings.Color = Color.FromHex("#9E9E9E");
linearScale.MajorTickSettings.Thickness = 1;

```

```
linearScale.MajorTickSettings.Length = 10;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.Thickness = 20;
barPointer.CornerRadiusType = CornerRadiusType.Start;
barPointer.CornerRadius = 10;
barPointer.EnableAnimation = false;
barPointer.Color = Color.FromHex("#36d1dc");
linearScale.Pointers.Add(barPointer);
linearGauge.Scales.Add(linearScale);
```



### Setting gradient color for bar pointer

You can give smooth color transition to bar pointer to specifying the different colors based on bar pointer's value by using [GradientStops](#) property.

### XML

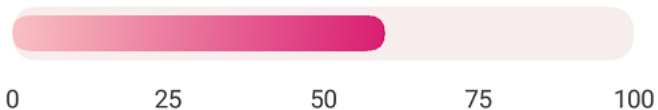
```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#F7EDED" LabelColor="Black"
      ScaleBarSize="40" CornerRadius="20" CornerRadiusType="Both"
      LabelFontSize="14" MinimumValue="0" MaximumValue="100" Interval="25"
      LabelOffset="-10" ShowTicks="False">
      <gauge:LinearScale.Pointers>
        <gauge:BarPointer Value="75" CornerRadiusType="Both" CornerRadius="15"
          Thickness="30" EnableAnimation="False">
          <gauge:BarPointer.GradientStops>
            <gauge:GaugeGradientStop Value="10" Color="#f8babf"/>
            <gauge:GaugeGradientStop Value="40" Color="#ee89a7"/>
            <gauge:GaugeGradientStop Value="50" Color="#e4548c"/>
            <gauge:GaugeGradientStop Value="60" Color="#db2575"/>
          </gauge:BarPointer.GradientStops>
        </gauge:BarPointer>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

**C#**

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromRgb(247, 237, 237);
linearScale.LabelColor = Color.Black;
linearScale.ScaleBarSize = 40;
linearScale.CornerRadius = 20;
linearScale.CornerRadiusType = CornerRadiusType.Both;
linearScale.LabelFontSize = 14;
linearScale.MinimumValue = 0;
linearScale.MaximumValue = 100;
linearScale.Interval = 25;
linearScale.LabelOffset = -10;
linearScale.ShowTicks = false;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.CornerRadiusType = CornerRadiusType.Both;
barPointer.CornerRadius = 15;
barPointer.Thickness = 30;
barPointer.EnableAnimation = false;
GaugeGradientStop gaugeGradientStop = new GaugeGradientStop();
gaugeGradientStop.Value = 10;
gaugeGradientStop.Color = Color.FromRgb(248, 186, 191);
barPointer.GradientStops.Add(gaugeGradientStop);
GaugeGradientStop gaugeGradientStop1 = new GaugeGradientStop();
gaugeGradientStop1.Value = 40;
gaugeGradientStop1.Color = Color.FromRgb(238, 137, 167);
barPointer.GradientStops.Add(gaugeGradientStop1);
GaugeGradientStop gaugeGradientStop2 = new GaugeGradientStop();
gaugeGradientStop2.Value = 50;
gaugeGradientStop2.Color = Color.FromRgb(228, 84, 140);
barPointer.GradientStops.Add(gaugeGradientStop2);
GaugeGradientStop gaugeGradientStop3 = new GaugeGradientStop();
gaugeGradientStop3.Value = 60;
gaugeGradientStop3.Color = Color.FromRgb(219, 37, 117);
barPointer.GradientStops.Add(gaugeGradientStop3);
linearScale.Pointers.Add(barPointer);
linearGauge.Scales.Add(linearScale);

```



Adding symbol pointer to scale

In [SymbolPointer](#), the value is pointed by a symbol on the scale.

**XML**

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e9f9" LabelColor="#424242" ScaleBarSize
= "40" LabelFontSize ="14" MinorTicksPerInterval ="0">

```

```

<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="#9E9E9E" Length="10"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.Pointers>
<gauge:BarPointer Value="75" CornerRadiusType="End" CornerRadius="10"
EnableAnimation="false" Color="#36d1dc" Thickness="20"/>
<gauge:SymbolPointer Value="30" EnableAnimation="False" Color="#5b86e5"/>
</gauge:LinearScale.Pointers>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

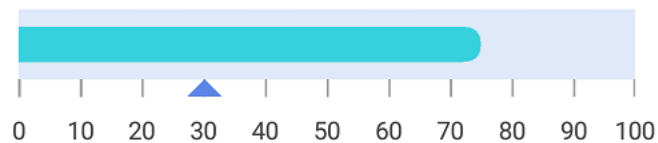
```

## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e9f9");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.ScaleBarSize = 40;
linearScale.LabelFontSize = 14;
linearScale.MinimumValue = 0;
linearScale.MaximumValue = 100;
linearScale.Interval = 10;
linearScale.MinorTicksPerInterval = 0;
linearScale.MajorTickSettings.Color = Color.FromHex("#9E9E9E");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 10;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.Thickness = 20;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.CornerRadius = 10;
barPointer.EnableAnimation = false;
barPointer.Color = Color.FromHex("#36d1dc");
linearScale.Pointers.Add(barPointer);
SymbolPointer symbolPointer = new SymbolPointer();
symbolPointer.Value = 30;
symbolPointer.EnableAnimation = false;
symbolPointer.Color = Color.FromHex("#5b86e5");
linearScale.Pointers.Add(symbolPointer);
linearGauge.Scales.Add(linearScale);

```



### Symbol pointer customization

You can modify the symbol pointer's size using the [Thickness](#) property. The color of the symbol pointer is changed using the [Color](#) property.

### XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
      <gauge:LinearScale.Pointers>
        <gauge:SymbolPointer Value="70" Color="DeepSkyBlue" Thickness="15"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

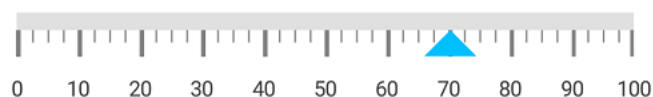
```

## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
SymbolPointer symbolPointer = new SymbolPointer();
symbolPointer.Value = 70;
symbolPointer.Color = Color.DeepSkyBlue;
symbolPointer.Thickness = 15;
linearScale.Pointers.Add(symbolPointer);
linearGauge.Scales.Add(linearScale);

```



### Positioning symbol pointer

You can position the [SymbolPointer](#) by using the following two ways:

#### Setting symbol pointer position

You can customize the position of the [SymbolPointer](#) by using the [SymbolPointerPosition](#). The default symbol pointer position is **Far**.

## XML

```

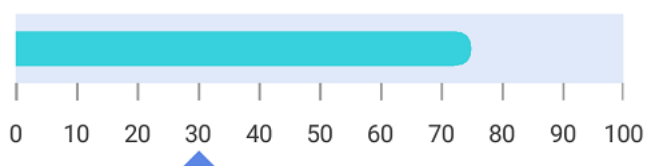
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e9f9" LabelColor="#424242" ScaleBarSize
      = "40" LabelFontSize="14" MinorTicksPerInterval="0">
      <gauge:LinearScale.MajorTickSettings>

```

```
<gauge:LinearTickSettings Thickness="1" Color="#9E9E9E" Length="10"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.Pointers>
<gauge:BarPointer Value="75" CornerRadiusType="End" CornerRadius="10"
EnableAnimation="false" Color="#36d1dc" Thickness="20"/>
<gauge:SymbolPointer Value="30" EnableAnimation="False" Color="#5b86e5"
SymbolPointerPosition="Away"/>
</gauge:LinearScale.Pointers>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

## C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e9f9");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.ScaleBarSize = 40;
linearScale.LabelFontSize = 14;
linearScale.MinorTicksPerInterval = 0;
linearScale.MajorTickSettings.Color = Color.FromHex("#9E9E9E");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 10;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.Thickness = 20;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.CornerRadius = 10;
barPointer.EnableAnimation = false;
barPointer.Color = Color.FromHex("#36d1dc");
linearScale.Pointers.Add(barPointer);
SymbolPointer symbolPointer = new SymbolPointer();
symbolPointer.Value = 30;
symbolPointer.EnableAnimation = false;
symbolPointer.SymbolPointerPosition = SymbolPointerPosition.Away;
symbolPointer.Color = Color.FromHex("#5b86e5");
linearScale.Pointers.Add(symbolPointer);
linearGauge.Scales.Add(linearScale);
```



### Setting offset for symbol pointer

You can move the [SymbolPointer](#) by using the [Offset](#) property.

## XML

```
<gauge:SfLinearGauge>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarColor="#e0e9f9" LabelColor="#424242" ScaleBarSize
="40" LabelFontSize="14" MinorTicksPerInterval="0">
```



```

<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="#9E9E9E" Length="10"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.Pointers>
<gauge:BarPointer Value="75" CornerRadiusType="End" CornerRadius="10"
EnableAnimation="false" Color="#36d1dc" Thickness="20"/>
<gauge:SymbolPointer Value="30" EnableAnimation="False" Color="#5b86e5"
Offset="40"/>
</gauge:LinearScale.Pointers>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

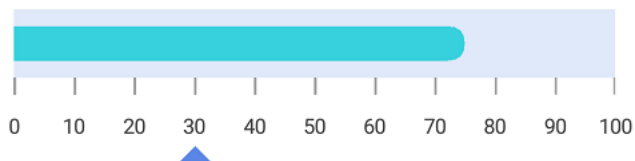
```

## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e9f9");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.ScaleBarSize = 40;
linearScale.LabelFontSize = 14;
linearScale.MinorTicksPerInterval = 0;
linearScale.MajorTickSettings.Color = Color.FromHex("#9E9E9E");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 10;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.Thickness = 20;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.CornerRadius = 10;
barPointer.EnableAnimation = false;
barPointer.Color = Color.FromHex("#36d1dc");
linearScale.Pointers.Add(barPointer);
SymbolPointer symbolPointer = new SymbolPointer();
symbolPointer.Value = 30;
symbolPointer.EnableAnimation = false;
symbolPointer.Offset = 40;
symbolPointer.Color = Color.FromHex("#5b86e5");
linearScale.Pointers.Add(symbolPointer);
linearGauge.Scales.Add(linearScale);

```



## Change symbol pointer shapes

Different types of shapes are used in [SymbolPointer](#) to mark the pointer value in scale. You can change the shape of [SymbolPointer](#) by using the [MarkerShape](#) property in pointer.

## XML

```

<gauge:SfLinearGauge>

```

```

<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarColor="#e0e9f9" LabelColor="#424242" ScaleBarSize
= "40" LabelFontSize="14" MinorTicksPerInterval="0">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="#9E9E9E" Length="10"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.Pointers>
<gauge:BarPointer Value="75" CornerRadiusType="End" CornerRadius="10"
EnableAnimation="false" Color="#36d1dc" Thickness="20"/>
<gauge:SymbolPointer Value="30" EnableAnimation="False" Color="#5b86e5"
SymbolPointerPosition="Away" MarkerShape="Circle"/>
</gauge:LinearScale.Pointers>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

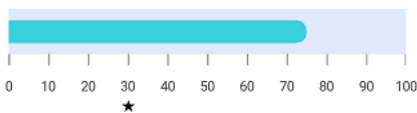
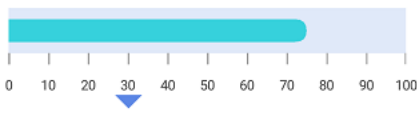
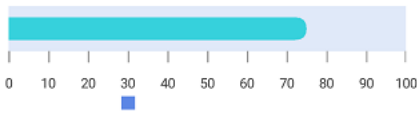
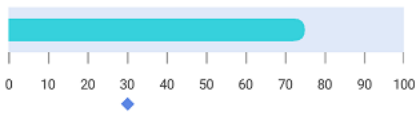
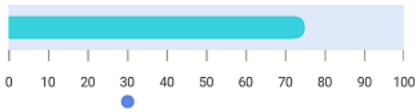
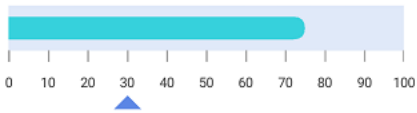
```

**C#**

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e9f9");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.ScaleBarSize = 40;
linearScale.LabelFontSize = 14;
linearScale.MinorTicksPerInterval = 0;
linearScale.MajorTickSettings.Color = Color.FromHex("#9E9E9E");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 10;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.Thickness = 20;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.CornerRadius = 10;
barPointer.EnableAnimation = false;
barPointer.Color = Color.FromHex("#36d1dc");
linearScale.Pointers.Add(barPointer);
SymbolPointer symbolPointer = new SymbolPointer();
symbolPointer.Value = 30;
symbolPointer.EnableAnimation = false;
symbolPointer.Thickness = 12;
symbolPointer.MarkerShape = MarkerShape.Circle;
symbolPointer.SymbolPointerPosition = SymbolPointerPosition.Away;
symbolPointer.Color = Color.FromHex("#5b86e5");
linearScale.Pointers.Add(symbolPointer);
linearGauge.Scales.Add(linearScale);

```



### Setting image shape for symbol pointer

You can achieve the image shape by setting the [MarkerShape](#) property to `Image` and setting image path to [ImageSource](#) property in [SymbolPointer](#).

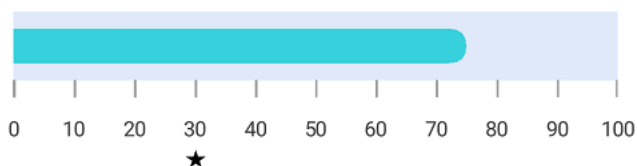
### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e9f9" LabelColor="#424242" ScaleBarSize
    = "40" LabelFontSize="14" MinorTicksPerInterval="0">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="#9E9E9E" Length="10"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.Pointers>
        <gauge:BarPointer Value="75" CornerRadiusType="End" CornerRadius="10"
        EnableAnimation="false" Color="#36d1dc" Thickness="20"/>
        <gauge:SymbolPointer Value="30" EnableAnimation="False" Color="#5b86e5"
        Offset="40" MarkerShape="Image" ImageSource="location.png"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

```
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e9f9");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.ScaleBarSize = 40;
linearScale.LabelFontSize = 14;
linearScale.MinorTicksPerInterval = 0;
linearScale.MajorTickSettings.Color = Color.FromHex("#9E9E9E");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 10;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.Thickness = 20;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.CornerRadius = 10;
barPointer.EnableAnimation = false;
barPointer.Color = Color.FromHex("#36d1dc");
linearScale.Pointers.Add(barPointer);
SymbolPointer symbolPointer = new SymbolPointer();
symbolPointer.Value = 30;
symbolPointer.EnableAnimation = false;
symbolPointer.Thickness = 12;
symbolPointer.MarkerShape = MarkerShape.Image;
symbolPointer.ImageSource = "location.png";
symbolPointer.Offset = 40;
symbolPointer.Color = Color.FromHex("#5b86e5");
linearScale.Pointers.Add(symbolPointer);
linearGauge.Scales.Add(linearScale);
```



### Adding multiple pointers

In addition to the default pointer, you can add n number of pointers to a linear scale by using the [Pointers](#) property.

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e9f9" LabelColor="#424242"
      CornerRadiusType="End" CornerRadius="20" ScaleBarSize = "40" LabelFontSize
      ="14" MinorTicksPerInterval ="0">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color ="#9E9E9E" Length = "10"/>
      </gauge:LinearScale.MajorTickSettings>
```

```

<gauge:LinearScale.Pointers>
<gauge:BarPointer Value="75" CornerRadiusType="End" EnableAnimation =
"false" Color = "#36d1dc" Thickness = "30"/>
<gauge:SymbolPointer Value="30" EnableAnimation="False" Color= "#5b86e5"
SymbolPointerPosition="Away" MarkerShape="Triangle"/>
</gauge:LinearScale.Pointers>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

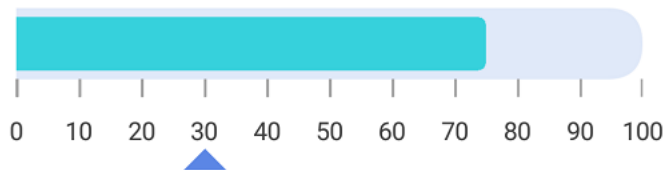
```

## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e9f9");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.ScaleBarSize = 40;
linearScale.CornerRadius = 20;
linearScale.CornerRadiusType = CornerRadiusType.End;
linearScale.LabelFontSize = 14;
linearScale.MinorTicksPerInterval = 0;
linearScale.MajorTickSettings.Color = Color.FromHex("#9E9E9E");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 10;
BarPointer barPointer = new BarPointer();
barPointer.Value = 75;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.Thickness = 30;
barPointer.EnableAnimation = false;
barPointer.Color = Color.FromHex("#36d1dc");
linearScale.Pointers.Add(barPointer);
SymbolPointer symbolPointer = new SymbolPointer();
symbolPointer.Value = 30;
symbolPointer.Thickness = 12;
symbolPointer.EnableAnimation = false;
symbolPointer.Color = Color.FromHex("#5b86e5");
symbolPointer.MarkerShape = MarkerShape.Triangle;
symbolPointer.SymbolPointerPosition = SymbolPointerPosition.Away;
linearScale.Pointers.Add(symbolPointer);
linearGauge.Scales.Add(linearScale);

```



## Change Orientation

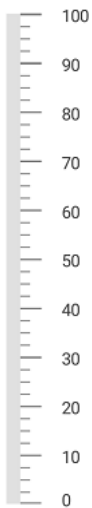
[SfLinearGauge](#) supports horizontal and vertical orientations. By default, [SfLinearGauge](#) is rendered with horizontal orientation. You can change the orientation by using the [Orientation](#) property.

## XML

```
<gauge:SfLinearGauge Orientation="OrientationVertical">
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale Interval="10" ScaleBarLength="350"
      ScaleBarColor="#e0e0e0" LabelColor="#424242">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" />
      </gauge:LinearScale.MajorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

## C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
linearGauge.Orientation = Orientation.OrientationVertical;
LinearScale linearScale = new LinearScale();
linearScale.Interval = 10;
linearScale.ScaleBarLength = 350;
linearScale.MajorTickSettings.Thickness = 1;
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearGauge.Scales.Add(linearScale);
```



## Header

You can add a title to gauge using the [LinearHeader](#) option to provide information to users about the data that is being plotted in the linear gauge.

Adding header to linear gauge

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Header >
    <gauge:LinearHeader Text="Thermometer"/>
  </gauge:SfLinearGauge.Header>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      ScaleBarLength="350">
```

```

<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

```

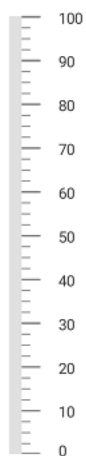
## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
linearGauge.Orientation = Orientation.OrientationVertical;
LinearHeader linearHeader = new LinearHeader();
linearHeader.Text = "Thermometer";
linearGauge.Header = linearHeader;
LinearScale linearScale = new LinearScale();
linearScale.Interval = 10;
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.ScaleBarLength = 350;
linearScale.LabelColor = Color.FromHex("#424242");
linearGauge.Scales.Add(linearScale);

```

Thermometer



## Positioning the header

To change the position of header, use the [Offset](#) property in the linear header. It ranges from 0 to 1. By default, the header will be positioned on the top of linear gauge.

## XML

```

<gauge:SfLinearGauge>

```

```

<gauge:SfLinearGauge.Header >
<gauge:LinearHeader Text="Thermometer" Offset="0.4,0.4"/>
</gauge:SfLinearGauge.Header>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
ScaleBarLength ="350">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

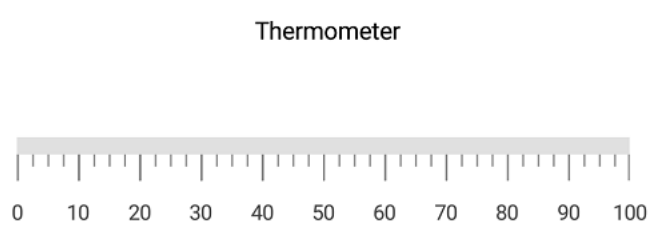
```

## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearHeader linearHeader = new LinearHeader();
linearHeader.Text = "Thermometer";
linearHeader.Offset = new Point(0.4, 0.4);
linearGauge.Header = linearHeader;
LinearScale linearScale = new LinearScale();
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearGauge.Scales.Add(linearScale);

```



## Customizing header text

You can customize the text of [LinearHeader](#) by using the [FontFamily](#), [FontAttribute](#), [TextSize](#), and [ForegroundColor](#) properties as shown in the following code snippet.

## XML

```

<gauge:SfLinearGauge>
<gauge:SfLinearGauge.Header >
<gauge:LinearHeader Text="Thermometer" TextSize="18"
ForegroundColor="DarkCyan" FontAttributes="Bold" Offset="0.35,0.4">
<gauge:LinearHeader.FontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Chalkduster"
Android="algerian.ttf" WinPhone="Chiller" />

```



```

</gauge:LinearHeader.FontFamily>
</gauge:LinearHeader>
</gauge:SfLinearGauge.Header>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242">
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

```

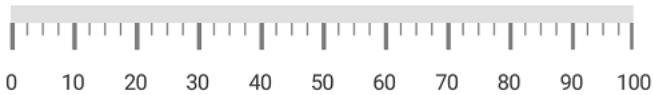
## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearHeader linearHeader = new LinearHeader();
linearHeader.Text = "Thermometer";
linearHeader.ForegroundColor = Color.DarkCyan;
linearHeader.FontFamily = Device.RuntimePlatform == Device.iOS ?
"Chalkduster" : Device.RuntimePlatform == Device.Android ? "algerian.ttf" :
"Chiller";
linearHeader.FontAttributes = FontAttributes.Bold;
linearHeader.TextSize = 18;
linearHeader.Offset = new Point(0.35, 0.4);
linearGauge.Header = linearHeader;
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearGauge.Scales.Add(linearScale);

```

### THERMOMETER



## Annotations

[SfLinearGauge](#) supports [Annotations](#), which is used to mark the specific area of interest in the gauge area with texts, shapes, or images. You can add any number of annotations to the gauge.

### Annotation

By using the [View](#) property of annotation object, you can specify the new element that needs to be displayed in the gauge area.

## XML

```

<gauge:SfLinearGauge>
<gauge:SfLinearGauge.Annotations>
<gauge:LinearGaugeAnnotation OffsetX = "0.5" OffsetY = "0.45">
<gauge:LinearGaugeAnnotation.View>
<Label Text="CPU Utilization" TextColor="Black" FontSize = "20"/>
</gauge:LinearGaugeAnnotation.View>
</gauge:LinearGaugeAnnotation>
</gauge:SfLinearGauge.Annotations>
<gauge:SfLinearGauge.Scales>

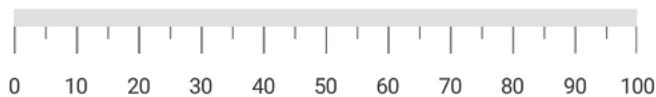
```

```
<gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
MinorTicksPerInterval = "1">
  <gauge:LinearScale.MajorTickSettings>
    <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
  </gauge:LinearScale.MajorTickSettings>
  <gauge:LinearScale.MinorTickSettings>
    <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
  </gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

## C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearGaugeAnnotation linearGaugeAnnotation = new LinearGaugeAnnotation();
linearGaugeAnnotation.OffsetX = 0.5;
linearGaugeAnnotation.OffsetY = 0.45;
linearGaugeAnnotation.View = new Label() { Text = "CPU Utilization",
TextColor = Color.Black, FontSize = 20 };
linearGauge.Annotations.Add(linearGaugeAnnotation);
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.MinorTicksPerInterval = 1;
linearGauge.Scales.Add(linearScale);
```

## CPU Utilization



## Positioning the annotation

You can place the annotation anywhere in gauge area by using the `Offset` or `ScaleValue` property.

### Change annotation position by using offset

You can position the annotation anywhere in the linear gauge by using the `OffsetX` and `OffsetY` properties. It ranges from 0 to 1.

## XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Annotations>
    <gauge:LinearGaugeAnnotation OffsetX = "0.5" OffsetY = "0.6">
      <gauge:LinearGaugeAnnotation.View>
        <Label Text="CPU Utilization" TextColor="Black" FontSize = "20"/>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
  </gauge:SfLinearGauge.Annotations>
</gauge:SfLinearGauge>
```

```

</gauge:LinearGaugeAnnotation>
</gauge:SfLinearGauge.Annotations>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
MinorTicksPerInterval = "1">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

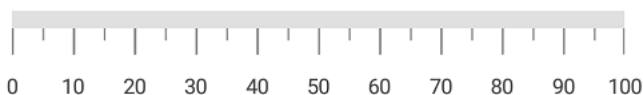
```

## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearGaugeAnnotation linearGaugeAnnotation = new LinearGaugeAnnotation();
linearGaugeAnnotation.OffsetX = 0.5;
linearGaugeAnnotation.OffsetY = 0.6;
linearGaugeAnnotation.View = new Label() { Text = "CPU Utilization",
TextColor = Color.Black, FontSize = 20 };
linearGauge.Annotations.Add(linearGaugeAnnotation);
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.MinorTicksPerInterval = 1;
linearGauge.Scales.Add(linearScale);

```



CPU Utilization

*Change annotation position by using scale value*

You can also place the annotation by specifying the [ScaleValue](#) property.

## XML

```

<gauge:SfLinearGauge>
<gauge:SfLinearGauge.Annotations>
<gauge:LinearGaugeAnnotation ScaleValue = "60">
<gauge:LinearGaugeAnnotation.View>
<Label Text="60%" FontAttributes = "Bold" TextColor="Black" FontSize = "15"/>
</gauge:LinearGaugeAnnotation.View>

```

```

</gauge:LinearGaugeAnnotation>
</gauge:SfLinearGauge.Annotations>
<gauge:SfLinearGauge.Scales>
<gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
MinorTicksPerInterval = "1" ScaleBarSize = "40">
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
<gauge:LinearScale.Pointers>
<gauge:BarPointer Value="60" Color = "#f95c85" Thickness = "20"
CornerRadius = "10" CornerRadiusType = "End"/>
</gauge:LinearScale.Pointers>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

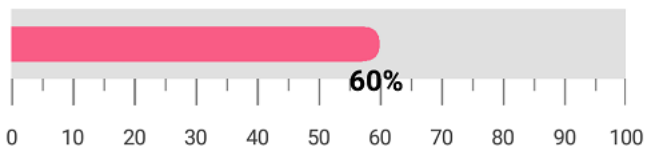
```

## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearGaugeAnnotation linearGaugeAnnotation = new LinearGaugeAnnotation();
linearGaugeAnnotation.ScaleValue = 60;
linearGaugeAnnotation.View = new Label() { Text = "60%", FontAttributes =
FontAttributes.Bold, TextColor = Color.Black, FontSize = 15 };
linearGauge.Annotations.Add(linearGaugeAnnotation);
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.ScaleBarSize = 40;
linearScale.MinorTicksPerInterval = 1;
linearGauge.Scales.Add(linearScale);
BarPointer barPointer = new BarPointer();
barPointer.Value = 60;
barPointer.Thickness = 20;
barPointer.CornerRadius = 10;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.Color = Color.FromRgb(249, 92, 133);
linearScale.Pointers.Add(barPointer);

```



### Set margin to the annotation

You can adjust the annotation by specifying the [ViewMargin](#) property in pixel, which adjusts the annotation element from its current position.

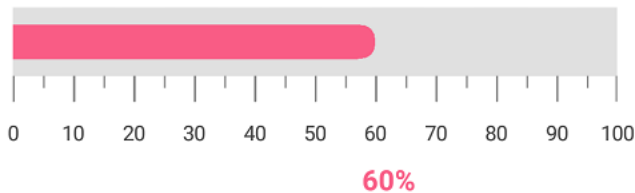
#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Annotations>
    <gauge:LinearGaugeAnnotation ScaleValue = "60" ViewMargin = "10,60">
      <gauge:LinearGaugeAnnotation.View>
        <Label Text="60%" FontAttributes = "Bold" TextColor= "#F95C85" FontSize
          ="15"/>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
  </gauge:SfLinearGauge.Annotations>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      MinorTicksPerInterval = "1" ScaleBarSize = "40">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
      <gauge:LinearScale.Pointers>
        <gauge:BarPointer Value="60" Color = "#F95C85" Thickness = "20"
          CornerRadius = "10" CornerRadiusType = "End"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

#### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearGaugeAnnotation linearGaugeAnnotation = new LinearGaugeAnnotation();
linearGaugeAnnotation.ScaleValue = 60;
linearGaugeAnnotation.ViewMargin = new Point(10, 60);
linearGaugeAnnotation.View = Label() { Text = "60%", FontAttributes =
FontAttributes.Bold, TextColor = Color.FromHex("#F95C85"), FontSize = 15 };
linearGauge.Annotations.Add(linearGaugeAnnotation);
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.ScaleBarSize = 40;
linearScale.MinorTicksPerInterval = 1;
linearGauge.Scales.Add(linearScale);
BarPointer barPointer = new BarPointer();
barPointer.Value = 60;
barPointer.Thickness = 20;
```

```
barPointer.CornerRadius = 10;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.Color = Color.FromRgb(249, 92, 133);
linearScale.Pointers.Add(barPointer);
```



### Alignment of annotation

You can align the annotation using the [HorizontalViewAlignment](#) and [VerticalViewAlignment](#) properties.

#### Setting horizontal view alignment

##### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Annotations>
    <gauge:LinearGaugeAnnotation ScaleValue="60" HorizontalViewAlignment="Start">
      <gauge:LinearGaugeAnnotation.View>
        <Label Text="60%" FontAttributes="Bold" TextColor="Black" FontSize="15"/>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
  </gauge:SfLinearGauge.Annotations>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      MinorTicksPerInterval="1" ScaleBarSize="40">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
      <gauge:LinearScale.Pointers>
        <gauge:BarPointer Value="60" Color="#f95c85" Thickness="20"
          CornerRadius="10" CornerRadiusType="End"/>
      </gauge:LinearScale.Pointers>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

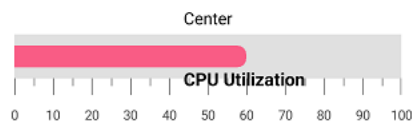
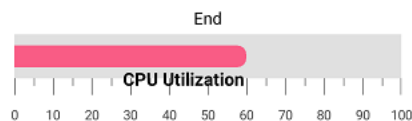
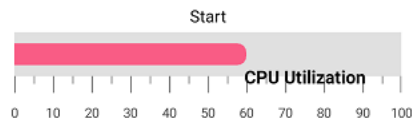
##### C#

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearGaugeAnnotation linearGaugeAnnotation = new LinearGaugeAnnotation();
linearGaugeAnnotation.ScaleValue = 60;
linearGaugeAnnotation.HorizontalViewAlignment = ViewAlignment.Start;
linearGaugeAnnotation.View = new Label() { Text = "60%", FontAttributes =
  FontAttributes.Bold, TextColor = Color.Black, FontSize = 15 };
linearGauge.Annotations.Add(linearGaugeAnnotation);
```

```

LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.ScaleBarSize = 40;
linearScale.MinorTicksPerInterval = 1;
linearGauge.Scales.Add(linearScale);
BarPointer barPointer = new BarPointer();
barPointer.Value = 60;
barPointer.Thickness = 20;
barPointer.CornerRadius = 10;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.Color = Color.FromRgb(249, 92, 133);
linearScale.Pointers.Add(barPointer);

```



### Setting vertical view alignment

#### XML

```

<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Annotations>
    <gauge:LinearGaugeAnnotation ScaleValue="60" VerticalViewAlignment="Start">
      <gauge:LinearGaugeAnnotation.View>
        <Label Text="60%" FontAttributes="Bold" TextColor="Black" FontSize="15"/>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
  </gauge:SfLinearGauge.Annotations>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      MinorTicksPerInterval="1" ScaleBarSize="40">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>

```

```

</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
</gauge:LinearScale.MinorTickSettings>
<gauge:LinearScale.Pointers>
<gauge:BarPointer Value="60" Color = "#f95c85" Thickness = "20"
CornerRadius ="10" CornerRadiusType ="End"/>
</gauge:LinearScale.Pointers>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

```

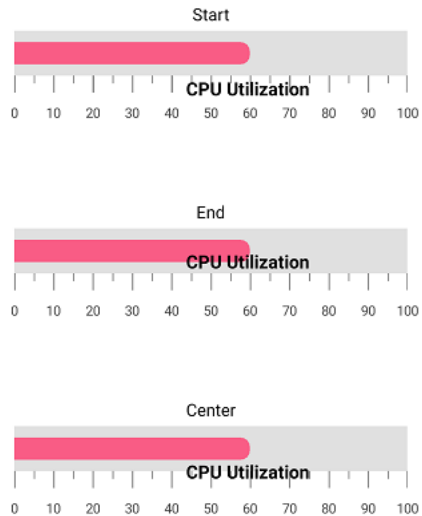
## C#

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearGaugeAnnotation linearGaugeAnnotation = new LinearGaugeAnnotation();
linearGaugeAnnotation.ScaleValue = 60;
linearGaugeAnnotation.VerticalViewAlignment = ViewAlignment.Start;
linearGaugeAnnotation.View = new Label() { Text = "60%", FontAttributes =
FontAttributes.Bold, TextColor = Color.Black, FontSize = 15 };
linearGauge.Annotations.Add(linearGaugeAnnotation);
LinearScale linearScale = new LinearScale();
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.ScaleBarSize = 40;
linearScale.MinorTicksPerInterval = 1;
linearGauge.Scales.Add(linearScale);
BarPointer barPointer = new BarPointer();
barPointer.Value = 60;
barPointer.Thickness = 20;
barPointer.CornerRadius = 10;
barPointer.CornerRadiusType = CornerRadiusType.End;
barPointer.Color = Color.FromRgb(249, 92, 133);
linearScale.Pointers.Add(barPointer);

```





Setting scale index for annotation

You can set the index for the scale by using [ScaleIndex](#)

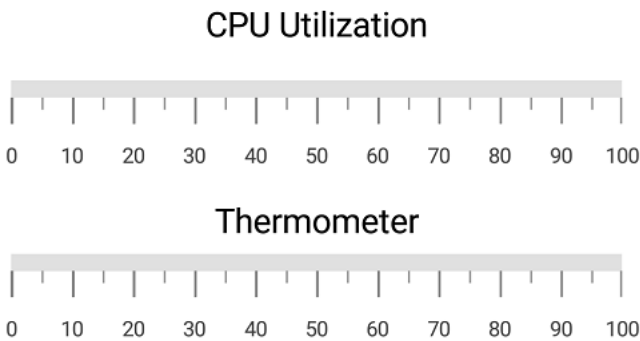
#### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Annotations>
    <gauge:LinearGaugeAnnotation OffsetX = "0.5" OffsetY ="0.45" ScaleIndex =
      "0">
      <gauge:LinearGaugeAnnotation.View>
        <Label Text="CPU Utilization" TextColor="Black" FontSize ="20"/>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
    <gauge:LinearGaugeAnnotation OffsetX = "0.5" OffsetY ="0.6" ScaleIndex ="1">
      <gauge:LinearGaugeAnnotation.View>
        <Label Text="Thermometer" TextColor="Black" FontSize ="20"/>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
  </gauge:SfLinearGauge.Annotations>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      MinorTicksPerInterval ="1">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
    <gauge:LinearScale ScaleBarColor="#e0e0e0" LabelColor="#424242"
      MinorTicksPerInterval ="1" Offset ="100">
      <gauge:LinearScale.MajorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="15"/>
      </gauge:LinearScale.MajorTickSettings>
      <gauge:LinearScale.MinorTickSettings>
        <gauge:LinearTickSettings Thickness="1" Color="Gray" Length="7"/>
      </gauge:LinearScale.MinorTickSettings>
    </gauge:LinearScale>
  </gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>
```

```
</gauge:SfLinearGauge>
```

**C#**

```
SfLinearGauge linearGauge = new SfLinearGauge();
LinearGaugeAnnotation linearGaugeAnnotation = new LinearGaugeAnnotation();
linearGaugeAnnotation.OffsetX = 0.5;
linearGaugeAnnotation.OffsetY = 0.45;
linearGaugeAnnotation.ScaleIndex = 0;
linearGaugeAnnotation.View = new Label() { Text = "CPU Utilization",
TextColor = Color.Black, FontSize = 18 };
linearGauge.Annotations.Add(linearGaugeAnnotation);
LinearGaugeAnnotation linearGaugeAnnotation1 = new LinearGaugeAnnotation();
linearGaugeAnnotation1.OffsetX = 0.5;
linearGaugeAnnotation1.OffsetY = 0.6;
linearGaugeAnnotation1.ScaleIndex = 1;
linearGaugeAnnotation1.View = new Label() { Text = "Thermometer", TextColor
= Color.Black, FontSize = 18 };
linearGauge.Annotations.Add(linearGaugeAnnotation1);
LinearScale linearScale = new LinearScale();
linearScale.MinimumValue = 0;
linearScale.MaximumValue = 100;
linearScale.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale.LabelColor = Color.FromHex("#424242");
linearScale.MajorTickSettings.Thickness = 1;
linearScale.MajorTickSettings.Length = 15;
linearScale.MajorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Color = Color.Gray;
linearScale.MinorTickSettings.Length = 7;
linearScale.MinorTickSettings.Thickness = 1;
linearScale.MinorTicksPerInterval = 1;
linearGauge.Scales.Add(linearScale);
LinearScale linearScale1 = new LinearScale();
linearScale1.MinimumValue = 0;
linearScale1.MaximumValue = 100;
linearScale1.ScaleBarColor = Color.FromHex("#e0e0e0");
linearScale1.LabelColor = Color.FromHex("#424242");
linearScale1.MajorTickSettings.Thickness = 1;
linearScale1.MajorTickSettings.Length = 15;
linearScale1.MajorTickSettings.Color = Color.Gray;
linearScale1.MinorTickSettings.Color = Color.Gray;
linearScale1.MinorTickSettings.Length = 7;
linearScale1.MinorTickSettings.Thickness = 1;
linearScale1.Offset = 100;
linearScale1.MinorTicksPerInterval = 1;
linearGauge.Scales.Add(linearScale1);
```



### Multiple annotations

You can add multiple annotations to the gauge as demonstrated below.

### XML

```
<gauge:SfLinearGauge>
  <gauge:SfLinearGauge.Annotations>
    <gauge:LinearGaugeAnnotation ScaleValue="15" ViewMargin ="0,30" >
      <gauge:LinearGaugeAnnotation.View>
        <Image Source="Low.png" WidthRequest="30" HeightRequest="30"/>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
    <gauge:LinearGaugeAnnotation ScaleValue="45" ViewMargin ="0,30">
      <gauge:LinearGaugeAnnotation.View>
        <Image Source="Moderate.png" WidthRequest="30" HeightRequest="30"/>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
    <gauge:LinearGaugeAnnotation ScaleValue="75" ViewMargin ="0,30">
      <gauge:LinearGaugeAnnotation.View>
        <Image Source="High.png" WidthRequest="30" HeightRequest="30"/>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
    <gauge:LinearGaugeAnnotation ScaleValue="15" ViewMargin ="0,80">
      <gauge:LinearGaugeAnnotation.View>
        <Label Text="Low" TextColor="#30b32d" FontSize ="18"></Label>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
    <gauge:LinearGaugeAnnotation ScaleValue="45" ViewMargin ="0,80">
      <gauge:LinearGaugeAnnotation.View>
        <Label Text="Moderate" TextColor="#ffdd00" FontSize ="18"></Label>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
    <gauge:LinearGaugeAnnotation ScaleValue="75" ViewMargin ="0,80">
      <gauge:LinearGaugeAnnotation.View>
        <Label Text="High" TextColor="#f03e3e" FontSize ="18"></Label>
      </gauge:LinearGaugeAnnotation.View>
    </gauge:LinearGaugeAnnotation>
  </gauge:SfLinearGauge.Annotations>
  <gauge:SfLinearGauge.Scales>
    <gauge:LinearScale MinimumValue="0" MaximumValue="90"
      ShowLabels="False" ScaleBarColor="Transparent"
      MinorTicksPerInterval="1" ScaleBarSize="13" ScalePosition="BackWard"
      ShowTicks ="False">
  </gauge:LinearScale>
</gauge:SfLinearGauge>
```

```

<gauge:LinearScale.Ranges>
<gauge:LinearRange StartValue="0" Color="#30b32d" EndValue="30"
StartWidth="60" EndWidth="60" />
<gauge:LinearRange StartValue="30" Color="#ffdd00" EndValue="60"
StartWidth="60" EndWidth="60" />
<gauge:LinearRange StartValue="60" Color="#f03e3e" EndValue="90"
StartWidth="60" EndWidth="60" />
</gauge:LinearScale.Ranges>
<gauge:LinearScale.Pointers>
<gauge:SymbolPointer Color="Red" MarkerShape="InvertedTriangle" Value="35"
Thickness="12" />
</gauge:LinearScale.Pointers>
<gauge:LinearScale.MajorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Transparent"
Length="0"></gauge:LinearTickSettings>
</gauge:LinearScale.MajorTickSettings>
<gauge:LinearScale.MinorTickSettings>
<gauge:LinearTickSettings Thickness="1" Color="Transparent"
Length="0"></gauge:LinearTickSettings>
</gauge:LinearScale.MinorTickSettings>
</gauge:LinearScale>
</gauge:SfLinearGauge.Scales>
</gauge:SfLinearGauge>

```

**C#**

```

SfLinearGauge linearGauge = new SfLinearGauge();
LinearGaugeAnnotation linearGaugeAnnotation = new LinearGaugeAnnotation();
linearGaugeAnnotation.ScaleValue = 75;
linearGaugeAnnotation.ViewMargin = new Point(0, 30);
linearGaugeAnnotation.View = new Image() { Source = "High.png",
HeightRequest = 30, WidthRequest = 30 };
linearGauge.Annotations.Add(linearGaugeAnnotation);
LinearGaugeAnnotation linearGaugeAnnotation1 = new LinearGaugeAnnotation();
linearGaugeAnnotation1.ScaleValue = 45;
linearGaugeAnnotation1.ViewMargin = new Point(0, 30);
linearGaugeAnnotation1.View = new Image() { Source = "Moderate.png",
HeightRequest = 30, WidthRequest = 30 };
linearGauge.Annotations.Add(linearGaugeAnnotation1);
LinearGaugeAnnotation linearGaugeAnnotation2 = new LinearGaugeAnnotation();
linearGaugeAnnotation2.ScaleValue = 15;
linearGaugeAnnotation2.ViewMargin = new Point(0, 30);
linearGaugeAnnotation2.View = new Image() { Source = "Low.png",
HeightRequest = 30, WidthRequest = 30 };
linearGauge.Annotations.Add(linearGaugeAnnotation2);
LinearGaugeAnnotation linearGaugeAnnotation3 = new LinearGaugeAnnotation();
linearGaugeAnnotation3.ScaleValue = 75;
linearGaugeAnnotation3.ViewMargin = new Point(0, 80);
linearGaugeAnnotation3.View = new Label() { Text = "High", TextColor =
Color.Red, FontSize = 18 };
linearGauge.Annotations.Add(linearGaugeAnnotation3);
LinearGaugeAnnotation linearGaugeAnnotation4 = new LinearGaugeAnnotation();
linearGaugeAnnotation4.ScaleValue = 45;
linearGaugeAnnotation4.ViewMargin = new Point(0, 80);
linearGaugeAnnotation4.View = new Label() { Text = "Moderate", TextColor =
Color.Yellow, FontSize = 18 };

```

```

linearGauge.Annotations.Add(linearGaugeAnnotation4);
LinearGaugeAnnotation linearGaugeAnnotation5 = new LinearGaugeAnnotation();
linearGaugeAnnotation5.ScaleValue = 15;
linearGaugeAnnotation5.ViewMargin = new Point(0, 80);
linearGaugeAnnotation5.View = new Label() { Text = "Low", TextColor =
Color.Green, FontSize = 18 };
linearGauge.Annotations.Add(linearGaugeAnnotation5);
LinearScale linearScale = new LinearScale();
linearScale.MinimumValue = 0;
linearScale.MaximumValue = 90;
linearScale.ShowLabels = false;
linearScale.ScaleBarColor = Color.Transparent;
linearScale.MinorTicksPerInterval = 1;
linearScale.ShowTicks = false;
linearScale.ScalePosition = ScalePosition.BackWard;
linearGauge.Scales.Add(linearScale);
LinearRange linearRange = new LinearRange();
linearRange.StartValue = 0;
linearRange.Color = Color.FromHex("#30b32d");
linearRange.EndValue = 30;
linearRange.StartWidth = 60;
linearRange.EndWidth = 60;
linearScale.Ranges.Add(linearRange);
LinearRange linearRange1 = new LinearRange();
linearRange1.StartValue = 30;
linearRange1.Color = Color.FromHex("#fdd00");
linearRange1.EndValue = 60;
linearRange1.StartWidth = 60;
linearRange1.EndWidth = 60;
linearScale.Ranges.Add(linearRange1);
LinearRange linearRange2 = new LinearRange();
linearRange2.StartValue = 60;
linearRange2.Color = Color.FromHex("#f03e3e");
linearRange2.EndValue = 90;
linearRange2.StartWidth = 60;
linearRange2.EndWidth = 60;
linearScale.Ranges.Add(linearRange2);

```



## SfListView

### SfListView

The SfListView for Xamarin.Forms renders set of data items with Xamarin.Forms views or custom templates. Data can be grouped, sorted, and filtered with ease.

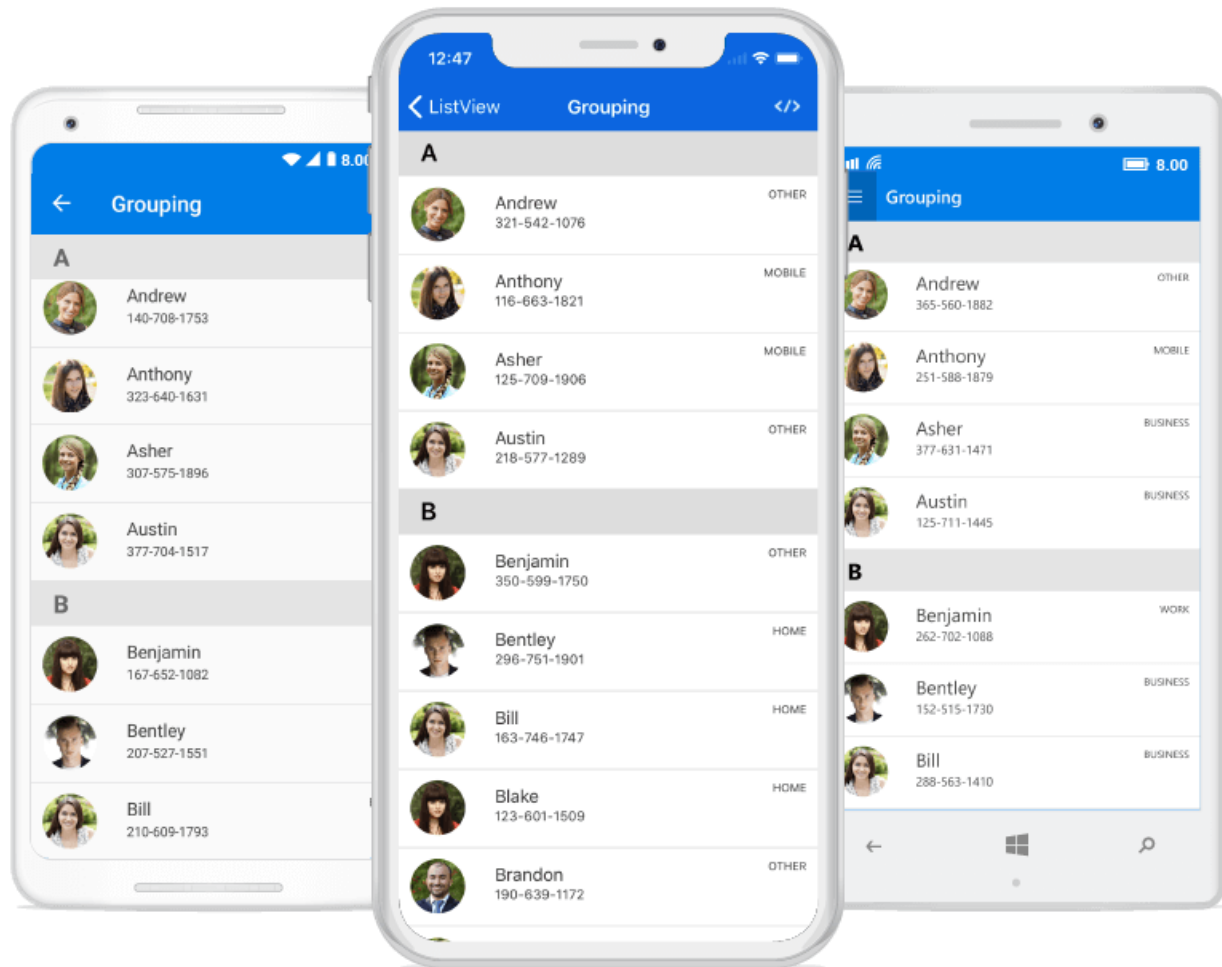
#### Key features

- Optimized view reusing strategy for enhanced performance.

- Item templating for rich UI(User Interface).
- Supports dynamic selection of UI for each item using the `DataTemplateSelector`.
- AutoFit items based on the content of 'ItemTemplate'.
- Supports linear layout and grid layout.
- Reordering items by dragging and dropping them in a linear layout with displaying custom UI in a template.
- Orientation support.
- Data operations such as sorting, grouping, and filtering.
- Customizable group header with option to stick in view.
- Selection with different selection modes and gestures.
- Swiping template for loading views with custom actions.
- Header and footer with sticky options.
- Supports to load more data either automatically or manually at bottom when end of the list is reached and manually at top.
- Supports for macOS with existing features along with key navigation.

#### Advantages of the SfListView over Xamarin.Forms ListView

- Horizontal orientation support.
- Support for different layouts such as linear layout and grid layout.
- Supports for item swiping.
- Supports for reordering items by drag and drop in linear layout.
- Supports to load more items when reached scroll end.
- Supports for sorting and filtering the data items.
- Supports to stick the header, group header, and footer in view.
- Supports to customize selection background color.
- Supports to notify the scroll state changes.



## Getting Started

This section provides a quick overview for getting started with the SfListView for Xamarin.Forms. Walk through the entire process of creating the real world SfListView.

### Assembly Deployment

After installing Essential Studio for Xamarin, you can find all the required assemblies in the {Syncfusion Essential Studio Installed location}\Essential Studio\{{ site.releaseversion }}\Xamarin\lib installation folder.

Eg: C:\Program Files (x86)\Syncfusion\Essential Studio\{{ site.releaseversion }}\Xamarin\lib

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

---

**Note:** Assemblies can be found in an unzipped package location in Mac.

---

### Adding SfListView reference

You can add SfListView reference using one of the following methods:

#### Method 1: Adding SfListView reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfListView). To add SfListView to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfListView](https://www.nuget.org/packages/Syncfusion.Xamarin.SfListView), and then install it.

![Adding SfListView reference from NuGet](SfListView\_images/Adding SfListView reference.png)

**Note:** Install the same version of SfListView NuGet in all the projects.

### Method 2: Adding SfListView reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfListView control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfListView assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.DataSource.Portable.dll Syncfusion.Core.XForms.dll Syncfusion.GridCommon.Portable.dll Syncfusion.SfListView.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.DataSource.Portable.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.GridCommon.Portable.dll Syncfusion.SfListView.XForms.Android.dll Syncfusion.SfListView.XForms.dll Syncfusion.Licensing.dll
iOS	Syncfusion.DataSource.Portable.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.GridCommon.Portable.dll Syncfusion.SfListView.XForms.iOS.dll Syncfusion.SfListView.XForms.dll Syncfusion.Licensing.dll
UWP	Syncfusion.DataSource.Portable.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.GridCommon.Portable.dll Syncfusion.SfListView.XForms.UWP.dll Syncfusion.SfListView.XForms.dll Syncfusion.Licensing.dll
macOS	pcl\Syncfusion.DataSource.Portable.dll pcl\Syncfusion.Core.XForms.dll pcl\Syncfusion.GridCommon.Portable.dll



	pcl\Syncfusion.SfListView.XForms.dll macOS\Syncfusion.Core.XForms.macOS.dll macOS\Syncfusion.SfListView.XForms.macOS Syncfusion.Licensing.dll
--	--

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** After adding the reference, an additional step is required for [iOS](#) and [UWP](#) projects. For UWP alone, one more additional step is required if the project is built-in release mode with .NET Native tool chain is enabled. You can refer to this [KB article](#) for more details. If you are adding the references from toolbox, this step is not needed.

### System Requirements

In the SfListView, current supported Xamarin.Forms version is 3.5.0.129452 and later.

The following table lists the platforms supported versions:

Platform	Supported versions
Android	API level 19 and later versions
iOS	iOS 9.0 and later versions
UWP	Windows 10 devices
macOS	10.11

### Launching the SfListView on Each Platform

To use this control inside an application, each platform application must initialize the SfListView renderer. This initialization step varies from platform to platform, and is discussed in the following sections:

#### Android

The Android launches the SfListView without any initialization, and is enough to only initialize the Xamarin.Forms Framework to launch the application.

#### iOS

To launch the SfListView in iOS, call the `SfListViewRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework initialization, and before the `LoadApplication` is called as demonstrated in the following code example:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
```

```
{
...
global::Xamarin.Forms.Forms.Init ();
SfListViewRenderer.Init();
LoadApplication (new App ());
...
}
```

### Universal Windows Platform (UWP)

To launch the SfListView in UWP, call the `SfListViewRenderer.Init()` in the `MainPage` constructor before the `LoadApplication` is called as demonstrated in the following code example:

#### C#

```
public MainPage ()
{
...
SfListViewRenderer.Init();
LoadApplication (new App ());
...
}
```

### macOS

To launch SfListView in macOS, call the `SfListViewRenderer.Init()` in the `DidFinishLaunching` override method of `AppDelegate` class, after calling the Xamarin.Forms Framework and `LoadApplication` initialization.

#### C#

```
public override void DidFinishLaunching(NSNotification notification)
{
Forms.Init();
LoadApplication(new App());
SfListViewRenderer.Init();
...
}
```

### ReleaseMode Issue in UWP Platform

The known Framework issue in UWP platform is the custom controls will not render when deployed the application in `Release Mode`.

The above problem can be resolved by initializing the SfListView assemblies in `App.xaml.cs` in UWP project as in the following code snippet:

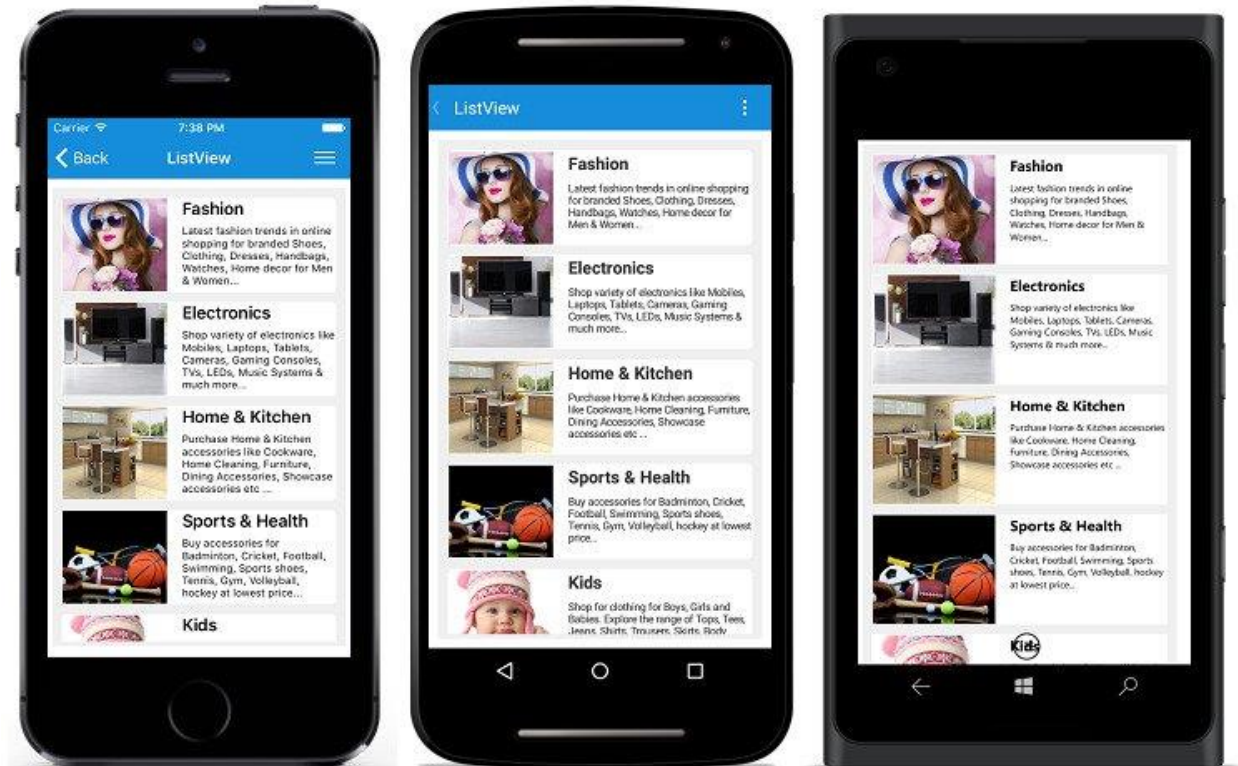
#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
...
rootFrame.NavigationFailed += OnNavigationFailed;
// you'll need to add `using System.Reflection;`
List<Assembly> assembliesToInclude = new List<Assembly>();
//Now, add all the assemblies your app uses
```

```
assembliesToInclude.Add(typeof(SfListViewRenderer).GetTypeInfo().Assembly);  
// replaces Xamarin.Forms.Forms.Init(e);  
Xamarin.Forms.Forms.Init(e, assembliesToInclude);  
...  
}
```

## Create a Simple SfListView

This section explains how to create a SfListView, and configure it. The SfListView control can be configured entirely in C# code, or by using XAML markup. This is how the control will look like on iOS, Android, and Windows devices.



In this walk through, you will create a new application with the SfListView that includes the following topics:

- [Creating the project](#)
- [Adding SfListView in Xamarin.Forms](#)
- [Creating Data Model](#)
- [Binding data](#)
- [Defining an ItemTemplate](#)
- [Layouts](#)
- [Sorting](#)
- [Filtering](#)
- [Grouping](#)
- [Selection](#)
- [Header and Footer](#)

### Creating the Project

Create a new blank [\(Xamarin.Forms.NET Standard\) application](#) in Xamarin Studio or Visual Studio for Xamarin.Forms.

### Adding the SfListView in Xamarin.Forms

1. Add the required assembly references to the corresponding projects as discussed in the [Assembly deployment](#) section.
2. Import the SfListView control namespace Syncfusion.ListView.XForms.
3. Set the SfListView control to the ContentPage.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.MainPage">
<syncfusion:SfListView x:Name="listView" />
</ContentPage>
```

### C#

```
using Syncfusion.ListView.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public class App : Application
    {
        SfListView listView;
        public App()
        {
            listView = new SfListView();
            MainPage = new ContentPage { Content = listView };
        }
    }
}
```

### Creating Data Model for the SfListView

Create a data model to bind it to the control.

Create a simple data source as shown in the following code example in a new class file, and save it as BookInfo.cs file:

### C#

```
public class BookInfo : INotifyPropertyChanged
{
    private string bookName;
    private string bookDesc;
    public string BookName
    {
```

```
get { return bookName; }
set
{
    bookName = value;
    OnPropertyChanged("BookName");
}
}
public string BookDescription
{
    get { return bookDesc; }
    set
    {
        bookDesc = value;
        OnPropertyChanged("BookDescription");
    }
}
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string name)
{
    if (this.PropertyChanged != null)
        this.PropertyChanged(this, new PropertyChangedEventArgs(name));
}
}
```

**Note:** If you want your data model to respond to property changes, then implement `INotifyPropertyChanged` interface in your model class.

Create a model repository class with `BookInfo` 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 `BookInfoRepository.cs` file:

### C#

```
public class BookInfoRepository
{
    private ObservableCollection<BookInfo> bookInfo;
    public ObservableCollection<BookInfo> BookInfo
    {
        get { return bookInfo; }
        set { this.bookInfo = value; }
    }
    public BookInfoRepository()
    {
        GenerateBookInfo();
    }
    internal void GenerateBookInfo()
    {
        bookInfo = new ObservableCollection<BookInfo>();
        bookInfo.Add(new BookInfo() { BookName = "Object-Oriented Programming in C#", BookDescription = "Object-oriented programming is a programming paradigm based on the concept of objects" });
        bookInfo.Add(new BookInfo() { BookName = "C# Code Contracts", BookDescription = "Code Contracts provide a way to convey code assumptions" });
    }
}
```

```
bookInfo.Add(new BookInfo() { BookName = "Machine Learning Using C#",  
BookDescription = "You'll learn several different approaches to applying  
machine learning" });  
bookInfo.Add(new BookInfo() { BookName = "Neural Networks Using C#",  
BookDescription = "Neural networks are an exciting field of software  
development" });  
bookInfo.Add(new BookInfo() { BookName = "Visual Studio Code",  
BookDescription = "It is a powerful tool for editing code and serves for  
end-to-end programming" });  
bookInfo.Add(new BookInfo() { BookName = "Android Programming",  
BookDescription = "It is provides a useful overview of the Android  
application life cycle" });  
bookInfo.Add(new BookInfo() { BookName = "iOS Succinctly", BookDescription =  
"It is for developers looking to step into frightening world of iPhone" });  
bookInfo.Add(new BookInfo() { BookName = "Visual Studio 2015",  
BookDescription = "The new version of the widely-used integrated development  
environment" });  
bookInfo.Add(new BookInfo() { BookName = "Xamarin.Forms", BookDescription =  
"Its creates mappings from its C# classes and controls directly" });  
bookInfo.Add(new BookInfo() { BookName = "Windows Store Apps",  
BookDescription = "Windows Store apps present a radical shift in Windows  
development" });  
}  
}
```

### Binding Data to the SfListView

To bind the data source of the SfListView, set the [SfListView.ItemsSource](#) property as shown as follows. You can bind the data source of the SfListView either from XAML or in code.

The following code example binds the collection created in previous step to the `SfListView.ItemsSource` property:

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:syncfusion="clr-  
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"  
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"  
x:Class="GettingStarted.MainPage">  
  <ContentPage.BindingContext>  
    <local:BookInfoRepository />  
  </ContentPage.BindingContext>  
  <syncfusion:SfListView x:Name="listView"  
    ItemsSource="{Binding BookInfo}" />  
</ContentPage>
```

#### C#

```
BookInfoRepository viewModel = new BookInfoRepository ();  
listView.ItemsSource = viewModel.BookInfo;
```

### Defining an ItemTemplate

By defining the [SfListView.ItemTemplate](#) of the SfListView, a custom user interface(UI) can be achieved to display the data items.

#### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding BookInfo}"
ItemSize="100">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Grid Padding="10">
<Grid.RowDefinitions>
<RowDefinition Height="0.4*" />
<RowDefinition Height="0.6*" />
</Grid.RowDefinitions>
<Label Text="{Binding BookName}" FontAttributes="Bold" TextColor="Teal"
FontSize="21" />
<Label Grid.Row="1" Text="{Binding BookDescription}" TextColor="Teal"
FontSize="15"/>
</Grid>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</ContentPage>
```

#### C#

```
using Syncfusion.ListView.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public class App : Application
    {
        SfListView listView;
        public App()
        {
            BookInfoRepository viewModel = new BookInfoRepository ();
            listView = new SfListView();
            listView.ItemSize = 100;
            listView.ItemsSource = viewModel.BookInfo;
            listView.ItemTemplate = new DataTemplate(() => {
                var grid = new Grid();
                var bookName = new Label { FontAttributes = FontAttributes.Bold,
                    BackgroundColor = Color.Teal, FontSize = 21 };
                bookName.SetBinding(Label.TextProperty, new Binding("BookName"));
                var bookDescription = new Label { BackgroundColor = Color.Teal, FontSize =
                    15 };
                bookDescription.SetBinding(Label.TextProperty, new
                    Binding("BookDescription"));
                grid.Children.Add(bookName);
                grid.Children.Add(bookDescription, 1, 0);
                return grid;
            });
        }
    }
}
```

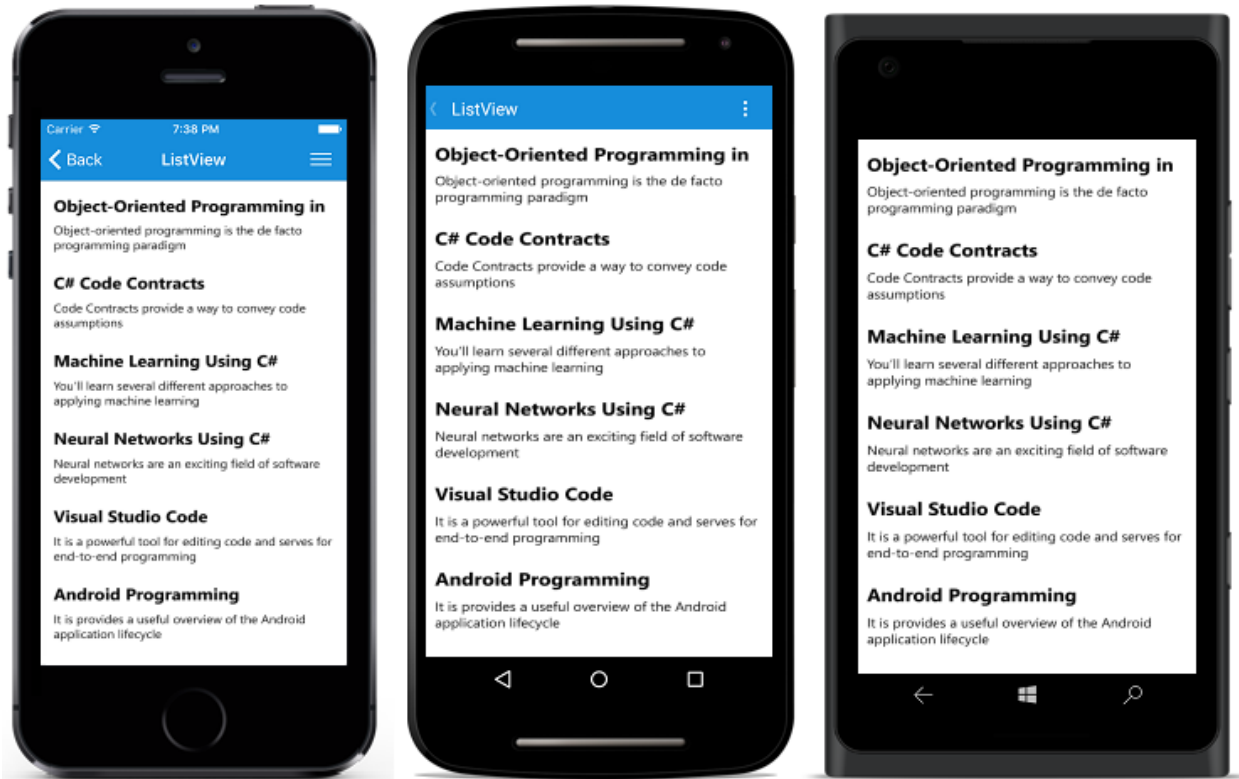
```

MainPage = new ContentPage { Content = listView };
}
}
}

```

Now, run the application to render the below output:

You can also download the entire source code of this demo [here](#).



## Layouts

SfListView supports different layouts such as linear layout and grid layout. The linear layout arranges the items in a single column, whereas the grid layout arranges the items in a predefined number of columns defined by the [SpanCount](#) property of [GridLayout](#).

The [SfListView.LayoutManager](#) property is used to define the layout of the SfListView. [LinearLayout](#) is default layout of this control.

## XML

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView"
    ItemsSource="{Binding BookInfo}"
    ItemSize="100">
    <syncfusion:SfListView.LayoutManager>
      <syncfusion:GridLayout SpanCount="3" />
    </syncfusion:SfListView.LayoutManager>
  </syncfusion:SfListView>
</ContentPage>

```



## C#

```
listView.LayoutManager = new GridLayout() { SpanCount = 3 };
```

## Sorting

The SfListView allows sorting on its data by using the [SfListView.DataSource.SortDescriptors](#) property. Create [SortDescriptor](#) for the property to be sorted, and add it into the [DataSource.SortDescriptors](#) collection.

Refresh the view by calling [SfListView.RefreshView](#) method.

SortDescriptor object holds the following three properties:

- [PropertyName](#): Describes name of the sorted property.
- [Direction](#): Describes an object of type [ListSortDirection](#) defines the sorting direction.
- [Comparer](#): Describes a comparer to be applied when sorting takes place.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.MainPage">
<syncfusion:SfListView x:Name="listView">
<syncfusion:SfListView.DataSource>
<data:DataSource>
<data:DataSource.SortDescriptors>
<data:SortDescriptor PropertyName="BookName" Direction="Ascending"/>
</data:DataSource.SortDescriptors>
</data:DataSource>
</syncfusion:SfListView.DataSource>
</syncfusion:SfListView>
</ContentPage>
```

## C#

```
listView.DataSource.SortDescriptors.Add(new SortDescriptor()
{
    PropertyName = "BookName",
    Direction = ListSortDirection.Ascending,
});
listView.RefreshView();
```

## Filtering

The SfListView supports to filter the records in view by setting predicate to the [SfListView.DataSource.Filter](#) property. Call the [DataSource.RefreshFilter](#) method after assigning the Filter property for refreshing the view.

To filter the items based on the Title property of the underlying data by using `FilterContacts` method, follow the code example:

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<SearchBar x:Name="filterText" HeightRequest="40"
Placeholder="Search here to filter"
TextChanged="OnFilterTextChanged" Grid.Row="0"/>
<syncfusion:SfListView x:Name="listView" Grid.Row="1" ItemsSource="{Binding
BookInfo}"/>
</Grid>
</ContentPage>
```

### C#

```
var grid = new Grid();
var viewModel = new BookInfoRepository ();
var searchBar = new SearchBar() { Placeholder = "Search here to filter" };
searchBar.TextChanged += OnFilterTextChanged;
listView = new SfListView();
listView.ItemsSource = viewModel.BookInfo;
grid.Children.Add(searchBar);
grid.Children.Add(listView, 0, 1);
...
private void OnFilterTextChanged(object sender, TextChangedEventArgs e)
{
searchBar = (sender as SearchBar);
if (listView.DataSource != null)
{
this.listView.DataSource.Filter = FilterContacts;
this.listView.DataSource.RefreshFilter();
}
}
private bool FilterContacts(object obj)
{
if (searchBar == null || searchBar.Text == null)
return true;
var bookInfo = obj as BookInfo;
if (bookInfo.BookName.ToLower().Contains(searchBar.Text.ToLower())
|| bookInfo.BookDescription.ToLower().Contains(searchBar.Text.ToLower()))
return true;
else
return false;
}
```

## Grouping

The SfListView allows displaying the items in a group using the [SfListView.DataSource.GroupDescriptors](#) property. Create [GroupDescriptor](#) for the property to be grouped, and add it in the [DataSource.GroupDescriptors](#) collection.

[GroupDescriptor](#) object holds the following properties:

- [PropertyName](#): Describes name of the property to be grouped.
- [KeySelector](#): Describes selector to return the group key.
- [Comparer](#): Describes comparer to be applied when sorting takes place.

It also supports to stick the group header by enabling the [SfListView.IsStickyGroupHeader](#) property.

## XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
  <syncfusion:SfListView x:Name="listView">
    <syncfusion:SfListView.DataSource>
      <data:DataSource>
        <data:DataSource.GroupDescriptors>
          <data:GroupDescriptor PropertyName="BookName"/>
        </data:DataSource.GroupDescriptors>
      </data:DataSource>
    </syncfusion:SfListView.DataSource>
  </syncfusion:SfListView>
</ContentPage>
```

## C#

```
listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()
{
    PropertyName = "BookName",
});
```

## Selection

The SfListView allows selecting the item by setting the [SfListView.SelectionMode](#) property. Set the [SfListView.SelectionMode](#) property to single, multiple, and none based on the requirements. Informations about the selected item can be tracked using the [SfListView.SelectedItem](#) and [SfListView.SelectedItems](#) properties. It also allows changing the selection highlight color by using the [SfListView.SelectionBackgroundColor](#).

The gesture type can be changed to select the item by setting the [SfListView.SelectionGesture](#) property. Set the [SfListView.SelectionGesture](#) property to Tap, DoubleTap, and Hold based on the requirements.

The selection operations can be handled with the help of [SelectionChanging](#) and [SelectionChanged](#) events of the SfListView.

## XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
SelectionMode="Single"
SelectionGesture="Tap"
SelectionBackgroundColor="#E4E4E4"/>
</ContentPage>
```

## C#

```
listView.SelectionMode = SelectionMode.Single;
listView.SelectionGesture = TouchGesture.Tap;
listView.SelectionBackgroundColor = Color.FromHex("#E4E4E4");
```

## Header and Footer

The SfListView allows setting the header and footer to the user interface(UI) view by setting the DataTemplate to the [HeaderTemplate](#) and [FooterTemplate](#).

The header and footer can be handled either by scrollable, or sticky to the view by enabling or disabling the [IsStickyHeader](#) and [IsStickyFooter](#) properties.

## XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding InboxInfo}"
IsStickyHeader="true"
IsStickyFooter="true">
<syncfusion:SfListView.HeaderTemplate>
<DataTemplate>
<Grid BackgroundColor="#4CA1FE" HeightRequest="45">
<Label Text="Inbox" FontAttributes="Bold" FontSize="18" TextColor="White" />
</Grid>
</DataTemplate>
</syncfusion:SfListView.HeaderTemplate>
<syncfusion:SfListView.FooterTemplate>
<DataTemplate>
<Grid BackgroundColor="#DC595F">
<Grid.ColumnDefinitions>
<ColumnDefinition />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<Image Grid.Column="0" Source="Edit.png" />
<Image Grid.Column="1" Source="Delete.png" />
</Grid>
</DataTemplate>
</syncfusion:SfListView.FooterTemplate>
</syncfusion:SfListView>
</ContentPage>
```

## C#

```
ViewModel viewModel = new ViewModel ();
listView.ItemsSource = viewModel.InboxInfo;
listView.IsStickyHeader = true;
listView.IsStickyFooter = true;
listView.HeaderTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    grid.BackgroundColor = Color.FromHex("#4CA1FE");
    var headerLabel = new Label { BackgroundColor = Color.White, FontSize = 18,
    FontAttributes = FontAttributes.Bold };
    headerLabel.Text = "Inbox";
    grid.Children.Add(headerLabel);
    return grid;
});
listView.FooterTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    grid.BackgroundColor = Color.FromHex("#DC595F");
    var editImage = new Image();
    editImage.Source = "Edit.png";
    var deleteImage = new Image();
    deleteImage.Source = "Delete.png";
    grid.Children.Add(editImage);
    grid.Children.Add(deleteImage, 0, 1);
    return grid;
});
```

## Working with SfListView

### Interacting with ListView Items

#### *Loaded event*

The [Loaded](#) event is raised when the SfListView is loading in view for the first time.

#### **C#**

```
listView.Loaded += ListView_Loaded;
private void ListView_Loaded(object sender, ListViewLoadedEventArgs e)
{
    listView.SelectedItems.Add(viewModel.Customers[2]);
}
```

The **Loaded** event used for the following use cases:

- To scroll the desired position or index by using the [ScrollTo](#) or [ScrollToRowIndex](#).
- To collapse all the groups.
- To find the sorted or grouped [DataSource.DisplayItems](#) of underlying bound data to SfListView.

#### *Tapped event*

The [ItemTapped](#) event will be triggered whenever tapping the item. Here, [TapCommandParameter](#) property sets the parameter for [SfListView.TapCommand](#) and its default value is [ItemTappedEventArgs](#). [ItemTappedEventArgs](#) has the following members which provides the information for [ItemTapped](#) event:

- [ItemType](#): It gets the type of the tapped item.
- [ItemData](#): The underlying data associated with the tapped item as its arguments.
- [Position](#): Gets the touch position in the tapped item.

### C#

```
listView.ItemTapped += ListView_ItemTapped;
private void ListView_ItemTapped(object sender,
Syncfusion.ListView.XForms.ItemTappedEventArgs e)
{
    if (e.ItemData == viewModel.InboxInfo[0])
        viewModel.InboxInfo.Remove(e.ItemData as ListViewInboxInfo);
}
```

The [ItemTapped](#) event is used for the following use cases:

- To show the context menu.
- To navigate to another page.
- To delete the item in the list view at runtime.
- To display the item details into another view.
- To expand the view like, accordion.
- To change the underlying bound data.
- To skip other events like selection events if the `Handled` property set to true.

### *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:

- [ItemType](#): It gets the type of double tapped item.
- [ItemData](#): The underlying data associated with the double tapped item as its arguments.
- [Position](#): Gets the touch position in the double tapped item.

### C#

```
listView.ItemDoubleTapped += ListView_ItemDoubleTapped;
private void ListView_ItemDoubleTapped(object sender,
ItemDoubleTappedEventArgs e)
{
    var listViewInboxInfo = new ListViewInboxInfo();
    listViewInboxInfo.Title = "Bryce Thomas";
    listViewInboxInfo.Subject = "Congratulations on the move!";
    viewModel.InboxInfo.Add(listViewInboxInfo);
}
```

The [ItemDoubleTapped](#) event is used for the following use cases:

- To show the context menu.
- To delete the item in the list view at runtime.
- To change the underlying bound data.

### *ItemHolding event*

The [ItemHolding](#) event will be triggered whenever long pressing the item. Here, [HoldCommandParameter](#) sets the parameter for [SfListView.HoldCommand](#) and its default value is [ItemHoldingEventArgs](#). [ItemHoldingEventArgs](#) has the following members which provides the information for [ItemHolding](#) event:

- [ItemType](#): It gets the type of the long pressed item.
- [ItemData](#): The underlying data associated with the holding item as its arguments.
- [Position](#): Gets the touch position in the holding item.

### **C#**

```
listView.ItemHolding += ListView_ItemHolding;
private void ListView_ItemHolding(object sender, ItemHoldingEventArgs e)
{
    if (e.ItemData == viewModel.InboxInfo[3])
        viewModel.InboxInfo.Remove(e.ItemData as ListViewInboxInfo);
}
```

{% endtabs%}

The [ItemHolding](#) event is used for the following use cases:

- To show the context menu.

### *ItemAppearing*

The [ItemAppearing](#) event is raised when the items are appearing in the view on scrolling, loading, and navigating from one page to another page. The [ItemAppearingEventArgs](#) has the following member which provides the information of appearing Items.

- [ItemData](#): The underlying data associated with the appearing item.

### **C#**

```
listView.ItemAppearing += listView_ItemAppearing;
private void listView_ItemAppearing(object sender,
Syncfusion.ListView.XForms.ItemAppearingEventArgs e)
{
    if (e.ItemData == viewModel.BookInfo[0])
        Debug.WriteLine((e.ItemData as BookInfo).BookName);
    // If the ItemData value is "Header", then it's a header item.
    if (e.ItemData == "Header")
        Debug.WriteLine("Header is Appeared");
}
```

The [ItemAppearing](#) event used for the following use cases:

- To find the item appears in the view.
- To customize the appearing item to change the background color using converter.

### ItemDisappearing

The [ItemDisappearing](#) event is raised when the items disappearing in the view on scrolling, disposing, and navigating from one page to another page. The [ItemDisappearingEventArgs](#) has the following member which provides the information on disappearing Items:

- [ItemData](#): The underlying data associated with the disappearing item.

### C#

```
listView.ItemDisappearing += listView_ItemDisappearing;
private void listView_ItemDisappearing(object sender,
Syncfusion.ListView.XForms.ItemDisappearingEventArgs e)
{
    if (e.ItemData == viewModel.BookInfo[0])
        Debug.WriteLine((e.ItemData as BookInfo).BookName);
    // If the ItemData value is "Footer" then it's a Footer item.
    if (e.ItemData == "Footer")
        Debug.WriteLine("Footer is Disappeared");
}
```

The [ItemDisappearing](#) event used for the following use cases:

- To find the item disappears in the view.

### Improving ListView performance

The SfListView has been built from the ground up with an optimized view reuse strategy for the best possible performance on the Xamarin platform even when loading large data sets. Following techniques are used to improve performance of the SfListView:

- Bind the ItemsSource property to an IList collection instead of an IEnumerable collection because IEnumerable collection do not support random access.
- The SfListView gets refreshed each and every time a new item added into the underlying collection. Because, when adding items at runtime, the DataSource gets refreshed. To avoid this behavior, use [BeginInit\(\)](#) to stop the RefreshView() calling in each time, and use [EndInit\(\)](#) to start the RefreshView() calling when adding number of finished items.
- Avoid loading complex layout in template of listview. For example, loading large-size images or nested containers degrades the scrolling performance. This practice commonly degrades performance in all platforms, and particularly more in Android version API level 19. So, use fewer elements and images with small size and resolution to achieve the maximum performance.
- Avoid placing the SfListView inside ScrollView for the following reasons:
  - The SfListView implement its own scrolling.
  - The SfListView will not receive any gestures as it will be handled by the parent ScrollView.
  - Should define size to the SfListView if it loads inside ScrollView.
  - Avoid changing the cell layout based on the BindingContext. This incurs large layout and initialization costs.
- Implement a model class inherited with [INotifyPropertyChanged](#) interface to notify the property changes at runtime.



### Loading ListView inside ScrollView

When the **SfListView** is loaded inside the **ScrollView** with the height of total items, scrolling will not occur in the SfListView only when [AllowSwiping](#) is set to **true**. The SfListView does not pass touch to the parent ScrollView in UWP, because swiping is handled in it.

When the SfListView is loaded inside the ScrollView the following scenarios, the height of the total items is set to **HeightRequest** of the SfListView.

- If the position of the SfListView is not in view when loading inside the StackLayout with more than one children, the SfListView will not be loaded. Because, the StackLayout passes the height for the **SfListView** as 1.
- When loading the SfListView inside the **Grid** with row definition as **Auto** in UWP, Grid passes the height for the SfListView to be **1**.

When the SfListView is loaded inside the ScrollView with sticky header and sticky group header, it changed to scrollable. The empty space remains after the **SfListView** items, when the device orientation is changed to horizontal. Because, the total extend is set to the ScrollView in horizontal orientation.

### XML

```
<local:ExtScrollView x:Name="scrollView" >
<sync:SfListView x:Name="listView" ItemsSource="{Binding BookInfo}"/>
</local:ExtScrollView>
```

### C#

```
public class ExtScrollView: ScrollView
{
    protected override void LayoutChildren(double x, double y, double width,
    double height)
    {
        this.Content.HeightRequest = height;
        base.LayoutChildren(x, y, width, height);
    }
}
```

Download the entire source code from GitHub [here](#).

### Loading ListView inside CarouselPage/Master detail page

When the SfListView is loaded in CarouselView with [SfListView.AllowSwiping](#) as false, it behaves in UWP platform as follows:

- When performing first swipe on the view, it will be handled by ScrollView to ensure whether scrolling is happened or not. If not means the manipulation to parent cannot be passed immediately due to UWP platform behavior. The second swipe will be listened by CarouselView, and the view gets swiped. This is the behavior of the SfListView.

When the SfListView is loaded in CarouselView with **SfListView.AllowSwiping** as true, it behaves as follows:

- When swiping in iOS, suddenly carousel swipe happened even if gesture is added to the carousel view. To swipe ListViewItem, touch and hold the item for some fraction of seconds (0.25 - 0.5 seconds) and then swipe.
- When swiping any Item, the SfListView handles the touch and swipe the ListViewItem.
- After swiping on ListViewItem, SwipeView will load along with it. If you swipe SwipeView element, Carousel view is swiped. Or else swipe on ListViewItem, control handles touching and swiping the item as usual.
- If you swipe header, footer or group header elements, Carousel view will swipe in Android platform. But in UWP, first swipe on those elements will be handled by SfListView itself, because manipulation to parent cannot be passed immediately. The second swipe will be listened by CarouselView.

### Context menu on items

The SfListView allows displaying a pop-up menu with different menu items to an item when it is long pressed by customizing the SfListView and by using custom view in it. For UWP platform, you can also display the pop-up menu when pressing right click button over the item. Display the pop-up menu by accessed the touch position in the item based on [Position](#) property from [ItemHolding](#) event.

### Defining SfPopUpView

#### C#

```
namespace SfListViewSample
{
    public class Behavior : Behavior<SfListView>
    {
        SfListView ListView;
        int sortOrder = 0;
        Contacts item;
        SfPopupLayout popupLayout;
        protected override void OnAttachedTo(SfListView listView)
        {
            ListView = listView;
            ListView.ItemHolding += ListView_ItemHolding;
            ListView.ScrollStateChanged += ListView_ScrollStateChanged;
            ListView.ItemTapped += ListView_ItemTapped;
            base.OnAttachedTo(listView);
        }
        private void ListView_ItemTapped(object sender,
            Syncfusion.ListView.XForms.ItemTappedEventArgs e)
        {
            popupLayout.Dismiss();
        }
        private void ListView_ScrollStateChanged(object sender,
            ScrollStateChangedEventArgs e)
        {
            popupLayout.Dismiss();
        }
        private void ListView_ItemHolding(object sender, ItemHoldingEventArgs e)
        {
            item = e.ItemData as Contacts;
            popupLayout = new SfPopupLayout();
            popupLayout.PopupView.HeightRequest = 100;
            popupLayout.PopupView.WidthRequest = 100;
        }
    }
}
```

```

popupLayout.PopupView.ContentTemplate = new DataTemplate(() =>
{
    var mainStack = new StackLayout();
    mainStack.BackgroundColor = Color.Teal;
    var deletedButton = new Button()
    {
        Text = "Delete",
        HeightRequest=50,
        BackgroundColor=Color.Teal,
        TextColor = Color.White
    };
    deletedButton.Clicked += DeletedButton_Clicked;
    var sortButton = new Button()
    {
        Text = "Sort",
        HeightRequest = 50,
        BackgroundColor = Color.Teal,
        TextColor=Color.White
    };
    sortButton.Clicked += SortButton_Clicked;
    mainStack.Children.Add(deletedButton);
    mainStack.Children.Add(sortButton);
    return mainStack;
});
popupLayout.PopupView.ShowHeader = false;
popupLayout.PopupView.ShowFooter = false;
if (e.Position.Y + 100 <= ListView.Height && e.Position.X + 100 >
    ListView.Width)
    popupLayout.Show((double)(e.Position.X - 100), (double)(e.Position.Y));
else if (e.Position.Y + 100 > ListView.Height && e.Position.X + 100 <
    ListView.Width)
    popupLayout.Show((double)e.Position.X, (double)(e.Position.Y - 100));
else if (e.Position.Y + 100 > ListView.Height && e.Position.X + 100 >
    ListView.Width)
    popupLayout.Show((double)(e.Position.X - 100), (double)(e.Position.Y -
    100));
else
    popupLayout.Show((double)e.Position.X, (double)(e.Position.Y));
}
private void SortButton_Clicked(object sender, EventArgs e)
{
    if (ListView == null)
        return;
    ListView.DataSource.SortDescriptors.Clear();
    popupLayout.Dismiss();
    ListView.DataSource.LiveDataUpdateMode =
    LiveDataUpdateMode.AllowDataShaping;
    if (sortOrder == 0)
    {
        ListView.DataSource.SortDescriptors.Add(new SortDescriptor { PropertyName =
        "ContactName", Direction = ListSortDirection.Descending });
        sortOrder = 1;
    }
    else
    {
        ListView.DataSource.SortDescriptors.Add(new SortDescriptor { PropertyName =
        "ContactName", Direction = ListSortDirection.Ascending });
    }
}

```

```

sortOrder = 0;
}
}
private void Dismiss()
{
popupLayout.IsVisible = false;
}
private void DeletedButton_Clicked(object sender, EventArgs e)
{
if (ListView == null)
return;
var source = ListView.ItemsSource as IList;
if (source != null && source.Contains(item))
source.Remove(item);
else
App.Current.MainPage.DisplayAlert("Alert", "Unable to delete the item",
"OK");
item = null;
source = null;
}
}
}

```

## Defining the SfListView

### XML












```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.BindingContext>
<local:ContactsViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<Grid>
<listView:SfListView x:Name="listView" ItemsSource="{Binding Items}" >
<listView:SfListView.ItemTemplate>
<DataTemplate>
<Grid>
<Image Source="{Binding ContactImage}"/>
<Label Text="{Binding ContactName}" />
<Label Text="{Binding ContactNumber}" />
</Grid>
</DataTemplate>
</listView:SfListView.ItemTemplate>
<listView:SfListView>
</Grid>
</ContentPage.Content>
</ContentPage>

```

You can download the entire source code of this demo [here](#).

**A**

	Aaron 790 - 3700	<div>SORT</div> <div>DELETE</div>
	Adam 747 - 3921	
	Adrian 720 - 3609	
	Aiden 752 - 3767	
	Alex 762 - 3419	
	Alexander 737 - 3728	
	Andrew 741 - 3044	
	Anthony 758 - 3972	
	Asher 785 - 3215	
	Austin 797 - 3236	
<b>B</b>		
	Benjamin 777 - 3333	

## Paging

The SfListView allows displaying paging using the [SfDataPager](#) control. It can be performed through loading data dynamically into ItemsSource of the SfListView using OnDemandLoading event for the current page by setting the [SfDataPager.UseOnDemandPaging](#) to 'True'. By using the [SfDataPager.PageSize](#) property, you can define the number of list items to be displayed in each page.

**Note:** For more details about paging refer to [here](#).

## XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:sfPager="clr-
namespace:Syncfusion.SfDataGrid.XForms.DataPager;assembly=Syncfusion.SfDataG
rid.XForms">
<ContentPage.Behaviors>
<local:SfListViewPagingBehavior/>
</ContentPage.Behaviors>
<ContentPage.Resources>
<ResourceDictionary>
<local:CurrencyFormatConverter x:Key="currencyFormatConverter"/>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<StackLayout>
<syncfusion:SfListView x:Name="listView" >
<syncfusion:SfListView.LayoutManager>
<syncfusion:GridLayout SpanCount = "2">
</syncfusion:SfListView.LayoutManager>
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Grid>
<Image Source="{Binding Image}" />
<Label Text="{Binding Name}" />
<Label Text="{Binding Weight}" />
</Grid>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
<sfPager:SfDataPager x:Name="dataPager" UseOnDemandPaging="True" DisplayMode
= "PreviousNextNumeric" NumericButtonCount = "4" PageSize = "6">
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using Syncfusion.ListView.XForms;
using Syncfusion.SfDataGrid.XForms.DataPager;
namespace Paging
{
    public class Paging : ContentPage
    {
        SfDataPager sfPager = new SfDataPager();
        SfListView listView = new SfListView();
        public Paging()
        {
        }
    }
}
```

```

{
    var stackLayout = new StackLayout();
    sfPager.DisplayMode = PagerDisplayMode.PreviousNextNumeric;
    sfPager.UseOnDemandPaging = true;
    sfPager.NumericButtonCount = 4;
    sfPager.PageSize = 6;
    listView = new SfListView();
    listView.LayoutManager = new GridLayout() { SpanCount = 2 };
    listView.ItemTemplate = new DataTemplate(() =>
    {
        var grid = new Grid();
        var image = new Image();
        image.SetBinding(Image.SourceProperty, new Binding("Image"));
        var label = new Label();
        label.SetBinding(Label.TextProperty, new Binding("Name"));
        var label1 = new Label();
        label1.SetBinding(Label.TextProperty, new Binding("Weight"));
        grid.Children.Add(image);
        grid.Children.Add(label);
        grid.Children.Add(label1);
        return grid;
    });
    stackLayout.Children.Add(listView);
    stackLayout.Children.Add(sfPager);
}
}
}

```

## C#

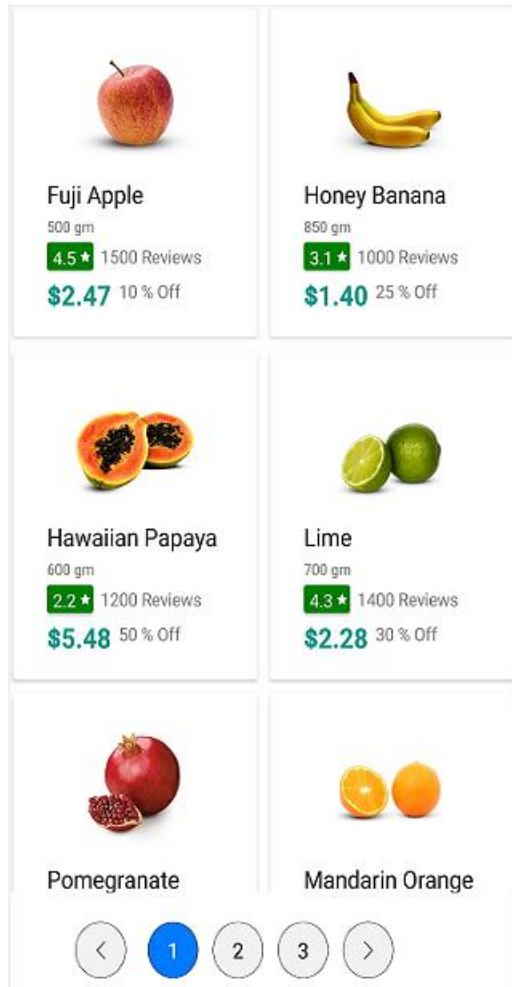
```

public class SfListViewPagingBehavior : Behavior<ContentPage>
{
    private Syncfusion.ListView.XForms.SfListView listView;
    private PagingViewModel PagingViewModel;
    private SfDataPager dataPager;
    protected override void OnAttachedTo(ContentPage bindable)
    {
        listView =
        bindable.FindByName<Syncfusion.ListView.XForms.SfListView>("listView");
        dataPager = bindable.FindByName<SfDataPager>("dataPager");
        PagingViewModel = new PagingViewModel();
        listView.BindingContext = PagingViewModel;
        dataPager.Source = PagingViewModel.pagingProducts;
        dataPager.OnDemandLoading += DataPager_OnDemandLoading;
        base.OnAttachedTo(bindable);
    }
    private void DataPager_OnDemandLoading(object sender,
    OnDemandLoadingEventArgs e)
    {
        var source =
        PagingViewModel.pagingProducts.Skip(e.StartIndex).Take(e.PageSize);
        listView.ItemsSource = source.AsEnumerable();
    }
    protected override void OnDetachingFrom(ContentPage bindable)
    {
        listView = null;
    }
}

```

```
PagingViewModel = null;  
dataPager = null;  
base.OnDetachingFrom(bindable);  
}  
}
```

Download the entire source code form GitHub [here](#).



#### Loading data from SQLite online database

The SfListView allows binding the data from online database with the help of **Azure App Service**. To perform this, follow the steps:

Step 1: Get URL to store the data online.

Step 2: Create table using **AppServiceHelpers**.

Step 3: Populate the data into the table using the ObservableCollection **Items**.

Step 4: Bind it to SfListView using [SfListView.ItemsSource](#) property.

Step 5: Refer to the following link to know more about working with **Azure App Service**.

<https://blog.xamarin.com/add-a-backend-to-your-app-in-10-minutes/>



---

**Note:** Install the following NuGet packages to successfully run the application:

---

- Microsoft.Azure.Mobile.Client(v.2.1.1)
- Microsoft.Azure.Mobile.Client.SQLiteStore(v.2.1.1)
- AppService.Helpers (Does not support UWP platform)
- AppService.Helpers.Forms (Does not support UWP platform)

Refer to the following code which illustrates, how to initialize the library with the URL of the Azure Mobile App and registering the Model with the client to create a table.

#### **C#**

```
public App()
{
    InitializeComponent();
    IEasyMobileServiceClient client = new EasyMobileServiceClient();
    client.Initialize("http://xamarin-todo-sample.azurewebsites.net");
    client.RegisterTable<ToDo>();
    client.FinalizeSchema();
    MainPage = new NavigationPage(new Pages.ToDoListPage(client));
}
```

Refer to the following code which illustrates, how to create a table using AppServiceHelpers and insert items in it.

#### **C#**

```
using AppServiceHelpers.Abstractions;
using AppServiceHelpers.Models;
public class BaseAzureViewModel<T> : INotifyPropertyChanged where T :
    EntityData
{
    IEasyMobileServiceClient client;
    ITableDataStore<T> table;
    public BaseAzureViewModel(IEasyMobileServiceClient client)
    {
        this.client = client;
        table = client.Table<T>();
    }
    // Returns an ObservableCollection of all the items in the table
    ObservableCollection<T> items = new ObservableCollection<T>();
    public ObservableCollection<T> Items
    {
        get { return items; }
        set
        {
            items = value;
            OnPropertyChanged("items");
        }
    }
    // Adds an item to the table.
    public async Task AddItemAsync(T item)
    {
        await table.AddAsync(item);
    }
}
```

```

// Deletes an item from the table.
public async Task DeleteItemAsync(T item)
{
    await table.DeleteAsync(item);
}
// Updates an item in the table.
public async Task UpdateItemAsync(T item)
{
    await table.UpdateAsync(item);
}
// Refresh the table and synchronize data with Azure.
Command refreshCommand;
public Command RefreshCommand
{
    get { return refreshCommand ?? (refreshCommand = new Command(async () =>
        await ExecuteRefreshCommand())); }
}
async Task ExecuteRefreshCommand()
{
    if (IsBusy)
        return;
    IsBusy = true;
    try
    {
        var _items = await table.GetItemsAsync();
        Items.Clear();
        foreach (var item in _items)
        {
            Items.Add(item);
        }
        IsBusy = false;
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage.DisplayAlert("Error", ex.Message, "OK");
    }
}
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string propertyName)
{
    if (PropertyChanged == null)
        return;
    PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
}

```

Refer to the following code which illustrates, how to bind the table contents into the SfListView.

#### C#

```

public partial class ToDoListPage : ContentPage
{
    public ToDoListPage(IEasyMobileServiceClient client)
    {
        InitializeComponent();
        var viewModel = new ViewModels.ToDosViewModel(client);
    }
}

```

```

BindingContext = viewModel;
listView.ItemsSource = viewModel.Items;
}
private void FetchButton_Clicked(object sender, EventArgs e)
{
    var viewModel1 = (TodosViewModel)BindingContext;
    viewModel1.RefreshCommand.Execute(null);
}
}

```

## XML

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.Content>
<StackLayout>
<syncfusion:SfListView x:Name="listView" SelectedItem="{Binding
SelectedToDoItem, Mode=TwoWay}" ItemSize="50">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout Orientation="Horizontal" HorizontalOptions="FillAndExpand"
VerticalOptions="CenterAndExpand">
<Label Text="{Binding Text}" />
<Switch IsToggled="{Binding Completed}" />
</StackLayout>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
<Button Text="Add New" Command="{Binding AddNewItemCommand}" />
<Button Text="Fetch" Command="{Binding FetchItemCommand}" />
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

Download the entire source code from GitHub [here](#).

### Loading data from SQLite offline database

The SfListView allows binding the data from local database by using SQLite. To perform this, follow the steps: SQLiteConnection

Step 1: Create a **SQLite database table**

Step 2: Populate the data into the table

Step 3: Store them as an **IEnumerable** collection

Step 4: Bind it to SfListView using [SfListView.ItemsSource](#) property.

Step 5: Refer to the following link to know how to create SQLite connection,

<http://developer.xamarin.com/guides/xamarin-forms/working-with/databases/>

**Note:** To run this sample in UWP, install [SQLite.Net.Pcl](#), version v1.0.10 (Only this version of SQLite supports UWP platform, later versions don't support UWP).

**C#**

```
using SQLite;
public class OrderItem
{
    public int ID { get; set; }
    public string Name { get; set; }
}
```

**C#**

```
using SQLite;
public class ViewModel
{
    SQLiteConnection database;
    IEnumerable<OrderItem> orderItemCollection;
    public IEnumerable<OrderItem> OrderItemCollection
    {
        get
        {
            if (orderItemCollection == null)
                orderItemCollection = GetItems();
            return orderItemCollection;
        }
    }
    public ViewModel()
    {
        database = DependencyService.Get<ISQLite>().GetConnection();
        // Create the table
        database.CreateTable<OrderItem>();
        // Insert items into table
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1001,'Antony')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1002,'Blake')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1003,'Catherine')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1004,'Jude')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1005,'Mark')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1006,'Anderson')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1007,'Wilson')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1008,'Jade')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1009,'Zachery')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1010,'Dhotis')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1011,'Trunks')");
        database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name) values (1012,'Kevin')");
    }
}
```

```

database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name)values
(1013,'Mathew')");
database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name)values
(1014,'Watson')");
database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name)values
(1015,'Chris')");
database.Query<OrderItem>("INSERT INTO OrderItem (ID,Name)values
(1016,'Phantom')");
}
public IEnumerable<OrderItem> GetItems()
{
    // Changing the database table items as ObservableCollection
    var table = (from i in database.Table<OrderItem>() select i);
    ObservableCollection<OrderItem> OrderList = new
    ObservableCollection<OrderItem>();
    foreach (var order in table)
    {
        OrderList.Add(new OrderItem()
        {
            ID = order.ID,
            Name = order.Name
        });
    }
    return OrderList;
}
}

```

Refer to the following code which illustrates, how to bind the data from the SQLite database to SfListView.

### XML

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.BindingContext>
<local:ViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<Grid>
<listView:SfListView x:Name="listView" ItemSize="70"
BackgroundColor="Teal"
ItemsSource="{Binding OrderItemCollection}">
<listView:SfListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="1" />
</Grid.RowDefinitions>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>

```

```

<Label LineBreakMode="NoWrap" Text="{Binding ID}" VerticalOptions="Center"/>
<Label LineBreakMode="NoWrap" Text="{Binding Name}" Grid.Column="1"
VerticalOptions="Center" />
</Grid>
<StackLayout Grid.Row="1" BackgroundColor="Gray" HeightRequest="1"/>
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</listView:SfListView.ItemTemplate>
</listView:SfListView>
</Grid>
</ContentPage.Content>
</ContentPage>

```

Download the entire source code from GitHub [here](#).

### ListView with Prism Framework

The SfListView allows the user to work with prism for MVVM Framework. Steps to be followed:

1. Replace your application to prism application in App.xaml file.
2. Inherit App.xaml.cs from prism application instead of your application.
3. Create a prism namespace library reference in xaml file of the ContentPage.
4. Connect view and view model instead of binding context by registering them.

### C#

```

public partial class App : PrismApplication
{
    public App(IPlatformInitializer initializer = null) : base(initializer) { }
    protected override void OnInitialized()
    {
        InitializeComponent();
    }
    protected override void RegisterTypes()
    {
        Container.RegisterTypeForNavigation<MainPage, MainPageViewModel>();
    }
}

```

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<prism:PrismApplication xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:prism="clr-namespace:Prism.Unity;assembly=Prism.Unity.Forms"
x:Class="ListViewPrism.App">
</prism:PrismApplication>

```

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

```

```
xmlns:local="clr-namespace:ListViewPrism;assembly=ListViewPrism"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
x:Class="ListViewPrism.Views.MainPage"
Title="MainPage">
<StackLayout HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
<syncfusion:SfListView x:Name="listView" ItemSize="70" ItemSpacing="0,0,5,0"
AutoFitMode="Height"
ItemsSource="{Binding ContactsInfo,}" IsStickyHeader="True"
AllowSwiping="True" IsStickyGroupHeader="True" GroupHeaderSize="50">
</syncfusion:SfListView>
</StackLayout>
</ContentPage>
```

For more details, refer to <https://xamgirl.com/prism-in-xamarin-forms-step-by-step-part-1>.

Download the entire source code from GitHub [here](#).

### Scrolling ListView without virtualization

ListView allows you to scroll by loading the entire collection of items inside the ScrollView and defining the total extend of its container to `HeightRequest` in the `Loaded` event.

#### XML

```
<ScrollView>
<sync:SfListView x:Name="listView" ItemsSource="{Binding BookInfo}"
Loaded="listView_Loaded"/>
</ScrollView>
```

#### C#

```
using Syncfusion.ListView.XForms.Control.Helpers;
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
    private void listView_Loaded(object sender, ListViewLoadedEventArgs e)
    {
        var container = listView.GetVisualContainer();
        var extent =
            (double)container.GetType().GetRuntimeProperties().FirstOrDefault(x =>
            x.Name == "TotalExtent").GetValue(container);
        listView.HeightRequest = extent;
    }
}
```

Download the entire source code from GitHub [here](#).

When ListView is in `AutoFitMode` as 'Height', the extend of the ListView will be updated only while scrolling. So you can resize the ListView in `VisualContainer` `PropertyChanged` method as like below.

#### XML

```
<ScrollView>
<sync:SfListView x:Name="listView" AutoFitMode="Height"
ItemsSource="{Binding BookInfo}" Loaded="listView_Loaded"/>
</ScrollView>
```

## C#

```
using Syncfusion.ListView.XForms.Control.Helpers;
public partial class MainPage : ContentPage
{
    VisualContainer visualContainer;
    bool loaded;
    public MainPage()
    {
        InitializeComponent();
        visualContainer = listView.GetVisualContainer();
        visualContainer.PropertyChanged += VisualContainer_PropertyChanged;
    }
    private void VisualContainer_PropertyChanged(object sender,
        System.ComponentModel.PropertyChangedEventArgs e)
    {
        if (e.PropertyName == "Height" && listView.HeightRequest !=
            visualContainer.Height && loaded)
        {
            listView.HeightRequest = visualContainer.Height;
        }
    }
    private void listView_Loaded(object sender, ListViewLoadedEventArgs e)
    {
        var extent =
            (double)visualContainer.GetType().GetRuntimeProperties().FirstOrDefault(x =>
                x.Name == "TotalExtent").GetValue(visualContainer);
        listView.HeightRequest = extent;
        loaded = true;
    }
}
```

You can download the entire source code of this demo [here](#).

**Note:** While loading in `AutoFitMode` make sure that the `ItemSize` property value is not specified, to avoid extra space below the list. Update the size of the container after ListView loaded to render all the list items in the view.

The following limitations should be noted when using the previous approaches:

- As the entire list items are loaded inside the parent `ScrollView` element, the `ItemAppearing` and `ItemDisappearing` events will not be fired when scrolling.
- When performing keyboard navigation, the view cannot be scrolled automatically while navigating out of view.
- Scrolling through the touch action will be handled in all platforms and ListView scrolling will be handled by the parent ScrollView.



### Working with nested ListView

ListView allows you to load another ListView inside its [ItemTemplate](#). When the [AutoFitMode](#) of the outer ListView is height, the size of the inner ListView will be allocated from the maximum screen size. Since the exact size for the inner list cannot not be obtained before loading the view.

To get a fixed height for the inner ListView, define a value in its [HeightRequest](#). If the items inside the inner ListView are less, allocate the total extend of the inner list to its [HeightRequest](#).

Follow the steps to set the size for the outer ListView based on the extend of inner ListView:

1. Define a property in the Model class and bind it to the [HeightRequest](#) of inner ListView, as the height for various inner ListView has to be identified while scrolling.
2. Hook the container's [PropertyChanged](#) event. When the height of the container changes, set the value of [TotalExtent](#) to the property bound to the [HeightRequest](#).
3. Call the [RefreshView](#) method asynchronously with few seconds delay in the [Loaded](#) event of the outer SfListView. The height requested for each inner SfListView will be set but the outer SfListView will not get any notification regarding the size change. So, call the [RefreshView](#) method asynchronously after loading the view.

### XML

```
<listView:SfListView x:Name="listView" ItemsSource="{Binding ContactInfo}"
AutoFitMode="Height" Loaded="listView_Loaded"
AllowGroupExpandCollapse="True">
<listView:SfListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid>
<local:ExtendedListView x:Name="list1" HeightRequest="{Binding
InnerListHeight}" ItemsSource="{Binding ContactDetails}" ItemSize="75">
<local:ExtendedListView.ItemTemplate>
<DataTemplate>
<StackLayout>
<Image Source="{Binding Image}" />
<Label Text="{Binding Name}" />
<Label Text="{Binding Number}" />
</StackLayout>
</DataTemplate>
</local:ExtendedListView.ItemTemplate>
</local:ExtendedListView>
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</listView:SfListView.ItemTemplate>
</listView:SfListView>
```

### C#

```
public partial class NestedListView : ContentPage
{
public NestedListView ()
```

```
{
InitializeComponent();
}
private async void listView_Loaded(object sender,
Syncfusion.ListView.XForms.ListViewLoadedEventArgs e)
{
await Task.Delay(2000);
listView.RefreshView();
}
}
```

## C#

```
using Syncfusion.ListView.XForms.Control.Helpers;
public class ExtendedListView : SfListView
{
VisualContainer container;
public ExtendedListView()
{
container = this.GetVisualContainer();
container.PropertyChanged += Container_PropertyChanged;
}
private void Container_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
{
Device.BeginInvokeOnMainThread(async () =>
{
var extent =
(double)container.GetType().GetRuntimeProperties().FirstOrDefault(container
=> container.Name == "TotalExtent").GetValue(container);
if (e.PropertyName == "Height")
(this.BindingContext as ContactInfo_NestedListView).InnerListHeight =
extent;
});
}
}
```

Download the entire source code from GitHub [here](#).

### Rendering ListView when loading in different layouts

The options are as follows:

- Creates the measurement and layout similar to Xamarin.Forms ListView, when the ListView is loaded inside the layouts such as StackLayout, ScrollView, and Grid, in which the RowDefinition or ColumnDefinition is set to 'Auto'. In all other layouts, the ListView size will be allocated from the framework.
- Sets the value of total extend to the HeightRequest of ListView, since the scrolling will be handled by the parent ScrollView, when ListView is loaded inside the StackLayout with base parent as ScrollView having multiple elements inside the StackLayout.

### How to

#### *Swipe and move an item to another listview on swipe item tapped action*

By using swipe view action, you can move an item from one listview to another listview.

**C#**

```
private void FavoriteTapped(object obj)
{
    var departureInfo = obj as DepartureInfo;
    var pinnedInfo = FirstLVCollection.Any(o => o.Name == departureInfo.Name) ?
    FirstLVCollection.First(o => o.Name == departureInfo.Name) : null;
    if (pinnedInfo == null)
    {
        FirstLVCollection.Add(new PinnedInfo() { Name = departureInfo.Name,
        RouteName = departureInfo.Name, Icon = departureInfo.Icon, IsFavorite = true
    });
    }
}
```

*Filter listview items based on another listview selection*

To filter the listview items based on the item selection in another listview, use the SfListView.DataSource.Filter property.

**C#**

```
private void ItemTapped(Syncfusion.ListView.XForms.ItemTappedEventArgs e)
{
    tappedPinnedInfo = e.ItemData as PinnedInfo;
    if (tappedPinnedInfo.IsFavorite)
    {
        secondLV.DataSource.Filter = FilterDepartures;
        tappedPinnedInfo.IsFavorite = false;
    }
    else
    {
        secondLV.DataSource.Filter = null;
        tappedPinnedInfo.IsFavorite = true;
    }
    secondLV.DataSource.RefreshFilter();
}

private bool FilterDepartures(object obj)
{
    var departureInfo = obj as DepartureInfo;
    if (tappedPinnedInfo == null)
        return true;
    if (departureInfo.Name.ToLower().Contains(tappedPinnedInfo.Name.ToLower())
    ||
    departureInfo.RouteName.ToLower().Contains(tappedPinnedInfo.RouteName.ToLower()
    ()))
        return true;
    else
        return false;
}
```

Download the entire source code from GitHub [here](#).

*Dispose listview*

You can dispose and release resources used by ListView by calling the [ListView.Dispose](#) method.

**C#**

```
protected override void OnDisappearing()  
{  
    listview.Dispose();  
    base.OnDisappearing();  
}
```

## View Appearance

The SfListView allows customizing appearance of the underlying data, and provides different functionalities to the end-user.

### Item template

A template can be used to present the data in a way that makes sense for the application by using different controls. SfListView allows customizing appearance of view by setting the [ItemTemplate](#) property.

### Data template selector

The SfListView allows customizing appearance of each item with different templates based on specific constraints by using the [DataTemplateSelector](#). You can choose a [DataTemplate](#) for each item at runtime based on the value of data-bound property using [DataTemplateSelector](#).

Here, an [ItemsCacheLimit](#) property maintains number of items reusing in the view. This cache limit is used to create and reuse the [ListViewItem](#) if different templates are used in [DataTemplateSelector](#) for better scrolling performance. Based on this value, [SfListView](#) creates number of [ListViewItem](#) for different templates in the view if new template is created while scrolling, and reuses it if same template is used for improving the scrolling performance.

### Create a data template selector

Create custom class that inherits from [DataTemplateSelector](#), and override the [OnSelectTemplate](#) method to return the [DataTemplate](#) for that item. At runtime, the SfListView invokes the [OnSelectTemplate](#) method for each item and passes the data object as parameter.

### C#

```
class MyDataTemplateSelector : Xamarin.Forms.DataTemplateSelector  
{  
    private readonly DataTemplate incomingDataTemplate;  
    private readonly DataTemplate outgoingDataTemplate;  
    public MyDataTemplateSelector()  
    {  
        this.incomingDataTemplate = new DataTemplate(typeof(IncomingViewCell));  
        this.outgoingDataTemplate = new DataTemplate(typeof(OutgoingViewCell));  
    }  
    protected override DataTemplate OnSelectTemplate(object item, BindableObject container)  
    {  
        var message = item as Message;  
        if (message == null)  
            return null;  
        return message.IsIncoming ? this.incomingDataTemplate :  
            this.outgoingDataTemplate;  
    }  
}
```

*Applying the data template selector*

Assign custom `DataTemplateSelector` to the [ItemTemplate](#) of the SfListView either in XAML or C#.

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="DataTemplateSelector.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:local="clr-
namespace:DataTemplateSelector;assembly=DataTemplateSelector">
<ContentPage.Resources>
<ResourceDictionary>
<local:MyDataTemplateSelector x:Key="MessageTemplateSelector" />
</ResourceDictionary>
</ContentPage.Resources>
<Grid>
<syncfusion:SfListView x:Name="ListView"
ItemTemplate="{StaticResource MessageTemplateSelector}"
ItemsSource="{Binding Messages}"
ItemSize="100">
</syncfusion:SfListView>
</Grid>
</ContentPage>
```

**C#**

```
public class MainPageCs : ContentPage
{
    public MainPageCs()
    {
        var viewModel = new MainPageViewModel();
        BindingContext = viewModel;
        Content = new SfListView()
        {
            ItemSize = 100,
            ItemsSource = viewModel.Messages,
            ItemTemplate = new MyDataTemplateSelector()
        };
    }
}
```

You can also download the entire source code of this demo [here](#).

Hi Jack. What are you doing?

04/12/2018 04:22 PM

Hi Mary. I'm filling out a job application.

04/12/2018 04:23 PM

Are you finished with school already?

04/12/2018 04:24 PM

No. I have one more semester, but it would be great to have a job lined up.

04/12/2018 04:25 PM

How is your day going?

04/12/2018 04:26 PM

Quite busy. I'm preparing for my exam tomorrow.

04/12/2018 04:27 PM

Good luck !.. for your exam.

04/12/2018 04:28 PM

Thank you buddy...Have a nice day

### Horizontal ListView

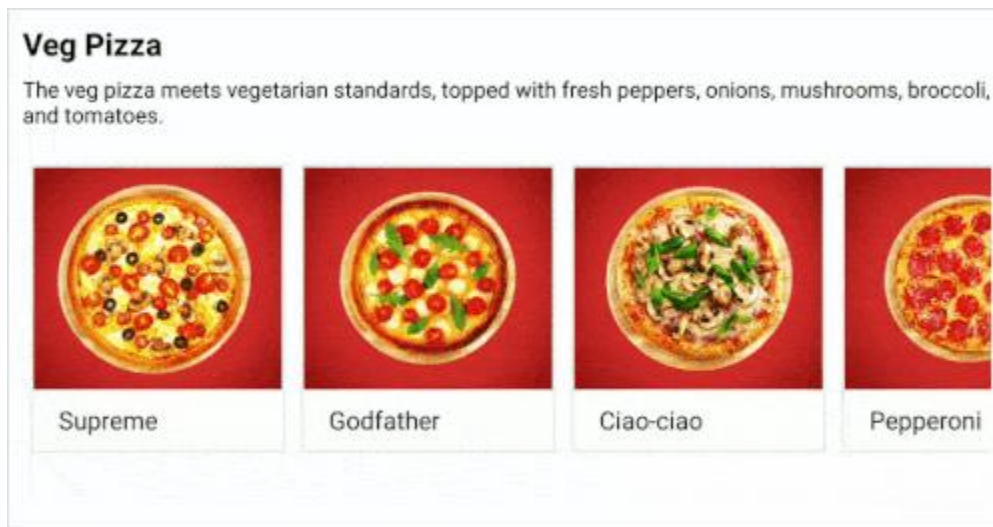
The SfListView allows you to layout every item in the [SfListView.ItemsSource](#) property either in vertical or horizontal orientation by setting the [SfListView.Orientation](#). The default orientation is **Vertical**.

#### XML

```
<syncfusion:SfListView x:Name="listView" Orientation="Horizontal" />
```

#### C#

```
listView.Orientation = Orientation.Horizontal;
```



### Navigate across views (like TabView)

The SfListView allows you to layout the items like **TabView** in the horizontal direction by setting the [Orientation](#) property as **Horizontal** using the [ItemTapped](#) event. It brings any desired view above the horizontal list as follows.

#### XML

```
<ContentPage xmlns:syncfusion="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <ContentPage.Content>
    <Grid x:Name="GridView">
      <Label Text="Tap image to expand"/>
      <Grid>
        <Image Source="{Binding ContactImage}" />
        <Label Text="{Binding ContactName}" />
        <Label Text="{Binding ContactNumber}" />
      </Grid>
      <syncfusion:SfListView x:Name="listView" ItemTapped="list_ItemTapped"
        ItemSize="70" ItemsSource="{Binding ContactsInfo}">
        <syncfusion:SfListView.ItemTemplate>
          <DataTemplate x:Name="ItemTemplate" x:Key="ItemTemplate">
            <ViewCell>
              <ViewCell.View>
                <Image Source="{Binding ContactImage}" />
              </ViewCell.View>
            </DataTemplate>
          </syncfusion:SfListView.ItemTemplate>
        </syncfusion:SfListView>
      </Grid>
    </ContentPage.Content>
  </ContentPage>
```

```

</ViewCell>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</Grid>
</ContentPage.Content>
</ContentPage>

```

## C#

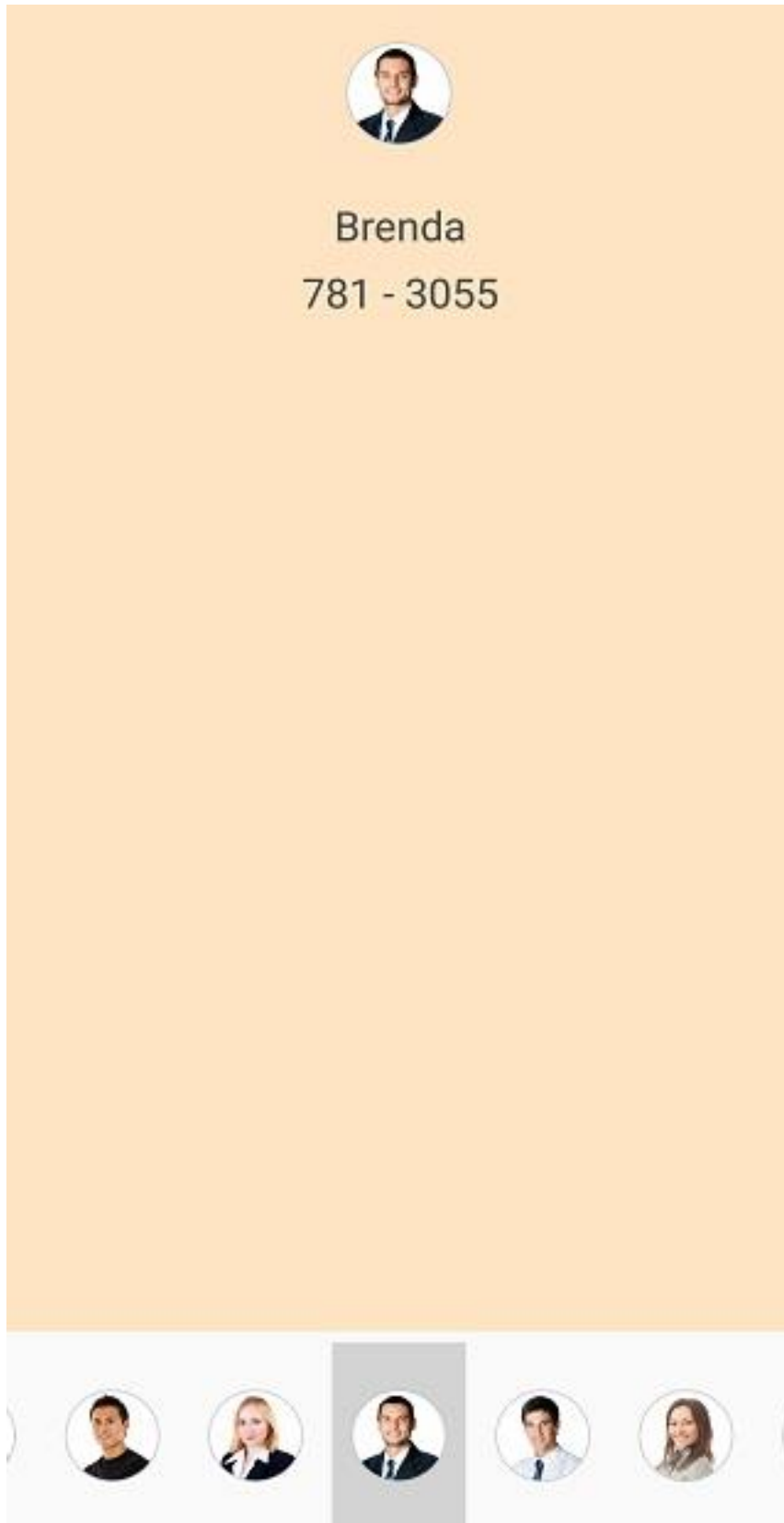
```

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        list.BindingContext = new ContactsViewModel();
        var grid = new Grid();
        var label = new Label();
        label.Text = "Tap image to expand";
        var grid1 = new Grid();
        var image = new Image();
        image.SetBinding(Image.SourceProperty, new Binding("ContactImage"));
        var label1 = new Label();
        label.SetBinding(Label.TextProperty, new Binding("ContactName"));
        var label2 = new Label();
        label1.SetBinding(Label.TextProperty, new Binding("ContactNumber"));
        grid1.Children.Add(image);
        grid1.Children.Add(label1);
        grid1.Children.Add(label2);
        var listView = new SfListView();
        ContactsViewModel contactsViewModel = new ContactsViewModel();
        listView.ItemsSource = contactsViewModel.ContactsInfo;
        listView.ItemSize = 70;
        listView.ItemTapped += ListView_ItemTapped;
        listView.ItemTemplate = new DataTemplate(() =>
        {
            var viewCell = new ViewCell();
            var image1 = new Image();
            image1.SetBinding(Image.SourceProperty, new Binding("ContactImage"));
            viewCell.View = image1;
            return viewCell;
        });
        grid.Children.Add(label);
        grid.Children.Add(grid1);
        grid.Children.Add(listView);
    }
    private void list_ItemTapped(object sender,
        Syncfusion.ListView.XForms.ItemTappedEventArgs e)
    {
        GridView.BindingContext = e.ItemData;
    }
}

```

Download the entire source code from GitHub [here](#).





### Horizontal list inside vertical list

The SfListView allows you to layout the items in horizontal list inside the vertical list. You can load the nested SfListView by customizing the [ItemTemplate](#) of outer SfListView.

You should define the size for each inner SfListView or set the [AutoFitMode](#) of inner SfListView as **Height**, and define the [ItemSize](#) for outer SfListView.

#### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.BindingContext>
<local:ListViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<Grid>
<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding OuterList}"
ItemSize="100">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Grid>
<Label Text="{Binding Label}" />
<syncfusion:SfListView ItemsSource="{Binding InnerList}" ItemSize="100"
Orientation="Horizontal">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Grid>
<Image Source="{Binding Image}"/>
</Grid>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</Grid>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</Grid>
</ContentPage>
```

#### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        var grid = new Grid();
        var label1 = new Label();
        label.SetBinding(Label.TextProperty, new Binding("Label"));
        var listView = new SfListView();
        listView.ItemsSource = OuterList;
        listView.ItemSize = 100;
        listView.ItemTemplate = new DataTemplate(() =>
        {
            var InnerListView = new SfListView();
            InnerListView.ItemSize = 100;
            InnerListView.ItemsSource = InnerList;
        });
    }
}
```

```
InnerListView.Orientation = Orientation.Horizontal;
listView.ItemTemplate = new DataTemplate(() =>
{
    var grid1 = new Grid();
    var image = new Image();
    image.SetBinding(Image.SourceProperty, new Binding("Image"));
    grid1.Children.Add(image);
}
return InnerListView;
});
grid.Children.Add(label1);
grid.Children.Add(listView);
}
```

Download the entire source code from GitHub [here](#).

Team1



Team2



Team3



Team4



Team5



Team6



Team7

### Item size

The SfListView allows customizing the size of items by setting the [ItemSize](#) property. The default value of this property is 40. This property can be customized at runtime.

#### XML

```
<syncfusion:SfListView x:Name="listView" ItemSize="60" />
```

#### C#

```
listView.ItemSize = 60;
```

---

**Note:** For vertical orientation, the item size is considered as height. For horizontal orientation, it will be considered as width.

---

### Item spacing

The SfListView allows specifying space between each item in the list by setting the [ItemSpacing](#) property. Generate the space around the item. The default value of this property is 0. This property can be customized at runtime.

#### XML

```
<syncfusion:SfListView x:Name="listView" ItemSpacing="5,0,0,0" />
```

#### C#

```
listView.ItemSpacing = new Thickness(5, 0, 0, 0)
```

<b>Aiden</b> 206-562-1210	OTHER
<b>Alexander</b> 242-798-1737	OTHER
<b>Andrew</b> 376-516-1839	OTHER
<b>Anthony</b> 331-566-1769	HOME
<b>Benjamin</b> 190-683-1509	MOBILE
<b>Bill</b> 180-694-1091	OTHER
<b>Brayden</b> 377-688-1085	MOBILE
<b>Brenda</b> 178-729-1352	MOBILE
<b>Caden</b> 348-779-1928	HOME
<b>Caleb</b>	MOBILE

### Alternate row styling

The SfListView allows applying alternate row styling for items by finding the index of the underlying object using `IValueConverter`.

### XML

```
<ContentPage xmlns:syncfusion="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <ContentPage.Resources>
    <ResourceDictionary>
      <local:IndexToColorConverter x:Key="IndexToColorConverter"/>
    </ResourceDictionary>
  </ContentPage.Resources>
</ContentPage.Content>
```

```

<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding Items}"
    ItemSize="50">
    <syncfusion:SfListView.ItemTemplate>
    <DataTemplate>
    <Grid BackgroundColor="{Binding ., Converter={StaticResource
    IndexToColorConverter}, ConverterParameter={x:Reference Name=listView}}">
    <Label Text="{Binding ContactName}" />
    <Label Text="{Binding ContactNumber}" />
    </Grid>
    </DataTemplate>
    </syncfusion:SfListView.ItemTemplate>
    </syncfusion:SfListView>
    </ContentPage.Content>
    </ContentPage>

```

**C#**

```

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        ContactsViewModel viewModel = new ContactsViewModel();
        var listView = new SfListView();
        listView.ItemsSource = viewModel.Items;
        listView.ItemSize = 50;
        listView.ItemTemplate = new DataTemplate(() =>
        {
            var grid = new Grid();
            var label1 = new Label();
            label1.SetBinding(Label.TextProperty, new Binding("ContactName"));
            var label2 = new Label();
            label2.SetBinding(Label.TextProperty, new Binding("ContactNumber"));
            grid.SetBinding(Grid.BackgroundColorProperty, new Binding(".",
            BindingMode.Default, new IndexToColorConverter(), listView));
            return grid;
        });
    }
}

```

**C#**

```

public class IndexToColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var listView = parameter as SfListView;
        var index = listView.DataSource.DisplayItems.IndexOf(value);
        if (index % 2 == 0)
            return Color.LightGray;
        return Color.Aquamarine;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
    }
}

```

```
}  
}
```

Download the entire source code from GitHub [here](#).



Kyle 760 - 3185
Gina 769 - 3408
Irene 792 - 3688
Katie 726 - 3905
Michael 763 - 3730
Oscar 769 - 3994
Josiah 795 - 3477
Oscar 792 - 3554
Hudson 790 - 3104
Nathaniel 771 - 3782
Bryson 784 - 3753
Ryder 790 - 3548

### Rounded corner on items

The SfListView allows customizing the item appearance like rounded corner by using the [Frame](#) layout in the [ItemTemplate](#) property. By defining the CornerRadius property of frame layout, you can perform the rounded corner for items.

#### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.Content>
<syncfusion:SfListView x:Name="listView" ItemSize="60" ItemsSource="{Binding
customerDetails}">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Frame x:Name="frame" CornerRadius="10" >
<StackLayout>
<Label Text="{Binding ContactName}" />
<Label Text="{Binding ContactNumber}" />
<Label Text="{Binding ContactType}" />
</StackLayout>
</Frame>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        var listView = new SfListView();
        listView.ItemsSource = customerDetails;
        listView.ItemSize = 60;
        listView.ItemTemplate = new DataTemplate(() =>
        {
            var frame = new Frame();
            frame.CornerRadius = 10;
            var stackLayout = new StackLayout();
            var label1 = new Label();
            label1.SetBinding(Label.TextProperty, new Binding("ContactName"));
            var label2 = new Label();
            label2.SetBinding(Label.TextProperty, new Binding("ContactName"));
            var label3 = new Label();
            label3.SetBinding(Label.TextProperty, new Binding("ContactType"));
            stackLayout.Children.Add(label1);
            stackLayout.Children.Add(label2);
            stackLayout.Children.Add(label3);
            frame.Children.Add(stackLayout);
            return frame;
        });
    }
}
```

Download the entire source code from GitHub [here](#).

<b>Aiden</b> 234-528-1200	WORK
<b>Alexander</b> 100-644-1704	HOME
<b>Andrew</b> 123-683-1865	WORK
<b>Anthony</b> 360-799-1553	MOBILE
<b>Benjamin</b> 158-765-1295	OTHER
<b>Bill</b> 168-696-1345	WORK
<b>Brayden</b> 253-624-1975	MOBILE
<b>Brenda</b> 103-724-1804	MOBILE
<b>Caden</b> 247-789-1208	OTHER
<b>Caleb</b> 110-785-1663	WORK
<b>Cameron</b> 303-769-1174	WORK
<b>Carter</b> 339-583-1608	OTHER

### Drop shadow effect on items

The SfListView allows customizing the item appearance like shadow effect for items by setting the shadow property of frame as true in [ItemTemplate](#) property.

**Note:** Define the frame within any view inside ItemTemplate with around some margin.

#### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.Content>
<syncfusion:SfListView x:Name="listView" ItemSize="60" ItemsSource="{Binding
customerDetails}">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Grid Padding="2" Margin="2" >
<Frame x:Name="frame" HasShadow="True" Padding="2" Margin="2">
<StackLayout>
<Label Text="{Binding ContactName}" />
<Label Text="{Binding ContactNumber}" />
<Label Text="{Binding ContactType}" />
</StackLayout>
</Frame>
</Grid>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        var listView = new SfListView();
        listView.ItemsSource = customerDetails;
        listView.ItemSize = 60;
        listView.ItemTemplate = new DataTemplate(() =>
        {
            var grid = new Grid();
            grid.Padding = 2;
            grid.Margin = 2;
            var frame = new Frame();
            frame.Padding = 2;
            frame.Margin = 2;
            frame.HasShadow = "True";
            var stackLayout = new StackLayout();
            var label1 = new Label();
            label.SetBinding(Label.TextProperty, new Binding("ContactName"));
            var label2 = new Label();
            label.SetBinding(Label.TextProperty, new Binding("ContactName"));
            var label3 = new Label();
            label1.SetBinding(Label.TextProperty, new Binding("ContactType"));
```

```
stackLayout.Children.Add(label1);  
stackLayout.Children.Add(label2);  
stackLayout.Children.Add(label3);  
frame.Children.Add(stackLayout);  
grid.Children.Add(frame);  
return grid;  
});  
}  
}
```

Download the entire source code from GitHub [here](#).

<b>Aiden</b> 119-687-1770	WORK
<b>Alexander</b> 186-753-1369	HOME
<b>Andrew</b> 365-593-1442	OTHER
<b>Anthony</b> 393-537-1549	HOME
<b>Benjamin</b> 262-645-1490	WORK
<b>Bill</b> 323-597-1395	WORK
<b>Brayden</b> 116-714-1991	WORK
<b>Brenda</b> 226-533-1609	WORK
<b>Caden</b> 310-519-1571	OTHER

### ListViewItem customization

The SfListView allows customizing the [ListViewItem](#) based on the [ItemType](#). To customize the Header, Footer, GroupHeader, LoadMore, and ListViewItem follow the code example.

#### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        this.listView.ItemGenerator = new ItemGeneratorExt(this.listView);
    }
}
```

### Extension class for ItemGenerator

#### C#

```
public class ItemGeneratorExt : ItemGenerator
{
    public SfListView listView;
    public ItemGeneratorExt(SfListView listView) : base(listView)
    {
        this.listView = listView;
    }
    protected override ListViewItem OnCreateListViewItem(int itemIndex, ItemType type, object data = null)
    {
        if (type == ItemType.Header)
            return new HeaderItemExt(this.listView);
        else if (type == ItemType.Footer)
            return new FooterItemExt(this.listView);
        else if (type == ItemType.GroupHeader)
            return new GroupHeaderItemExt(this.listView);
        else if (type == ItemType.LoadMore)
            return new LoadMoreItemExt(this.listView);
        else if (type == ItemType.Record)
            return new ListViewItemExt(this.listView);
        return base.OnCreateListViewItem(itemIndex, type, data);
    }
}
```

### Extension class for HeaderItem

#### C#

```
public class HeaderItemExt : HeaderItem
{
    private SfListView listView;
    public HeaderItemExt(SfListView listView)
    {
        this.listView = listView
    }
    protected override void OnItemAppearing()
    {
        base.OnItemAppearing();
    }
}
```



```
this.BackgroundColor = Color.Yellow;
}
}
```

*Extension class for FooterItem***C#**

```
public class FooterItemExt : FooterItem
{
    private SfListView listView;
    public FooterItemExt(SfListView listView)
    {
        this.listView = listView
    }
    protected override void OnItemAppearing()
    {
        base.OnItemAppearing();
        this.BackgroundColor = Color.Yellow;
    }
}
```

*Extension class for GroupHeaderItem***C#**

```
public class GroupHeaderItemExt : GroupHeaderItem
{
    private SfListView listView;
    public GroupHeaderItemExt(SfListView listView)
    {
        this.listView = listView
    }
    protected override void OnItemAppearing()
    {
        base.OnItemAppearing();
        this.BackgroundColor = Color.Yellow;
    }
}
```

*Extension class for LoadMoreItem***C#**

```
public class LoadMoreItemExt : LoadMoreItem
{
    private SfListView listView;
    public LoadMoreItemExt(SfListView listView)
    {
        this.listView = listView
    }
    protected override void OnItemAppearing()
    {
        base.OnItemAppearing();
        this.BackgroundColor = Color.Yellow;
    }
}
```

*Extension class for ListViewItem***C#**

```

public class ListViewItemExt : ListViewItem
{
    private SfListView listView;
    public ListViewItemExt(SfListView listView)
    {
        this.listView = listView
    }
    protected override void OnItemAppearing()
    {
        base.OnItemAppearing();
        this.BackgroundColor = Color.Yellow;
    }
}

```

*Accordion view*

The SfListView supports accordion view to display a list of items. Each item can be expanded or stretched to reveal the content associated with that item. There can be zero expanded items, exactly one item, or more than one item can be expanded at a time depending on the configuration.

**XML**

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
    <ContentPage.Behaviors>
        <local:SfListViewAccordionBehavior />
    </ContentPage.Behaviors>
    <ContentPage.Content>
        <Grid x:Name="mainGrid" BackgroundColor="#F0F0F0" Padding="4">
            <syncfusion:SfListView x:Name="listView" AutoFitMode="Height" SelectionMode
            ="None" IsScrollBarVisible="False" ItemSpacing="0">
                <syncfusion:SfListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <ViewCell.View>
                                <Grid Padding="2" Margin="1" BackgroundColor="#F0F0F0" >
                                    <Frame x:Name="frame" CornerRadius="2" Padding="1" Margin="1"
                                    OutlineColor="White">
                                        <Grid VerticalOptions="FillAndExpand" BackgroundColor="White"
                                        HorizontalOptions="FillAndExpand">
                                            <Grid.RowDefinitions>
                                                <RowDefinition Height="Auto" />
                                            </Grid.RowDefinitions>
                                            <Grid x:Name="grid" >
                                                <Grid.RowDefinitions>
                                                    <RowDefinition Height="60" />
                                                </Grid.RowDefinitions>
                                                <Grid RowSpacing="0">
                                                    <Grid.ColumnDefinitions>
                                                        <ColumnDefinition Width="60" />
                                                        <ColumnDefinition Width="*" />
                                                        <ColumnDefinition Width="50" />

```

```

</Grid.ColumnDefinitions>
<Image Grid.Row="0" Grid.Column="0" Source="{Binding ContactImage}"
VerticalOptions="Center"/>
<Grid Grid.Row="0" Grid.Column="1" VerticalOptions="CenterAndExpand">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Label Grid.Row="0" LineBreakMode="NoWrap" TextColor="#474747"
Text="{Binding ContactName}" />
<Label Grid.Row="1" TextColor="#474747" LineBreakMode="NoWrap"
Text="{Binding CallTime}" />
</Grid>
<Grid Grid.Row="0" Grid.Column="2" HorizontalOptions="Center"
VerticalOptions="Center">
<Image Source="{Binding PhoneImage}" HeightRequest="20" WidthRequest="20"
HorizontalOptions="Center" VerticalOptions="Center" />
</Grid>
</Grid>
</Grid>
<Grid IsVisible="{Binding IsVisible, Mode=TwoWay}" ColumnSpacing="0"
RowSpacing="0" Grid.Row="1" BackgroundColor="White"
HorizontalOptions="FillAndExpand" Padding="5"
VerticalOptions="FillAndExpand">
<Grid.RowDefinitions>
<RowDefinition Height="1" />
<RowDefinition Height="40" />
<RowDefinition Height="40" />
<RowDefinition Height="40" />
<RowDefinition Height="40" />
<RowDefinition Height="40" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions >
<ColumnDefinition Width="50" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<BoxView Grid.Row="0" Grid.Column="0" BackgroundColor="LightGray" />
<Image Grid.Row="1" Grid.Column="0" Source="{Binding NewContact}" />
<Image Grid.Row="2" Grid.Column="0" Source="{Binding AddContact}" />
<Image Grid.Row="3" Grid.Column="0" Source="{Binding SendMessage}" />
<Image Grid.Row="4" Grid.Column="0" Source="{Binding BlockSpan}" />
<Image Grid.Row="5" Grid.Column="0" Source="{Binding CallDetails}" />
<BoxView Grid.Row="0" Grid.Column="1" BackgroundColor="LightGray" />
<Label Grid.Row="1" Grid.Column="1" Text="Create new contact"
TextColor="#000000" />
<Label Grid.Row="2" Grid.Column="1" Text="Add to a contact"
TextColor="#000000"/>
<Label Grid.Row="3" Grid.Column="1" Text="Send a message"
TextColor="#000000" />
<Label Grid.Row="4" Grid.Column="1" Text="Block/report Spam"
TextColor="#000000" />
<Label Grid.Row="5" Grid.Column="1" Text="Call details" TextColor="#000000"
/>
</Grid>
</Grid>
</Frame>
</Grid>

```

```

</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</Grid>
</ContentPage.Content>
</ContentPage>

```

Accordion view can be displayed by defining two different ItemTemplates. The ItemTemplates can be enabled or disabled in the [ItemTapped](#) event.

## C#

```

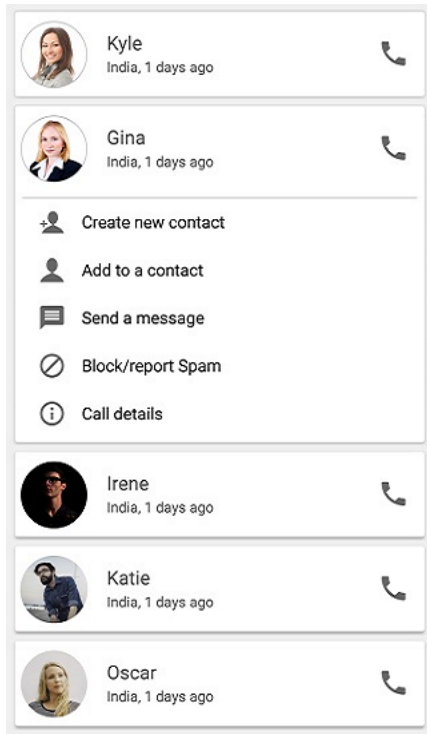
internal class SfListViewAccordionBehavior : Behavior<ContentPage>
{
    #region Fields
    private Contact tappedItem;
    private Syncfusion.ListView.XForms.SfListView listview;
    private AccordionViewModel AccordionViewModel;
    #endregion
    #region Properties
    public SfListViewAccordionBehavior()
    {
        AccordionViewModel = new AccordionViewModel();
    }
    #endregion
    #region Override Methods
    protected override void OnAttachedTo(ContentPage bindable)
    {
        listview =
        bindable.FindByName<Syncfusion.ListView.XForms.SfListView>("listView");
        listview.ItemsSource = AccordionViewModel.ContactsInfo;
        listview.ItemTapped += ListView_ItemTapped;
    }
    #endregion
    #region Private Methods
    using Syncfusion.ListView.XForms.Control.Helpers;
    private void ListView_ItemTapped(object sender,
        Syncfusion.ListView.XForms.ItemTappedEventArgs e)
    {
        if (tappedItem != null && tappedItem.IsVisible)
        {
            var previousIndex = listview.DataSource.DisplayItems.IndexOf(tappedItem);
            tappedItem.IsVisible = false;
            if (Device.RuntimePlatform != Device.macOS)
                Device.BeginInvokeOnMainThread(() => {
                    listview.RefreshListViewItem(previousIndex, previousIndex, false); });
        }
        if (tappedItem == (e.ItemData as Contact))
        {
            if (Device.RuntimePlatform == Device.macOS)
            {
                var previousIndex = listview.DataSource.DisplayItems.IndexOf(tappedItem);
                Device.BeginInvokeOnMainThread(() => {
                    listview.RefreshListViewItem(previousIndex, previousIndex, false); });
            }
        }
    }
}

```

```
}
tappedItem = null;
return;
}
tappedItem = e.ItemData as Contact;
tappedItem.IsVisible = true;
if (Device.RuntimePlatform == Device.macOS)
{
    var visibleLines =
    this.listView.GetVisualContainer().ScrollRows.GetVisibleLines();
    var firstIndex = visibleLines[visibleLines.FirstBodyVisibleIndex].LineIndex;
    var lastIndex = visibleLines[visibleLines.LastBodyVisibleIndex].LineIndex;
    Device.BeginInvokeOnMainThread(() => {
        listView.RefreshListViewItem(firstIndex, lastIndex, false);
    });
}
else
{
    var currentIndex = listView.DataSource.DisplayItems.IndexOf(e.ItemData);
    Device.BeginInvokeOnMainThread(() => {
        listView.RefreshListViewItem(currentIndex, currentIndex, false);
    });
}
#endregion
protected override void OnDetachingFrom(ContentPage bindable)
{
    listView.ItemTapped -= ListView_ItemTapped;
}
}
```

The `IsVisible` model property which is bound to the second template will be enabled when tapping the item and disabled when tapping again the same item.

Download the entire source code from GitHub [here](#).



### show busy indicator on list view

The SfListView allows displaying the [SfBusyIndicator](#) when loading the bounded items. The busy indicator can be enabled and disabled by using [IsBusy](#) property.

Create a IsLoading boolean property in view model and bind it to the IsBusy property. By setting the value to IsLoading property, the busy indicator will be enabled and disabled into the view till the items loaded in the SfListView.

### XML

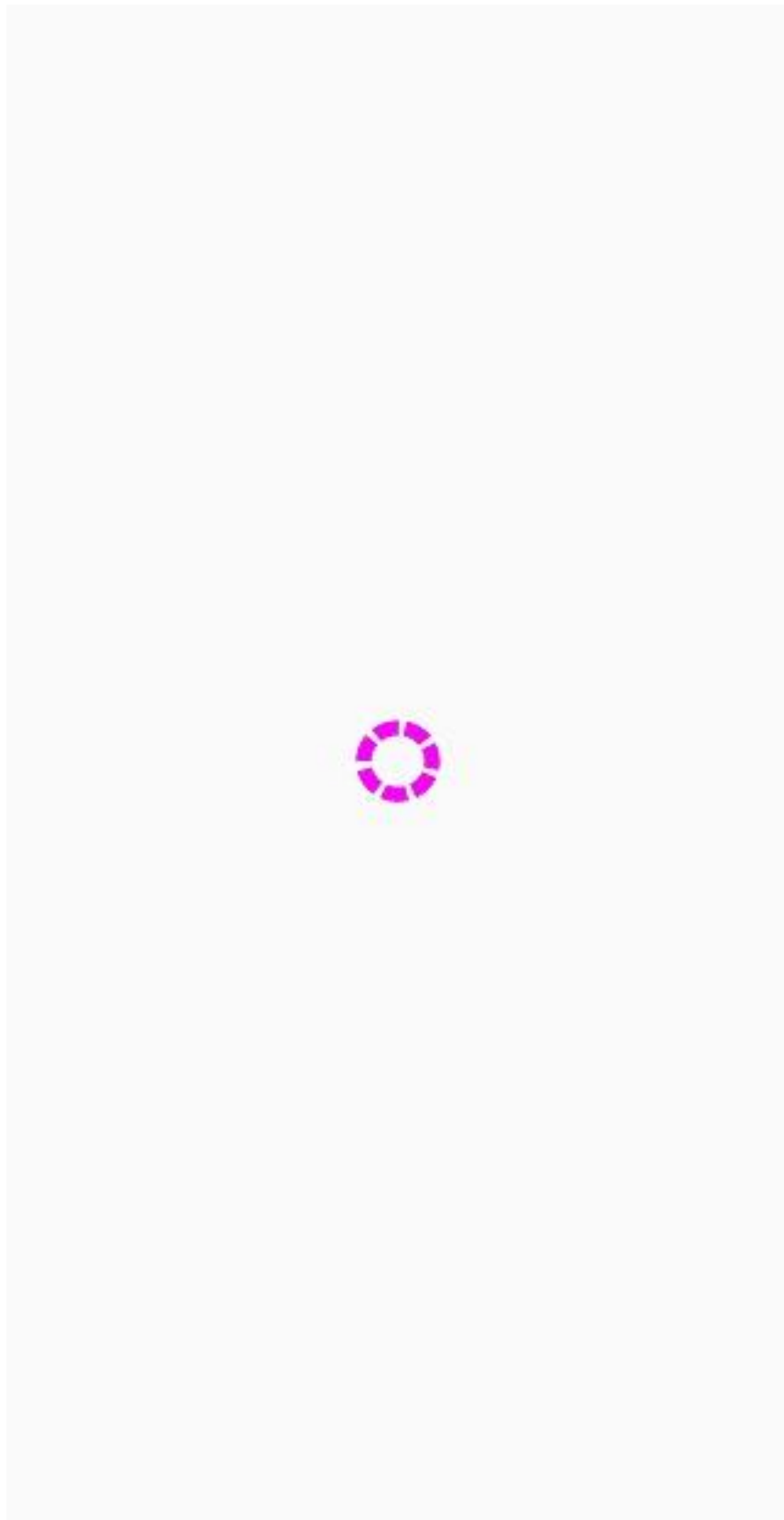
```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:busyIndicator="clr-
namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndica
tor.XForms">
<Grid>
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding ContactInfo}"
ItemSize="110">
</syncfusion:SfListView>
<busyIndicator:SfBusyIndicator x:Name="busyIndicator"
InputTransparent="True"
AnimationType="SingleCircle"
IsBusy="{Binding IsLoading, Mode=TwoWay}"
TextColor="Magenta"
ViewBoxWidth="50"
ViewBoxHeight="50"/>
</Grid>
</ContentPage>
```

### C#

```
public class ViewModel : INotifyPropertyChanged
{
    private bool isLoading = false;
    public bool IsLoading
    {
        get { return isLoading; }
        set
        {
            this.isLoading = value;
            OnPropertyChanged("IsLoading");
        }
    }
    private async void GenerateItems()
    {
        IsLoading = true;
        await Task.Delay(5000);
        for (int i = 0; i < 30; i++)
        {
            var contact = new Model(CustomerNames[i], ContactNumber[i]);
            ContactInfo.Add(contact);
        }
        IsLoading = false;
    }
}
```

**Note:** When both SfBusyIndicator and ListView loaded with same row and column, you need to set InputTransparent as True to SfBusyindicator in order to pass touch interaction to listview in iOS platform.

Download the entire source code from GitHub [here](#).





## **Object-Oriented Programming in C#**

Object-oriented programming is the de facto programming paradigm for many programming languages Microsoft Visual Studio 2015 is the new version of the widely-used integrated development environment for building modern.

### **C# Code Contracts**

Code Contracts provide a way to convey code assumptions in your .NET applications, Neural networks are an exciting field of software development used to calculate outputs from input data.

### **Machine Learning Using C#**

Object-oriented programming is the de facto programming paradigm for many programming languages Microsoft Visual Studio 2015 is the new version of the widely-used integrated development environment for building modern.

### **Neural Networks Using C#**

Code Contracts provide a way to convey code assumptions in your .NET applications, Neural networks are an exciting field of software development used to calculate outputs from input data.

### **Visual Studio Code**

You'll learn several different approaches to applying machine learning to data analysis.

## **Android Programming**

Neural networks are an exciting field of software development used to calculate outputs from input data.

## **iOS Succinctly**

It is a powerful tool for editing code and serves as a complete environment for end-to-end programming.

## **Visual Studio 2015**

In Android Programming Succinctly, Ryan Hodson provides

### show busy indicator on list view items

The SfListView allows displaying an activity indicator for an item when its data is being loaded in the background. To perform this, load both `ActivityIndicator` and a `Button` in the same row of a `Grid` element inside the [ItemTemplate](#) of the SfListView. The busy indicator and button can be enabled and disabled by using properties `IsButtonVisible` and `IsIndicatorVisible` respectively in the model class.

#### C#

```
public class BookInfo : INotifyPropertyChanged
{
    private string bookName;
    private string bookDescription;
    public bool isDescriptionVisible;
    public bool isButtonVisible;
    public bool isIndicatorVisible;
    public BookInfo()
    {
    }
    public string BookName
    {
        get { return bookName; }
        set
        {
            bookName = value;
            OnPropertyChanged("BookName");
        }
    }
    public bool IsDescriptionVisible
    {
        get { return isDescriptionVisible; }
        set
        {
            isDescriptionVisible = value;
            OnPropertyChanged("IsDescriptionVisible");
        }
    }
    public string BookDescription
    {
        get { return bookDescription; }
        set
        {
            bookDescription = value;
            OnPropertyChanged("BookDescription");
        }
    }
    public bool IsButtonVisible
    {
        get { return isButtonVisible; }
        set
        {
            isButtonVisible = value;
            OnPropertyChanged("IsButtonVisible");
        }
    }
    public bool IsIndicatorVisible
    {

```

```

get { return isIndicatorVisible; }
set
{
    isIndicatorVisible = value;
    OnPropertyChanged("IsIndicatorVisible");
}
}
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string name)
{
    if (this.PropertyChanged != null)
        this.PropertyChanged(this, new PropertyChangedEventArgs(name));
}
}

```

Disable the visibility of Description and ActivityIndicator initially while adding items into collection.

### C#

```

public class BookInfoRepository : INotifyPropertyChanged
{
    private ObservableCollection<BookInfo> newBookInfo;
    public event PropertyChangedEventHandler PropertyChanged;
    public ObservableCollection<BookInfo> NewBookInfo
    {
        get { return newBookInfo; }
        set { this.newBookInfo = value; }
    }
    public void OnPropertyChanged(string name)
    {
        if (this.PropertyChanged != null)
            this.PropertyChanged(this, new PropertyChangedEventArgs(name));
    }
    public BookInfoRepository()
    {
        GenerateNewBookInfo();
    }
    private void GenerateNewBookInfo()
    {
        NewBookInfo = new ObservableCollection<BookInfo>();
        NewBookInfo.Add(new BookInfo() { BookName = "Machine Learning Using C#",
            BookDescription = "You'll learn several different approaches to applying
            machine learning", IsIndicatorVisible = false, IsButtonVisible = true,
            IsDescriptionVisible = false });
        NewBookInfo.Add(new BookInfo() { BookName = "Object-Oriented Programming in
            C#", BookDescription = "Object-oriented programming is the de facto
            programming paradigm", IsIndicatorVisible = false, IsButtonVisible = true,
            IsDescriptionVisible = false });
        NewBookInfo.Add(new BookInfo() { BookName = "C# Code Contracts",
            BookDescription = "Code Contracts provide a way to convey code assumptions",
            IsIndicatorVisible = false, IsButtonVisible = true, IsDescriptionVisible =
            false });
    }
}

```

Bind the bool values for the IsVisible properties to switch between indicator and button while loading the description.

### XML

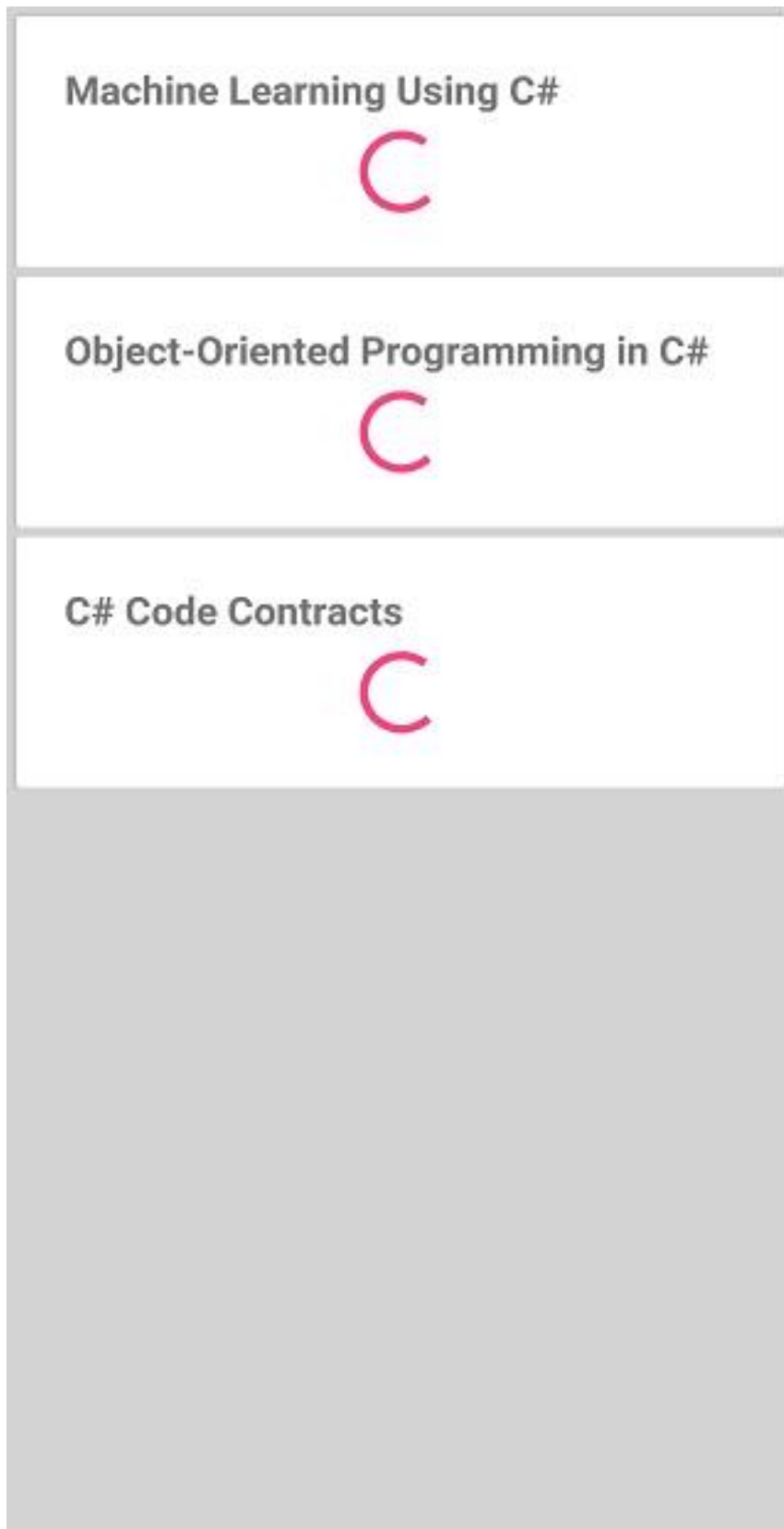
```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.BindingContext>
<local:BookInfoRepository x:Name="ViewModel" />
</ContentPage.BindingContext>
<sync:SfListView x:Name="listView" AutoFitMode="Height"
BackgroundColor="#d3d3d3" SelectionMode="None" ItemsSource="{Binding
NewBookInfo}">
<sync:SfListView.ItemTemplate>
<DataTemplate>
<Frame HasShadow="True" Margin="5,5,5,0">
<Grid Padding="5">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="2*" />
</Grid.RowDefinitions>
<Label Text="{Binding BookName}" FontAttributes="Bold" FontSize="19" />
<Button Grid.Row="1" Clicked="Button_Clicked" Text="Load Description"
IsVisible="{Binding IsButtonVisible}" HorizontalOptions="Center"
VerticalOptions="Center"/>
<Label Grid.Row="1" Text="{Binding BookDescription}" FontSize="15"
IsVisible="{Binding IsDescriptionVisible}" />
<ActivityIndicator Grid.Row="1" IsEnabled="True" IsRunning="True"
IsVisible="{Binding IsIndicatorVisible}" />
</Grid>
</Frame>
</DataTemplate>
</sync:SfListView.ItemTemplate>
</sync:SfListView>
</ContentPage>
```

In the Clicked event of the Button, get the row data from its BindingContext and alter the bool values accordingly.

### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
    private async void Button_Clicked(object sender, EventArgs e)
    {
        var model = ((sender as Button).BindingContext as BookInfo);
        model.IsIndicatorVisible = true;
        model.IsButtonVisible = false;
        await Task.Delay(2000);
        model.IsDescriptionVisible = true;
        model.IsIndicatorVisible = false;
    }
}
```

Download the entire source code from GitHub [here](#).



**Machine Learning Using C#**

You'll learn several different approaches to applying machine learning

**Object-Oriented Programming in C#**

Object-oriented programming is the de facto programming paradigm

**C# Code Contracts**

Code Contracts provide a way to convey code assumptions

### Show busy indicator on list view items using toggle switch

The SfListView allows to display **ActivityIndicator** for an item when loading its data in the background. To do this, load both **ActivityIndicator** and a toggle switch in the same row of a **Grid** element inside the **ItemTemplate** of SfListView. The busy indicator and toggle switch can be enabled and disabled by using the **IsButtonVisible** and **IsIndicatorVisible** properties respectively in the model class. The **ActivityIndicator** remains visible when the toggle switch is enabled.

#### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.BindingContext>
<local:ContactInfoRepository x:Name="ViewModel" />
</ContentPage.BindingContext>
<syncfusion:SfListView x:Name="listView" AutoFitMode="Height"
BackgroundColor="#d3d3d3" SelectionMode="None" ItemsSource="{Binding
NewContactInfo}">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Frame HasShadow="True" Margin="5,5,5,0">
<Grid Padding="5">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Label Text="{Binding ContactName}" FontAttributes="Bold" FontSize="19" />
<Switch Grid.Row="1" Grid.Column="1" IsVisible="{Binding IsButtonVisible}"
IsToggled="{Binding IsChecked}" Toggled="Switch_Toggled"/>
<Label Grid.Row="1" Text="{Binding ContactNo}" FontSize="15"
IsVisible="{Binding IsDescriptionVisible}" />
<ActivityIndicator Grid.Row="1" IsEnabled="True" IsRunning="True"
IsVisible="{Binding IsIndicatorVisible}" />
</Grid>
</Frame>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</ContentPage>
```

In the Toggled event of the switch, get the row data from its BindingContext and alter the Bool values accordingly.

#### C#

```
public partial class MainPage : ContentPage
{
    private Random random = new Random();
    public MainPage()
    {
        InitializeComponent();
    }
    private async void Switch_Toggled(object sender, ToggledEventArgs e)
    {
        var model = ((sender as Switch).BindingContext as ContactInfo);
        if (model.IsChecked == true)
```

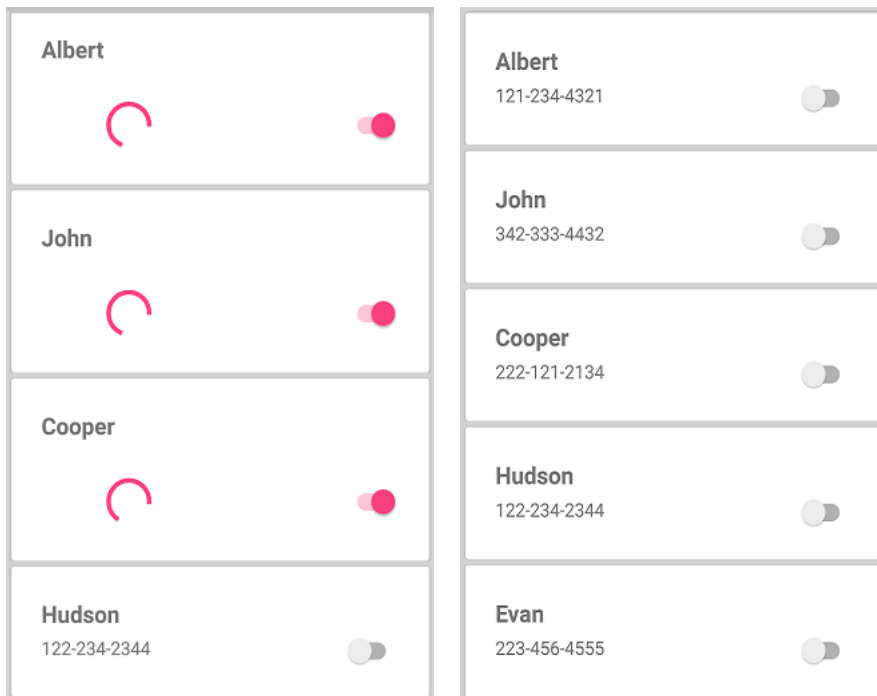


```

{
    model.ContactNo = random.Next(100, 400).ToString() + "-" + random.Next(500,
800).ToString() + "-" + random.Next(1000, 2000).ToString();
    model.IsDescriptionVisible = false;
    model.IsIndicatorVisible = true;
    await Task.Delay(2000);
    model.IsDescriptionVisible = true;
    model.IsIndicatorVisible = false;
    model.IsChecked = false;
}
else
{
    model.IsIndicatorVisible = false;
}
}
}

```

Download the entire source code from GitHub [here](#).



#### Item animation on appearing

The SfListView supports animating the items by using an [OnItemAppearing](#) virtual method. It is raised when the items appearing in the view on scrolling, loading, and navigating from one page to another page. To apply the animation effect for items, follow the steps:

#### Extension of ItemGenerator

##### C#

```

public class ItemGeneratorExt : ItemGenerator
{
    public SfListView listView;
    public ItemGeneratorExt(SfListView listView) : base(listView)
    {
    }
}

```

```

this.listView = listView;
}
protected override ListViewItem OnCreateListViewItem(int itemIndex, ItemType
type, object data = null)
{
if (type == ItemType.Record)
return new ListViewItemExt(this.listView);
return base.OnCreateListViewItem(itemIndex, type, data);
}
}

```

Initialize and assign ItemGenerator extension to ListView

**C#**

```

public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
this.listView.ItemGenerator = new ItemGeneratorExt(this.listView);
}
}

```

#### *Extension of ListViewItem*

To apply the animation for items while appearing, override the [OnItemAppearing](#) method.

**C#**

```

public class ListViewItemExt : ListViewItem
{
private SfListView listView;
public ListViewItemExt(SfListView listView)
{
this.listView = listView;
this.PropertyChanged += ListViewItemExt_PropertyChanged;
}
private void ListViewItemExt_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
{
if (e.PropertyName == "Visibility")
{
if (!this.Visibility)
this.AbortAnimation("FadeTo");
}
}
protected override void OnItemAppearing()
{
this.Opacity = 0;
this.FadeTo(1, 400, Easing.SinInOut);
base.OnItemAppearing();
}
}

```

Here **FadeTo** animation is applied for [ListViewItem](#), when comes in the view.

Download the entire source code from GitHub [here](#).

## C# Code Contracts

Code Contracts provide a way to convey code assumptions

## Machine Learning Using C#

You'll learn several different approaches to applying machine learning

## Neural Networks Using C#

Neural networks are an exciting field of software development

## Visual Studio Code

It is a powerful tool for editing code and serves for end-to-end programming

## Android Programming

It provides a useful overview of the Android application lifecycle

## iOS Succinctly

It is for developers looking to step into the frightening world of iPhone

### *Right to left(RTL)*

ListView supports to change the flow of text to the right-to-left direction by setting the [FlowDirection](#) property. ListView supports RTL in Xamarin.Forms version 3.0 and above.

### XML

```
<ContentPage xmlns:syncfusion="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView"
    ItemsSource="{Binding Products}"
    FlowDirection="RightToLeft"/>
</ContentPage>
```

### C#






```
this.FlowDirection = FlowDirection.RightToLeft;
```

In UWP platform, the ScrollView is not changed when RTL is enabled (framework issue). To overcome this issue, set the `FlowDirection` property in constructor of `MainPage` in UWP renderer as demonstrated in the following code example.

### C#

```
public MainPage()
{
    ...
    SfListViewRenderer.Init();
    this.FlowDirection = FlowDirection.RightToLeft;
    LoadApplication (new App ());
    ...
}
```

**Note:** When a label is loaded in the `ItemTemplate`, the right-to-left direction is not applied due to the framework issue. It has been reported to the Xamarin team; for more details about this, refer to this [link](#). To overcome this issue, set the `HorizontalOptions` to `StartAndExpand` in Label.

A		
MOBILE	Anthony 100-693-1167	
HOME	Asher 301-649-1747	
WORK	Austin 182-605-1921	
B		
C		
MOBILE	Caden 210-684-1734	
BUSINESS	Caleb 327-750-1357	

### Limitations

- ListView does not support the right-to-left(RTL) direction when [SfListView.Orientation](#) is `Horizontal`.

## Scrolling

### Programmatic scrolling

#### Scrolling to row index

The SfListView allows programmatically scrolling based on the index by using the [ScrollToRowIndex](#) method for both linear and grid layouts. It also enables and disables the scrolling animation when changing the view. By default, the scrolling will be animated.

You can set position of item in view while scrolling by passing [ScrollToPosition](#) to [ScrollToRowIndex](#) method. Below are four different types of positions:

- **MakeVisible**: Scrolls a specific item to make visible in the view. If the item is already in view, scrolling will not occur.
- **Start**: Scrolls a specific item to be positioned at the begin of the view.
- **End**: Scrolls a specific item to be positioned at the end of the view.
- **Center**: Scrolls a specific item to be positioned at the center of the view.

You can also scroll to specified data in [SfListView](#) using the [ScrollTo](#) method.

#### C#

```
int index =  
listView.DataSource.DisplayItems.IndexOf(viewModel.Customers[2]);  
// Programmatic scrolling based on the item index.  
listView.LayoutManager.ScrollToRowIndex(index,  
Syncfusion.ListView.XForms.ScrollToPosition.Center, true);  
// Programmatic scrolling based on the item data.  
listView.ScrollTo(viewModel.Customers[index],  
Syncfusion.ListView.XForms.ScrollToPosition.Center, true);
```

**Note:** If grouping is enabled, get the desired row index by passing the underlying data in the [DisplayItems.IndexOf](#) method.

#### C#

```
int index =  
listView.DataSource.DisplayItems.IndexOf(viewModel.Customers[2]);  
listView.LayoutManager.ScrollToRowIndex(index, true);
```

### Limitations

- When [AutoFitMode](#) is [Height](#) or grouping is enabled, the scroll animation will be disabled by default in Android and iOS platforms.
- If [ScrollToRowIndex](#) method is called when loading the [SfListView](#), set [disableAnimation](#) to [true](#) to scroll to the appropriate row index, or else view does not scrolled in Android.
- If the [ScrollToRowIndex](#) method is applied to a particular item index while the item is in Grouping or AutoFit mode, the particular item will get displayed in view but not in the exact position when the [ScrollToPosition](#) property is set as [MakeVisible](#) or [Center](#) for first time.
- In UWP platform, when you hover and scroll the inner listview, the outer listview will not be scrolled by default. To overcome this, set the [InputTransparent](#) to [True](#) for the parent element

loaded in the `ItemTemplate`. If you set `InputTransparent`, then the inner listview scroll will not work. This is the default behavior of the listview.

### Scrollbar visibility

The SfListView provides an option to enable or disable the `Scrollbar` visibility by using the [IsScrollBarVisible](#) property. By default, the value will be `true`.

**Note:** Due to some restrictions in native ScrollView renderer in Xamarin.Forms, you cannot change the `IsScrollBarVisible` value at runtime. It can be defined only when initializing the SfListView.

### XML

```
<syncfusion:SfListView x:Name="listView" IsScrollBarVisible="False" />
```

### C#

```
listView.IsScrollBarVisible = false;
```

### Identifying scroll state changes

The SfListView will notify the scrolling state changes by using the [ScrollStateChanged](#) event.

Following states will be notified via [ScrollState](#) property in the event argument.

- Dragging: Specifies that SfListView is currently being dragged in the view.
- Flung: Specifies that fling action is performed on the SfListView.
- Idle: Specifies that SfListView is not currently scrolling.
- Programmatic: Specifies that scrolling is performed by using [ScrollTo](#) or [ScrollToRowIndex](#) method.

### C#

```
listView.ScrollStateChanged += ListView_ScrollStateChanged;
private void ListView_ScrollStateChanged(object sender,
ScrollStateChangedEventArgs e)
{
    if (e.ScrollState == ScrollState.Idle)
    {
        DisplayAlert("ScrollState", "Scrolling has stopped", "OK");
    }
}
```

### Identify when end of the list is reached on scrolling

The SfListView allows notifying when scrolling using [Changed](#) event of [ScrollAxisBase](#) in [VisualContainer](#) of the SfListView. By using this event, you can find whether reached the last item in the list in the SfListView based on [LastBodyVisibleLineIndex](#) property and underlying collection count.

### C#

```
using Syncfusion.ListView.XForms.Control.Helpers;
public partial class MainPage : ContentPage
{
    VisualContainer visualContainer;
```

```

bool showAlertShown = false;
public MainPage()
{
    InitializeComponent();
    visualContainer = listView.GetVisualContainer();
    visualContainer.ScrollRows.Changed += ScrollRows_Changed;
}
///<summary>
///To notify when end reached
///</summary>
private void ScrollRows_Changed(object sender, ScrollChangedEventArgs e)
{
    var lastIndex = visualContainer.ScrollRows.LastBodyVisibleLineIndex;
    //To include header if used
    var header = (listView.HeaderTemplate != null && !listView.IsStickyHeader) ?
    1 : 0;
    //To include footer if used
    var footer = (listView.FooterTemplate != null && !listView.IsStickyFooter) ?
    1 : 0;
    var totalItems = listView.DataSource.DisplayItems.Count + header + footer;
    if ((lastIndex == totalItems - 1))
    {
        if (!isAlertShown)
        {
            DisplayAlert("Alert", "End of list reached...", "Ok");
            showAlertShown = true;
        }
    }
    else
    {
        showAlertShown = false;
    }
}
}

```

Download the entire source code from GitHub [here](#).

Maintain the scroll position while updating ItemsSource at runtime

The SfListView have scrolled to top automatically when changing the ItemsSource at runtime. However, you can maintain the same scrolled position by using the **SCROLLY** value of ExtendedScrollView from the [VisualContainer](#). After changing the ItemsSource, you can pass the SCROLLY value to [ScrollTo](#) method of SfListView and scroll back to the same position.

For horizontal orientation, use the **SCROLLX** value of ExtendedScrollView.

By using [Reflection](#), get the value of **ScrollOwner** from **VisualContainer** and use it.

## C#

```

using Syncfusion.ListView.XForms.Control.Helpers;
public partial class MainPage : ContentPage
{
    ExtendedScrollView scrollView;
    public MainPage()
    {
        InitializeComponent();
        scrollView = listView.GetScrollView();
    }
}

```

```
private void ChangeItemsSource_Clicked(object sender, EventArgs e)
{
    var viewModel = new ContactsViewModel();
    listView.ItemsSource = viewModel.EmployeeInfo;
    listView.ScrollTo(scrollView.ScrollY);
}
}
```

Download the entire source code from GitHub [here](#).

### Item Size Customization

This section explains how to customize the item size in the SfListView.

#### Customize item size of a particular item on-demand

The SfListView allows customizing the size of the item on-demand by the [SfListView.QueryItemSize](#) event using the item index. This event is raised whenever items come to view and triggered with [QueryItemSizeEventArgs](#).

The [SfListView.QueryItemSize](#) event provides the following properties in their arguments:

- [ItemIndex](#): Identifies a particular item in the SfListView.
- [ItemData](#): Identifies the underlying data bound to that item.
- [ItemSize](#): Identifies size of the queried item. For vertical orientation, it will be considered as the item height. For horizontal orientation, it will be considered as the item width.
- [ItemType](#): Identifies the item type of the queried item.
- [Handled](#): Decides whether the specified size can be set to the item or not. The default value is false. When this property is not set, the decided size will not set to the item.

### C#

```
this.listView.QueryItemSize += ListView_QueryItemSize;
private void ListView_QueryItemSize(object sender,
Syncfusion.ListView.XForms.QueryItemSizeEventArgs e)
{
    if(e.ItemIndex == 1)
    {
        e.ItemSize = 300;
        e.Handled = true;
    }
}
```

Download the entire source code from GitHub [here](#).



## AutoFitItems



### Neural Networks Using C#

James McCaffrey

Neural networks are an exciting field of software development used to calculate outputs from input data.



### C# Code Contracts

Dirk Strauss

Code Contracts provide a way to convey code assumptions in your .NET applications. In C# Code Contracts Succinctly, author Dirk Strauss demonstrates how to use Code Contracts to validate logical correctness in code, how they can be integrated with abstract classes and interfaces, and even how they can be used to make writing documentation less painful. Learn to write applications that support different languages and cultures, with an emphasis on .NET development. With the help of author Jonas Gauffin, Localization for .NET Succinctly will help you become an effective developer in the global community. Data Structures is your concise guide.



### Machine Learning Using C#

James McCaffrey

In Machine Learning Using C# Succinctly, you'll learn several different approaches to applying machine learning to data analysis and



### Object-Oriented Programming in C#

Sander Rossel

Object-oriented programming is the de facto programming paradigm for many programming languages. Object-Oriented Programming in



### Visual Studio Code

Alessandro Del Sole

Visual Studio Code is a powerful tool for editing code and serves as a complete environment for

### AutoFit the items based on the content

The SfListView allows dynamically adjusting size of items based on the content loaded in the [SfListView.ItemTemplate](#) by defining the [SfListView.AutoFitMode](#) property.

The control contains the following three types of [AutoFitMode](#):

- Height: AutoFit the items based on the content.
- DynamicHeight: AutoFit the items based on the content if size of the content is changed at run time.
- None: The [SfListView](#) items are layout by [SfListView.ItemSize](#).

---

**Note:** If you define any size manually to the view loaded in [SfListView.ItemTemplate](#), the [SfListView](#) will return that size as the item size for each item.

---

### *AutoFitMode as Height*

AutoFit considers height of the item when [SfListView.Orientation](#) is vertical. When [SfListView.Orientation](#) is horizontal, it considers width of the item. The [SfListView.GridLayout](#) AutoFit all the items in a row and takes the maximum item height of the row and applies to all other items in the row.

### **XML**

```
<ContentPage xmlns:syncfusion="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView"
    ItemSize="200"
    AutoFitMode="Height"
    ItemsSource="{Binding BookInfo}" />
</ContentPage>
```

### **C#**

```
listView.AutoFitMode = AutoFitMode.Height;
```

Download the entire source code from GitHub [here](#).

## AutoFitItems



### Neural Networks Using C#

James McCaffrey

Neural networks are an exciting field of software development used to calculate outputs from input data.



### C# Code Contracts

Dirk Strauss

Code Contracts provide a way to convey code assumptions in your .NET applications. In C# Code Contracts Succinctly, author Dirk Strauss demonstrates.



### Machine Learning Using C#

James McCaffrey

In Machine Learning Using C# Succinctly, you'll learn several different approaches to applying machine learning to data analysis and prediction problems.



### Object-Oriented Programming in C#

Sander Rossel

Object-oriented programming is the de facto programming paradigm for many programming languages. Object-Oriented Programming in C# Succinctly provides an introduction to OOP for C# developers.



### Visual Studio Code

Alessandro Del Sole

Visual Studio Code is a powerful tool for editing code and serves as a complete environment for end-to-end programming. Alessandro Del Sole Visual Studio Code Succinctly will guide readers to mastery of this valuable tool so that they can make full use of its features.

### *AutoFitMode as DynamicHeight*

AutoFit considers height of the item when `SfListView.Orientation` is vertical. When `SfListView.Orientation` is horizontal, it considers width of the item. The `SfListView.GridLayout` AutoFit all the items in a row and takes the maximum item height of the row and applies to all other items in the row.

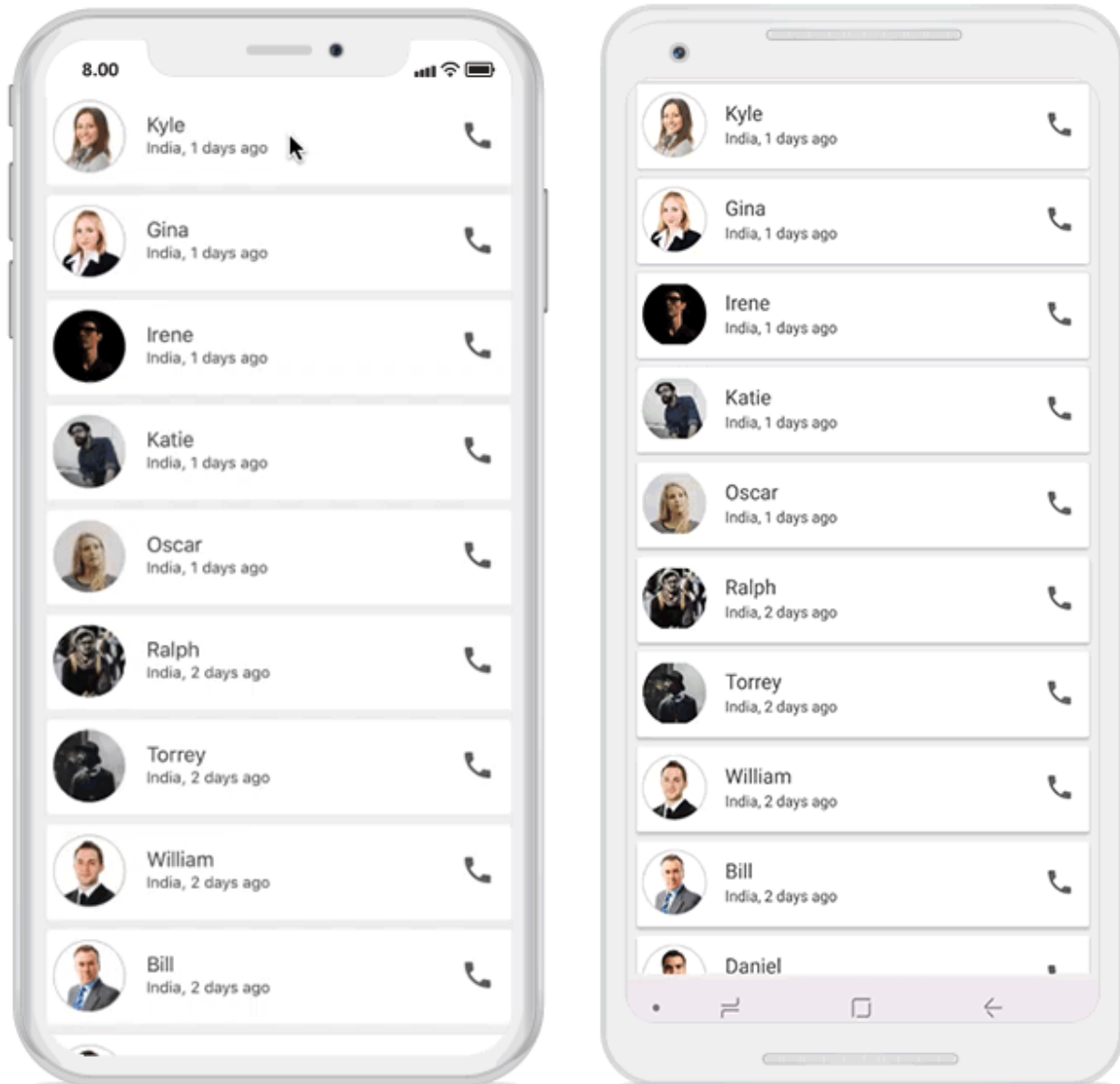
#### **XML**

```
<ContentPage xmlns:syncfusion="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView"
    ItemSize="200"
    AutoFitMode="DynamicHeight"
    ItemsSource="{Binding BookInfo}" />
</ContentPage>
```

#### **C#**

```
listView.AutoFitMode = AutoFitMode.DynamicHeight;
```

You can download the entire source code of this demo [here](#).



Updating the listview item size based on font at runtime

ListView allows you to resize the item size based on the change in font size of the label element at runtime when `SfListView.AutoFitMode` is [DynamicHeight](#).

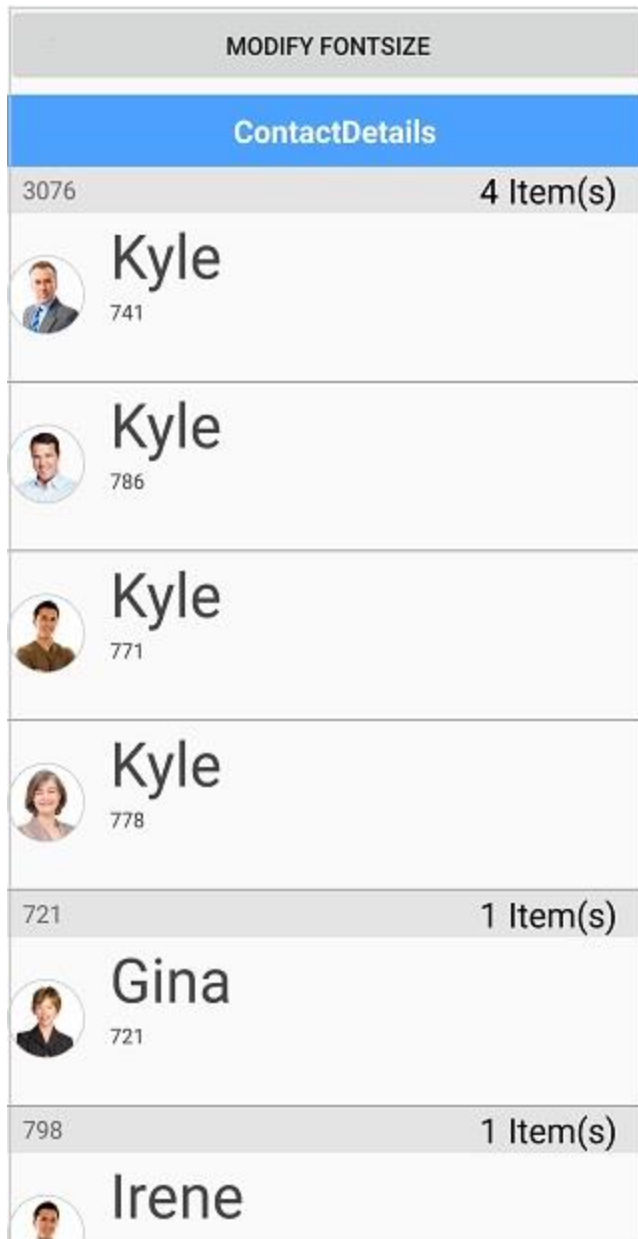
#### **XML**

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="50"/>
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Button Text="Modify FontSize" Clicked="Button_Clicked"/>
    <syncfusion:SfListView x:Name="listView"
      ItemsSource="{Binding Items}"
```

```
BackgroundColor="#FFE8E8EC"
AutoFitMode="DynamicHeight"
ItemSize="60">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Grid x:Name="grid" RowSpacing="1">
<Label LineBreakMode="NoWrap"
TextColor="#474747"
FontSize="{Binding BindingContext.FontSize, Source={x:Reference
Name=listView}}"
Text="{Binding ContactName}">
</Label>
</Grid>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</ContentPage>
```

### C#

```
private void Button_Clicked(object sender, EventArgs e)
{
    ViewModel.FontSize += 25;
}
```



Updating the Header and Footer height based on font at runtime

ListView allows you to resize the header and footer item size based on the change in font size of the label element at runtime by calling [RefreshListViewItem](#) method asynchronously when [SfListView.AutoFitMode](#) is Height.

#### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="50"/>
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
```

```

<Button Text="Change FontSize" Command="{Binding ResizeHeaderFooterCommand}"
CommandParameter="{x:Reference listView}"/>
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding Contacts}"
BackgroundColor="#FFE8E8EC"
AutoFitMode="Height">
<syncfusion:SfListView.HeaderTemplate>
<DataTemplate>
<ViewCell>
<Grid>
<Label Text="Contact Details"
FontSize="{Binding BindingContext.FontSize, Source={x:Reference
listView}}"/>
</Grid>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.HeaderTemplate>
<syncfusion:SfListView.FooterTemplate>
<DataTemplate>
<ViewCell>
<Grid >
<Label Text="Contacts Count" FontSize="{Binding BindingContext.FontSize,
Source={x:Reference listView}}"/>
<Label Text="{Binding Contacts.Count}" FontSize="{Binding
BindingContext.FontSize, Source={x:Reference listView}}"/>
</Grid>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.FooterTemplate>
</syncfusion:SfListView>
</ContentPage>

```

## C#

```

namespace SfListViewSample
{
    public class ContactsViewModel : INotifyPropertyChanged
    {
        public Command ResizeHeaderFooterCommand { get; set; }
        private double MaxPhone = 70;
        private double MinPhone = 20;
        private double MaxTablet = 100;
        private double MinTablet = 30;
        public ContactsViewModel()
        {
            ResizeHeaderFooterCommand = new Command(ResizeHeaderFooter);
        }
        private void ResizeHeaderFooter(object obj)
        {
            list = obj as SfListView;
            var maxFont = Device.Idiom == TargetIdiom.Phone ? MaxPhone : MaxTablet;
            var minFont = (Device.Idiom == TargetIdiom.Phone) ? MinPhone : MinTablet;
            if (FontSize >= maxFont)
            {
                FontSize = minFont;
            }
        }
    }
}

```



```
else
{
    FontSize += 10;
}
list.RefreshListViewItem(-1, -1, true);
}
}
```

You can download entire source code from [GitHub](#).



#### Load images with autofit mode

By default, the image is not loaded with the actual size in autofit mode, as it measures the size before the layout. So, the size of the child view changes cannot be found from the parent view. It is a known issue in listview, but this can be overcome by calling the `RefreshListViewItem` method in the loaded event of listview.

#### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <ContentPage.Content>
    <Grid>
      <syncfusion:SfListView x:Name="listView"
        AutoFitMode="Height"
        ItemsSource="{Binding ContactsInfo}"
        Loaded="ListView_Loaded">
        <syncfusion:SfListView.ItemTemplate>
```

```

<DataTemplate>
<StackLayout>
<StackLayout>
<Label Text="{Binding ContactName}" />
<Label Text="{Binding ContactNumber}" />
</StackLayout>
<Image Source="{Binding ContactImage}" />
</StackLayout>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</Grid>
</ContentPage.Content>
</ContentPage>

```

## C#

```

private void ListView_Loaded(object sender,
Syncfusion.ListView.XForms.ListViewLoadedEventArgs e)
{
Device.BeginInvokeOnMainThread(async () =>
{
await Task.Delay(100);
listView.RefreshListViewItem();
});
}

```

## Limitations

- Defines the size of the image when loading image in the [SfListView.ItemTemplate](#). Because, it does not return actual measured size when measuring before layout the item.
- Avoids SfListView inside the SfListView if SfListView.AutoFitMode is Height or DynamicHeight. Because, the inner SfListView does not return actual measured size when measuring before layout the item.

## Layouts

The SfListView supports different layouts such as linear layout and grid layout. The [SfListView.LayoutManager](#) property is used to define the layout.

### Linear Layout

Linear layout arrange items in a single column. Initialize the [LinearLayout](#), and assign it to the [SfListView.LayoutManager](#) property to load the SfListView in linear layout. It is the default layout.

## XML

```

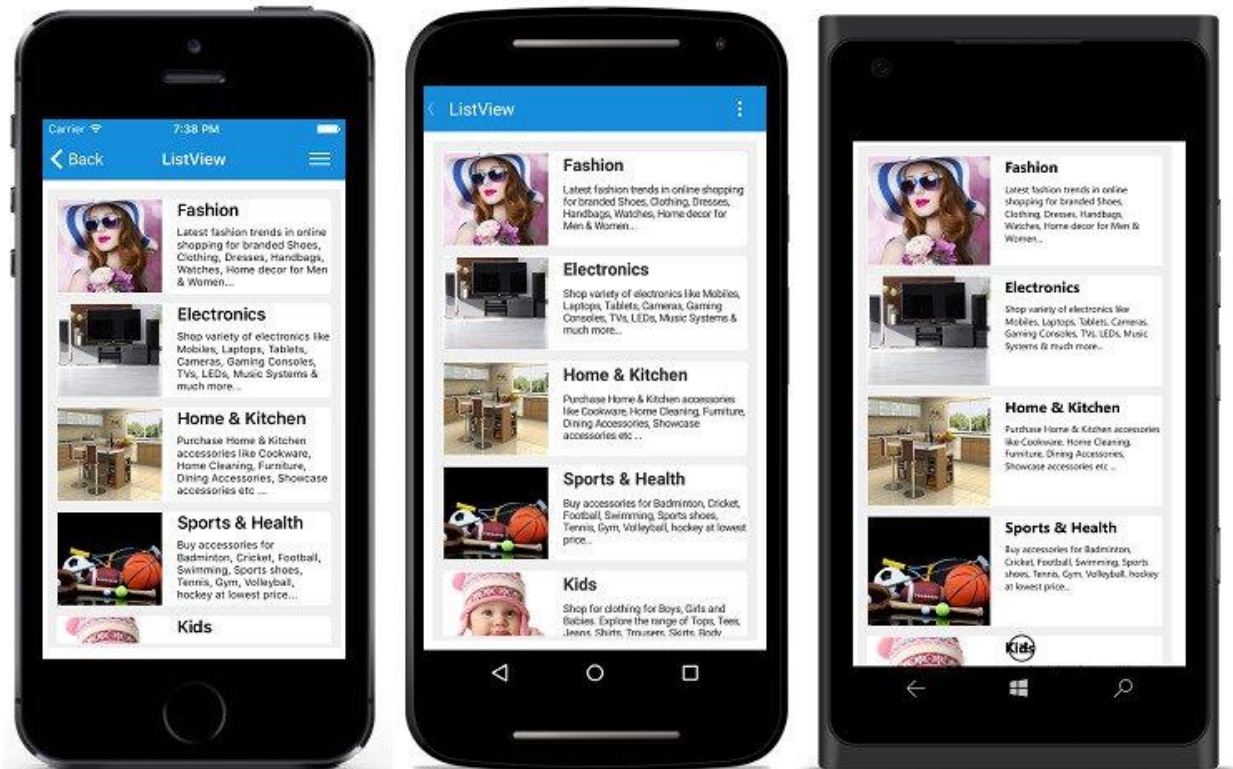
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding GalleryInfo}"
ItemSize="100">
<syncfusion:SfListView.LayoutManager>
<syncfusion:LinearLayout />
</syncfusion:SfListView.LayoutManager>

```

```
</syncfusion:SfListView>
</ContentPage>
```

## C#

```
listView.LayoutManager = new LinearLayout();
```



## Grid Layout

Grid layout arrange items in a predefined number of columns. Initialize the [GridLayout](#), and assign it to the [SfListView.LayoutManager](#) property to load the SfListView in grid layout.

The number of columns can be defined by using the [SpanCount](#) property of [GridLayout](#). Default [SpanCount](#) is 2.

In horizontal orientation, [SpanCount](#) defines the number of rows.

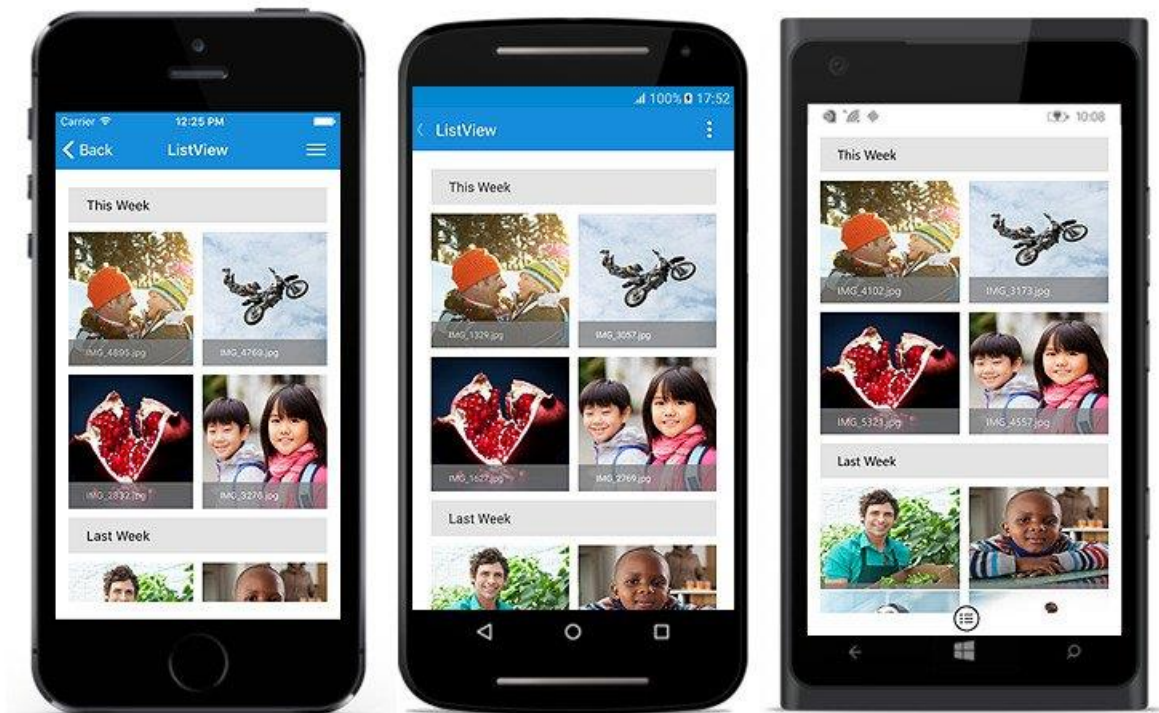
## XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding GalleryInfo}"
ItemSize="100">
<syncfusion:SfListView.LayoutManager>
<syncfusion:GridLayout SpanCount="2" />
</syncfusion:SfListView.LayoutManager>
</syncfusion:SfListView>
</ContentPage>
```

**C#**

```
listView.LayoutManager = new GridLayout() { SpanCount = 2 };
```

Download the entire source code from GitHub [here](#).



### Customize span count based on platform

The [SpanCount](#) property of the [GridLayout](#) can be customized based on the specified platform to avoid squeezed problem of listview item in phone and tablet devices or windows desktop.

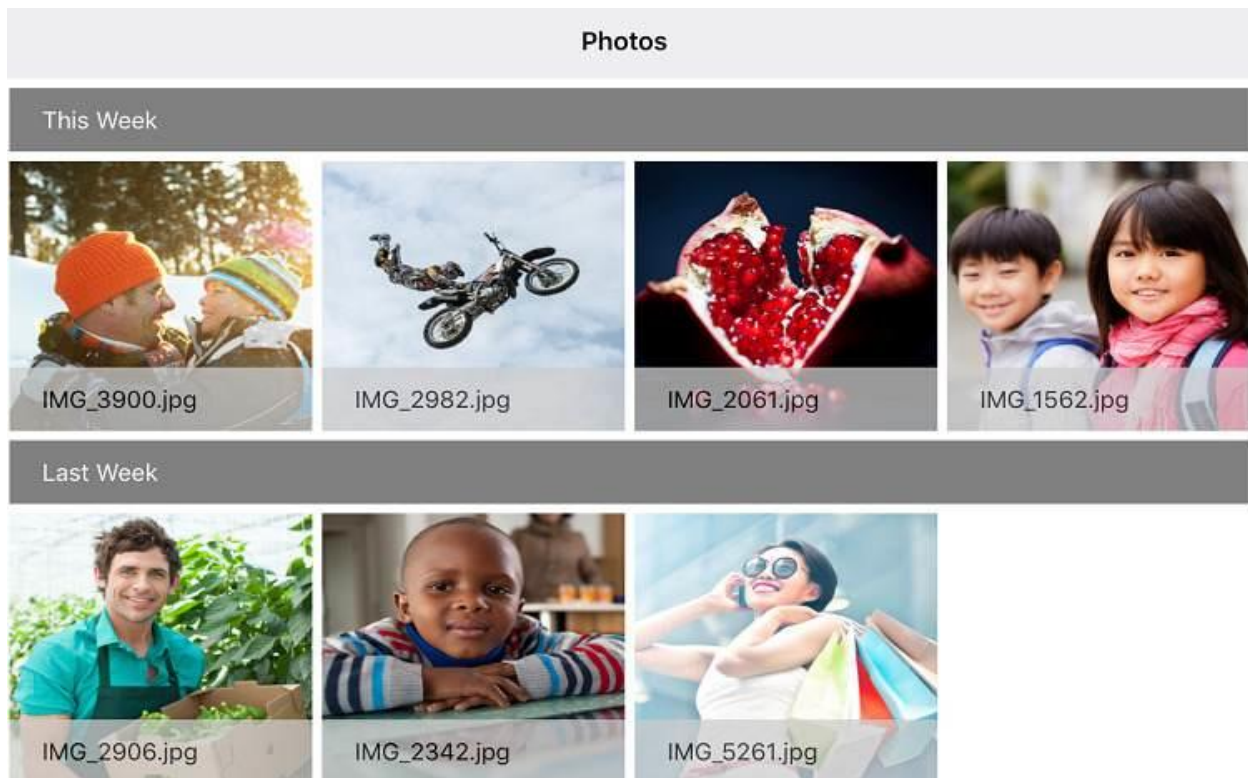
**XML**

```
<ContentPage xmlns:syncfusion="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView.LayoutManager>
    <syncfusion:GridLayout>
      <syncfusion:GridLayout.SpanCount>
        <OnPlatform x:TypeArguments="x:Int32">
          <OnPlatform.WinPhone>
            <OnIdiom x:TypeArguments="x:Int32" Phone="2" Tablet="4" Desktop="4"/>
          </OnPlatform.WinPhone>
          <OnPlatform.Android>
            <OnIdiom x:TypeArguments="x:Int32" Phone="2" Tablet="4" />
          </OnPlatform.Android>
          <OnPlatform.iOS>
            <OnIdiom x:TypeArguments="x:Int32" Phone="2" Tablet="4" />
          </OnPlatform.iOS>
        </OnPlatform>
      </syncfusion:GridLayout.SpanCount>
    </syncfusion:GridLayout>
  </syncfusion:SfListView.LayoutManager>
```

```
</syncfusion:SfListView>
</ContentPage>
```

**C#**

```
GridLayout gridLayout = new GridLayout();
if (Device.OS == TargetPlatform.Android || Device.OS == TargetPlatform.iOS)
gridLayout.SpanCount = Device.Idiom == TargetIdiom.Phone ? 2 : 4;
else if (Device.OS == TargetPlatform.Windows)
gridLayout.SpanCount = Device.Idiom == TargetIdiom.Desktop || Device.Idiom
== TargetIdiom.Tablet ? 4 : 2;
listView.LayoutManager = gridLayout;
```



Change span count based on screen size

In the SfListView, the [GridLayout](#) allows changing the span count based on the view size of application with orientation in either portrait or landscape mode.

**C#**

```
public partial class GridLayoutPage : ContentPage
{
    protected override void OnSizeAllocated(double width, double height)
    {
        base.OnSizeAllocated(width, height);
        if (width > 0 && pageWidth != width)
        {
            var size = Application.Current.MainPage.Width / listView.ItemSize;
            gridLayout.SpanCount = (int)size;
            listView.LayoutManager = gridLayout;
        }
    }
}
```

```
}
}
}
```

You can download the entire sample code [here](#).

## Item Drag and Drop

The SfListView allows reordering by dragging and dropping items. It supports displaying the customized view in a template while dragging the item. It can be enabled by setting the [SfListView.DragStartMode](#) property to **OnHold**. The drag and drop options are listed as follows:

- None: Disables drag and drop. This is the default value.
- OnHold: Allows dragging and dropping by holding the item.
- OnDragIndicator: Allows dragging and dropping by loading the [DragIndicatorView](#) within [SfListView.ItemTemplate](#).

---

**Note:** The [GridLayout](#) does not support drag and drop.

---

The drag and drop scenarios are as follows:

- Items can be reordered to any position by auto-scrolling.
- Items can be reordered in same group or in other groups but, no groups can be added to other groups.
- Groups, header, and footer cannot be reordered.

To enable drag and drop using 'OnHold', follow the code example:

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding ToDoList}"
DragStartMode="OnHold"
BackgroundColor="#FFE8E8EC"
ItemSize="60" />
</ContentPage>
```

### C#

```
listView.DragStartMode = DragStartMode.OnHold;
```

To enable drag and drop using both 'OnHold' and 'OnDragIndicator', follow the code example:

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding ToDoList}"
DragStartMode="OnHold, OnDragIndicator"
BackgroundColor="#FFE8E8EC"
```



```
ItemSize="60" />
</ContentPage>
```

## C#

```
listView.DragStartMode = DragStartMode.OnHold |
DragStartMode.OnDragIndicator;
```

**Note:** Reordering changes made only in view, and not in underlying collection. Thus, the changes will be reverted when performing sorting, grouping, or any other operation that refreshes view. You can update underlying collection by setting UpdateSource to true.

### Drag indicator view

To drag and drop the items by [DragIndicatorView](#), set the [SfListView.DragStartMode](#) property to OnDragIndicator. To display the dragging item, define any custom user interface(UI) in DragIndicatorView.

**Note:** You must set the SfListView instance as reference to the [ListView](#) property in DragIndicatorView.

## XML

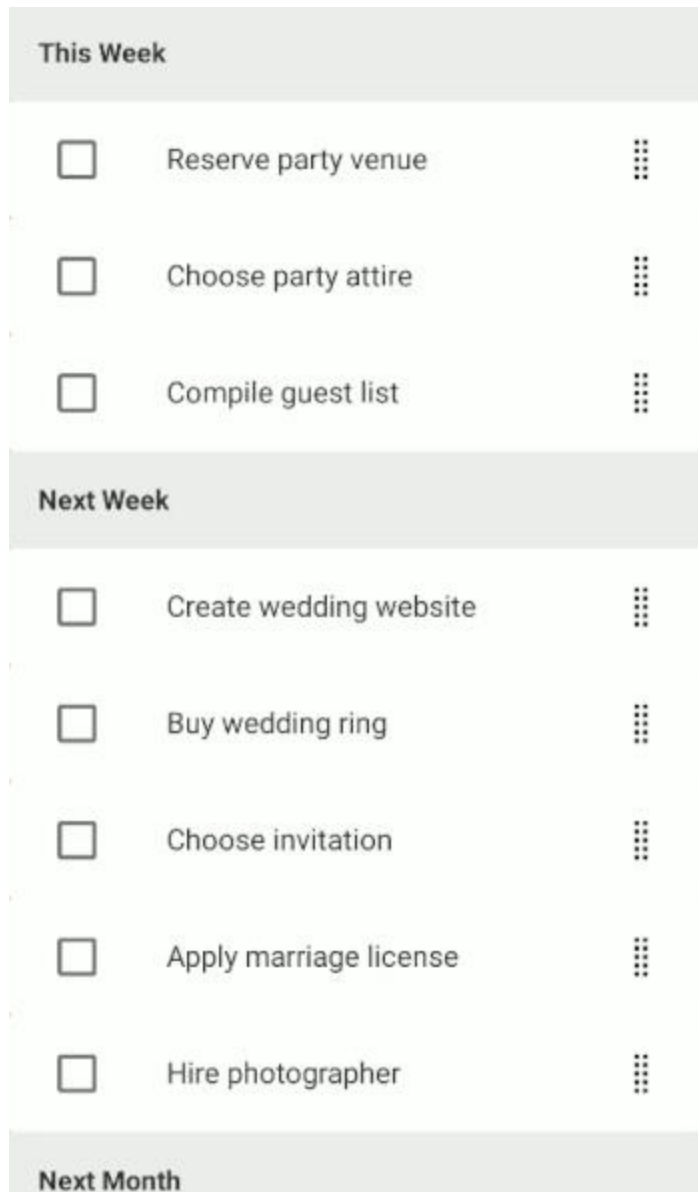
```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding ToDoList}"
DragStartMode="OnDragIndicator"
BackgroundColor="#FFE8E8EC"
ItemSize="60">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Grid Padding="10">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="60" />
</Grid.ColumnDefinitions>
<Label x:Name="textLabel" Text="{Binding Name}" Grid.Column="1"
FontSize="15" TextColor="#333333" />
<syncfusion:DragIndicatorView Grid.Column="2" ListView="{x:Reference
listView}"
HorizontalOptions="Center"
VerticalOptions="Center">
<Grid Padding="10, 20, 20, 20">
<Image Source="DragIndicator.png" />
</Grid>
</syncfusion:DragIndicatorView>
</Grid>
</Grid>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</ContentPage>
```

## C#

```
listView.ItemTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var name = new Label
    {
        FontSize = 15,
        VerticalOptions = LayoutOptions.Center
    };
    name.SetBinding(Label.TextProperty, new Binding("Name"));
    var indicatorGrid = new Grid()
    {
        Padding = new Thickness(10, 20, 20, 20),
        HorizontalOptions = LayoutOptions.End,
        VerticalOptions = LayoutOptions.Center
    };
    var dragIndicatorView = new DragIndicatorView() { ListView = this.listView };
    var indicator = new Image() { Source = "DragIndicator.png" };
    indicatorGrid.Children.Add(indicator);
    dragIndicatorView.Content = indicatorGrid;
    grid.Children.Add(name);
    grid.Children.Add(dragIndicatorView, 1, 0);
    return grid;
});
```

The screenshot shows the output of the reordering items by drag and drop. Download the entire source code from GitHub [here](#).





### Drag item customization

By defining the [SfListView.DragItemTemplate](#) of the SfListView, displays the custom User Interface(UI) when performing drag and drop operations. The template can be defined either in code or XAML.

**Note:** If BackgroundColor is set to DragItemTemplate or [DragIndicatorView](#), set InputTransparent to true. Since, dragging does not happen when performing by DragIndicatorView in UWP.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView"
    ItemsSource="{Binding ToDoList}"
    DragStartMode="OnHold"
    BackgroundColor="#FFE8E8EC"
    ItemSize="60">
    <syncfusion:SfListView.DragItemTemplate>
```

```
<DataTemplate>
<Grid Padding="10">
<Label x:Name="textLabel" Text="{Binding Name}" FontSize="15" />
</Grid>
</DataTemplate>
</syncfusion:SfListView.DragItemTemplate>
</syncfusion:SfListView>
</ContentPage>
```

## C#

```
listView.ItemTemplate = new DataTemplate(() => {
var grid = new Grid();
var name = new Label { FontSize = 15 };
name.SetBinding(Label.TextProperty, new Binding("Name"));
grid.Children.Add(name);
return grid;
});
```

## Event

The [ItemDragging](#) event is raised while dragging and dropping the item in the [SfListView](#). The [ItemDraggingEventArgs](#) has the following members which provides the information for the [ItemDragging](#) event:

- [Action](#): Returns the drag [Action](#) such as start, dragging, and drop.
- [Bounds](#): Return bounds of drag item when dragging and dropping.
- [Handled](#): If this member is set to true, dragging can be handled. It is applicable only if [Action](#) is [Dragging](#).
- [ItemData](#): Returns the underlying data of the dragging item.
- [NewIndex](#): Returns the item index of the [DataSource.DisplayItems](#) where dragging item is going to be dropped.
- [OldIndex](#): Returns the item index of the [DataSource.DisplayItems](#) where dragging item started. The OldIndex and NewIndex will be same if [Action](#) is [Start](#).
- [Position](#): Returns the touch position of the drag item from screen coordinates.

## Auto scroll options

### Auto scroll margin

To adjust auto scroll margin, set a value to the [ScrollMargin](#) property of [AutoScroller](#) to enable auto-scrolling while dragging. The default value is 15. Auto-scrolling will be enabled when reaching [ScrollMargin](#) from view bounds while dragging.

To disable auto-scrolling, set the value to 0 for [ScrollMargin](#).

## C#

```
this.listView.AutoScroller.ScrollMargin = 20;
```

### Auto scroll interval

To adjust auto-scroll interval while dragging, set the [Interval](#) property of [AutoScroller](#). The default value is 150 milliseconds.

**C#**

```
this.listView.AutoScroller.Interval = new TimeSpan(0, 0, 0, 0, 200);
```

*Disable outside scroll*

To disable auto-scroll when dragging item moves outside the SfListView while dragging, set the [AllowOutsideScroll](#) property of [AutoScroller](#) to true. The default value is true.

**C#**

```
this.listView.AutoScroller.AllowOutsideScroll = false;
```

*Disable dragging for particular item*

To disable dragging for a particular item, handle the [ItemDragging](#) event based on the conditions of [Action](#) event argument.

**C#**

```
private void ListView_ItemDragging(object sender, ItemDraggingEventArgs e)
{
    // Disable the dragging for 4th item.
    if (e.Action == DragAction.Start && e.NewIndex == 3)
        e.Cancel = true;
}
```

*Cancel dropping for the dragged item*

To cancel dropping for the dragged item, handle the [ItemDragging](#) event based on the conditions of [Action](#) event argument.

**C#**

```
using Syncfusion.ListView.XForms.Control.Helpers;
private void ListView_ItemDragging(object sender, ItemDraggingEventArgs e)
{
    // Cancel the dropping if drop the drag item into out of view.
    var listView = sender as ListView;
    var totalExtent = listView.GetVisualContainer().Bounds.Bottom;
    if (e.Action == DragAction.Drop && (e.Bounds.Y < -30 || e.Bounds.Bottom >
        totalExtent + 40))
        e.Cancel = true;
}
```

*Reorder the underlying collection*

The underlying collection can be reordered directly by setting the [UpdateSource](#) property to true. The default value is false.

**C#**

```
this.listView.DragDropController.UpdateSource = true;
```

We can able to update collection even when [UpdateSource](#) is false. Like, user can decide where dragged item should be dropped actually by handling the [ItemDragging](#) event with [DragAction.Drop](#).

**C#**

```
private void ListView_ItemDragging(object sender, ItemDraggingEventArgs e)
{
    if (e.Action == DragAction.Drop)
    {
        ViewModel.ToDoList.MoveTo(1, 5);
    }
}
```

**Note:** Underlying collection will not be updated when any data operation like sorting or grouping is performed. The order will be maintained only in DisplayItems of data source. When drag and drop an item between groups, the value of the property in which grouping is performed is updated in data object.

Delete item when dropping in particular view

To delete the dragged item when dropping into a particular view, handle the [ItemDragging](#) event based on the conditions of [Action](#) and [Bounds](#) event arguments.

To delete the dragged item from the underlying collection when dropping into delete icon, follow the code example. It will enable or disable whenever drag started, and dropped by `IsVisible` property in `ViewModel`.

**XML**

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid BackgroundColor="#2196F3">
<Label Text="To Do Items" x:Name="headerLabel" TextColor="White"
FontAttributes="Bold" VerticalOptions="Center" HorizontalOptions="Center" />
<StackLayout x:Name="stackLayout" IsVisible="{Binding IsVisible}"
Orientation="Horizontal" VerticalOptions="Center"
HorizontalOptions="Center">
<Image Source="Delete.png" HeightRequest="30" WidthRequest="30" />
<Label x:Name="deleteLabel" Text="Delete Item" FontAttributes="Bold"
TextColor="White" />
</StackLayout>
</Grid>
<syncfusion:SfListView x:Name="listView" Grid.Row="1"
ItemsSource="{Binding ToDoList}"
DragStartMode="OnHold"
BackgroundColor="#FFE8E8EC"
ItemSize="60" />
</Grid>
</ContentPage>
```

**C#**

```
public partial class MainPage : ContentPage
{
```

```
public MainPage()
{
    InitializeComponent();
    var grid = new Grid();
    grid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(40) });
    grid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
        GridUnitType.Star) });
    var grid1 = new Grid();
    var headerLabel = new Label()
    {
        Text = "To Do Items"
    };
    var stackLayout = new StackLayout();
    stackLayout.SetBinding(StackLayout.IsVisibleProperty, new
        Binding("IsVisible"));
    var image = new Image() { Source = "Delete.png" };
    var deleteLabel = new Label() { Text = "DeleteItem" };
    stackLayout.Children.Add(image);
    stackLayout.Children.Add(deleteLabel);
    grid1.Children.Add(headerLabel);
    grid1.Children.Add(stackLayout);
    var listView = new SfListView()
    {
        DragStartMode = DragStartMode.OnHold,
        ItemSize = 60,
        SelectionMode = SelectionMode.None,
        BackgroundColor = Color.FromHex("#FFE8E8EC")
    };
    listView.SetBinding(ListView.ItemsSourceProperty, new Binding("ToDoList"));
    grid.Children.Add(grid1);
    grid.Children.Add(listView, 0, 1);
}
```

## C#

```
private async void ListView_ItemDragging(object sender,
    ItemDraggingEventArgs e)
{
    var viewModel = this.listView.BindingContext as ViewModel;
    if (e.Action == DragAction.Start)
    {
        viewModel.IsVisible = true;
        this.stackLayout.Opacity = 0.25;
    }
    if (e.Action == DragAction.Dragging)
    {
        var position = new Point(e.Position.X - this.listView.Bounds.X, e.Position.Y
            - this.listView.Bounds.Y);
        if (this.stackLayout.Bounds.Contains(position))
            this.deleteLabel.TextColor = Color.Red;
        else
            this.deleteLabel.TextColor = Color.White;
    }
    if (e.Action == DragAction.Drop)
    {

```

```
var position = new Point(e.Position.X - this.listView.Bounds.X, e.Position.Y - this.listView.Bounds.Y);  
if (this.stackLayout.Bounds.Contains(position))  
{  
    await Task.Delay(100);  
    viewModel.ToDoList.Remove(e.ItemData as ToDoItem);  
}  
viewModel.IsVisible = false;  
this.deleteLabel.TextColor = Color.White;  
this.headerLabel.IsVisible = true;  
}  
}
```

Download the sample from GitHub [here](#).

To Do Items		
This Week		
<input type="checkbox"/>	Reserve party venue	⋮
<input type="checkbox"/>	Choose party attire	⋮
<input type="checkbox"/>	Compile guest list	⋮
<input type="checkbox"/>	Choose invitation	⋮
Next Week		
<input type="checkbox"/>	Buy wedding ring	⋮
<input type="checkbox"/>	Apply marriage license	⋮
<input type="checkbox"/>	Hire photographer	⋮
Next Month		

### Skip dragging item into another group

To skip dragging from one group to another group, handle the [ItemDragging](#) event based on the conditions of [Action](#) and [Bounds](#) event arguments.

---

**Note:** While auto-scrolling, dragging item cannot be skipped.

---

Skip the dragging item by bounds of dragging item, and bounds of current and next group item.

### C#

```
using Syncfusion.ListView.XForms.Control.Helpers;
private async void ListView_ItemDragging(object sender,
ItemDraggingEventArgs e)
{
    if (e.Action == DragAction.Dragging)
    {
        var currentGroup = this.GetGroup(e.ItemData);
        var container = this.ListView.GetVisualContainer();
        var groupIndex = this.ListView.DataSource.Groups.IndexOf(currentGroup);
        var nextGroup = (groupIndex + 1 < this.ListView.DataSource.Groups.Count) ?
this.ListView.DataSource.Groups[groupIndex + 1] : null;
        ListViewItem groupItem = null;
        ListViewItem nextGroupItem = null;
        foreach (ListViewItem item in container.Children)
        {
            if (item.BindingContext == null || !item.Visibility)
                continue;
            if (item.BindingContext.Equals(currentGroup))
                groupItem = item;
            if (nextGroup != null && item.BindingContext.Equals(nextGroup))
                nextGroupItem = item;
        }
        if (groupItem != null && e.Bounds.Y <= groupItem.Y + groupItem.Height ||
nextGroupItem != null && (e.Bounds.Y + e.Bounds.Height >= nextGroupItem.Y))
            e.Handled = true;
    }
}

private GroupResult GetGroup(object itemData)
{
    GroupResult itemGroup = null;
    foreach (var item in this.listView.DataSource.DisplayItems)
    {
        if (item is GroupResult)
            itemGroup = item as GroupResult;
        if (item == itemData)
            break;
    }
    return itemGroup;
}
```

Download sample from GitHub [here](#)

## Drag and drop customization

### Adjust drag item axis

To adjust drag item coordinates (X and Y) while dragging, returns true from virtual method [CanAdjustDragItemAxis](#) of [DragDropController](#). By default, Y coordinates can be adjusted if [SfListView.Orientation](#) is [Vertical](#), and X coordinates can be adjusted if [Orientation](#) is [Horizontal](#).

#### C#

```
this.listView.DragDropController = new DragDropControllerExt(this.listView);
public class DragDropControllerExt : DragDropController
{
    public DragDropControllerExt(SfListView listView) : base(listView)
    {
    }
    protected override bool CanAdjustDragItemAxis()
    {
        return true;
    }
}
```

### Layout item on dragging

In the SfListView, layout the [ListViewItem](#) with different animations, and time on dragging by virtual method [OnLayoutItem](#).

#### C#

```
this.listView.DragDropController = new DragDropControllerExt(this.listView);
public class DragDropControllerExt : DragDropController
{
    public DragDropControllerExt(SfListView listView) : base(listView)
    {
    }
    protected override Task<bool> OnLayoutItem(View element, Rectangle rect)
    {
        return element.LayoutTo(rect, 250, Easing.BounceIn);
    }
}
```

## Sorting

The SfListView supports sorting the data either in ascending or descending order by using [DataSource.SortDescriptors](#) property and by using the custom logic.

---

**Note:** When ItemsSource changed for ListView, [DataSource.SortDescriptors](#) will be cleared by default. You need to add [DataSource.SortDescriptors](#) again after changing ItemsSource if you want to retain sorting in listview.

---

### Programmatic sorting

Sorting the data by creating the [SortDescriptor](#) with required property name and direction and adding it into the [DataSource.SortDescriptors](#) property.

[SortDescriptor](#) object holds the following three properties:

- [PropertyName](#): Describes the name of the sorted property.



- [Direction](#): Describes an object of type [ListSortDirection](#) that defines the sorting direction.
- [Comparer](#): Describes the comparer to be applied when sorting take place.

### XML

```
<ContentPage xmlns:syncfusion="clr-  
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"  
xmlns:data="clr-  
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">  
  <syncfusion:SfListView x:Name="listView">  
    <syncfusion:SfListView.DataSource>  
      <data:DataSource>  
        <data:DataSource.SortDescriptors>  
          <data:SortDescriptor PropertyName="ContactName" Direction="Ascending"/>  
        </data:DataSource.SortDescriptors>  
      </data:DataSource>  
    </syncfusion:SfListView.DataSource>  
  </syncfusion:SfListView>  
</ContentPage>
```

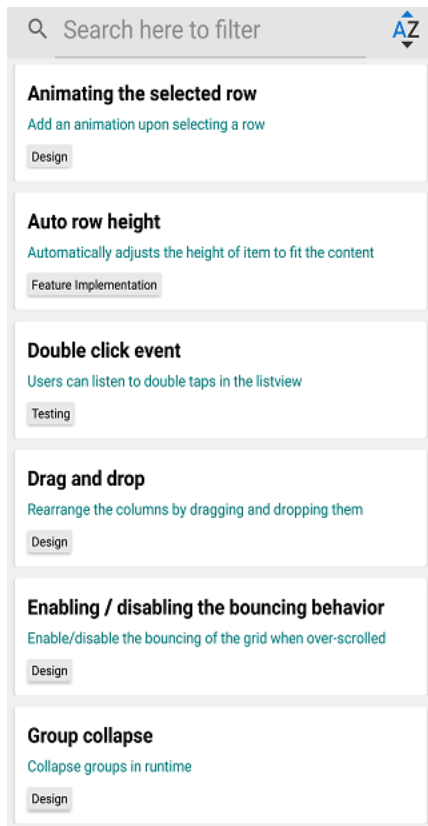
### C#

```
listView.DataSource.SortDescriptors.Add(new SortDescriptor()  
{  
    PropertyName = "ContactName",  
    Direction = ListSortDirection.Ascending,  
});  
listView.RefreshView();
```

---

**Note:** It is mandatory to specify the `PropertyName` of `SortDescriptor`.

---



### Custom sorting

Sort the items based on the custom logic and it can be applied to either [SfListView.DataSource.SortComparer](#) property or [SortDescriptor.Comparer](#) which is added into the [DataSource.SortDescriptors](#) collection.

You can download the entire sample code from the [github](#).

**Note:** If the `PropertyName` in the [SortDescriptor](#) and `GroupDescriptor` are same then, [GroupResult](#) will be passed as parameters for the `SortDescriptor.Comparer`. Otherwise data objects are passed. To sort the data items alone, set the different `PropertyName` in both `SortDescriptor` and `GroupDescriptor` properties.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
  <ContentPage.Resources>
    <ResourceDictionary>
      <local:CustomSortComparer x:Key="CustomSortComparer" />
    </ResourceDictionary>
  </ContentPage.Resources>
  <syncfusion:SfListView x:Name="listView"
  <syncfusion:SfListView.DataSource>
    <data:DataSource>
      <data:DataSource.SortDescriptors>
        <data:SortDescriptor Comparer="{StaticResource CustomSortComparer}"/>
      </data:SortDescriptors>
    </data:DataSource>
  </syncfusion:SfListView>
</ContentPage>
```

```

</data:DataSource.SortDescriptors>
</data:DataSource>
</syncfusion:SfListView.DataSource>
</syncfusion:SfListView>
</ContentPage>

```

**C#**

```

listView.DataSource.SortDescriptors.Add(new SortDescriptor()
{
    Comparer = new CustomSortComparer()
});

```

**C#**

```

public class CustomSortComparer : IComparer<object>
{
    public int Compare(object x, object y)
    {
        if (x.GetType() == typeof(ListViewContactsInfo))
        {
            var xitem = (x as ListViewContactsInfo).ContactName;
            var yitem = (y as ListViewContactsInfo).ContactName;
            if (xitem.Length > yitem.Length)
            {
                return 1;
            }
            else if (xitem.Length < yitem.Length)
            {
                return -1;
            }
            else
            {
                if (string.Compare(xitem, yitem) == -1)
                {
                    return -1;
                }
                else if (string.Compare(xitem, yitem) == 1)
                {
                    return 1;
                }
            }
            return 0;
        }
    }
}

```

For more information about custom sorting of groups, please refer the documentation [here](#).

Sort the items on header tapped

To apply the sorting when tapping the header, handle the [ItemTapped](#) event of the SfListView.

**XML**

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
<syncfusion:SfListView x:Name="listView" ItemSize="60"

```

```

ItemsSource="{Binding customerDetails}"
ItemTapped="ListView_ItemTapped"
IsStickyHeader="True">
<syncfusion:SfListView.HeaderTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<StackLayout BackgroundColor="Teal">
<Label TextColor="White" FontSize="20" FontAttributes="Bold"
Text="CustomerDetails" />
</StackLayout>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.HeaderTemplate>
</syncfusion:SfListView>
</ContentPage>

```

**C#**

```

listView = new SfListView();
listView.ItemsSource = viewModel.customerDetails;
listView.ItemSize = 60;
listView.ItemTapped += ListView_ItemTapped;
listView.IsStickyHeader = true;
listView.HeaderTemplate = new DataTemplate(() =>
{
    var stackLayout = new StackLayout { BackgroundColor = Color.Teal };
    var label = new Label { Text = "CustomerDetails", TextColor = Color.White,
        FontAttributes = FontAttributes.Bold, FontSize = 20 };
    stackLayout.Children.Add(label);
    return stackLayout;
});

```

When the `ItemTapped` event is raised for the Header, add the [SortDescriptor](#) and refresh the view.

**C#**

```

private void ListView_ItemTapped(object sender,
Syncfusion.ListView.XForms.ItemTappedEventArgs e)
{
    //Applying sorting to the underlying data when the header item is tapped.
    if (e.ItemType == ItemType.Header && listView.IsStickyHeader)
    {
        listView.DataSource.SortDescriptors.Clear();
        listView.DataSource.SortDescriptors.Add(new SortDescriptor()
        {
            PropertyName = "ContactName",
            Direction = ListSortDirection.Ascending
        });
    }
}

```

Sort the items along with grouping

The SfListView allows sorting the items along with grouping by adding the [DataSource.GroupDescriptors](#) and the [DataSource.SortDescriptors](#) with required property name.

Sorting with grouping by year

Sorting the items along with grouping by using [KeySelector](#) based on returning the year value of the data-time property.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
  <ContentPage.Content>
    <syncfusion:SfListView x:Name="listView" ItemsSource="{Binding Items}"
    ItemSize="50">
      <syncfusion:SfListView.GroupHeaderTemplate>
        <DataTemplate>
          <Grid>
            <Label Text= "{Binding Key}" BackgroundColor="Teal" FontAttributes="Bold"
            TextColor="White"/>
          </Grid>
        </DataTemplate>
      </syncfusion:SfListView.GroupHeaderTemplate>
    </syncfusion:SfListView>
  </ContentPage.Content>
</ContentPage>
```

### C#

```
var listView = new SfListView();
listView.ItemSize = 50;
listView.ItemsSource = viewModel.Items;
listView.GroupHeaderTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var headerLabel = new Label
    {
        TextColor = Color.White,
        FontAttributes = FontAttributes.Bold,
        BackgroundColor=Color.Teal
    };
    headerLabel.SetBinding(Label.TextProperty, new Binding("key"));
    grid.Children.Add(headerLabel);
    return grid;
});
listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()
{
    PropertyName = "DateOfBirth",
    KeySelector = (object obj1) =>
    {
        var item = (obj1 as Contacts);
        return item.DateOfBirth.Year;
    },
});
```

```
this.listView.DataSource.SortDescriptors.Add(new SortDescriptor()
{
    PropertyName = "DateOfBirth",
    Direction = ListSortDirection.Ascending
});
```

The following screenshot shows the output when items are sorted by year. Download the entire source code from GitHub [here](#)

2012	
Bill	25/09/2012
Holly	22/10/2012
Gina	03/12/2012
2013	
Katie	27/01/2013
Fiona	28/01/2013
Danielle	23/03/2013
Kyle	02/04/2013
Frank	26/04/2013
Michael	01/11/2013

### Sorting with grouping by month and year

Sorting the items along with grouping by using `KeySelector` based on retuning the month and year value of the data-time property.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
  <ContentPage.Content>
    <syncfusion:SfListView x:Name="listView">
      <syncfusion:SfListView.DataSource>
        <data:DataSource>
          <data:DataSource.GroupDescriptors>
            <data:GroupDescriptor PropertyName="ContactName" />
          </data:DataSource.GroupDescriptors>
          <data:DataSource.SortDescriptors>
            <data:SortDescriptor PropertyName="ContactName" Direction="Ascending"/>
          </data:DataSource.SortDescriptors>
        </data:DataSource>
      </syncfusion:SfListView.DataSource>
    </syncfusion:SfListView>
  </ContentPage.Content>
</ContentPage>
```

```
</data:DataSource.SortDescriptors>
</data:DataSource>
</syncfusion:SfListView.DataSource>
</syncfusion:SfListView>
</ContentPage.Content>
</ContentPage>
```

## C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()
        {
            PropertyName = "DateOfBirth",
            KeySelector = (object obj1) =>
            {
                var item = (obj1 as Contacts);
                return item.DateOfBirth.Month + "/" + item.DateOfBirth.Year;
            },
            Comparer = new CustomGroupComparer()
        });
        this.listView.DataSource.SortDescriptors.Add(new SortDescriptor()
        {
            PropertyName = "DateOfBirth",
            Direction = ListSortDirection.Ascending
        });
    }
}
```

The following screenshot shows the output when items are sorted by month and year.

2012/11
Bill
02/11/2012
2012/12
Fiona
27/12/2012
2013/2
Holly
24/02/2013
Kyle
27/02/2013
2013/10
Jennifer
27/10/2013
2013/12
William
23/12/2013
2014/1

## Filtering

This section explains how to filter the data and its related operations in the SfListView.

### Programmatic filtering

The [SfListView](#) supports to filter the data by setting the [SfListView.DataSource.Filter](#) property. You have to call the [SfListView.DataSource.RefreshFilter\(\)](#) method after assigning the [Filter](#) property for refreshing the view.

The [FilterChanged](#) event is raised once filtering is applied to the SfListView.

The [FilterContacts](#) method filters the data contains the filter text value. Assign [FilterContacts](#) method to [SfListView.DataSource.Filter](#) predicate to filter the [ContactName](#). To apply filtering in the SfListView, follow the code example:

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <SearchBar x:Name="filterText" HeightRequest="40"
      Placeholder="Search here to filter"
      TextChanged="OnFilterTextChanged" Grid.Row="0"/>
    <syncfusion:SfListView x:Name="listView" Grid.Row="1" ItemSize="60"
      ItemsSource="{Binding customerDetails}"/>
  </Grid>
</ContentPage>
```



```
</Grid>
</ContentPage>
```

### C#

```
var grid = new Grid();
var searchBar = new SearchBar() { Placeholder = "Search here to filter" };
searchBar.TextChanged += OnFilterTextChanged;
var listView = new SfListView();
listView.ItemsSource = viewModel.customerDetails;
listView.ItemSize = 60;
grid.Children.Add(searchBar);
grid.Children.Add(listView, 0, 1);
```

The following code example illustrates code for filtering the data using `FilterContacts` method in the ViewModel:

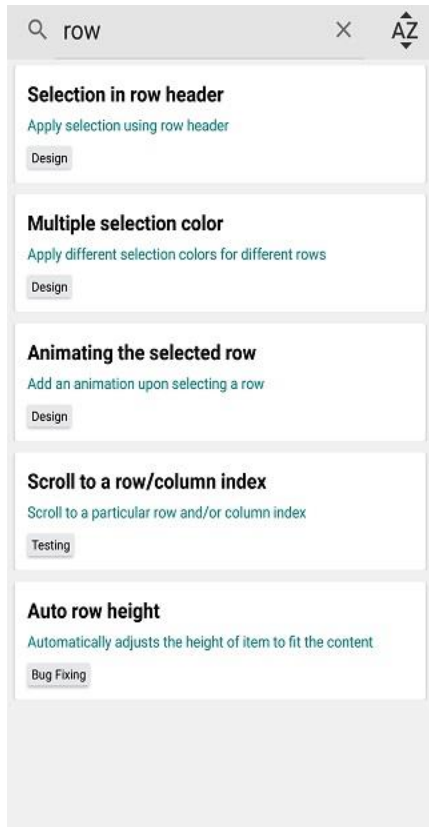
### C#

```
SearchBar searchBar = null;
private void OnFilterTextChanged(object sender, TextChangedEventArgs e)
{
    searchBar = (sender as SearchBar);
    if (listView.DataSource != null)
    {
        this.listView.DataSource.Filter = FilterContacts;
        this.listView.DataSource.RefreshFilter();
    }
}

private bool FilterContacts(object obj)
{
    if (searchBar == null || searchBar.Text == null)
        return true;
    var contacts = obj as Contacts;
    if (contacts.ContactName.ToLower().Contains(searchBar.Text.ToLower())
        || contacts.ContactName.ToLower().Contains(searchBar.Text.ToLower()))
        return true;
    else
        return false;
}
```

Download the entire source code from GitHub [here](#)

The following screenshot shows the output rendered when the items are filtered:



### *Filter based on multiple criteria*

The [SfListView](#) allows filtering the items based on multiple criteria. To filter the data using multiple properties, follow the code example:

#### **C#**

```
private bool FilterContacts(object obj)
{
    if (searchBar == null || searchBar.Text == null)
        return true;
    var contacts = obj as Contacts;
    if (contacts.ContactName.ToLower().Contains(searchBar.Text.ToLower())
        || contacts.ContactNumber.ToLower().Contains(searchBar.Text.ToLower()))
        return true;
    else
        return false;
}
```

### Getting the filtered data

You can get filtered items from the view and modify it in the [SfListView.DataSource.FilterChanged](#) event. When filter is applied, the filtered items are available in the [SfListView.DataSource.DisplayItems](#).

#### **C#**

```
listView.DataSource.FilterChanged += DataSource_FilterChanged;
...
private void DataSource_FilterChanged(object sender,
    NotifyCollectionChangedEventArgs e)
```

```
{  
    //Contacts is model class  
    ObservableCollection<Contacts> contacts = new  
    ObservableCollection<Contacts>();  
    // Get the filtered items  
    var items = (sender as DataSource).DisplayItems;  
    foreach (var item in items)  
        contacts.Add(item as Contacts);  
}
```

### Clear filtering

The SfListView allows clearing the filters by setting the [DataSource.Filter](#) to null, and call the [DataSource.RefreshFilter](#) method.

#### C#

```
listView.DataSource.Filter = null;  
listView.DataSource.RefreshFilter();
```

### Sort the filtered items

The order of the filtered items can be rearranged in the [FilterChanged](#) event by adding [SortDescriptor](#). To sort the filtered items, follow the code example:

#### C#

```
private void DataSource_FilterChanged(object sender,  
    NotifyCollectionChangedEventArgs e)  
{  
    listView.DataSource.SortDescriptors.Add(new SortDescriptor { PropertyName =  
        "ContactName",  
        Direction = ListSortDirection.Ascending });  
    listView.RefreshView();  
}
```

## Grouping

A group represents a collection of items belongs to a category. When grouping is applied, the data is organized into different groups based on key values. Each group is identified by its [Key](#), by which you can get the underlying data in the group.

---

**Note:** When ItemsSource changed for ListView, [DataSource.GroupDescriptors](#) will be cleared by default. You need to add `DataSource.GroupDescriptors` again after changing ItemsSource if you want to retain grouping in listview.

---

### Programmatic grouping

The SfListView allows programmatic grouping by defining the [GroupDescriptor](#) object, and adding it into the [DataSource.GroupDescriptors](#) collection. The `GroupDescriptor` object holds the following properties:

- [PropertyName](#): Describes the name of the property to be grouped.
- [KeySelector](#): Describes selector to return the group key.
- [Comparer](#): Describes comparer to be applied in when sorting take place.

## XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
  <syncfusion:SfListView x:Name="listView">
    <syncfusion:SfListView.DataSource>
      <data:DataSource>
        <data:DataSource.GroupDescriptors>
          <data:GroupDescriptor PropertyName="BookName"/>
        </data:DataSource.GroupDescriptors>
      </data:DataSource>
    </syncfusion:SfListView.DataSource>
  </syncfusion:SfListView>
</ContentPage>
```

## C#

```
listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()
{
    PropertyName = "ContactName",
});
```

### Custom grouping

ListView supports grouping the items based on custom logic for each [GroupDescriptor](#) by using [KeySelector](#). Below topics explains how to achieve different custom grouping use cases with code examples,

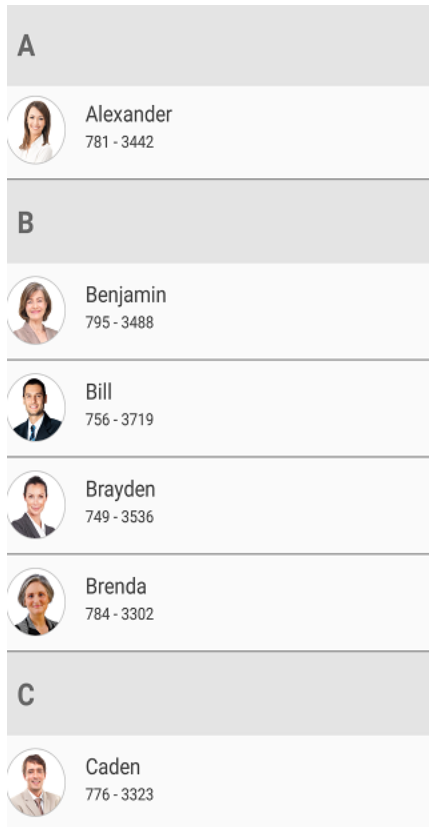
#### Grouping based on first character

The SfListView supports grouping the items based on first character of the value assigned to the property name in [GroupDescriptor](#) by using [KeySelector](#).

## C#

```
listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()
{
    PropertyName = "ContactName",
    KeySelector = (object obj1) =>
    {
        var item = (obj1 as Contacts);
        return item.ContactName[0].ToString();
    }
    Comparer = new CustomGroupComparer()
});
```

The following screenshot shows the output when grouping based on first character.



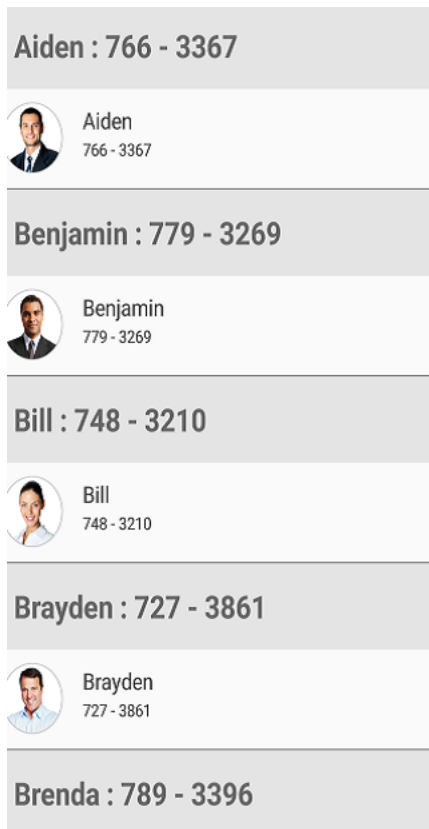
*Grouping based on more than one property in the data object*

Group the items by binding multiple properties to the property name of [GroupDescriptor](#) by using [KeySelector](#) in which the group header items can be created with multiple data model object effectively.

### C#

```
listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()  
{  
    PropertyName = "Designation",  
    KeySelector = (object obj1) =>  
    {  
        var item = (obj1 as Employee);  
        return item.Designation + item.Level;  
    }  
});
```

The following screenshot shows the output when grouping based on more than one property .



#### *Grouping by ignoring case sensitivity*

Grouping the items by ignoring case sensitive by using the [KeySelector](#) property in the [GroupDescriptor](#). While returning the [KeySelector](#), convert the required property name in the data model to group either as Upper or Lower case. The items will be grouped based on the [KeySelector](#) with returned case sensitive.

#### **C#**

```
listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()  
{  
    PropertyName = "ContactName",  
    KeySelector = (object obj) =>  
    {  
        return (obj as Contacts).ContactName.ToUpper()[0];  
    }  
});
```

The following screenshot shows grouping by ignoring case sensitivity.

Download the entire source code from GitHub [here](#).

KYLE
Kyle 933 - 82721
kYLe 887 - 97995
GINA
Gina 946 - 87450
ginA 947 - 99886
IRENE
Irene 978 - 93395
ireNE 981 - 88912
KATIE
KATie 939 - 94294
KaTie 977 - 90541

### Sorting the groups

ListView sorts the groups using default sorting logic of List.

#### *Custom sorting of groups*

The SfListView supports sorting the groups based on custom logic applied to either [SfListView.DataSource.GroupComparer](#) property or [GroupDescriptor.Comparer](#) added to the `DataSource.GroupDescriptors` collection.

In custom group comparer, all the items present in a group compares each other based on the items count to each group sorted accordingly. You can download the entire sample code [here](#)

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable"
xmlns:local="clr-namespace:CustomGrouping">
<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding
ContactsInfo}">
<syncfusion:SfListView.DataSource>
<dataSource:DataSource>
<dataSource:DataSource.GroupDescriptors>
<dataSource:GroupDescriptor PropertyName="ContactType">
<dataSource:GroupDescriptor.Comparer>
<local:CustomGroupComparer/>
</dataSource:GroupDescriptor.Comparer>
</dataSource:GroupDescriptor>
</dataSource:DataSource.GroupDescriptors>
```

```

</dataSource:DataSource>
</syncfusion:SfListView.DataSource>
</syncfusion:SfListView>
</ContentPage>

```

**C#**

```

public class CustomGroupComparer : IComparer<GroupResult>
{
    public int Compare(GroupResult x, GroupResult y)
    {
        if (x.Count > y.Count)
        {
            //GroupResult y is stacked into top of the group i.e., Ascending.
            //GroupResult x is stacked at the bottom of the group i.e., Descending.
            return 1;
        }
        else if (x.Count < y.Count)
        {
            //GroupResult x is stacked into top of the group i.e., Ascending.
            //GroupResult y is stacked at the bottom of the group i.e., Descending.
            return -1;
        }
        return 0;
    }
}

```

*Sorting the items within group*

Group the items of underlying collection along with sorting by adding the [DataSource.GroupDescriptors](#) and the [DataSource.SortDescriptors](#) with required properties.

**C#**

```

public GroupingPage()
{
    InitializeComponent();
    listView.DataSource.SortDescriptors.Add(new SortDescriptor { PropertyName =
"ContactName", Direction = ListSortDirection.Ascending });
    //Applying custom grouping
    listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()
    {
        PropertyName = "ContactName",
        KeySelector = (object obj1) =>
        {
            var item = (obj1 as Contacts);
            return item.ContactName[0].ToString();
        },
    });
}

```

*Group header summary**Aggregate summary*

For each group, display the sum of values of the property from model object in the [SfListView.GroupHeaderTemplate](#) by using converter.



**XML**

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
<syncfusion:SfListView x:Name="listView">
<syncfusion:SfListView.GroupHeaderTemplate>
<DataTemplate x:Name="GroupHeaderTemplate" x:Key="GroupHeaderTemplate">
<ViewCell>
<ViewCell.View>
<Grid BackgroundColor="#E4E4E4">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Label Text="{Binding Key}" />
<Label Text="{Binding Items,Converter={StaticResource Converter}}"
Grid.Column="1" />
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.GroupHeaderTemplate>
</syncfusion:SfListView>
</ContentPage>

```

**C#**

```

listView.GroupHeaderTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var label1 = new Label();
    label1.SetBinding(Label.TextProperty, new Binding("Key"));
    var label2 = new Label();
    Binding binding = new Binding("Items");
    binding.Converter = new Converter();
    label2.SetBinding(Label.TextProperty, binding);
    grid.Children.Add(label1);
    grid.Children.Add(label2, 1, 0);
    return grid;
});
public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
{
    int result = 0;
    var items = value as IEnumerable;
    if(items != null)
    {
        var items = items.ToList<object>().ToList<object>();
        if (items != null)
        {
            for (int i = 0; i < items.Count; i++)
            {
                var contact = items[i] as Contacts;
                result += contact.ContactNumber;
            }
        }
    }
}

```







```

}
}
return result
}

```

The following screenshot shows grouping by sum of property value.

Download the entire sample code [here](#).

A Sum of salary: 38391	
	Adam 10,913
	Adrian 14,659
	Aiden 12,819
B Sum of salary: 34377	
	Bill 12,989
	Brenda 10,234
	Bryson 11,154

#### Displaying items count

The total number of items in each group will be displayed in the group header by binding the [Count](#) property in the [SfListView.GroupHeaderTemplate](#).

#### XML

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
  <syncfusion:SfListView>
    <syncfusion:SfListView.GroupHeaderTemplate>
      <DataTemplate>
        <Grid BackgroundColor="#E4E4E4">
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
          </Grid.ColumnDefinitions>
          <StackLayout Orientation="Horizontal" HorizontalOptions="Start"
            VerticalOptions="Center" >

```

```

<Label Text="{Binding Key}" TextColor="Black" />
<Label Text="Year" TextColor="Black" />
</StackLayout>
<StackLayout Orientation="Horizontal" Grid.Column="1"
HorizontalOptions="EndAndExpand" VerticalOptions="Center">
<Label Text="{Binding Count}" TextColor="Black" />
<Label Text="Item(s)" TextColor="Black" />
</StackLayout>
</Grid>
</DataTemplate>
</syncfusion:SfListView.GroupHeaderTemplate>
</syncfusion:SfListView>
</ContentPage>

```

## C#










```

listView.GroupHeaderTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var stack1 = new StackLayout()
    {
        VerticalOptions = LayoutOptions.Center,
        HorizontalOptions = LayoutOptions.Start,
        Orientation = StackOrientation.Horizontal
    };
    var yearLabel = new Label
    {
        TextColor = Color.Black,
    };
    var yearlabel2 = new Label() { Text="Year", TextColor=Color.Black};
    yearLabel.SetBinding(Label.TextProperty, new Binding("key"));
    var stack2 = new StackLayout()
    {
        VerticalOptions = LayoutOptions.Center,
        HorizontalOptions = LayoutOptions.EndAndExpand,
        Orientation = StackOrientation.Horizontal
    };
    var countLabel = new Label
    {
        TextColor = Color.Black,
    };
    countLabel.SetBinding(Label.TextProperty, new Binding("Count"));
    var countlabel2 = new Label() { Text="Item's" , TextColor=Color.Black};
    grid.Children.Add(stack1);
    grid.Children.Add(stack2, 1, 0);
    return grid;
});

```

The following screenshot shows the output when displaying items count at group header.

Download entire sample code from GitHub [here](#).

24 Year	19 Item(s)
28 Year	22 Item(s)
27 Year	16 Item(s)
 Katie 750 - 3102	
 Michael 763 - 3691	
 Oscar 724 - 3177	
 Fiona 752 - 3489	
 Jack 761 - 3156	
 Holly 728 - 3321	
 Jackson 789 - 3789	
 Jayden 736 - 3781	
 Joseph 795 - 3086	

### Multi-level grouping

The SfListView supports multiple level grouping by adding multiple [GroupDescriptor](#) objects into the [DataSource.GroupDescriptors](#) collection. The grouped items will be displayed in hierarchical structure by customizing the [SfListView.GroupHeaderTemplate](#) property. In the [GroupHeaderTemplate](#), set the [Padding](#) property to the custom view to arrange the group header items and sub-group header items in the hierarchical structure.

**Note:** Multi-level grouping is only applicable for [LinearLayout](#) in the SfListView.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
  <ContentPage.Resources>
    <ResourceDictionary>
      <local:GroupHeaderConverter x:Key="TemplateConverter"/>
    </ResourceDictionary>
  </ContentPage.Resources>
  <syncfusion:SfListView ItemsSource="{Binding EmployeeInfo}" ItemSize="60">
    <syncfusion:SfListView.DataSource>
      <data:DataSource>
        <data:DataSource.GroupDescriptors>
          <data:GroupDescriptor PropertyName="Designation" />
          <data:GroupDescriptor PropertyName="Level" />
        </data:DataSource.GroupDescriptors>
      </data:DataSource>
    </syncfusion:SfListView.DataSource>
  </syncfusion:SfListView>
</ContentPage>
```

```

</syncfusion:SfListView.DataSource>
<syncfusion:SfListView.GroupHeaderTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<StackLayout BackgroundColor="{Binding Level,Converter={StaticResource
TemplateConverter}}"
Padding="{Binding Level,Converter={StaticResource TemplateConverter}}">
<Label Text="{Binding Key}"
VerticalOptions="Center" HorizontalOptions="Start"/>
</StackLayout>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.GroupHeaderTemplate>
</syncfusion:SfListView>
</ContentPage>

```

**C#**

```

listView.ItemsSource = viewModel.EmployeeInfo;
listView.ItemSize = 60;
listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()
{
    PropertyName = "Designation",
});
listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()
{
    PropertyName = "Designation",
});
listView.GroupHeaderTemplate = new DataTemplate(() =>
{
    var stack = new StackLayout();
    Binding binding = new Binding("Level");
    binding.Converter = new TemplateConverter();
    stack.SetBinding(StackLayout.BackgroundColorProperty, binding);
    stack.SetBinding(StackLayout.PaddingProperty, binding);
    var label = new Label() {
        VerticalOptions=LayoutOptions.Center,HorizontalOptions=LayoutOptions.Start};
    label.SetBinding(Label.TextProperty, new Binding("Key"));
    return stack;
});
public class GroupHeaderConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        if (targetType.Name == "Color")
        {
            if ((int)value == 1)
                return Color.FromHex("#D3D3D3");
            else
                return Color.Transparent;
        }
        else
        {

```

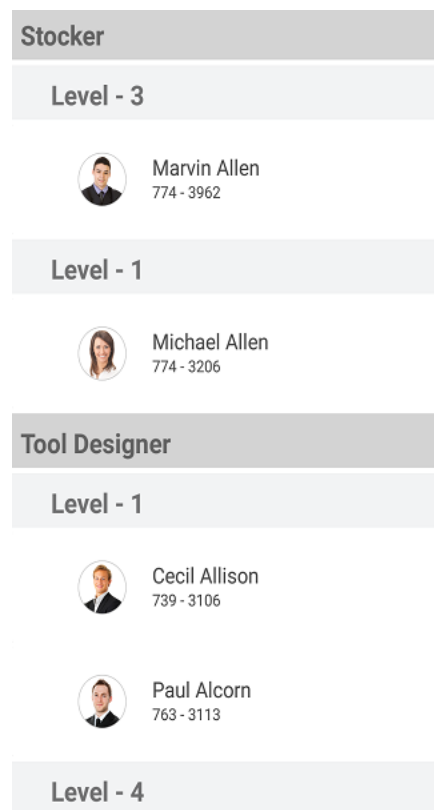
```

if ((int)value == 1)
return new Thickness(5, 5, 5, 0);
else
return new Thickness((int)value * 15, 5, 5, 0);
}
}
public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
{
throw new NotImplementedException();
}
}

```

The following screenshot shows the output for multi-level grouping.

Download the entire source code from GitHub [here](#).



### Group expand and collapse

By default, the groups will be in expanded state in the SfListView. You can expand or collapse the group at runtime by setting the [SfListView.AllowGroupExpandCollapse](#) to true. So, when tapping the group header, the group gets collapse if the group is in expand state and vice-versa.

### XML

```

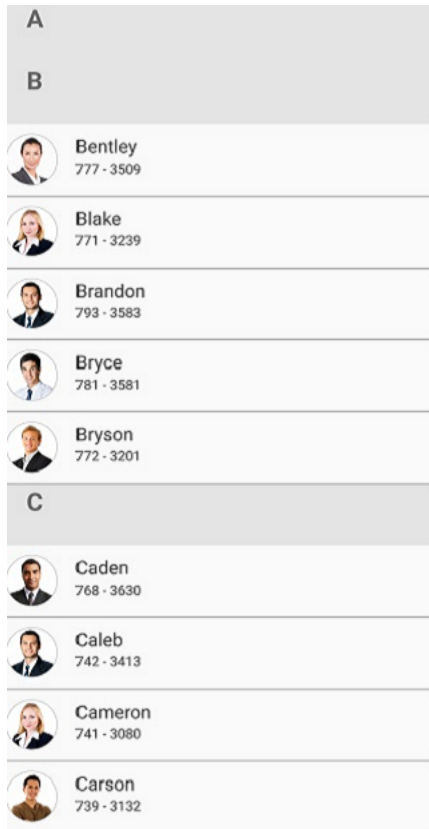
<syncfusion:SfListView x:Name="listView" ItemSize="70"
AllowGroupExpandCollapse="True"
ItemsSource="{Binding contactsInfo}" />

```

**C#**

```
listView.AllowGroupExpandCollapse = true;
```

The following screenshot shows the output when the groups are collapsed.

*Programmatic expand and collapse**Expand or collapse all the groups*

Expand or collapse all the groups programmatically at runtime by using the [SfListView.ExpandAll](#) method and [SfListView.CollapseAll](#) method.

**C#**

```
listView.ExpandAll();  
listView.CollapseAll();
```

*Expand or collapse a specific group*

Expand or collapse a specific group by using the [SfListView.ExpandGroup](#) method and [SfListView.CollapseGroup](#) method.

**C#**

```
var group = listView.DataSource.Groups[0];  
listView.ExpandGroup(group);  
listView.CollapseGroup(group);
```

### Expand or collapse all groups by default

Expand or collapse all the groups by default using the [SfListView.Loaded](#) event.

#### C#

```
listView.Loaded += ListView_Loaded;
private void ListView_Loaded(object sender, ListViewLoadedEventArgs e)
{
    listView.CollapseAll();
    listView.ExpandAll();
}
```

### Keeping only one group in expanded state

To keep any one specific group alone in the expanded state, use the [SfListView.GroupExpanding](#) event. The particular selected group can be get from [GroupExpandCollapseChangingEventArgs](#), by which you can compare and collapse all other groups, and expand the particular selected group.

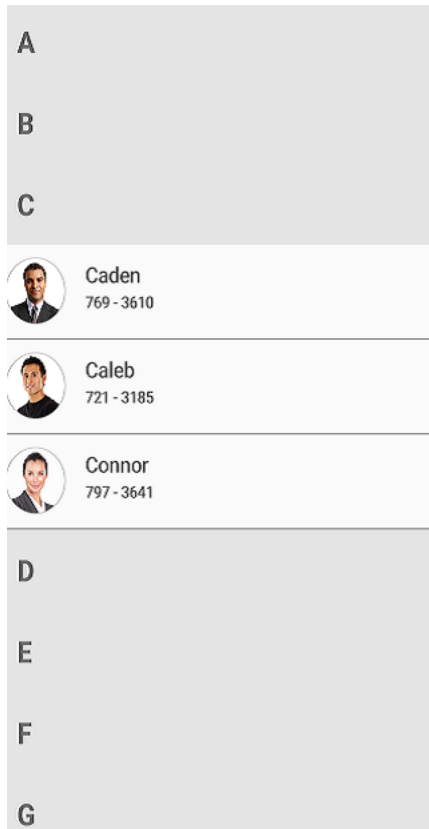
#### C#

```
private void ListView_GroupExpanding(object sender,
GroupExpandCollapseChangingEventArgs e)
{
    if (e.Groups.Count > 0)
    {
        var group = e.Groups[0];
        if (expandedGroup == null || group.Key != expandedGroup.Key)
        {
            foreach (var otherGroup in listView.DataSource.Groups)
            {
                if (group.Key != otherGroup.Key)
                {
                    listView.CollapseGroup(otherGroup);
                }
            }
            expandedGroup = group;
            listView.ExpandGroup(expandedGroup);
        }
    }
}
```

The following screenshot shows the output when one group in expanded state.

Download the entire source code from GitHub [here](#).





### Events

#### GroupExpanding Event

The [SfListView.GroupExpanding](#) event occurs when the group is being expanded.

The [GroupExpandCollapseChangingEventArgs](#) of the [GroupExpanding](#) event provides the information about the expanding group and it has the following members:

[Groups](#): Gets a list of groups being expanded.

[Cancel](#): Decides whether to cancel the group expansion or not.

The [GroupExpanding](#) event used for the following use case.

- Keeps any one specific group in the expanded state by comparing and collapsing all other groups.

You can cancel the group expansion by setting [GroupExpandCollapseChangingEventArgs.Cancel](#) to true.

### XML

```
<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding contactsInfo}"
    GroupExpanding="ListView_GroupExpanding" />
```

### C#

```
listView.GroupExpanding += ListView_GroupExpanding;
```

**C#**

```
private void ListView_GroupExpanding(object sender,
GroupExpandCollapseChangingEventArgs e)
{
    if (e.Groups[0] == listView.DataSource.Groups[0])
        e.Cancel = true;
}
```

*GroupExpanded Event*

The [SfListView.GroupExpanded](#) event occurs after expanding the group.

The [GroupExpandCollapseChangedEventArgs](#) of the [GroupExpanded](#) event provides the information about the expanded group and it has the following member:

[Groups](#): Gets a list of expanded groups.

*GroupCollapsing Event*

The [SfListView.GroupCollapsing](#) event occurs when the group is being collapsed.

The [GroupExpandCollapseChangingEventArgs](#) of the [GroupCollapsing](#) event provides the information about the collapsing group and it contains the following members:

[Groups](#): Gets a list of groups being collapsed.

[Cancel](#): Decides whether to cancel the group collapsing or not.

You can cancel the group is being collapsed by using the [GroupExpandCollapseChangingEventArgs.Cancel](#) of [GroupCollapsing](#) event.

**XML**

```
<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding
contactsInfo}"
GroupCollapsing="ListView_GroupCollapsing" />
```

**C#**

```
listView.GroupCollapsing += ListView_GroupCollapsing;
```

**C#**

```
private void ListView_GroupCollapsing(object sender,
GroupExpandCollapseChangingEventArgs e)
{
    if (e.Groups[0] == listView.DataSource.Groups[0])
        e.Cancel = true;
}
```

*GroupCollapsed Event*

The [SfListView.GroupCollapsed](#) event occurs after the group is collapsed.

The [GroupExpandCollapseChangedEventArgs](#) of the [GroupCollapsed](#) event provides the information about collapsed group and it contains the following member.

[Groups](#): Gets a list of collapsed groups.

#### Stick group header

To stick the group header to view, enable the property [SfListView.IsStickyGroupHeader](#). If [IsStickyGroupHeader](#) is true, the corresponding group header will be in view until the last item of the group goes out of view, and sticky group header will move when another group header leads while scrolling.

**Information:** If sticky group header is enabled and [AutoFitMode](#) is [Height](#), the panning experience will not be smooth or item's layout will not work as expected. To make the panning experience smooth, set the same size for all group header items by handling the [QueryItemSize](#) event.

**Note:** When the [IsStickyGroupHeader](#) is set to true, the [IsStickyHeader](#) property will be changed to true because the header item can not be scrolled. When the [IsStickyHeader](#) is set to false, if [IsStickyGroupHeader](#) is true then it will be changed to false because the group header item cannot be sticky.

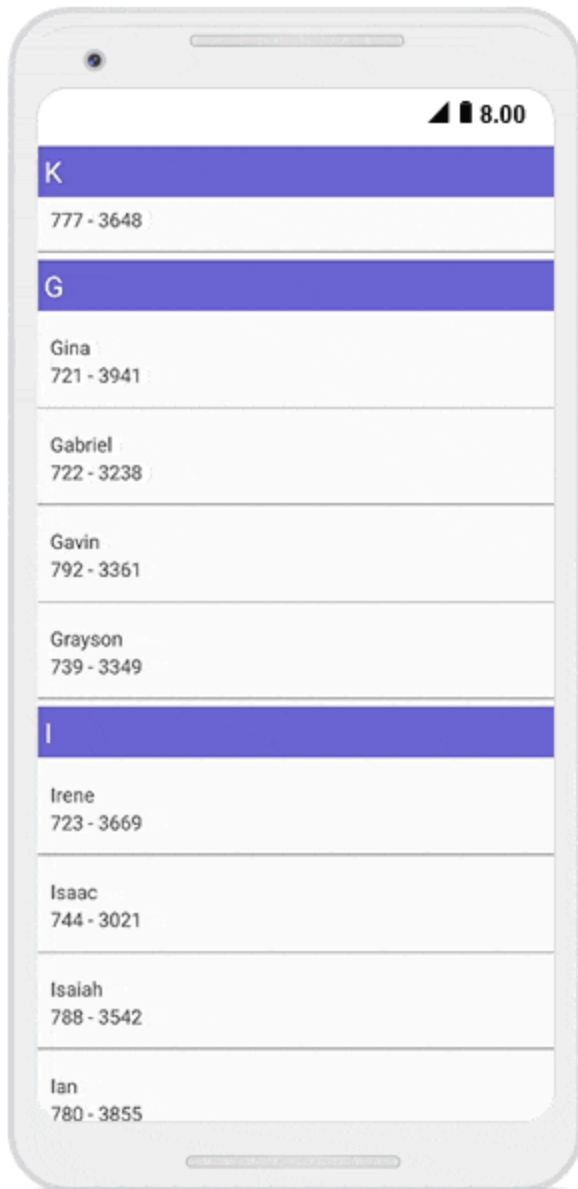
#### XML

```
<syncfusion:SfListView x:Name="listView" ItemSize="70"
IsStickyGroupHeader="True"
ItemsSource="{Binding contactsInfo}" />
```

#### C#

```
listView.IsStickyGroupHeader = true;
```

The following screenshot illustrates the output when the group headers are sticky.



## Group header customization

### *Appearance customization*

The User Interface (UI) for the group header items can be customized by using the [SfListView.GroupHeaderTemplate](#) property.

To customize the view for group header items and binding the [Key](#) to it, follow the code example.

### **XML**

```
<ContentPage xmlns:syncfusion="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView">
    <syncfusion:SfListView.GroupHeaderTemplate>
      <DataTemplate>
        <ViewCell>
```

```
<ViewCell.View>
<StackLayout BackgroundColor="#E4E4E4">
<Label Text="{Binding Key}"
FontSize="22"
FontAttributes="Bold"
VerticalOptions="Center"
HorizontalOptions="Start"
Margin="20,0,0,0" />
</StackLayout>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.GroupHeaderTemplate>
</syncfusion:SfListView>
</ContentPage>
```

## C#

```
listView.GroupHeaderTemplate = new DataTemplate(() =>
{
    var grid = new Grid { BackgroundColor = Color.FromHex("#E4E4E4") };
    var label = new Label
    {
        VerticalOptions = LayoutOptions.Center,
        HorizontalOptions = LayoutOptions.Start,
        Margin = new Thickness(20, 0, 0, 0),
    };
    label.SetBinding(Label.TextProperty, new Binding("Key"));
    grid.Children.Add(label);
    return grid;
});
```

The following screenshot shows the output when the groups header appearance customized by key.

### Network Administrator



Marvin Allen  
778 - 3607



Thomas Armstrong  
745 - 3072



Frances Adams  
731 - 3200

### Senior Tool Designer



Michael Allen  
752 - 3012

### Master Scheduler



Cecil Allison  
731 - 3048

#### *Expand and collapse icon in group header*

Expand and collapse the group when tapping icon in the group header item by customizing the [SfListView.GroupHeaderTemplate](#) with the help of converter.

#### **XML**

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <ContentPage.Resources>
    <ResourceDictionary>
      <local:BoolToImageConverter x:Key="BoolToImageConverter"/>
      <DataTemplate x:Name="GroupHeaderTemplate" x:Key="GroupHeaderTemplate">
        <ViewCell>
          <ViewCell.View>
            <Grid>
              <Grid.ColumnDefinitions>
                <ColumnDefinition Width="30" />
                <ColumnDefinition Width="*" />
              </Grid.ColumnDefinitions>
              <Image x:Name="NormalImage" Grid.Column="0" HorizontalOptions="Center"
                Source="{Binding IsExpand, Converter={StaticResource BoolToImageConverter}}"
                VerticalOptions="Center"/>
              <Label Text="{Binding Key}" Grid.Column="1"/>
            </Grid>
          </ViewCell.View>
        </ViewCell>
      </DataTemplate>
    </ResourceDictionary>
  </ContentPage.Resources>
```

```

<ContentPage.Content>
<Grid>
<syncfusion:SfListView x:Name="listView" ItemSize="70"
GroupHeaderSize="60"
GroupHeaderTemplate="{StaticResource GroupHeaderTemplate}"
ItemsSource="{Binding contactsInfo}"
AllowGroupExpandCollapse="True">
</syncfusion:SfListView>
</Grid>
</ContentPage.Content>
</ContentPage>

```

## C#

```

listView = new SfListView();
viewModel = new ContactsViewModel();
listView.ItemsSource = viewModel.contactsInfo;
listView.GroupHeaderTemplate = new DataTemplate(() =>
{
var grid = new Grid { BackgroundColor = Color.FromHex("#E4E4E4") };
var column0 = new ColumnDefinition { Width = 30 };
var column1 = new ColumnDefinition { Width = new GridLength(1,
GridUnitType.Star) };
grid.ColumnDefinitions.Add(column0);
grid.ColumnDefinitions.Add(column1);
var image = new Image();
Binding binding = new Binding("IsExpand");
binding.Converter = new BoolToImageConverter();
image.SetBinding(Image.SourceProperty, binding);
image.HeightRequest = 30;
image.WidthRequest = 30;
image.VerticalOptions = LayoutOptions.Center;
image.HorizontalOptions = LayoutOptions.Center;
var label = new Label
{
FontAttributes = FontAttributes.Bold,
FontSize = 22,
VerticalOptions = LayoutOptions.Center,
HorizontalOptions = LayoutOptions.Start,
Margin = new Thickness(20, 0, 0, 0),
};
label.SetBinding(Label.TextProperty, new Binding("Key"));
grid.Children.Add(image, 0, 0);
grid.Children.Add(label, 1, 0);
return grid;
});

```

You can switch the expand or collapse group icon based on `IsExpand` property using the converter.

## C#

```

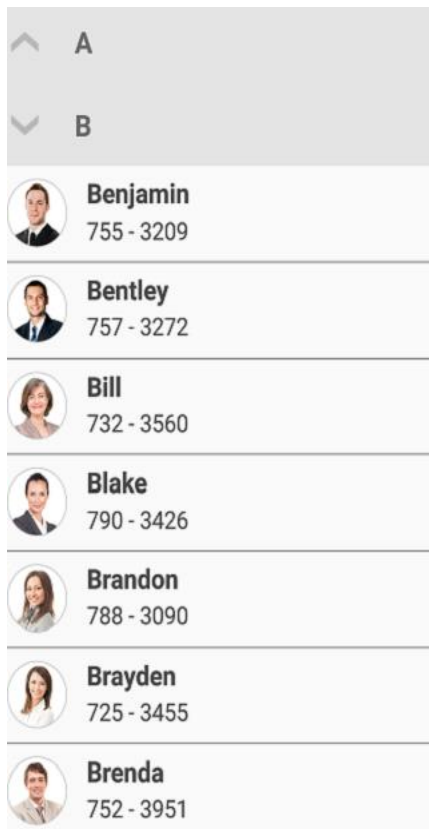
public class BoolToImageConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
{

```

```
if ((bool) value)
return ImageSource.FromResource("ListViewSample.Images. GroupExpand.png");
else
return ImageSource.FromResource("ListViewSample.Images. GroupCollapse.png");
}
public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
{
throw new NotImplementedException();
}
}
```

The following screenshot shows the output when grouping with expand collapse icon in group header.

Download entire source code from GitHub [here](#).



#### *Height customization*

The size of the group header items can be customized by setting the [SfListView.GroupHeaderSize](#) property. The default value of this property is 40. This property responds to runtime changes.

#### **XML**

```
<syncfusion:SfListView x:Name="listView" GroupHeaderSize="60" />
```

#### **C#**

```
listView.GroupHeaderSize = 60;
```



**Note:** For Vertical orientation, the group header size is considered as height and for Horizontal orientation, it will be considered as width.

#### *CheckBox in group header*

ListView supports selecting each group and items in the group like a checkBox selection by customizing the [SfListView.GroupHeaderTemplate](#) and the [ItemTemplate](#) respectively. The checkbox state will be update by using converter.

#### **XML**

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView>
<syncfusion:SfListView.GroupHeaderTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid BackgroundColor="#d3d3d3">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="30" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<Label Text="{Binding Key}" Grid.Column="1" VerticalTextAlignment="Center"/>
<Image Grid.Column="2" IsVisible="{Binding SelectionMode,
Source={x:Reference listView}}"
HorizontalOptions="Center" VerticalOptions="Center"
Source="{Binding ., Converter={StaticResource GroupingSelectionConverter},
ConverterParameter={x:Reference listView}}">
<Image.GestureRecognizers>
<TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped"/>
</Image.GestureRecognizers>
</Image>
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.GroupHeaderTemplate>
</syncfusion:SfListView>
</ContentPage>
```

#### **C#**

```
listView.GroupHeaderTemplate = new DataTemplate(() =>
{
var grid = new Grid();
var label = new Label() { VerticalTextAlignment=TextAlignment.Center};
label.SetBinding(Label.TextProperty, new Binding("Key"));
var image = new Image() { VerticalOptions=LayoutOptions.Center,
HorizontalOptions=LayoutOptions.Center};
Binding binding = new Binding(".");
binding.Converter = new GroupingSelectionConverter();
binding.ConverterParameter = listView;
image.SetBinding(Image.SourceProperty, binding);
Binding bind = new Binding("SelectionMode");
bind.Source = listView;
```

```

image.SetBinding(Image.IsVisibleProperty, bind);
var tapped = new TapGestureRecognizer();
tapped.Tapped += Image_Tapped;
image.GestureRecognizers.Add(tapped);
grid.Children.Add(label);
grid.Children.Add(image, 2, 0);
return grid;
});

```

The checkBox state in the `GroupHeaderTemplate` will be updated whenever items select and deselect by using converter.

### C#

```

public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
{
    if (value == null)
        return value;
    GroupResult groupResult = value as GroupResult;
    SfListView list = parameter as SfListView;
    var items = new List<MusicInfo>(groupResult.Items.ToList<MusicInfo>());
    if ((items.All(item => item.IsSelected == false)))
    {
        for (int i = 0; i < items.Count(); i++)
        {
            var item = items[i];
            (item as MusicInfo).IsSelected = false;
            list.SelectedItems.Remove(item);
        }
        return ImageSource.FromResource("CustomSelection.Images.NotSelected.png");
    }
    else if ((items.All(item => item.IsSelected == true)))
    {
        for (int i = 0; i < items.Count(); i++)
        {
            var item = items[i];
            (item as MusicInfo).IsSelected = true;
            list.SelectedItems.Add(item);
        }
        return ImageSource.FromResource("CustomSelection.Images.Selected.png");
    }
    else
        return ImageSource.FromResource("CustomSelection.Images.Intermediate.png");
}

```

To select and deselect all the items of group by tap the checkbox in the group header, follow the code example.

### C#

```

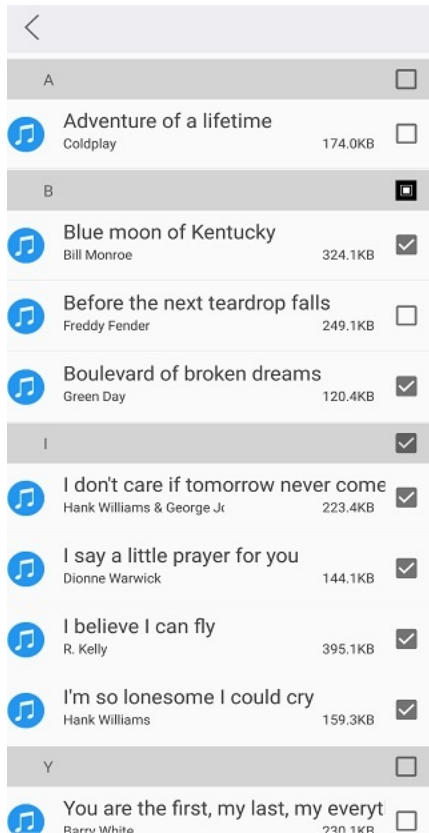
using Syncfusion.ListView.XForms.Control.Helpers;
private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
{
    var image = (sender as Image);
    var groupResult = image.BindingContext as GroupResult;
}

```

```
if (groupResult == null)
return;
var items = groupResult.Items.ToList<MusicInfo>().ToList();
if ((items.All(listItem => listItem.IsSelected == true)))
{
for (int i = 0; i < items.Count(); i++)
{
var item = items[i];
(item as MusicInfo).IsSelected = false;
}
}
else if ((items.All(listItem => listItem.IsSelected == false)))
{
for (int i = 0; i < items.Count(); i++)
{
var item = items[i];
(item as MusicInfo).IsSelected = true;
}
}
this.RefreshGroupHeader(groupResult);
listView.RefreshView();
}
private void RefreshGroupHeader(GroupResult group)
{
foreach (var item in this.listView.GetVisualContainer().Children)
{
if (item.BindingContext == group)
{
item.BindingContext = null;
(item as GroupHeaderItem).Content.BindingContext = null;
}
}
}
```

The following screenshot shows the output when checking items in group header.

Download entire source code from GitHub [here](#).



*Changing group header appearance when expanding*

Change the [SfListView.GroupHeaderTemplate](#) appearance like `BackgroundColor` of view while expanding the particular group with the help of Converter.

#### **XML**

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:data="clr-
namespace:Syncfusion.DataSource;assembly=Syncfusion.DataSource.Portable">
<ContentPage.Resources>
<ResourceDictionary>
<local:SelectionBoolToBackgroundColorConverter
x:Key="BoolToColorConverter"/>
</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfListView x:Name="listView" ItemSize="90"
AllowGroupExpandCollapse="True"
ItemSpacing="2" ItemsSource="{Binding Items}">
<syncfusion:SfListView.DataSource>
<data:DataSource>
<data:DataSource.GroupDescriptors>
<data:GroupDescriptor PropertyName="DisplayString"/>
</data:DataSource.GroupDescriptors>
</data:DataSource>
</syncfusion:SfListView.DataSource>
<syncfusion:SfListView.GroupHeaderTemplate>
<DataTemplate>
<StackLayout BackgroundColor="{Binding Path=IsExpand,
```

```

Converter={StaticResource BoolToColorConverter}}">
<Label Text="{Binding Key}"
VerticalOptions="Center" HorizontalOptions="Start" />
</StackLayout>
</DataTemplate>
</syncfusion:SfListView.GroupHeaderTemplate>
</syncfusion:SfListView>
</ContentPage>

```

**C#**

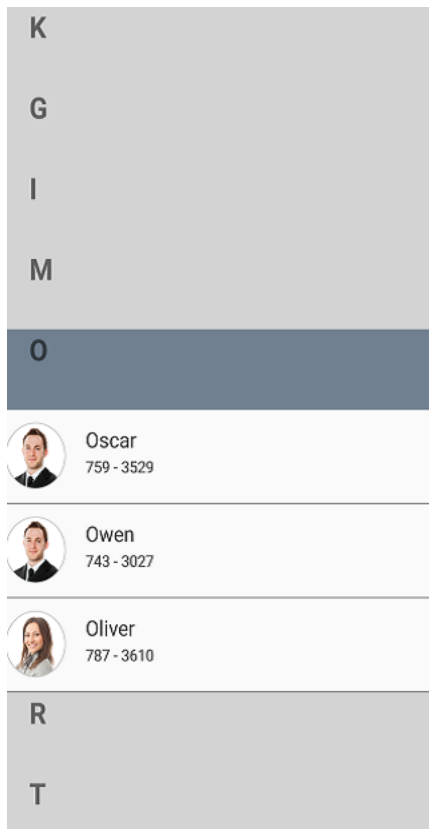
```

listView.DataSource.GroupDescriptors.Add(new GroupDescriptor()
{
    PropertyName = "BookName",
});
listView.GroupHeaderTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var stack = new StackLayout();
    Binding binding = new Binding("IsExpand");
    binding.Converter = new BoolToColorConverter();
    stack.SetBinding(StackLayout.BackgroundColorProperty, binding);
    var label = new Label() {
        VerticalOptions=LayoutOptions.Center,HorizontalOptions=LayoutOptions.Start};
    label.SetBinding(Label.TextProperty, new Binding("Key"));
    stack.Children.Add(label);
    grid.Children.Add(stack);
    return grid;
});
public class SelectionBoolToBackgroundColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return (bool)value == true ? Color.FromHex("#E4E4E4") :
            Color.FromHex("#ADD8E6");
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

The following screenshot shows the output when group header appearance changed by expanding.

Download entire source code from GitHub [here](#)



#### *Providing Indentation for GroupHeader items*

ListView allows you to provide space between the group header items by using the [Margin](#) property of parent view in the [GroupHeaderTemplate](#) property. For example, in the following code snippet, StackLayout is considered as parent view and spacing is provided by setting its margin.

#### **XML**

```
<syncfusion:SfListView x:Name="listView" ItemSize="90"
AllowGroupExpandCollapse="True"
ItemSpacing="2" ItemsSource="{Binding Items}">
  <syncfusion:SfListView.GroupHeaderTemplate>
    <DataTemplate>
      <StackLayout BackgroundColor="#E4E4E4" Margin="2,0,0,0">
        <Label Text="{Binding Key}" FontSize="22" />
      </StackLayout>
    </DataTemplate>
  </syncfusion:SfListView.GroupHeaderTemplate>
</syncfusion:SfListView>
```

#### How To

##### *Allow to select only one item in a group at a time*

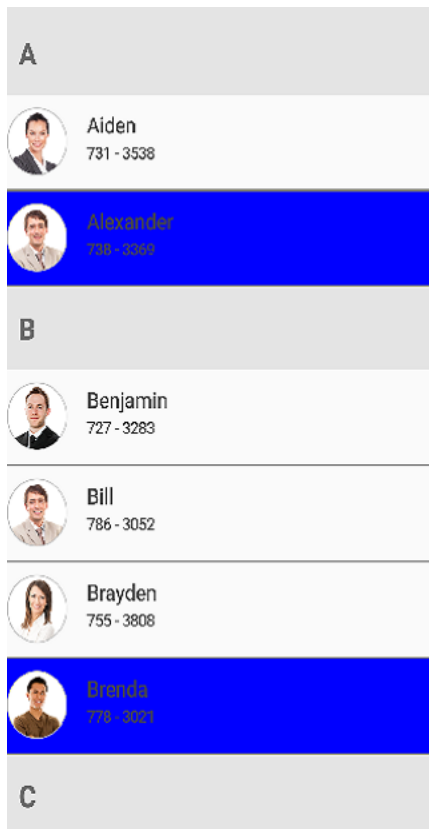
To select only one item in a group at a time, use the [ItemSelectionChangingEventArgs](#) event. If more than one item in the same group gets selected, already selected item will be removed from ListView's SelectedItems.

#### **C#**

```
private void ListView_SelectionChanging(object sender,
ItemSelectionChangingEventArgs e)
{
    GroupResult actualGroup = null;
    object key = null;
    var selectedItems = listView.SelectedItems;
    //To Cancel the Deselection
    if (e.RemovedItems.Count > 0 && selectedItems.Contains(e.RemovedItems[0]))
    {
        e.Cancel = true;
        return;
    }
    //To return when SelectedItems is zero
    if (e.AddedItems.Count <= 0)
        return;
    var itemData = (e.AddedItems[0] as Contacts);
    var descriptor = listView.DataSource.GroupDescriptors[0];
    if (descriptor.KeySelector == null)
    {
        var Collection = new PropertyInfoCollection(itemData.GetType());
        key = Collection.GetValue(itemData, descriptor.PropertyName);
    }
    else
        key = descriptor.KeySelector(itemData);
    for (int i = 0; i < listView.DataSource.Groups.Count; i++)
    {
        var group = listView.DataSource.Groups[i];
        if ((group.Key != null && group.Key.Equals(key)) || group.Key == key)
        {
            actualGroup = group;
            break;
        }
    }
    if (selectedItems.Count > 0)
    {
        foreach (var item in actualGroup.Items)
        {
            var groupItem = item;
            if (selectedItems.Contains(groupItem))
            {
                listView.SelectedItems.Remove(groupItem);
                break;
            }
        }
    }
}
```

The following screenshot shows the output when only one item in a group gets selected.

Download entire source code from GitHub [here](#).



#### *Add an item at the specific index in a group*

The SfListView allows adding an item at the specific index in a group by finding the group with the help of [Key](#) value of the group.

#### **C#**

```
internal void GetGroupResult(object ItemData)
{
    var descriptor = listView.DataSource.GroupDescriptors[0];
    object key;
    if (descriptor.KeySelector == null)
    {
        var propertyInfoCollection = new PropertyInfoCollection(ItemData.GetType());
        key = propertyInfoCollection.GetValue(ItemData, descriptor.PropertyName);
    }
    else
    {
        key = descriptor.KeySelector(ItemData);
        for (int i = 0; i < this.listView.DataSource.Groups.Count; i++)
        {
            var group = this.listView.DataSource.Groups[i];
            if ((group.Key != null && group.Key.Equals(key)) || group.Key == key)
            {
                itemGroup = group;
                break;
            }
            group = null;
        }
        itemGroup = this.listView.DataSource.Groups.FirstOrDefault(x => x.Key == key);
    }
}
```



```
descriptor = null;  
key = null;  
}
```

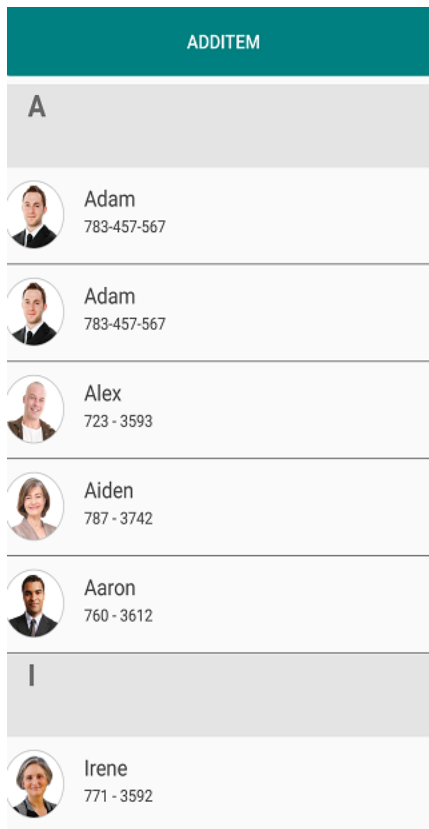
To add item at specific index in a group, follow the code example

### C#

```
using Syncfusion.ListView.XForms.Control.Helpers;  
private void AddItem_Clicked(object sender, EventArgs e)  
{  
    var contact = new Contacts();  
    contact.ContactName = "Adam";  
    contact.ContactNumber = "783-457-567";  
    contact.DisplayString = "A";  
    contact.ContactImage = ImageSource.FromResource("Grouping.Images.Image" + 25  
    + ".png");  
    ViewModel.ContactItems.Add(contact);  
    GetGroupResult(contact);  
    if (itemGroup == null)  
        return;  
    var items = itemGroup.GetType().GetRuntimeProperties().FirstOrDefault(x =>  
    x.Name == "ItemList").GetValue(itemGroup) as List<object>;  
    InsertItemInGroup(items, contact, 0);  
}  
internal void InsertItemInGroup(List<object> items, object Item, int  
InsertAt)  
{  
    visualContainer = listView.GetVisualContainer();  
    items.Remove(Item);  
    items.Insert(InsertAt, Item);  
    visualContainer.ForceLayout();  
}
```

The following screenshot shows the output when item added at specified index.

Download entire source code from GitHub [here](#).



## Header and Footer

This section explains how to define and customize the header and footer in the SfListView.

### Header and footer customization

The [SfListView](#) allows adding and customizing appearance of the header and footer by setting the [SfListView.HeaderTemplate](#) and [SfListView.FooterTemplate](#) properties.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView"
    ItemsSource="{Binding InboxInfo}"
    ItemSize="100">
    <syncfusion:SfListView.HeaderTemplate>
      <DataTemplate>
        <Grid BackgroundColor="#4CA1FE" HeightRequest="45">
          <Label LineBreakMode="NoWrap"
            Margin="10,0,0,0" Text="Inbox" FontAttributes="Bold"
            FontSize="18" TextColor="White" HorizontalOptions="Center"
            VerticalOptions="Center"/>
        </Grid>
      </DataTemplate>
    </syncfusion:SfListView.HeaderTemplate>
  </syncfusion:SfListView>
</ContentPage>
```

### C#

```
viewModel = new ViewModel ();
listView = new SfListView();
listView.ItemsSource = viewModel.InboxInfo;
listView.HeaderTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    grid.BackgroundColor = Color.FromHex("#4CA1FE");
    var headerLabel = new Label
    {
        BackgroundColor = Color.White,
        FontSize = 18,
        FontAttributes = FontAttributes.Bold,
        Text = "Inbox"
    };
    grid.Children.Add(headerLabel);
    return grid;
});
```

### Header and footer items appearance

The [SfListView](#) allows customizing size of the header and footer items by setting the [SfListView.HeaderSize](#) and [SfListView.FooterSize](#) properties. The default value is 40. These properties can be customized at runtime.

#### XML

```
<syncfusion:SfListView x:Name="listView" HeaderSize="70" FooterSize="60" />
```

#### C#

```
listView.HeaderSize = 70;
listView.FooterSize = 60;
```

**Note:** For vertical orientation, the header and footer size are considered as height. For horizontal orientation, that will be considered as width.

### Sticky header and footer

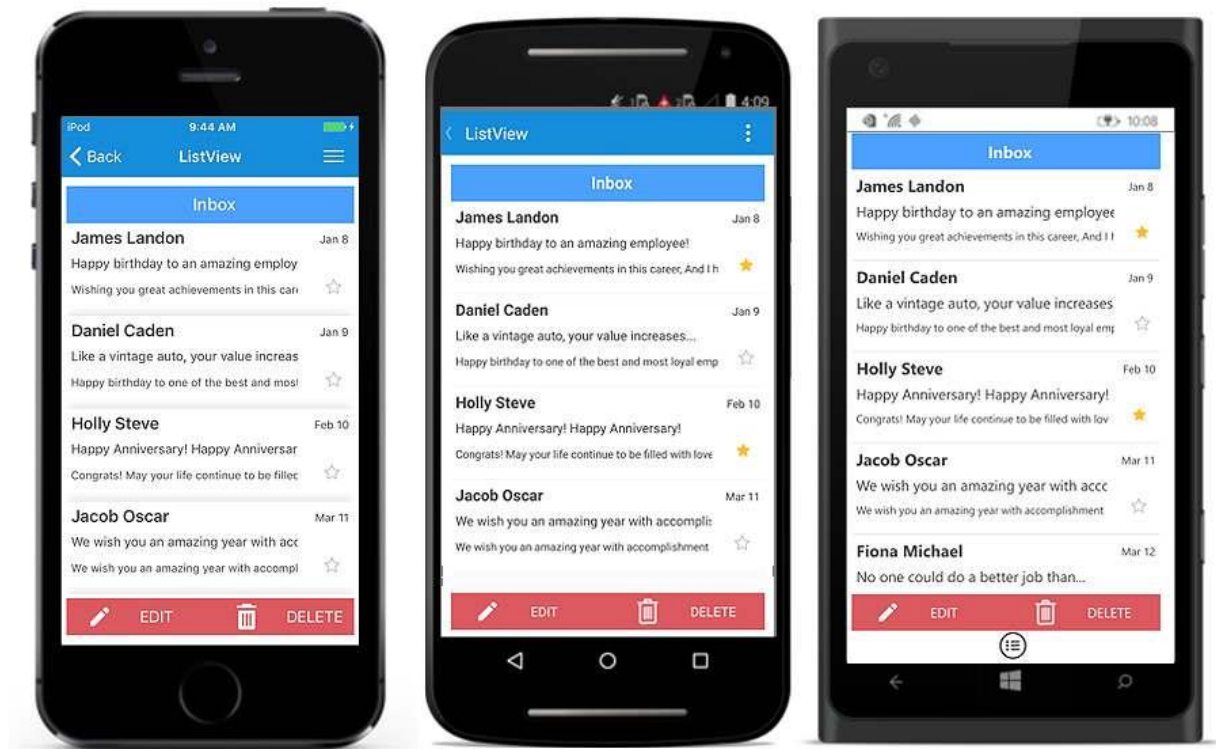
The [SfListView](#) allows sticking the header and footer items to view by enabling the [SfListView.IsStickyHeader](#) and [SfListView.IsStickyFooter](#) properties. If the [SfListView.IsStickyHeader](#) is **true**, the header item will stick to the top of the view. If the [SfListView.IsStickyFooter](#) is **true**, the footer item will stick to the bottom of the view.

#### XML

```
<syncfusion:SfListView x:Name="listView" IsStickyHeader="True"
IsStickyFooter="True" />
```

#### C#

```
listView.IsStickyHeader = true;
listView.IsStickyFooter = true;
```



### Sticky footer position

The [SfListView](#) allows to position the footer item by using the [StickyFooterPosition](#) property.

The [StickyFooterPosition](#) property has two options:

- **Body:** The footer item will be positioned inside the body of the ListView when the items are less than the view size.
- **Default:** The footer item will be positioned at the bottom of the ListView.

### XML

```
<syncfusion:SfListView x:Name="listView" StickyFooterPosition="Body"/>
```

### C#

```
listView.StickyFooterPosition = FooterPosition.Body;
```



How to

*TabView appearance using listview header*

The [SfListView](#) allows performing tab view structure by customizing the [HeaderTemplate](#) to load any view.

#### **XML**

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <ContentPage.BindingContext>
    <local:BookInfoRepository />
  </ContentPage.BindingContext>
  <ContentPage.Resources>
    <ResourceDictionary>
      <DataTemplate x:Key="template">
        <ViewCell>
```

```

<ViewCell.View>
<Grid x:Name="grid" RowSpacing="0" Margin="5">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="1" />
</Grid.RowDefinitions>
<Grid RowSpacing="0" Grid.Row="0">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Label Grid.Row="0" Text="{Binding BookName}" FontAttributes="Bold"
TextColor="Teal" FontSize="21" />
<Label Grid.Row="1" Text="{Binding BookDescription}" TextColor="Teal"
FontSize="15"/>
</Grid>
<StackLayout Grid.Row="1" BackgroundColor="#E4E4E4" HeightRequest="1"/>
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfListView x:Name="listView" IsStickyHeader="True"
IsStickyFooter="True"
HeaderSize="80" FooterSize="60" SelectionBackgroundColor="LightBlue"
ItemSize="90" SelectionMode="Single" ItemTemplate="{StaticResource
template}">
<syncfusion:SfListView.HeaderTemplate>
<DataTemplate>
<Grid BackgroundColor="#4CA1FE" HeightRequest="60">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="2" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="2" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="2" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="2" />
</Grid.ColumnDefinitions>
<Grid Grid.Column="1" HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" >
<Label x:Name="info" BackgroundColor="Transparent" Text="Informations"
FontAttributes="Bold" FontSize="18" TextColor="White"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center" />
<Grid.GestureRecognizers>
<TapGestureRecognizer Tapped="InfoTapGestureRecognizerTapped"
NumberOfTapsRequired="1"/>
</Grid.GestureRecognizers>
</Grid>
<Grid Grid.Column="3" HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" >
<Label x:Name="status" BackgroundColor="Transparent" Text="Status"
FontAttributes="Bold" FontSize="18" TextColor="White"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center" />
<Grid.GestureRecognizers>

```

```

<TapGestureRecognizer Tapped="StatusTapGestureRecognizerTapped"
NumberOfTapsRequired="1"/>
</Grid.GestureRecognizers>
</Grid>
<Grid Grid.Column="5" HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" >
<Label x:Name="contacts" BackgroundColor="Transparent" Text="Contacts"
FontAttributes="Bold" FontSize="18" TextColor="White"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"/>
<Grid.GestureRecognizers>
<TapGestureRecognizer Tapped="ContactsTapGestureRecognizerTapped"
NumberOfTapsRequired="1"/>
</Grid.GestureRecognizers>
</Grid>
<BoxView Grid.Column="0" BackgroundColor="White" />
<BoxView Grid.Column="2" BackgroundColor="White" />
<BoxView Grid.Column="4" BackgroundColor="White" />
<BoxView Grid.Column="6" BackgroundColor="White" />
</Grid>
</DataTemplate>
</syncfusion:SfListView.HeaderTemplate>
</syncfusion:SfListView>
</ContentPage>

```

## C#

```

public partial class MainPage : ContentPage
{
    BookInfoRepository viewModel = new BookInfoRepository();
    BookInfoRepository1 viewModel1 = new BookInfoRepository1();
    EmployeeViewModel viewModel2 = new EmployeeViewModel();
    public MainPage()
    {
        InitializeComponent();
        this.listView.ItemsSource = viewModel.BookInfo;
    }
    public void InfoTapGestureRecognizerTapped(object sender, EventArgs e)
    {
        this.listView.ItemsSource = viewModel.BookInfo;
        this.listView.ItemTemplate = this.Resources["template"] as DataTemplate;
    }
    public void StatusTapGestureRecognizerTapped(object sender, EventArgs e)
    {
        this.listView.ItemsSource = viewModel1.BookInfo1;
        this.listView.ItemTemplate = this.Resources["template"] as DataTemplate;
    }
    public void ContactsTapGestureRecognizerTapped(object sender, EventArgs e)
    {
        this.listView.ItemsSource = viewModel2.EmployeeInfo;
        InitializeTemplate();
    }
    private void InitializeTemplate()
    {
        listView.ItemTemplate = new DataTemplate(() => { return
        CreateItemTemplate(); });
        listView.ItemSize = 70;
    }
}

```

```

listView.GroupHeaderSize = 60;
listView.IsStickyGroupHeader = true;
listView.AllowGroupExpandCollapse = true;
listView.ItemSpacing = new Thickness(0, 0, 5, 0);
listView.IsStickyHeader = true;
listView.GroupHeaderTemplate = new DataTemplate(() =>
{
    var grid = new Grid { BackgroundColor = Color.FromHex("#E4E4E4") };
    var label = new Label
    {
        FontAttributes = FontAttributes.Bold,
        FontSize = 22,
        VerticalOptions = LayoutOptions.Center,
        HorizontalOptions = LayoutOptions.Start,
        Margin = new Thickness(20, 0, 0, 0),
    };
    label.SetBinding(Label.TextProperty, new Binding("Key"));
    grid.Children.Add(label);
    return grid;
});
private Grid CreateItemTemplate()
{
    var gridView = new Grid();
    gridView.RowSpacing = 1;
    var row0 = new RowDefinition { Height = new GridLength(1, GridUnitType.Star) };
    var row1 = new RowDefinition { Height = 1 };
    gridView.RowDefinitions.Add(row0);
    gridView.RowDefinitions.Add(row1);
    var grid = new Grid();
    var column0 = new ColumnDefinition { Width = 50 };
    var column1 = new ColumnDefinition { Width = new GridLength(1, GridUnitType.Star) };
    var column2 = new ColumnDefinition { Width = 70 };
    grid.ColumnDefinitions.Add(column0);
    grid.ColumnDefinitions.Add(column1);
    grid.ColumnDefinitions.Add(column2);
    var contactImage = new Image();
    contactImage.SetBinding(Image.SourceProperty, new Binding("EmployeeImage"));
    contactImage.HeightRequest = 50;
    contactImage.VerticalOptions = LayoutOptions.Center;
    contactImage.HorizontalOptions = LayoutOptions.Center;
    var gridView = new Grid();
    gridView.RowSpacing = 1;
    gridView.Padding = new Thickness(10, 0, 0, 0);
    gridView.VerticalOptions = LayoutOptions.Center;
    var rowdefinition0 = new RowDefinition { Height = new GridLength(1, GridUnitType.Star) };
    var rowdefinition1 = new RowDefinition { Height = new GridLength(1, GridUnitType.Star) };
    gridView.RowDefinitions.Add(rowdefinition0);
    gridView.RowDefinitions.Add(rowdefinition1);
    var contactName = new Label();
    contactName.SetBinding(Label.TextProperty, new Binding("EmployeeName"));
    contactName.LineBreakMode = LineBreakMode.NoWrap;
    contactName.TextColor = Color.FromHex("#474747");
}

```



```
contactName.FontSize = 20;
var contactNumber = new Label();
contactNumber.FontAttributes = FontAttributes.None;
contactNumber.FontSize = 18;
contactNumber.LineBreakMode = LineBreakMode.NoWrap;
contactNumber.SetBinding(Label.TextProperty, new Binding("ContactNumber"));
contactNumber.TextColor = Color.FromHex("#474747");
var contactType = new Label();
contactType.Margin = new Thickness(5);
contactType.FontAttributes = FontAttributes.None;
contactType.LineBreakMode = LineBreakMode.NoWrap;
contactType.VerticalOptions = LayoutOptions.End;
contactType.VerticalTextAlignment = TextAlignment.End;
contactType.HorizontalOptions = LayoutOptions.End;
contactType.SetBinding(Label.TextProperty, new Binding("Designation"));
contactType.TextColor = Color.FromHex("#474747");
contactType.FontSize = 12;
var stackLayout = new StackLayout();
stackLayout.HeightRequest = 1;
stackLayout.BackgroundColor = Color.Gray;
gridView.Children.Add(contactName, 0, 0);
gridView.Children.Add(contactNumber, 0, 1);
grid.Children.Add(contactImage);
grid.Children.Add(gridView, 1, 0);
gridView.Children.Add(grid);
gridView.Children.Add(stackLayout, 0, 1);
return gridView;
}
}
```

You can download the entire source code of this demo [here](#).

**Informations****Status****Contacts**

## Object-Oriented Programming in C#

Object-oriented programming is the de facto programming paradigm

## C# Code Contracts

Code Contracts provide a way to convey code assumptions

## Machine Learning Using C#

You'll learn several different approaches to applying machine learning

## Neural Networks Using C#

Neural networks are an exciting field of software development

## Visual Studio Code

It is a powerful tool for editing code and serves for end-to-end programming

## Android Programming

It provides a useful overview of the Android application lifecycle

## iOS Succinctly

It is for developers looking to step into the frightening world of iPhone

*Header and footer in vertical mode when listview in horizontal mode*

By customizing the [SfListView](#), layout the header and footer items in **Vertical** mode with listview items in **Horizontal** mode.

### XML

```
<ContentPage >
<ContentPage.BindingContext>
<local:ListViewAutoFitContentViewModel x:Name="ViewModel"/>
</ContentPage.BindingContext>
<local:SfListViewExt x:Name="listView" ItemSize="180" GroupHeaderSize="80"
ItemSpacing="2" Orientation="Horizontal"
ItemsSource="{Binding BookInfo}"
SelectionMode="None"
IsStickyGroupHeader="False"
SelectionBackgroundColor="#d3d3d3">
<local:SfListViewExt.ItemTemplate>
<DataTemplate>
<Grid RowSpacing="0">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="1" />
</Grid.RowDefinitions>
<Grid Grid.Row="0" RowSpacing="0" Padding="5,10,5,5">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<StackLayout Padding="10,0,0,0" Orientation="Vertical" Grid.Column="1">
<Label Text="{Binding BookName}" FontAttributes="Bold" FontSize="18"
TextColor="#474747" VerticalTextAlignment="Center"/>
<Label Text="By Syncfusion Software" FontSize="12" TextColor="#474747"
VerticalTextAlignment="Center"/>
<Label Text="Published on: March 22, 2017" FontSize="12" TextColor="#474747"
VerticalTextAlignment="Center"/>
</StackLayout>
</Grid>
<StackLayout Padding="5,10,0,5" Grid.Row="1" Orientation="Vertical">
<Label Text="Description" FontSize="15" FontAttributes="Bold"
TextColor="#474747" VerticalTextAlignment="Center"/>
<Label Text="{Binding BookDescription}" TextColor="#474747" FontSize="14"
VerticalTextAlignment="Center"/>
</StackLayout>
<BoxView Grid.Row="2" HeightRequest="1" BackgroundColor="#474747" />
</Grid>
</DataTemplate>
</local:SfListViewExt.ItemTemplate>
</local:SfListViewExt>
</ContentPage>
```

### C#

```
public class SfListViewExt : SfListView
{
    Grid headerGrid;
```

```
Grid footerGrid;
public SfListViewExt()
{
    headerGrid = new Grid();
    headerGrid.BackgroundColor = Color.Teal;
    Label headerLabel = new Label();
    headerLabel.Text = "Header Item";
    headerLabel.FontSize = 18;
    headerLabel.TextColor = Color.White;
    headerLabel.HorizontalOptions = LayoutOptions.Center;
    headerLabel.VerticalOptions = LayoutOptions.Center;
    headerGrid.Children.Add(headerLabel);
    footerGrid = new Grid();
    footerGrid.BackgroundColor = Color.Teal;
    Label footerLabel = new Label();
    footerLabel.Text = "Footer Item";
    footerLabel.FontSize = 18;
    footerLabel.TextColor = Color.White;
    footerLabel.HorizontalOptions = LayoutOptions.Center;
    footerLabel.VerticalOptions = LayoutOptions.Center;
    footerGrid.Children.Add(footerLabel);
    this.Children.Add(headerGrid);
    this.Children.Add(footerGrid);
}
protected override void LayoutChildren(double x, double y, double width,
double height)
{
    headerGrid.Layout(new Rectangle(0, 0, width, 70));
    footerGrid.Layout(new Rectangle(0, height - 70, width, 70));
    base.LayoutChildren(0, 70, width, height);
}
}
```

Download the entire source code from GitHub [here](#).

Header Item		
<p><b>Windows Store Apps</b></p> <p>By Syncfusion Software</p> <p>Published on: March 2017</p> <p><b>Description</b></p> <p>Objective-C Succinctly is the only book you need for getting started with Objective-C—the primary language beneath all iPad, and iPhone</p>	<p><b>Visual Studio Code</b></p> <p>By Syncfusion Software</p> <p>Published on: March 22, 2017</p> <p><b>Description</b></p> <p>James McCaffrey's SciPy Programming Succinctly offers readers a quick, thorough grounding in knowledge of the Python open source extension SciPy. The SciPy library, accompanied by its interdependent NumPy, offers Python programmers advanced functions that work with arrays and matrices. Each section presents a complete demo program for programmers to experiment with, carefully chosen examples to best illustrate each function, and resources for further learning.</p>	<p><b>Entity Framework First</b></p> <p>By Syncfusion</p> <p>Published on: March 22, 2017</p> <p><b>Description</b></p> <p>Data Structures: A concise guide to lists, hash tables, priority queues, trees, and B-</p>
Footer Item		

## Selection

This section explains how to perform selection and its related operations in the SfListView.

### UI selection

The SfListView allows selecting items either programmatically or touch interactions by setting the [SfListView.SelectionMode](#) property value to other than **None**. The control has different selection modes to perform selection operations as listed as follows:

- **None**: Allows disabling selection.
- **Single**: Allows selecting single item only. When clicking on the selected item, selection not will not be cleared. This is the default value for [SfListView.SelectionMode](#).
- **SingleDeselect**: Allows selecting 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.












The SfListView allows selecting items on different gestures such as tap, double tap, and hold by setting the [SfListView.SelectionGesture](#). The default value for the [SfListView.SelectionGesture](#) is [TouchGesture.Tap](#).

### XML

```
<syncfusion:SfListView x:Name="listView"
    SelectionMode="Multiple"
    SelectionGesture="Hold"/>
```

### C#

```
listView.SelectionMode = SelectionMode.Multiple;
listView.SelectionGesture = TouchGesture.Hold;
```

	Adventure of a lifetime Coldplay	50.3KB
	Blue moon of Kentucky Bill Monroe	158.0KB
	I don't care if tomorrow never comes Hank Williams & George Jones	397.0KB
	You are the first, my last, my everything Barry White	84.3KB
	Words don't come easy to me F. R. David	364.2KB
	Everybody's free to wear sunscreen Baz Luhrmann	135.4KB
	Before the next teardrop falls Freddie Fender	424.1KB
	You've lost that lovin' feeling Righteous Brothers	100.4KB
	Underneath your clothes Shakira	228.0KB
	Try to remember Andy Williams	315.2KB
	The hanging tree James Newton Howard ft. Jeni	210.1KB

### Programmatic selection

When the [SfListView.SelectionMode](#) is other than `None`, the item or items in the SfListView can be selected from the code by setting the [SfListView.SelectedItem](#), or adding items to the [SfListView.SelectedItems](#) property based on the `SfListView.SelectionMode`.

When the selection mode is `Single`, programmatically select an item by setting the underlying object to the [SfListView.SelectedItem](#) property.

#### C#

```
//Perform selection using selected item  
listView.SelectedItem = viewModel.Items[5];
```

When the selection mode is `Multiple`, programmatically select more than one item by adding the underlying object to the [SfListView.SelectedItems](#) property.

#### C#

```
//Perform multiple selection using selected items  
listView.SelectedItems.Add (viewModel.Items [4]);  
listView.SelectedItems.Add (viewModel.Items [5]);
```

All items of the SfListView can be selected using the [SelectAll](#) method.

#### C#

```
listView.SelectAll();
```

---

**Note:** When programmatically select an item then the selection related [events](#) will not be triggered. It triggers only on UI interactions.

---

However, get the notification from the SelectedItems collection changed event which will be triggered when add an item at runtime.

### Selected items

#### *Get selected items*

The SfListView gets all the selected items through the [SfListView.SelectedItems](#) property and gets the single item by using the [SfListView.SelectedItem](#) property.

#### *Clear selected items*

The selected items can be cleared by calling the [SelectedItem.Clear\(\)](#) method.

### C#

```
listView.SelectedItems.Clear();
```

#### *CurrentItem vs SelectedItem*

The SfListView gets the selected item by using the [SfListView.SelectedItem](#) and [SfListView.CurrentItem](#) properties. Both [SfListView.SelectedItem](#) and [SfListView.CurrentItem](#) returns the same data object when selecting single item. When selecting more than one item, the [SfListView.SelectedItem](#) property returns the first selected item, and the [SfListView.CurrentItem](#) property returns the last selected item.

---

**Warning:** If you select an item when [SfListView.SelectionMode](#) is none or if you select multiple items when [SfListView.SelectionMode](#) is single, exception will be thrown.

---

### Selected item customization

The SfListView supports customizing the selection background color for the selected items by using the [SfListView.SelectedItemTemplate](#) if the background color is set to view loaded in the [SfListView.ItemTemplate](#).

### XML

```
<ContentPage xmlns:syncfusion="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView">
    <syncfusion:SfListView.SelectedItemTemplate>
      <DataTemplate>
        <Grid x:Name="grid" BackgroundColor="RoyalBlue">
          <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="1" />
          </Grid.RowDefinitions>
          <Label Text="{Binding SongTitle}" />
          <Label Text="{Binding SongAuthor}" Grid.Row="1"/>
          <Frame Grid.Row="2" HasShadow="True" HeightRequest="1"/>
        </Grid>
      </DataTemplate>
    </syncfusion:SfListView.SelectedItemTemplate>
  </syncfusion:SfListView>
</ContentPage>
```



**C#**

```
listView.SelectedItemTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    grid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
        GridUnitType.Star) });
    grid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
        GridUnitType.Star) });
    grid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1) });
    grid.BackgroundColor = Color.RoyalBlue;
    var songTitle = new Label();
    songTitle.SetBinding(Label.TextProperty, new Binding("SongTitle"));
    var songAuthor = new Label();
    songAuthor.SetBinding(Label.TextProperty, new Binding("songAuthor"));
    var frame = new Frame()
    {
        HeightRequest = 1,
        HasShadow = true,
    };
    grid.Children.Add(songTitle);
    grid.Children.Add(songAuthor, 0, 1);
    grid.Children.Add(frame, 0, 2);
    return grid;
});
```

Download the entire sample from GitHub [here](#).



*Show checked circle on selected items*

To customize the appearance of the selected item or items by using the appearance of [SfListView.SelectedItemTemplate](#). The following customizations should give an idea to customize the appearance of selected items in the control.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView">
<syncfusion:SfListView.SelectedItemTemplate>
<DataTemplate>
<Grid x:Name="grid">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="40" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<Image Source="{Binding SongThumbnail}" />
<Grid Grid.Column="1">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Label Text="{Binding SongTitle}" />
<Grid Grid.Row="1">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
```

```

</Grid.ColumnDefinitions>
<Label Text="{Binding SongAuthor}" />
<Label Grid.Column="1" Text="{Binding SongSize}"/>
</Grid>
</Grid>
<Image Grid.Column="2" x:Name="selectionImage" IsVisible="True"
Source="Selected.png"/>
</Grid>
</DataTemplate>
</syncfusion:SfListView.SelectedItemTemplate>
</syncfusion:SfListView>
</ContentPage>

```

## C#












```

listView.SelectedItemTemplate = new DataTemplate(() =>
{
    var grid1 = new Grid();
    grid1.ColumnDefinitions.Add(new ColumnDefinition { Width = new
    GridLength(40) });
    grid1.ColumnDefinitions.Add(new ColumnDefinition { Width = new GridLength(1,
    GridUnitType.Star) });
    grid1.ColumnDefinitions.Add(new ColumnDefinition { Width = new GridLength(1,
    GridUnitType.Auto) });
    var songThumbnail = new Image();
    songThumbnail.SetBinding(Image.SourceProperty, new
    Binding("SongThumbnail"));
    var grid2 = new Grid();
    grid2.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
    GridUnitType.Star) });
    grid2.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
    GridUnitType.Star) });
    grid2.Padding = new Thickness(15, 0, 0, 0);
    var songTitle = new Label();
    songTitle.SetBinding(Label.TextProperty, new Binding("SongTitle"));
    var grid3 = new Grid();
    grid3.ColumnDefinitions.Add(new ColumnDefinition { Width = new GridLength(1,
    GridUnitType.Star) });
    grid3.ColumnDefinitions.Add(new ColumnDefinition { Width = new GridLength(1,
    GridUnitType.Star) });
    var songAuthor = new Label();
    songAuthor.SetBinding(Label.TextProperty, new Binding("songAuthor"));
    var songSize = new Label();
    songSize.SetBinding(Label.TextProperty, new Binding("SongSize"));
    grid3.Children.Add(songAuthor);
    grid3.Children.Add(songSize, 1, 0);
    grid2.Children.Add(songTitle);
    grid2.Children.Add(grid3, 0, 1);
    var selectionImage = new Image()
    {
        IsVisible = true,
        Source = "Selected.png"
    };
    grid1.Children.Add(songThumbnail);
    grid1.Children.Add(grid2, 1, 0);
    grid1.Children.Add(selectionImage, 2, 0);
}

```

```
return grid1;
});
```

Download the entire source code from GitHub [here](#).

	Adventure of a lifetime Coldplay	238.4KB	<input type="radio"/>
	Blue moon of Kentucky Bill Monroe	183.4KB	<input checked="" type="radio"/>
	I don't care if tomorrow never come Hank Williams & George Jr	217.1KB	<input type="radio"/>
	You are the first, my last, my everyt Barry White	277.4KB	<input checked="" type="radio"/>
	Words don't come easy to me F. R. David	477.4KB	<input type="radio"/>
	Everybody's free to wear sunscreen Baz Luhrmann	199.0KB	<input checked="" type="radio"/>
	Before the next teardrop falls Freddy Fender	468.3KB	<input type="radio"/>
	You've lost that lovin' feeling Righteous Brothers	360.1KB	<input checked="" type="radio"/>
	Underneath your clothes Shakira	130.2KB	<input type="radio"/>
	Try to remember Andy Williams	526.3KB	<input type="radio"/>
	The hanging tree James Newton Howard ft.	514.1KB	<input type="radio"/>

### Selected item style

#### *Selection background*












The SfListView allows changing the selection background color for the selected items by using the [SfListView.SelectionBackgroundColor](#) property. You can also change the selection background color at runtime.

#### **C#**

```
<syncfusion:SfListView x:Name="listView"
SelectionBackgroundColor="Khaki"/>
```

#### **C#**

```
listView.SelectionBackgroundColor = Color.Khaki;
```

	Adventure of a lifetime Coldplay	256.3KB
	Blue moon of Kentucky Bill Monroe	181.2KB
	I don't care if tomorrow never comes Hank Williams & George Jones	190.2KB
	You are the first, my last, my everything Barry White	341.1KB
	Words don't come easy to me F. R. David	406.1KB
	Everybody's free to wear sunscreen Baz Luhrmann	250.3KB
	Before the next teardrop falls Freddie Fender	571.1KB
	You've lost that lovin' feeling Righteous Brothers	489.1KB
	Underneath your clothes Shakira	594.2KB
	Try to remember Andy Williams	481.4KB
	The hanging tree James Newton Howard ft. Jeni	546.4KB

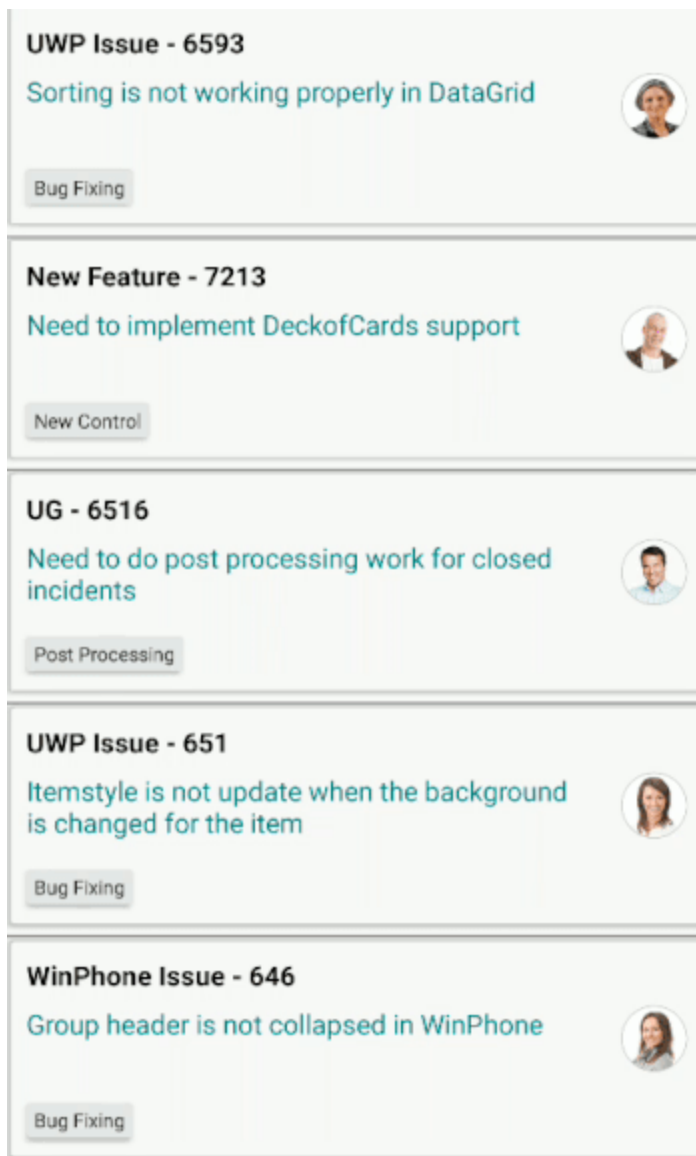
### *Programmatic animation*

The SfListView allows programmatic animation in selection at runtime by using the virtual method [AnimateSelectedItem](#) of [SelectionController](#) class.

### **C#**

```
listView.SelectionController = new SelectionControllerExt(listView);  
public class SelectionControllerExt : SelectionController  
{  
    public SelectionControllerExt(SfListView listView) : base(listView)  
    {  
    }  
    protected override void AnimateSelectedItem(ListViewItem  
selectedListViewItem)  
    {  
        base.AnimateSelectedItem(selectedListViewItem);  
        selectedListViewItem.Opacity = 0;  
        selectedListViewItem.FadeTo(1, 3000, Easing.SinInOut);  
    }  
}
```

Download the entire source code from GitHub [here](#).



## Events

### *SelectionChanging event*

The [SelectionChanging](#) event is raised while selecting an item at the execution time.

[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.

The `SelectionChanging` event is used for the following use case:

- Disable the selection of the particular item based on the underlying data.

### C#

```
listView.SelectionChanging += ListView_SelectionChanging;  
private void ListView_SelectionChanging(object sender,  
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 SfListView. [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.

The [SelectionChanged](#) event used for the following use cases:

- Clears all the selected item.
- Removes the particular selected item.
- Gets the index of the selected item.

### C#

```
listView.SelectionChanged += ListView_OnSelectionChanged;  
private void ListView_OnSelectionChanged(object sender,  
ItemSelectionChangedEventArgs e)  
{  
listView.SelectedItems.Clear();  
}
```

---

**Note:** [SelectionChanging](#) and [SelectionChanged](#) events will be triggered only on UI interactions.

---

### Key navigation

The [AllowKeyNavigation](#) property enables navigation through keyboard buttons. When the [AllowKeyNavigation](#) property is [true](#), navigation gets enabled. Otherwise, set to [false](#). Behavior of key navigation in UWP and macOS are explained as follows :

- When the [SfListView.SelectionMode](#) is [Single](#), the selected item is highlighted with [FocusBorderColor](#) around the item while key navigation.
- When the [SelectionMode](#) is [SingleDeSelect](#) or [Multiple](#), the [FocusBorderColor](#) will set to the [CurrentItem](#) only on key navigation.

- If focusable elements i.e. `Entry`, `SearchBar`, etc. are loaded in the page, the `Focus` will not be changed either to other elements in the view or to the next `ListViewItem` when `Tab` or `Shift+Tab` key is pressed in the `LinearLayout`.
- In the `GridLayout` with span count greater than 1, the `FocusBorderColor` will navigate to the next or previous `ListViewItem` when pressing `Tab` or `Shift+Tab` key.
- In macOS, need to move the focus manually to perform key navigation.

#### *FocusBorderColor*

`FocusBorderColor` used to set the border color for the current focused item. For Android and iOS platform, the default color is `Color.Transparent` and for macOS and UWP platform, the default color is `Color.FromRgb(76, 161, 254)`.

#### *FocusBorderThickness*

`FocusBorderThickness` used to set the border thickness for the current focused item. For Android and iOS platform, the default thickness is 0 and for macOS and UWP platform, the default thickness is 1.

#### MacOS support

##### *Known issues*

- If `SfListView` flings with more `inertia` or the scrollbar reaches either the bottom or top of the view, the listview items will not layout properly. Since, it occurs in simple custom control and `Xamarin.Forms.ListView`. So, a defect report is logged to `Bugzilla` team. Find the bug report: <https://github.com/xamarin/Xamarin.Forms/issues/2403>
- When `Grouping` is enabled, the listview items and group header items will not layout properly while scrolling the `SfListView`.
- Application will get crash when `Linker` option is set as `Link All`.
- If an `Image` is loaded in the `ItemTemplate` property, it is necessary to define a definite size for the `Image`.

#### *Not yet implemented*

The following features are not yet implemented due to some limitations in the `Xamarin.Forms.macOS` platform:

- `ScrollStateChanged` event
- `Pull-To-Refresh`

#### How to

##### *Disable selection on particular item*

The selection of a particular set of items can be disabled based on the `SfListView.SelectedItems` of the underlying collections.

#### **C#**

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
}
```



```
private void listView_SelectionChanging(object sender,
Syncfusion.ListView.XForms.ItemSelectionChangingEventArgs e)
{
    if (e.AddedItems.Count > 0 && (e.AddedItems[0] as Contacts).Category ==
        "Non-Selectable items")
        e.Cancel = true;
}
```

Download the entire sample from GitHub [here](#).

*Automatically scroll to bring a selected item into the view*

To bring the [SfListView.SelectedItem](#) automatically into the view when it changed at runtime by calling the [ScrollToRowIndex](#) method.

In linear layout, you can get the row index of [SfListView.SelectedItem](#) and resolve if header and group header are used.

### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        listView.PropertyChanged += listView_PropertyChanged;
    }
    private void listView_PropertyChanged(object sender,
        System.ComponentModel.PropertyChangedEventArgs e)
    {
        if (e.PropertyName == "SelectedItem")
        {
            var selectedItemIndex =
                listView.DataSource.DisplayItems.IndexOf(listView.SelectedItem);
            selectedItemIndex += (listView.HeaderTemplate != null &&
                !listView.IsStickyHeader || !listView.IsStickyGroupHeader) ? 1 : 0;
            selectedItemIndex -= (listView.GroupHeaderTemplate != null &&
                listView.IsStickyGroupHeader) ? 1 : 0;
            (listView.LayoutManager as
                LinearLayout).ScrollToRowIndex(selectedItemIndex);
        }
    }
}
```

Download the entire sample from GitHub [here](#).

*Gets the index of selected item*

When performing selection, you can get the index of the selected item by using the [SelectionChanged](#) event from the [DataSource.DisplayItems](#).

### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
```

```
InitializeComponent();
listView.SelectionChanged += ListView_SelectionChanged;
}
private void ListView_SelectionChanged(object sender,
ItemSelectionChangedEventArgs e)
{
var items = e.AddedItems;
var index = listView.DataSource.DisplayItems.IndexOf(items[0]);
entry.Text = index.ToString();
}
}
```

Download the entire sample from GitHub [here](#).

#### *Display selection when ItemTemplate contains image*

When [ItemTemplate](#) contains only image, then the selection color will not be visible in the view when select an image. To see selection, add any layout such as Grid or StackLayout above the image, and set margin or padding to it.

#### **XML**

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView
ItemsSource="{Binding BookInfo}"
ItemSize="100">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Grid Margin="10">
<Image Source="{Binding Image}" Aspect="Fill"/>
</Grid>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</ContentPage>
```

#### **C#**

```
public partial class MainPage : ContentPage
{
SfListView listView;
public MainPage()
{
InitializeComponent();
listView = new SfListView();
listView.ItemSize = 100;
listView.ItemsSource = viewModel.BookInfo;
listView.ItemTemplate = new DataTemplate(() =>
{
var grid = new Grid() { Margin = 10 };
var image = new Image() { Aspect = Aspect.Fill};
image.SetBinding(Image.SourceProperty, new Binding("Image"));
grid.Children.Add(image);
return grid;
});
this.Content = listView;
}
```

```
}
}
```

## 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 ListView instead of [ItemTemplate](#).
- When the ListView contains duplicated items in the collection, only the first item whose instance was created initially will be selected or deselected.

## Swiping

### Overview

The SfListView allows swiping items to do custom actions such as deleting the data, adding the data, editing the data, etc. To enable swiping, set the [SfListView.AllowSwiping](#) property to `true`. Swipe views are displayed when swiping from left to right or right to left (for horizontal orientation, top to bottom or bottom to top) on the item.

It provides customizable swipe templates for swiping on left and right sides. You can restrict the layout of swipe view up to a certain position when swiping the item by setting the [SfListView.SwipeThreshold](#) property. You can set size of the swipe views by setting the [SfListView.SwipeOffset](#) property.

**Note:** When [SfListView.AutoFitMode](#) is `AutoFitMode.Height` for main listview, the height of inner listview will change while scrolling the view and items will be refreshed.

**Note:** When tap a swiped item, the [SelectionChanging](#) and [SelectionChanged](#) events will not occur since the swiped item is reset at this time.

### Assigning left and right swipe templates

The User Interface (UI) for swiping can be customized by using swipe templates [SfListView.LeftSwipeTemplate](#) when swiping towards right and [SfListView.RightSwipeTemplate](#) when swiping towards left. The contents inside the swipe template are arranged based on the offset values when swiping an item. You can reset the swiping item or swiped item by calling the [SfListView.ResetSwipe](#) method.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView" AllowSwiping="True">
    <syncfusion:SfListView.LeftSwipeTemplate>
      <DataTemplate x:Name="LeftSwipeTemplate">
        <Grid>
          <Grid BackgroundColor="#009EDA" HorizontalOptions="Fill"
            VerticalOptions="Fill" Grid.Column="0">
            <Grid VerticalOptions="Center" HorizontalOptions="Center">
              <Image Grid.Column="0"
                Grid.Row="0"
                BackgroundColor="Transparent"
                HeightRequest="35"
                WidthRequest="35"/>
```

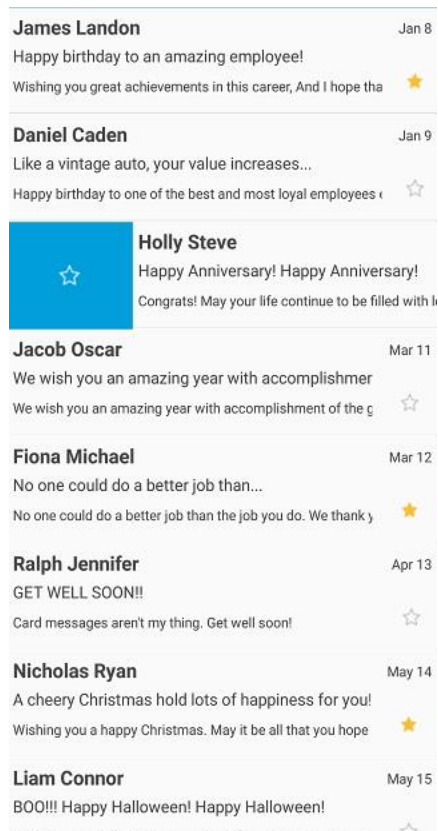
```
Source="Favorites.png" />
</Grid>
</Grid>
</Grid>
</DataTemplate>
</syncfusion:SfListView.LeftSwipeTemplate>
</syncfusion:SfListView>
</ContentPage>
```

## C#

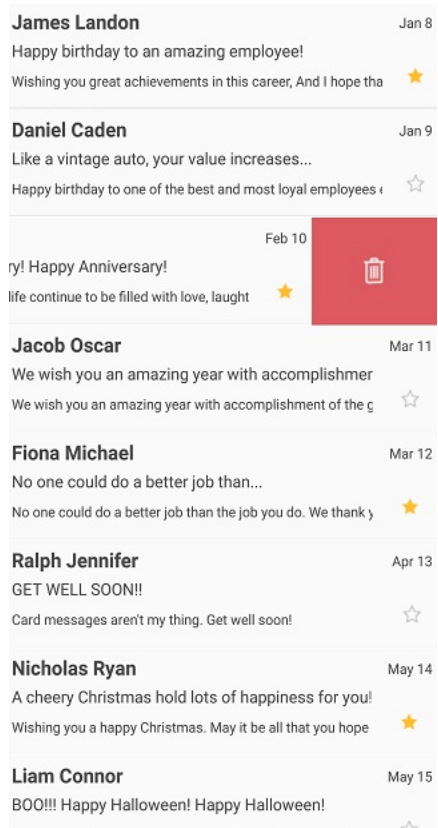
```
//Defining left swipe template
listView.LeftSwipeTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var grid1 = new Grid() { BackgroundColor = Color.FromHex("#009EDA"),
        HorizontalOptions = LayoutOptions.Fill,
        VerticalOptions = LayoutOptions.Fill };
    var favoriteGrid = new Grid() { HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.Center };
    var favoriteImage = new Image() { BackgroundColor = Color.Transparent,
        HeightRequest = 35, WidthRequest = 35 };
    favoriteImage.Source =
        ImageSource.FromResource("Swiping.Images.Favorites.png");
    favoriteGrid.Children.Add(favoriteImage);
    grid1.Children.Add(favoriteGrid);
    grid.Children.Add(grid1);
    return grid;
});
```

**Note:** Similarly, the UI for swiping towards left can be customized by using the [SfListView.RightSwipeTemplate](#).

**Note:** Swipe Template is mandatory to perform swiping in the SfListView.



Download the entire source code from GitHub [here](#).



**Note:** To customize the appearance of each swipe item with different templates based on specific constraints by using the [DataTemplateSelector](#).

### Working with multiple views in swipe template

The swipe templates allows customizing with custom actions such as deleting the data, adding the data, editing the data, etc. by loading multiple views.

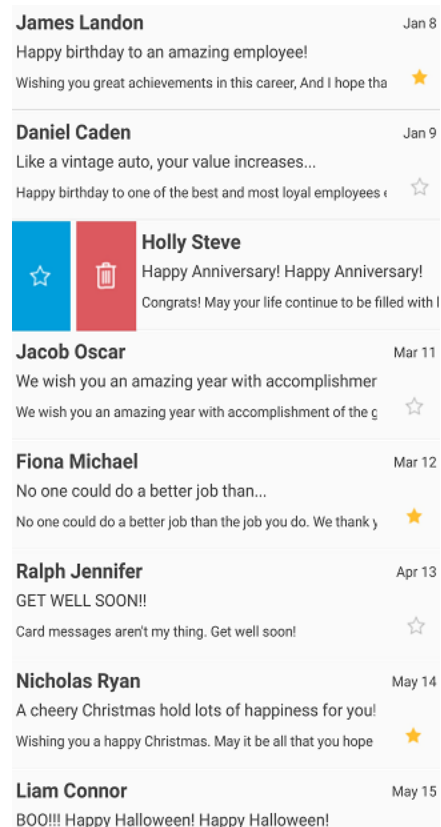
#### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView x:Name="listView">
    <syncfusion:SfListView.LeftSwipeTemplate>
      <DataTemplate x:Name="LeftSwipeTemplate">
        <Grid>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
          </Grid.ColumnDefinitions>
          <Grid BackgroundColor="#009EDA" HorizontalOptions="Fill"
VerticalOptions="Fill" Grid.Column="0">
            <Grid VerticalOptions="Center" HorizontalOptions="Center">
              <Image Grid.Column="0"
Grid.Row="0"
BackgroundColor="Transparent"
HeightRequest="35"
WidthRequest="35"
BindingContextChanged="leftImage_BindingContextChanged"
Source="Favorites.png" />
            </Grid>
          </Grid>
          <Grid BackgroundColor="#DC595F" HorizontalOptions="Fill"
VerticalOptions="Fill" Grid.Column="1">
            <Grid VerticalOptions="Center" HorizontalOptions="Center">
              <Image Grid.Column="0"
Grid.Row="0"
HeightRequest="35"
WidthRequest="35"
BackgroundColor="Transparent"
BindingContextChanged="rightImage_BindingContextChanged"
Source="Delete.png" />
            </Grid>
          </Grid>
        </Grid>
      </DataTemplate>
    </syncfusion:SfListView.LeftSwipeTemplate>
  </syncfusion:SfListView>
</ContentPage>
```

#### C#

```
listView.LeftSwipeTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var grid1 = new Grid()
    {
```

```
BackgroundColor = Color.FromHex("#009EDA"),
HorizontalOptions = LayoutOptions.Fill,
VerticalOptions = LayoutOptions.Fill
};
var favoriteGrid = new Grid() { HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center };
var favoriteImage = new Image() { BackgroundColor = Color.Transparent,
HeightRequest = 35, WidthRequest = 35 };
favoriteImage.Source =
ImageSource.FromResource("Swiping.Images.Favorites.png");
favoriteImage.BindingContextChanged += FavoriteImage_BindingContextChanged;
favoriteGrid.Children.Add(favoriteImage);
grid1.Children.Add(favoriteGrid);
var grid2 = new Grid()
{
BackgroundColor = Color.FromHex("#DC595F"),
HorizontalOptions = LayoutOptions.Fill,
VerticalOptions = LayoutOptions.Fill
};
var deleteGrid = new Grid() { HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center };
var deleteImage = new Image() { BackgroundColor = Color.Transparent,
HeightRequest = 35, WidthRequest = 35 };
deleteImage.Source = ImageSource.FromResource("Swiping.Images.Delete.png");
deleteImage.BindingContextChanged += DeleteImage_BindingContextChanged;
deleteGrid.Children.Add(deleteImage);
grid1.Children.Add(deleteGrid);
grid.Children.Add(grid1);
grid.Children.Add(grid2, 1, 0);
return grid;
});
```



To delete the item when **Delete** image is tapped and setting favorites to item when **Favorites** image is tapped, follow the code example.

### C#

```
Image leftImage;
Image rightImage;
int itemIndex = -1;
private void SetFavorites()
{
    if (itemIndex >= 0)
    {
        var item = viewModel.InboxInfo[itemIndex];
        item.IsFavorite = !item.IsFavorite;
    }
    this.listView.ResetSwipe();
}
private void Delete()
{
    if (itemIndex >= 0)
    {
        viewModel.InboxInfo.RemoveAt(itemIndex);
        this.listView.ResetSwipe();
    }
}
private void ListView_SwipeStarted(object sender, SwipeStartedEventArgs e)
{
    itemIndex = -1;
}
private void ListView_SwipeEnded(object sender, SwipeEndedEventArgs e)
{
}
```



```

itemIndex = e.ItemIndex;
}
private void leftImage_BindingContextChanged(object sender, EventArgs e)
{
    if (leftImage == null)
    {
        leftImage = sender as Image;
        (leftImage.Parent as View).GestureRecognizers.Add(new TapGestureRecognizer()
        { Command = new Command(SetFavorites) });
        leftImage.Source = ImageSource.FromResource("Swiping.Images.Favorites.png");
    }
}
private void rightImage_BindingContextChanged(object sender, EventArgs e)
{
    if (rightImage == null)
    {
        rightImage = sender as Image;
        (rightImage.Parent as View).GestureRecognizers.Add(new
        TapGestureRecognizer() { Command = new Command>Delete) });
        rightImage.Source = ImageSource.FromResource("Swiping.Images.Delete.png");
    }
}

```

### Performing swipe delete operation

To delete an item in view while swiping the item from one extent to other by using [SfListView.SwipeEnded](#) event. By setting the [SfListView.SwipeOffset](#) value to the view size to swipe the item upto end of the item.

### XML

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
AllowSwiping="True" SelectionMode="None"
SwipeOffset="360" SwipeThreshold="30"
SwipeStarted="ListView_SwipeStarted"
SwipeEnded="ListView_SwipeEnded"
Swiping="ListView_Swiping">
<syncfusion:SfListView.RightSwipeTemplate>
<DataTemplate x:Name="RightSwipeTemplate">
<Grid BackgroundColor="#DC595F" HorizontalOptions="Fill"
VerticalOptions="Fill">
<Grid VerticalOptions="Center" HorizontalOptions="Center">
<Image Grid.Column="0"
Grid.Row="0"
HeightRequest="35"
WidthRequest="35"
BackgroundColor="Transparent"
Source="Delete.png" />
</Grid>
</Grid>
</DataTemplate>
</syncfusion:SfListView.RightSwipeTemplate>
</syncfusion:SfListView>
</ContentPage>

```

**C#**

```

listView.AllowSwiping = true;
listView.SelectionMode = SelectionMode.None;
listView.SwipeOffset = 360;
listView.SwipeThreshold = 30;
listView.SwipeStarted += ListView_SwipeStarted;
listView.SwipeEnded += ListView_SwipeEnded;
listView.Swiping += ListView_Swiping;
listView.RightSwipeTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var grid1 = new Grid()
    {
        BackgroundColor = Color.FromHex("#DC595F"),
        HorizontalOptions = LayoutOptions.Fill,
        VerticalOptions = LayoutOptions.Fill
    };
    var deleteGrid = new Grid() { HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.Center };
    var deleteImage = new Image() { BackgroundColor = Color.Transparent,
        HeightRequest = 35, WidthRequest = 35 };
    deleteImage.Source = ImageSource.FromResource("Swiping.Images.Delete.png");
    deleteGrid.Children.Add(deleteImage);
    grid1.Children.Add(deleteGrid);
    grid.Children.Add(grid1);
    return grid;
});
private void ListView_SwipeEnded(object sender, SwipeEndedEventArgs e)
{
    if (e.SwipeOffset >= 360)
    {
        viewModel.InboxInfo.RemoveAt(e.ItemIndex);
        listView.ResetSwipe();
    }
}

```

## Events

*SwipeStarted Event*

The [SfListView.SwipeStarted](#) event is raised when the swipe offset changes from its initial value.

The **SwipeStarted** event provides the following properties in their arguments:

- [ItemIndex](#): Defines the swiping item index.
- [ItemData](#): Defines the underlying data associated with the swiped item. as its arguments.
- [SwipeDirection](#): Defines the swipe direction of the swiped item.

The **SwipeStarted** event is used for the following use case:

- To cancel the swipe action for a particular item by setting the Cancel property of the [SwipeStartedEventArgs](#).

**XML**

```
<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding InboxInfo}"  
SwipeStarted="ListView_SwipeStarted" />
```

### C#

```
listView.SwipeStarted += ListView_SwipeStarted;
```

### C#

```
private void ListView_SwipeStarted(object sender, SwipeStartedEventArgs e)  
{  
    if (e.ItemIndex == 1)  
        e.Cancel = true;  
}
```

### Swiping Event

The [SfListView.Swiping](#) event is raised while swiping an item is in progress. This event is triggered with [SwipingEventArgs](#).

The [Swiping](#) event provides the following properties in their arguments:

- [ItemIndex](#): Defines the swiping item index.
- [ItemData](#): Defines the underlying data associated with the swiped item as its arguments.
- [SwipeDirection](#): Defines the swipe direction of the swiped item.
- [SwipeOffset](#): Defines the current swipe offset of the item being swiped.
- [Handled](#): Defines that if it is true, current swipe offset value remains same for the swiped item until the [SwipeEnded](#) event is raised.

The [Swiping](#) event used for the following use cases:

- To maintain the current offset value for the swiped item till the [SwipeEnded](#) event gets called.
- To hold the swipe view being swiping by setting the [Handled](#) property if swipe offset meet certain position.

### XML

```
<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding InboxInfo}"  
Swiping="ListView_Swiping" />
```

### C#

```
listView.Swiping += ListView_Swiping;
```

### C#

```
private void ListView_Swiping(object sender, SwipingEventArgs e)  
{  
    if (e.ItemIndex == 1 && e.SwipeOffset > 70)  
        e.Handled = true;  
}
```

### SwipeEnded Event

The [SfListView.SwipeEnded](#) event is fired when completing the swipe action. This event is triggered with [SwipeEndedEventArgs](#).

The **SwipeEnded** event provides the following properties in their arguments:

- [ItemIndex](#): Defines the swiping item index.
- [ItemData](#): Defines the underlying data associated with the swiped item as its arguments.
- [SwipeDirection](#): Defines the swipe direction of the swiped item.
- [SwipeOffset](#): Defines the current swipe offset of the item being swiped.

The **SwipeEnded** event is used for the following use cases:

- To insert the data or edit the data after swiped.
- To delete the item from view after swiping it to certain extent.
- To reset the swipe view automatically for the swiped item.

### XML

```
<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding InboxInfo}"  
SwipeEnded="ListView_SwipeEnded" />
```

### C#

```
listView.SwipeEnded += ListView_SwipeEnded;
```

### C#

```
private void ListView_SwipeEnded(object sender, SwipeEndedEventArgs e)  
{  
    if (e.SwipeOffset > 70)  
        listView.ResetSwipe();  
}
```

### SwipeReset Event

The [SfListView.SwipeReset](#) event is fired when swiping gets reset. The SwipeReset action can be canceled by setting the Cancel property of the ResetSwipeEventArgs to true. This event is triggered with [ResetSwipeEventArgs](#).

**SwipeReset** event provides the following properties in their arguments:

- [ItemIndex](#): Defines the swiping item index.
- [ItemData](#): Defines the underlying data associated with the swiped item as its arguments.
- [SwipeOffset](#): Defines the current swipe offset of the item being swiped.

The **SwipeReset** event used for the following use case:

- To skip the reset operation for a swiped item.

### XML

```
<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding InboxInfo}"  
SwipeReset="ListView_SwipeReset" />
```

### C#

```
listView.SwipeReset += ListView_SwipeReset;
```

### C#

```
private void ListView_SwipeReset(object sender, ResetSwipeEventArgs e)  
{  
    if (e.ItemIndex == 1)  
        e.Cancel = true;  
}
```

### Limitations

When ListView is loaded in CarouselView with [SfListView.AllowSwiping](#) as false, it behaves in UWP platform as follows:

- While performing first swipe on the view, it will be handled by ScrollView to ensure whether scrolling is happened or not. If not means the manipulation to parent cannot be passed immediately due to UWP platform behavior. The second swipe will be listened by CarouselView, and the view gets swiped. This is the behavior of the SfListView.

When ListView is loaded in CarouselView with [AllowSwiping](#) as true, it behaves as follows:

- When swiping in iOS, suddenly carousel swipe happened. To swipe ListViewItem, touch and hold the item for some fraction of seconds (0.25 - 0.5 seconds) and then swipe.
- When swiping any Item, the SfListView handles the touch and swipe the ListViewItem.
- After swiping on ListViewItem, SwipeView will load along with it. If you swipe SwipeView element, Carousel view is swiped. Or else swipe on ListViewItem, control handles touching and swiping the item as usual.
- If you swipe header, footer, or group header elements, Carousel view will swipe in Android platform. But in UWP, first swipe on those elements will be handled by SfListView itself, because manipulation to parent cannot be passed immediately. The second swipe will be listened by CarouselView.

When ListView is loaded in MasterDetailPage with [AllowSwiping](#) as true, it behaves as follows:

- In iOS platform, when swiping a ListViewItem, touch and hold the item for some fraction of seconds (0.25 - 0.5 seconds) and then swipe.

When ListView [ItemTemplate](#) contains button with [AllowSwiping](#) as true, it behaves as follows:

- While swiping in Android and UWP, button click event executes after swiping.
- While swiping in iOS, swipe action does not happened when clicking and swiping a button.

- Swiping will be reset whenever the listview is refreshed like scrolling or any size change is made.

How to reset swipe view automatically?

Swiped item can be reset by defining the [SfListView.SwipeOffset](#) argument of [SfListView.SwipeEnded](#) event to 0 when the swiping action is completed.

**C#**

```
private void ListView_SwipeEnded(object sender, SwipeEndedEventArgs e)
{
    if (e.SwipeOffset > 70)
        e.SwipeOffset = 0;
}
```

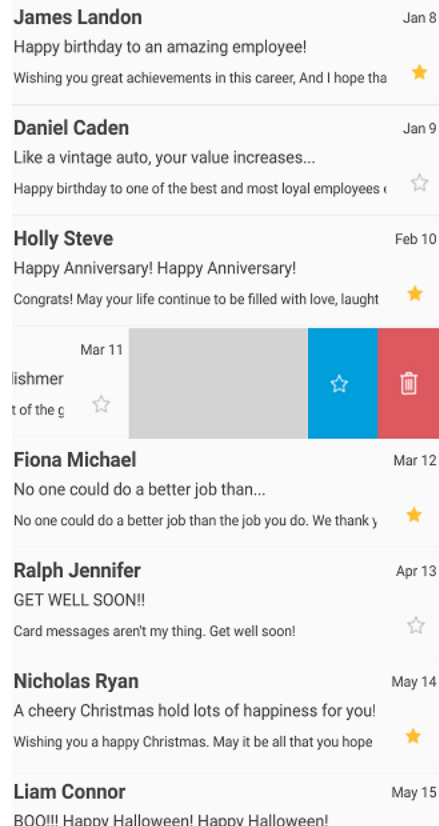
How to swipe an item indefinitely?

To swipe an item indefinitely, set the [SfListView.SwipeOffset](#) property by considering the width or height of the SfListView with [SfListView.Orientation](#) accordingly.

**C#**

```
ListView.PropertyChanged += ListView_PropertyChanged;
private void ListView_PropertyChanged(object sender,
PropertyChangedEventArgs e)
{
    if (e.PropertyName == "Width" && ListView.Orientation ==
Orientation.Vertical && ListView.SwipeOffset != ListView.Width)
        ListView.SwipeOffset = ListView.Width;
    else if (e.PropertyName == "Height" && ListView.Orientation ==
Orientation.Horizontal && ListView.SwipeOffset != ListView.Height)
        ListView.SwipeOffset = ListView.Height;
}
```

Download the entire source code from GitHub [here](#)



### How to edit data by swiping?

The SfListView allows editing the item data using either [SfListView.RightSwipeTemplate](#) or [SfListView.LeftSwipeTemplate](#) by loading edit view into the respective template after swiping the item.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
  <syncfusion:SfListView >
    <syncfusion:SfListView.RightSwipeTemplate>
      <DataTemplate x:Name="RightSwipeTemplate">
        <Grid BackgroundColor="#DC595F" HorizontalOptions="Fill"
          VerticalOptions="Fill">
          <Grid>
            <Label Grid.Row="0"
              HeightRequest="50"
              WidthRequest="50"
              BackgroundColor="Transparent"
              Text="EditItem">
            <Grid.GestureRecognizers>
              <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped"/>
            </Grid.GestureRecognizers>
          </Label>
        </Grid>
      </DataTemplate>
    </syncfusion:SfListView.RightSwipeTemplate>
  </syncfusion:SfListView>
</ContentPage>
```

**C#**

```
listView.RightSwipeTemplate = new DataTemplate(() =>
{
    var grid = new Grid()
    {
        BackgroundColor = Color.FromHex("#009EDA"),
        HorizontalOptions = LayoutOptions.Fill,
        VerticalOptions = LayoutOptions.Fill
    };
    var grid1 = new Grid();
    TapGestureRecognizer tapped = new TapGestureRecognizer();
    grid1.GestureRecognizers.Add(tapped);
    tapped.Tapped += Grid_Tapped;
    var label = new Label()
    {
        HeightRequest = 50,
        WidthRequest = 50,
        BackgroundColor = Color.Transparent,
        Text = "EditItem"
    };
    grid1.Children.Add(label);
    grid.Children.Add(grid1);
    return grid;
});
```

To set tapped items binding context for pop-up page, follow the code example.

**C#**

```
private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
{
    var popupPage = new SfPopUpView();
    popupPage.BindingContext = (sender as Grid).BindingContext;
    Navigation.PushAsync(popupPage);
}
```

Download entire source code from GitHub [here](#).

**PullToRefresh**

The [SfPullToRefresh](#) refreshing control allows interacting and refreshing the loaded view. When the SfListView is loaded inside the [SfPullToRefresh](#), it refreshes the item when performing the pull-to-refresh action.

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

Refer [initializing pull-to-refresh](#) to launch pull-to-refresh on each platform.

**SfListView inside the SfPullToRefresh**

The SfListView supports refreshing the data in view when performing the pull-to-refresh action at runtime by loading it directly into the [SfPullToRefresh.PullableContent](#) of the [SfPullToRefresh](#).

---

**Note:** You should load the SfListView as first children of [PullableContent](#) for the SfPullToRefresh.

---



## XML

```
<ContentPage xmlns:pulltoRefresh="clr-
namespace:Syncfusion.SfPullToRefresh.XForms;assembly=Syncfusion.SfPullToRefr
esh.XForms"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
>
<pulltoRefresh:SfPullToRefresh x:Name="pullToRefresh"
ProgressBackgroundColor="#428BCA" RefreshContentHeight="50"
RefreshContentWidth="50" TransitionMode="Push" IsRefreshing="False">
<pulltoRefresh:SfPullToRefresh.PullableContent>
<syncfusion:SfListView x:Name="listView" ItemSize="120"
SelectionMode="None">
</syncfusion:SfListView>
</pulltoRefresh:SfPullToRefresh.PullableContent>
</pulltoRefresh:SfPullToRefresh>
</ContentPage>
```

## C#

```
public ListViewPullToRefresh()
{
InitializeComponent();
//Initializing the PullToRefresh control.
SfPullToRefresh pullToRefresh = new SfPullToRefresh();
pullToRefresh.RefreshContentHeight = 50;
pullToRefresh.RefreshContentWidth = 50;
pullToRefresh.TransitionMode = TransitionType.Push;
pullToRefresh.IsRefreshing = false;
//Initializing the SfListView control.
var listView = new SfListView();
listView.ItemSize = 120;
listView.SelectionMode = SelectionMode.None;
//loading listview into pulltoRefresh
pullToRefresh.PullableContent = listView;
}
```

### Loading data when refreshing

To refresh the data in view at runtime, use the [SfPullToRefresh.Refreshing](#) event. The **Refreshing** event gets triggered once the progress bar meets 100 %. The data can be added into the underlying collection, and the data gets updated in view once the **Refreshing** event gets completed.

## C#

```
pullToRefresh.Refreshing += PullToRefresh_Refreshing;
private async void PullToRefresh_Refreshing(object sender, EventArgs args)
{
pullToRefresh.IsRefreshing = true;
await Task.Delay(2000);
for (int i = 0; i < 3; i++)
{
var blogsCount = pullToRefreshViewModel.BlogsInfo.Count;
var item = new ListViewBlogsInfo()
{
```

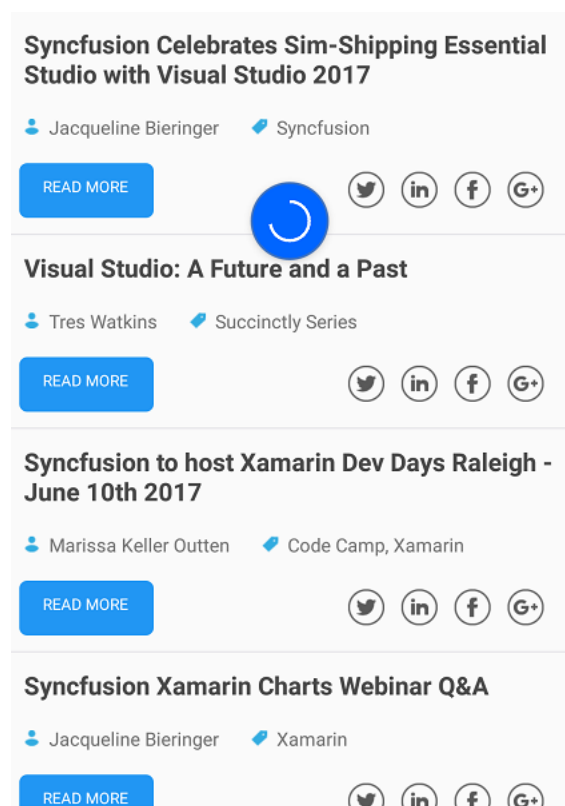
```

BlogTitle = pullToRefreshViewModel.BlogsTitle[blogsTitleCount - blogsCount],
BlogAuthor = pullToRefreshViewModel.BlogsAuthors[blogsAuthorCount -
blogsCount],
BlogCategory = pullToRefreshViewModel.BlogsCategory[blogsCategoryCount -
blogsCount],
ReadMoreContent =
pullToRefreshViewModel.BlogsReadMoreInfo[blogsReadMoreCount - blogsCount],
};
pullToRefreshViewModel.BlogsInfo.Insert(0, item);
}
pullToRefresh.IsRefreshing = false;
}

```

Run the application to render the following output.

Download the entire source code from GitHub [here](#).



SfListView inside the SfPullToRefresh with ScrollView

The SfListView allows loading as a [SfPullToRefresh.PullableContent](#) of the [SfPullToRefresh](#) with ScrollView and refresh the data in view at runtime.

### XML

```

<ContentPage xmlns:pulltoRefresh="clr-
namespace:Syncfusion.SfPullToRefresh.XForms;assembly=Syncfusion.SfPullToRefr
esh.XForms"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
>
<ScrollView>

```

```

<pulltoRefresh:SfPullToRefresh x:Name="pullToRefresh"
ProgressBackgroundColor="#428BCA"
TransitionMode="SlideOnTop"
IsRefreshing="False">
<pulltoRefresh:SfPullToRefresh.PullableContent>
<syncfusion:SfListView x:Name="listView"
ItemSize="120"
AutoFitMode="Height"
SelectionMode="None">
</syncfusion:SfListView>
</pulltoRefresh:SfPullToRefresh.PullableContent>
</pulltoRefresh:SfPullToRefresh>
</ScrollView>
</ContentPage>

```

## C#

```

public ListViewPullToRefresh()
{
InitializeComponent();
//Initializing the PullToRefresh control.
SfPullToRefresh pullToRefresh = new SfPullToRefresh();
pullToRefresh.RefreshContentHeight = 50;
pullToRefresh.RefreshContentWidth = 50;
pullToRefresh.TransitionMode = TransitionType.Push;
pullToRefresh.IsRefreshing = false;
//Initializing the SfListView control.
var listView = new SfListView();
listView.ItemSize = 120;
listView.SelectionMode = SelectionMode.None;
//loading listView into pulltorefresh
pullToRefresh.PullableContent = listView;
}

public App()
{
InitializeComponent();
SfPullToRefresh pullToRefresh = new SfPullToRefresh();
MainPage = new ContentPage { Content = new ScrollView { Content =
pullToRefresh.PullableContent } };
}

```

## Limitation

The horizontal ListView does not support the [SfPullToRefresh](#).

## How To

### *Pull-to-refresh with SearchBar at Top*

When the SearchBar or any view placed above the [SfPullToRefresh](#) control, pulling action does not work on the SfListView because touches directly passed to the SfListView instead of SfPullToRefresh control. You can overcome this problem by placing the SfListView inside the Grid and place that Grid as [SfPullToRefresh.PullableContent](#).

## XML

```

<ContentPage xmlns:pulltoRefresh="clr-
namespace:Syncfusion.SfPullToRefresh.XForms;assembly=Syncfusion.SfPullToRefr
esh.XForms"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
>
<ContentPage.Content>
<Grid RowSpacing="0" ColumnSpacing="0" Padding="0" Margin="0">
<Grid.RowDefinitions>
<RowDefinition Height="50" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.Behaviors>
<local:SfListViewPullToRefreshBehavior />
</Grid.Behaviors>
<SearchBar x:Name="filterText" Placeholder="Search" FontSize="14" />
<pulltoRefresh:SfPullToRefresh x:Name="pullToRefresh" Grid.Row="1"
ProgressBackgroundColor="#428BCA" RefreshContentHeight="50"
RefreshContentWidth="50" TransitionMode="Push" IsRefreshing="False">
<pulltoRefresh:SfPullToRefresh.PullableContent>
<Grid>
<syncfusion:SfListView x:Name="listView" AllowSwiping="True"
AutoFitMode="Height">
</syncfusion:SfListView>
</Grid>
</pulltoRefresh:SfPullToRefresh.PullableContent>
</pulltoRefresh:SfPullToRefresh>
</Grid>
</ContentPage.Content>
</ContentPage>

```

## C#

```

public ListViewPullToRefresh()
{
    InitializeComponent();
    //Initializing the PullToRefresh control.
    SfPullToRefresh pullToRefresh = new SfPullToRefresh();
    pullToRefresh.RefreshContentHeight = 50;
    pullToRefresh.RefreshContentWidth = 50;
    pullToRefresh.TransitionMode = TransitionType.Push;
    pullToRefresh.IsRefreshing = false;
    //Initializing the SfListView control.
    var listView = new SfListView();
    listView.ItemSize = 120;
    listView.SelectionMode = SelectionMode.None;
    //loading listview into pulltorefresh
    pullToRefresh.PullableContent = listView;
    var grid = new Grid();
    var searchBar = new SearchBar() { Placeholder = "Search here to filter" };
    grid.Children.Add(searchBar);
    grid.Children.Add(pullToRefresh, 0, 1);
}

```

Download entire source code from GitHub [here](#).

*Pull-to-refresh with Grouping*

The [SfPullToRefresh](#) has its pullable content as SfListView along with [SfListView.GroupHeaderTemplate](#). When refreshing the items in the listview, the newly added item loads directly into respective groups.

**XML**

```
<ContentPage xmlns:pulltoRefresh="clr-
namespace:Syncfusion.SfPullToRefresh.XForms;assembly=Syncfusion.SfPullToRefr
esh.XForms"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
>
<ContentPage.Content>
<Grid RowSpacing="0" ColumnSpacing="0" Padding="0" Margin="0">
<Grid.Behaviors>
<local:SfListViewPullToRefreshBehavior />
</Grid.Behaviors>
<Grid.RowDefinitions>
<RowDefinition Height="60"/>
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<SearchBar x:Name="filterText" Placeholder="Search" FontSize="14" />
<pulltoRefresh:SfPullToRefresh x:Name="pullToRefresh"
ProgressBackgroundColor="#0065FF"
RefreshContentHeight="50"
PullingThreshold="150"
RefreshContentWidth="50"
TransitionMode="Push"
IsRefreshing="False"
Grid.Row="1">
<pulltoRefresh:SfPullToRefresh.PullableContent>
<syncfusion:SfListView x:Name="listView" ItemSize="120"
AutoFitMode="Height" SelectionMode="None" AllowGroupExpandCollapse="True">
<syncfusion:SfListView.GroupHeaderTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid HeightRequest="150">
<Label Text="{Binding Key}" TextColor="Red" BackgroundColor="#d3d3d3"/>
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.GroupHeaderTemplate>
</syncfusion:SfListView>
</pulltoRefresh:SfPullToRefresh.PullableContent>
</pulltoRefresh:SfPullToRefresh>
</Grid>
</ContentPage.Content>
</ContentPage>
```

**C#**

```
public ListViewPullToRefresh ()
{
InitializeComponent();
```

```
//Initializing the PullToRefresh control.
SfPullToRefresh pullToRefresh = new SfPullToRefresh();
pullToRefresh.RefreshContentHeight = 50;
pullToRefresh.RefreshContentWidth = 50;
pullToRefresh.TransitionMode = TransitionType.Push;
pullToRefresh.IsRefreshing = false;
//Initializing the SfListView control.
var listView = new SfListView();
listView.ItemSize = 120;
listView.SelectionMode = SelectionMode.None;
listView.GroupHeaderTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    grid.HeightRequest = 150;
    var headerLabel = new Label
    {
        TextColor = Color.Red,
        BackgroundColor=Color.White
    };
    headerLabel.SetBinding(Label.TextProperty, new Binding("key"));
    grid.Children.Add(headerLabel);
    return grid;
});
//loading listview into pulltorefresh
pullToRefresh.PullableContent = listView;
var grid = new Grid();
var searchBar = new SearchBar() { Placeholder = "Search here to filter" };
grid.Children.Add(searchBar);
grid.Children.Add(pullToRefresh, 0, 1);
}
```

## Load More

The **ListView** enables **Load More** view by setting the [SfListView.LoadMoreOption](#) and [SfListView.LoadMoreCommand](#) properties. This can be displayed either on the top or bottom of the view by setting the [SfListView.LoadMorePosition](#) property. This view will be displayed when reaching the end of the list when the **LoadMorePosition** is bottom. This provides an option to add the items at runtime. If the **SfListView.LoadMorePosition** property is set as top, the items will be loaded only by using the **LoadMoreOption.Manual** mode.

The **SfListView.LoadMoreOption** property contains the following four different modes of operations:

- **None**: Disables the load more button. This is the default value.
- **Manual**: Displays the load more button when reaching the end of the list and execute **SfListView.LoadMoreCommand** when tapping the button.
- **Auto**: Automatically execute the **SfListView.LoadMoreCommand** when reaching end of the list.
- **AutoOnScroll**: Executes **SfListView.LoadMoreCommand** when users interact with listview and reach to the end of list.

The **SfListView.LoadMorePosition** property has two positions:

- Top: Positioned on the top of list.
- Bottom: Positioned on the bottom of list when reaching the end of the list. This is the default value.

`SfListView.LoadMoreCommand` executes when the listview is empty. This is the default behavior of Manual and Auto.

Load more automatically

To automatically load more items using the [SfListView.LoadMoreCommand](#) and [SfListView.LoadMoreCommandParameter](#) when reaching end of the list, set the [SfListView.LoadMoreOption](#) property as `Auto`.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
    ItemSize="120"
    LoadMoreOption="Auto"
    LoadMoreCommand="{Binding LoadMoreItemsCommand}"
    LoadMoreCommandParameter="{Binding Source={x:Reference listView}}}"
    ItemsSource="{Binding Products}"/>
</ContentPage>
```

### C#

```
listView.LoadMoreOption = LoadMoreOption.Auto;
listView.LoadMoreCommand = viewModel.LoadMoreItemsCommand;
//ViewModel.cs
LoadMoreItemsCommand = new Command<object>(LoadMoreItems, CanLoadMoreItems);
private bool CanLoadMoreItems(object obj)
{
    if (Products.Count >= totalItems)
        return false;
    return true;
}
private async void LoadMoreItems(object obj)
{
    var listView = obj as Syncfusion.ListView.XForms.SfListView;
    listView.IsBusy = true;
    await Task.Delay(2500);
    var index = Products.Count;
    var count = index + 3 >= totalItems ? totalItems - index : 3;
    AddProducts(index, count);
    listView.IsBusy = false;
}
private void AddProducts(int index, int count)
{
    for (int i = index; i < index + count; i++)
    {
        var name = Names[i];
        var p = new Product()
        {
            Name = name,
            Weight = Weights[i],
        }
    }
}
```

```

Price = Prices[i],
Image = ImageSource.FromResource("LoadMoreUG.LoadMore." + name.Replace(" ",
string.Empty) + ".jpg")
};
Products.Add(p);
}
}

```

### Load more manually

To load more items manually using the [SfListView.LoadMoreCommand](#) and [SfListView.LoadMoreCommandParameter](#) when tapping the load more button at end of the list, set the [SfListView.LoadMoreOption](#) property as **Manual**.

### XML

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
ItemSize="120"
LoadMoreOption="Manual"
LoadMoreCommand="{Binding LoadMoreItemsCommand}"
LoadMoreCommandParameter="{Binding Source={x:Reference listView}}}"
ItemsSource="{Binding Products}"/>
</ContentPage>

```

### C#

```

listView.LoadMoreOption = LoadMoreOption.Manual;
listView.LoadMoreCommand = viewModel.LoadMoreItemsCommand;
//ViewModel.cs
LoadMoreItemsCommand = new Command<object>(LoadMoreItems, CanLoadMoreItems);
private bool CanLoadMoreItems(object obj)
{
    if (Products.Count >= totalItems)
        return false;
    return true;
}
private async void LoadMoreItems(object obj)
{
    var listView = obj as Syncfusion.ListView.XForms.SfListView;
    listView.IsBusy = true;
    await Task.Delay(2500);
    var index = Products.Count;
    var count = index + 3 >= totalItems ? totalItems - index : 3;
    AddProducts(index, count);
    listView.IsBusy = false;
}
private void AddProducts(int index, int count)
{
    for (int i = index; i < index + count; i++)
    {
        var name = Names[i];
        var p = new Product()
        {
            Name = name,

```



```

Weight = Weights[i],
Price = Prices[i],
Image = ImageSource.FromResource("LoadMoreUG.LoadMore." + name.Replace(" ",
string.Empty) + ".jpg")
};
Products.Add(p);
}
}

```

### Load more when user interacts

To load more items only when users interact with the listview and reach to the end of list using `SfListView.LoadMoreCommand` and `SfListView.LoadMoreCommandParameter`, set the `SfListView.LoadMoreOption` property to [AutoOnScroll](#).

The `SfListView.LoadMoreCommand` will not execute when the listview is initially loaded. The `SfListView.LoadMoreCommand` will execute only when users interact and reach to the end of list.

The `SfListView.LoadMoreTemplate` is not displayed when the listview is initially loaded. When users interact, the `SfListView.LoadMoreTemplate` is displayed for the initially loaded items. Then, the visibility of `SfListView.LoadMoreTemplate` will be handled by the `CanExecute` method.

### XML

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="listView"
ItemSize="120"
LoadMoreOption="AutoOnScroll"
LoadMoreCommand="{Binding LoadMoreItemsCommand}"
LoadMoreCommandParameter="{Binding Source={x:Reference listView}}}"
ItemsSource="{Binding Products}"/>
</ContentPage>

```

### C#

```

listView.LoadMoreOption = LoadMoreOption.AutoOnScroll;
listView.LoadMoreCommand = viewModel.LoadMoreItemsCommand;
//ViewModel.cs
LoadMoreItemsCommand = new Command<object>(LoadMoreItems, CanLoadMoreItems);
/// <summary>
/// When AutoOnScroll load more is enabled, the CanExecute method will be
called only when the user interacts and reach to the end of list.
/// Based on return value, the visibility of the LoadMoreTemplate is handled
and the Execute method is called.
/// </summary>
/// <param name="obj">ListView is passed as default parameter.</param>
/// <returns>Returns true if the list has items to load, else returns
false.</returns>
private bool CanLoadMoreItems(object obj)
{
if (Products.Count >= totalItems)
return false;
return true;
}

```

```

/// <summary>
/// The Execute method is called based on the return value of the CanExecute
/// method. If CanExecute returns false, the Execute method will not be
/// executed.
/// </summary>
/// <param name="obj">ListView is passed as default parameter.</param>
private async void LoadMoreItems(object obj)
{
    var listView = obj as Syncfusion.ListView.XForms.SfListView;
    //Enables LoadMoreIndicator to the LoadMoreTemplate.
    listView.IsBusy = true;
    await Task.Delay(2500);
    var index = Products.Count;
    var count = index + 3 >= totalItems ? totalItems - index : 3;
    //Adding the items to the list.
    AddProducts(index, count);
    //Disables LoadMoreIndicator after adding the items.
    listView.IsBusy = false;
}

private void AddProducts(int index, int count)
{
    for (int i = index; i < index + count; i++)
    {
        var name = Names[i];
        var p = new Product()
        {
            Name = name,
            Weight = Weights[i],
            Price = Prices[i],
            Image = ImageSource.FromResource("LoadMoreUG.LoadMore." + name.Replace(" ",
            string.Empty) + ".jpg")
        };
        Products.Add(p);
    }
}

```

### Show loading indicator

The [SfListView.LoadMoreIndicator](#) will be displayed when loading more items in the list.

By using the [SfListView.IsBusy](#) property, you can interchange the visibility of the button and busy indicator when creating the load more view. You can set the value of the [SfListView.IsBusy](#) property to true before adding items to the list and set it to false, after adding the items. You can also bind the [IsBusy](#) property through ViewModel.

### C#

```

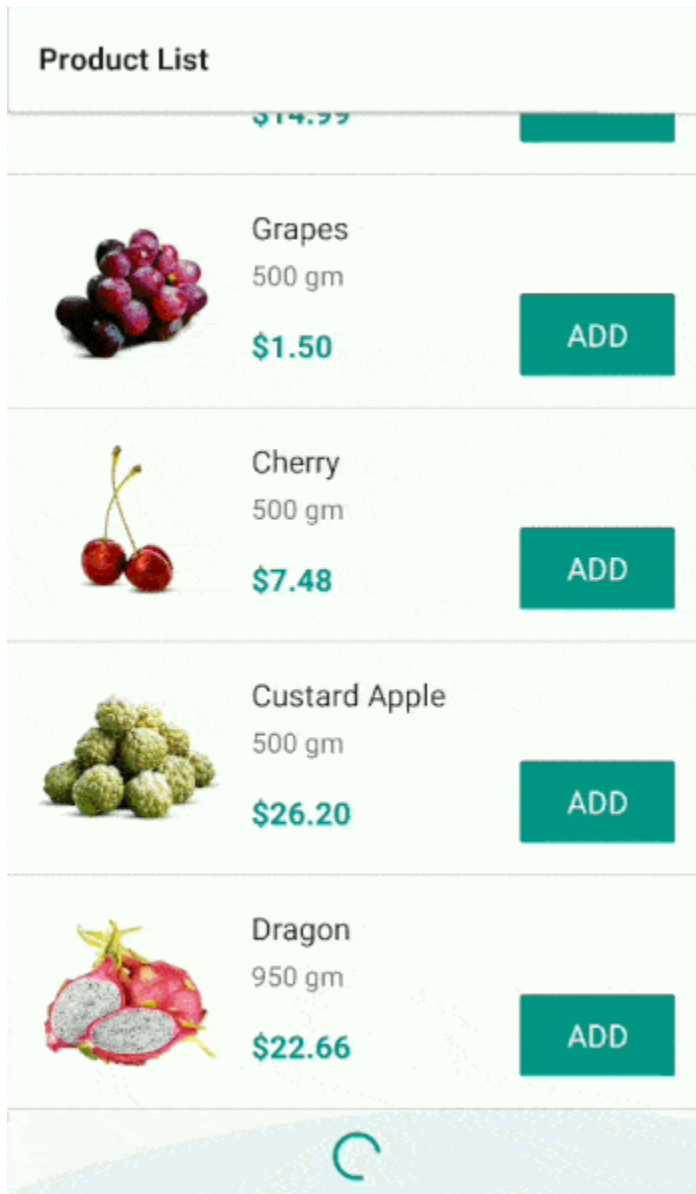
private async void LoadMoreItems(object obj)
{
    var listView = obj as Syncfusion.ListView.XForms.SfListView;
    listView.IsBusy = true;
    await Task.Delay(2500);
    var index = Products.Count;
    var count = index + 3 >= totalItems ? totalItems - index : 3;
    AddProducts(index, count);
    listView.IsBusy = false;
}

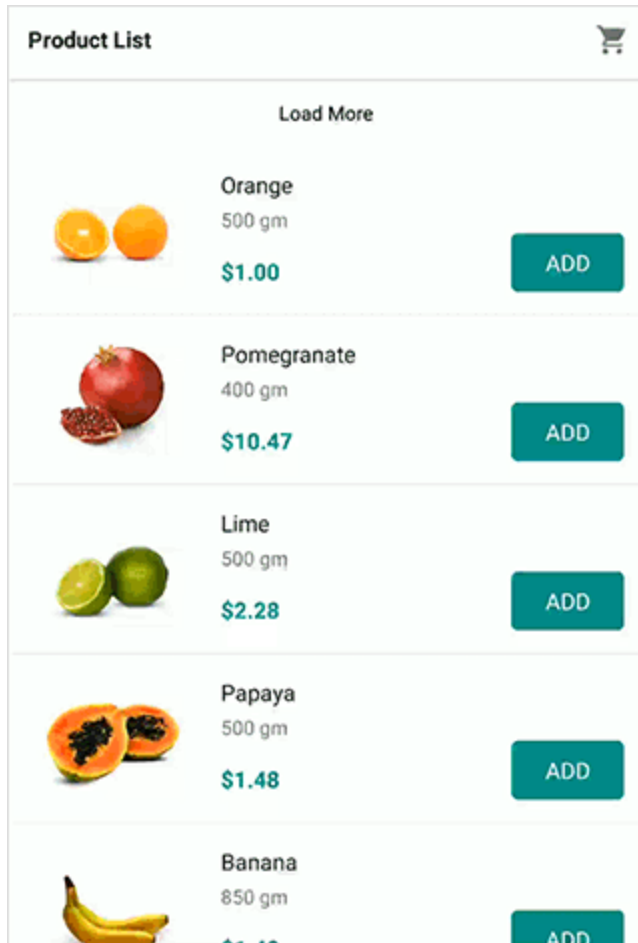
```

```
}
```

Download the entire source code from GitHub [here](#).

Items can be loaded either on the top or bottom of the view.





### Load more view customization

The SfListView allows customizing User Interface(UI) of **Load More** view.

#### *Load more button*

To customize the load more button, add the custom UI in the [SfListView.LoadMoreTemplate](#) property.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="Sample.MainPage"
xmlns:helper="clr-
namespace:Syncfusion.ListView.XForms.Helpers;assembly=Syncfusion.SfListView.
XForms"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.BindingContext>
<local:LoadMoreViewModel/>
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<helper:InverseBoolConverter x:Key="inverseBoolConverter"/>
</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfListView x:Name="listView"
```

```

ItemSize="120"
LoadMoreOption="Manual"
LoadMoreCommand="{Binding LoadMoreItemsCommand}"
LoadMoreCommandParameter="{Binding Source={x:Reference listView}}"
ItemsSource="{Binding Products}">
<syncfusion:SfListView.LoadMoreTemplate>
<DataTemplate>
<Grid>
<Label Text="Load More Items..." TextColor="Black"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
IsVisible="{Binding IsBusy, Converter={StaticResource inverseBoolConverter},
Source={x:Reference Name=listView}}" />
</Grid>
</DataTemplate>
</syncfusion:SfListView.LoadMoreTemplate>
</syncfusion:SfListView>
</ContentPage>

```

## C#

```

listView.LoadMoreTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var label = new Label
    {
        Text = "Load More Items...",
        FontSize = 20,
        BackgroundColor = Color.AliceBlue,
        HorizontalTextAlignment = TextAlignment.Center,
        VerticalTextAlignment = TextAlignment.Center
    };
    label.SetBinding(Label.IsVisibleProperty, new Binding("IsBusy",
        BindingMode.Default, new InverseBoolConverter(), null, null, listView));
    grid.Children.Add(label);
    return grid;
});

```

## Loading Indicator

To customize the loading indicator, add the custom UI in the [SfListView.LoadMoreTemplate](#) property.

## XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="Sample.MainPage"
xmlns:helper="clr-
namespace:Syncfusion.ListView.XForms.Helpers;assembly=Syncfusion.SfListView.
XForms"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.BindingContext>
<local:LoadMoreViewModel/>
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<helper:InverseBoolConverter x:Key="inverseBoolConverter"/>

```

```

</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfListView x:Name="listView"
    ItemSize="120"
    LoadMoreOption="Manual"
    LoadMoreCommand="{Binding LoadMoreItemsCommand}"
    LoadMoreCommandParameter="{Binding Source={x:Reference listView}}}"
    ItemsSource="{Binding Products}">
    <syncfusion:SfListView.LoadMoreTemplate>
    <DataTemplate>
    <Grid>
    <Label Text="Load More Items" TextColor="Black"
        HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
        IsVisible="{Binding IsBusy, Converter={StaticResource inverseBoolConverter},
        Source={x:Reference Name=listView}}" />
    <syncfusion:LoadMoreIndicator IsRunning="{Binding IsBusy,
        Source={x:Reference Name=listView}}" IsVisible="{Binding IsBusy,
        Source={x:Reference Name=listView}}" Color="Red" VerticalOptions="Center"/>
    </Grid>
    </DataTemplate>
    </syncfusion:SfListView.LoadMoreTemplate>
    </syncfusion:SfListView>
</ContentPage>

```

## C#

```

listView.LoadMoreTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var label = new Label
    {
        Text = "Load More Items...",
        FontSize = 20,
        BackgroundColor = Color.AliceBlue,
        HorizontalTextAlignment = TextAlignment.Center,
        VerticalTextAlignment = TextAlignment.Center
    };
    label.SetBinding(Label.IsVisibleProperty, new Binding("IsBusy",
        BindingMode.Default, new InverseBoolConverter(), null, null, listView));
    var loadMoreIndicator = new LoadMoreIndicator();
    loadMoreIndicator.Color = Color.Red;
    loadMoreIndicator.VerticalOptions = LayoutOptions.Center;
    loadMoreIndicator.SetBinding(LoadMoreIndicator.IsRunningProperty, new
        Binding("IsBusy", BindingMode.Default, null, null, null, listView));
    loadMoreIndicator.SetBinding(LoadMoreIndicator.IsVisibleProperty, new
        Binding("IsBusy", BindingMode.Default, null, null, null, listView));
    grid.Children.Add(label);
    grid.Children.Add(loadMoreIndicator);
    return grid;
});

```

### *Customize the size of load more view and indicator*

Listview allows customizing the size of the load more item by the [SfListView.QueryItemSize](#) event using the item type.

**C#**

```

this.listView.QueryItemSize += ListView_QueryItemSize;
private void ListView_QueryItemSize(object sender,
Syncfusion.ListView.XForms.QueryItemSizeEventArgs e)
{
    if(e.ItemType == ItemType.LoadMore)
    {
        e.ItemSize = 300;
        e.Handled = true;
    }
}

```

To customize the size of the loading indicator, add the custom UI to the [SfListView.LoadMoreTemplate](#) property and assign the height and width for the grid and loading indicator.

**XML**

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="Sample.MainPage"
xmlns:helper="clr-
namespace:Syncfusion.ListView.XForms.Helpers;assembly=Syncfusion.SfListView.
XForms"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.BindingContext>
<local:LoadMoreViewModel/>
</ContentPage.BindingContext>
<ContentPage.Resources>
<ResourceDictionary>
<helper:InverseBoolConverter x:Key="inverseBoolConverter"/>
</ResourceDictionary>
</ContentPage.Resources>
<syncfusion:SfListView x:Name="listView"
ItemSize="120"
LoadMoreOption="Manual"
LoadMoreCommand="{Binding LoadMoreItemsCommand}"
LoadMoreCommandParameter="{Binding Source={x:Reference listView}}}"
ItemsSource="{Binding Products}">
<syncfusion:SfListView.LoadMoreTemplate>
<DataTemplate>
<Grid HeightRequest="100" WidthRequest="100">
<Label Text="Load More Items" TextColor="Black"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
IsVisible="{Binding IsBusy, Converter={StaticResource inverseBoolConverter},
Source={x:Reference Name=listView}}" />
<syncfusion:LoadMoreIndicator IsRunning="{Binding IsBusy,
Source={x:Reference Name=listView}}" IsVisible="{Binding IsBusy,
Source={x:Reference Name=listView}}" Color="Red" VerticalOptions="Center"
HeightRequest="100" WidthRequest="100"/>
</Grid>
</DataTemplate>
</syncfusion:SfListView.LoadMoreTemplate>
</syncfusion:SfListView>
</ContentPage>

```

**C#**

```
listView.LoadMoreTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    grid.HeightRequest = 100;
    grid.WidthRequest = 100;
    var label = new Label
    {
        Text = "Load More Items...",
        FontSize = 20,
        BackgroundColor = Color.AliceBlue,
        HorizontalTextAlignment = TextAlignment.Center,
        VerticalTextAlignment = TextAlignment.Center
    };
    label.SetBinding(Label.IsVisibleProperty, new Binding("IsBusy",
        BindingMode.Default, new InverseBoolConverter(), null, null, listView));
    var loadMoreIndicator = new LoadMoreIndicator();
    loadMoreIndicator.Color = Color.Red;
    loadMoreIndicator.VerticalOptions = LayoutOptions.Center;
    loadMoreIndicator.SetBinding(LoadMoreIndicator.IsRunningProperty, new
        Binding("IsBusy", BindingMode.Default, null, null, null, listView));
    loadMoreIndicator.SetBinding(LoadMoreIndicator.IsVisibleProperty, new
        Binding("IsBusy", BindingMode.Default, null, null, null, listView));
    loadMoreIndicator.HeightRequest = 100;
    loadMoreIndicator.WidthRequest = 100;
    grid.Children.Add(label);
    grid.Children.Add(loadMoreIndicator);
    return grid;
});
```

## Disable load more at runtime

To disable the **Load More** view, return the 'CanExecute' method of the [SfListView.LoadMoreCommand](#) to false.

If you reach maximum count (for example, totalItems = 22) in the list, follow the code example to disable the **Load More** view.

**C#**

```
LoadMoreItemsCommand = new Command<object>(LoadMoreItems, CanLoadMoreItems);
private async void LoadMoreItems(object obj)
{
    var listView = obj as Syncfusion.ListView.XForms.SfListView;
    listView.IsBusy = true;
    await Task.Delay(2500);
    var index = Products.Count;
    var count = index + 3 >= totalItems ? totalItems - index : 3;
    AddProducts(index, count);
    listView.IsBusy = false;
}
private bool CanLoadMoreItems(object obj)
{
    if (Products.Count >= totalItems)
```



```
return false;
return true;
}
```

## Limitations

- SfListView does not support to set Manual in [SfListView.LoadMoreOption](#) when [SfListView.Orientation](#) is Horizontal.
- SfListView supports to set Auto and AutoOnScroll in [SfListView.LoadMoreOption](#) only when [SfListView.LoadMorePosition](#) is set to Bottom.
- Handle [LoadMoreCommand](#) execution by implementing [CanExecute](#) predicate of command.

## How to

### *Load more on infinite scroll*

The SfListView allows adding more items infinite times either manually or automatically.

## C#

```
public class LoadMoreViewModel:INotifyPropertyChanged
{
    public ObservableCollection<Product> Products { get; set; }
    public Command<object> LoadMoreItemsCommand { get; set; }
    public LoadMoreViewModel()
    {
        Products = new ObservableCollection<Product>();
        AddProducts(0, 10);
        LoadMoreItemsCommand = new Command<object>(LoadMoreItems);
    }
    private async void LoadMoreItems(object obj)
    {
        var listview = obj as Syncfusion.ListView.XForms.SfListView;
        listview.IsBusy = true;
        await Task.Delay(2500);
        AddProducts(11, 21);
        listview.IsBusy = false;
    }
    private void AddProducts(int value, int count)
    {
        Random rand= new Random();
        for (int i = value; i < count; i++)
        {
            var name = Names[rand.Next(1,22)];
            var p = new Product()
            {
                Name = name,
                Weight = Weights[i],
                Price = Prices[i],
                Image = ImageSource.FromResource("LoadMoreUG.LoadMore." + name.Replace(" ",
                    string.Empty) + ".jpg")
            };
            Products.Add(p);
        }
    }
}
```

```
}
```

Download the entire sample from GitHub [here](#).

*Load more items automatically from up direction*

The SfListView allows loading more items automatically when reaching top of the list by showing the busy indicator by loading in the [HeaderTemplate](#).

#### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="ListView" IsBusy="True"
ItemsSource="{Binding Messages}"
AutoFitMode="Height">
<syncfusion:SfListView.HeaderTemplate>
<DataTemplate>
<ViewCell>
<Grid>
<syncfusion:LoadMoreIndicator Color="Red" IsRunning="True"
IsVisible="{Binding IndicatorIsVisible}"/>
</Grid>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.HeaderTemplate>
</syncfusion:SfListView>
</ContentPage>
```

#### C#

```
public partial class MainPage : ContentPage
{
    MainPageViewModel ViewModel;
    public MainPage()
    {
        InitializeComponent();
        ViewModel = new MainPageViewModel();
        ListView.IsBusy = true;
        ListView.ItemsSource = ViewModel.Messages;
        ListView.AutoFitMode = AutoFitMode.Height;
        ListView.HeaderTemplate = new DataTemplate(() =>
        {
            var grid = new Grid();
            var loadMoreIndicator = new LoadMoreIndicator()
            {
                Color = Color.Red,
                IsRunning = true
            };
            loadMoreIndicator.SetBinding(LoadMoreIndicator.IsVisibleProperty, new
            Binding("IndicatorIsVisible"));
            grid.Children.Add(loadMoreIndicator);
            return grid;
        });
    }
}
```

Insert each new item in the 0th position of the underlying collection bound to the SfListView.ItemsSource property.

### C#

```
using Syncfusion.ListView.XForms.Control.Helpers;
public partial class MainPage : ContentPage
{
    MainPageViewModel ViewModel;
    VisualContainer visualContainer;
    public bool isScrolled;
    HeaderItem headerItem;
    public MainPage()
    {
        InitializeComponent();
        ViewModel = new MainPageViewModel();
        BindingContext = ViewModel;
        ViewModel.ListView = this.ListView;
        ListView.Loaded += ListView_Loaded;
        visualContainer = ListView.GetVisualContainer();
    }
    private void HeaderItem_PropertyChanged(object sender,
        System.ComponentModel.PropertyChangedEventArgs e)
    {
        if(e.PropertyName=="Visibility")
        {
            if (headerItem.Visibility && isScrolled)
                LoadMoreOnTop();
        }
    }
    private async void LoadMoreOnTop()
    {
        //To get the current first item which is visible in the View.
        var firstItem = ListView.DataSource.DisplayItems[0];
        ViewModel.IndicatorIsVisible = true;
        await Task.Delay(4000);
        var r = new Random();
        //To avoid layout calls for arranging each and every items to be added in the View.
        ListView.DataSource.BeginInit();
        for (int i = 0; i < 5; i++)
        {
            var collection = new Message();
            collection.Text = ViewModel.MessageText[r.Next(0,
                ViewModel.MessageText.Count() - 1)];
            collection.IsIncoming = i % 2 == 0 ? true : false;
            collection.MessageDateTime = DateTime.Now.ToString();
            ViewModel.Messages.Insert(0, collection);
        }
        ListView.DataSource.EndInit();
        var firstItemIndex = ListView.DataSource.DisplayItems.IndexOf(firstItem);
        var header = (ListView.HeaderTemplate != null && !ListView.IsStickyHeader) ?
            1 : 0;
        var totalItems = firstItemIndex + header;
        //Need to scroll back to previous position else the ScrollViewer moves to top of the list.
        ListView.LayoutManager.ScrollToRowIndex(totalItems, true);
    }
}
```

```

ViewModel.IndicatorIsVisible = false;
}
private void ListView_Loaded(object sender,
Syncfusion.ListView.XForms.ListViewLoadedEventArgs e)
{
    //To avoid loading items initially when page loaded.
    if (!isScrolled)
    (ListView.LayoutManager as
LinearLayout).ScrollToRowIndex(ViewModel.Messages.Count - 1, true);
    headerItem = visualContainer.Children[0] as HeaderItem;
    headerItem.PropertyChanged += HeaderItem_PropertyChanged;
    isScrolled = true;
}
}

```

Download the entire source code from GitHub [here](#).



*Load more items manually from up direction*

The SfListView allows loading more items when tapping the button loaded in the [HeaderTemplate](#) when reaching top of the list and shows the busy indicator till the items are added into the collection.

#### XML

```

<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<syncfusion:SfListView x:Name="ListView" IsBusy="True"
ItemTemplate="{StaticResource MessageTemplateSelector}"
ItemsSource="{Binding Messages}"

```

```

ItemSize="100">
<syncfusion:SfListView.HeaderTemplate>
<DataTemplate>
<ViewCell>
<Grid>
<Grid BackgroundColor="#d3d3d3" IsVisible="{Binding GridIsVisible}">
<Button Text="Load More" Clicked="Button_Clicked"
HorizontalOptions="CenterAndExpand" VerticalOptions="CenterAndExpand"/>
</Grid>
<syncfusion:LoadMoreIndicator Color="Red" IsRunning="True"
IsVisible="{Binding IndicatorIsVisible}"/>
</Grid>
</ViewCell>
</DataTemplate>
</syncfusion:SfListView.HeaderTemplate>
</syncfusion:SfListView>
</ContentPage>

```

**C#**

```

public partial class MainPage : ContentPage
{
    MainPageViewModel ViewModel;
    public MainPage()
    {
        InitializeComponent();
        ViewModel = new MainPageViewModel();
        ListView.IsBusy = true;
        ListView.ItemsSource = ViewModel.Messages;
        ListView.AutoFitMode = AutoFitMode.Height;
        ListView.HeaderTemplate = new DataTemplate(() =>
        {
            var grid = new Grid();
            grid.BackgroundColor = Color.FromHex("#d3d3d3");
            grid.SetBinding(Grid.IsVisibleProperty, new Binding("GridIsVisible"));
            var loadMore = new Button()
            {
                HorizontalOptions = LayoutOptions.CenterAndExpand,
                VerticalOptions = LayoutOptions.CenterAndExpand,
                Text = "LoadMore",
            };
            loadMore.Clicked += Button_Clicked;
            grid.Children.Add(loadMore);
            var grid1 = new Grid();
            var loadMoreIndicator = new LoadMoreIndicator()
            {
                Color = Color.Red,
                IsRunning = true
            };
            loadMoreIndicator.SetBinding(LoadMoreIndicator.IsVisibleProperty, new
            Binding("IndicatorIsVisible"));
            grid1.Children.Add(loadMoreIndicator);
            grid1.Children.Add(grid);
            return grid1;
        });
    }
}

```

```
}

```

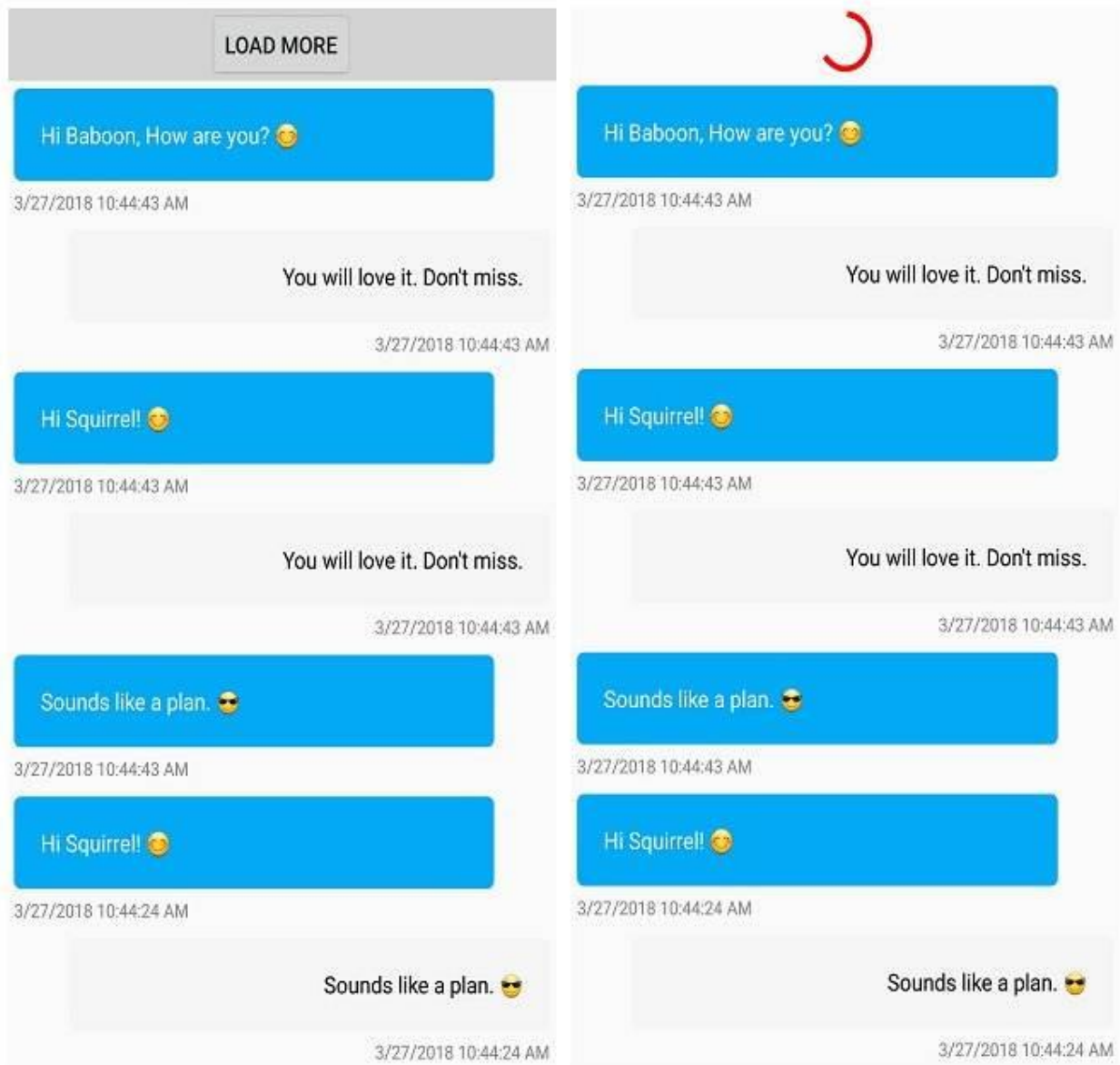
Insert each new item in the 0th position of the underlying collection bound to the SfListView.ItemsSource property.

### C#

```
public partial class MainPage : ContentPage
{
    MainPageViewModel ViewModel;
    VisualContainer visualContainer;
    public MainPage()
    {
        InitializeComponent();
        ViewModel = new MainPageViewModel();
        BindingContext = ViewModel;
        ViewModel.ListView = this.ListView;
        ListView.Loaded += ListView_Loaded;
    }
    private void ListView_Loaded(object sender,
        Syncfusion.ListView.XForms.ListViewLoadedEventArgs e)
    {
        (ListView.LayoutManager as
        LinearLayout).ScrollToRowIndex(ViewModel.Messages.Count - 1, true);
    }
    private async void Button_Clicked(object sender, EventArgs e)
    {
        //To get the current first item which is visible in the View.
        var firstItem = ListView.DataSource.DisplayItems[0];
        ViewModel.GridIsVisible = false;
        ViewModel.IndicatorIsVisible = true;
        await Task.Delay(2000);
        var r = new Random();
        //To avoid layout calls for arranging each and every items to be added in the View.
        ListView.DataSource.BeginInit();
        for (int i = 0; i < 5; i++)
        {
            var collection = new Message();
            collection.Text = ViewModel.MessageText[r.Next(0,
            ViewModel.MessageText.Count() - 1)];
            collection.IsIncoming = i % 2 == 0 ? true : false;
            collection.MessageDateTime = DateTime.Now.ToString();
            ViewModel.Messages.Insert(0, collection);
        }
        ListView.DataSource.EndInit();
        var firstItemIndex = ListView.DataSource.DisplayItems.IndexOf(firstItem);
        var header = (ListView.HeaderTemplate != null && !ListView.IsStickyHeader) ?
        1 : 0;
        var totalItems = firstItemIndex + header;
        //Need to scroll back to previous position else the ScrollViewer moves to top of the list.
        ListView.LayoutManager.ScrollToRowIndex(totalItems, true);
        ViewModel.GridIsVisible = true;
        ViewModel.IndicatorIsVisible = false;
    }
}
```

}

Download the entire source code from GitHub [here](#).



*How to disable LoadMoreCommand execution when ListView is Empty?*

You can skip the load more action by checking the underlying collection count in the execute method.

**C#**

```
//ViewModel.cs
LoadMoreItemsCommand = new Command<object>(LoadMoreItems, CanLoadMoreItems);
private bool CanLoadMoreItems(object obj)
{
    if (Products.Count >= totalItems)
        return false;
    return true;
}
```

```
private async void LoadMoreItems(object obj)
{
    if (Products.Count == 0)
        return;
    var listView = obj as Syncfusion.ListView.XForms.SfListView;
    listView.IsBusy = true;
    await Task.Delay(2500);
    var index = Products.Count;
    var count = index + 3 >= totalItems ? totalItems - index : 3;
    AddProducts(index, count);
    listView.IsBusy = false;
}

private void AddProducts(int index, int count)
{
    for (int i = index; i < index + count; i++)
    {
        var name = Names[i];
        var p = new Product()
        {
            Name = name,
            Weight = Weights[i],
            Price = Prices[i],
            Image = ImageSource.FromResource("LoadMoreUG.LoadMore." + name.Replace(" ",
                string.Empty) + ".jpg")
        };
        Products.Add(p);
    }
}
```

Download the GitHub sample from GitHub [here](#)

### Right to left(RTL)

ListView supports to change the flow of text to the right-to-left direction by setting the [FlowDirection](#) to [RightToLeft](#) in both [Vertical](#) and [Horizontal](#) orientations. ListView supports RTL in Xamarin.Forms version 3.0 and above. It also supports RTL when device's flow direction is changed.

**Note:** Specific platform setup is required to enable right-to-left localization. For platform settings you can refer [here](#).

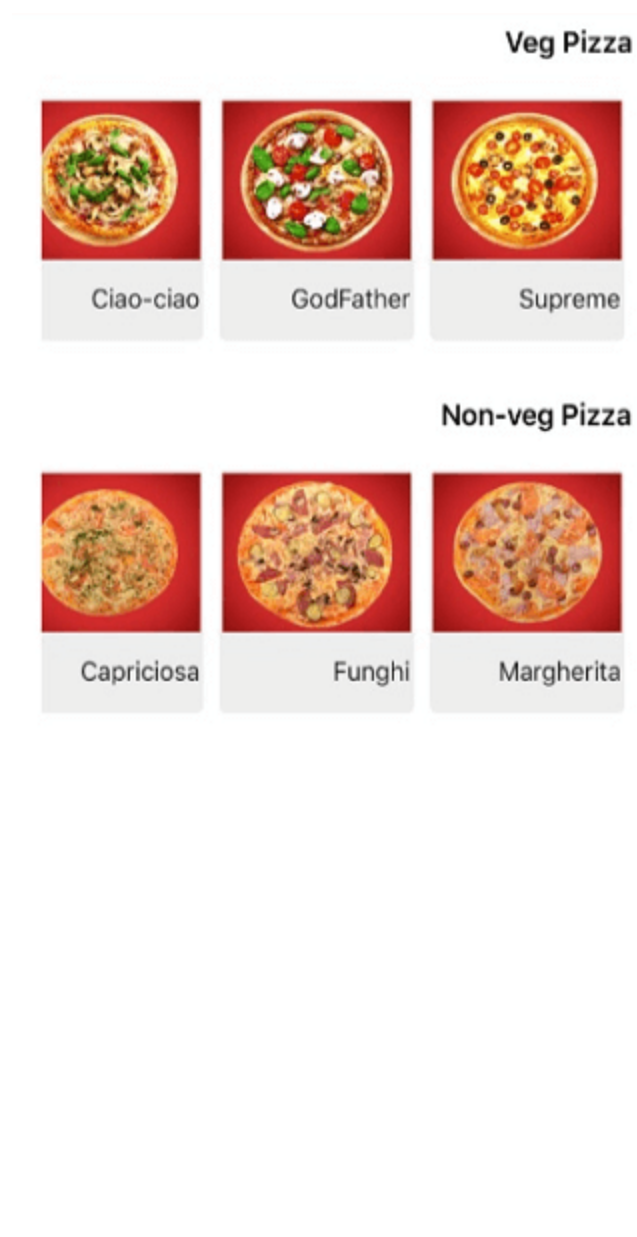
### XML

```
<ContentPage xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms">
<ContentPage.Content>
<syncfusion:SfListView x:Name="listView" FlowDirection="RightToLeft"/>
</ContentPage.Content>
</ContentPage>
```

### C#

```
listView.FlowDirection = FlowDirection.RightToLeft;
```





You can download the entire source code of this demo from [here](#).

**Note:** When a label is loaded in the `ItemTemplate`, the right-to-left direction is not applied due to the framework issue. It has been reported to the Xamarin team; for more details about this, refer to this [link](#). To overcome this issue, set the `HorizontalOptions` to `StartAndExpand` in Label.

#### Limitation

- ListView item does not arrange from right to left direction in `Horizontal` orientation, when the `AutoFitMode` is `Height`.

## MVVM

### Commands

#### Tap command

The [TapCommand](#) will be triggered whenever tapping the item and passing the [ItemTappedEventArgs](#) as parameter.

#### C#

```
listView.TapCommand = viewModel.TappedCommand;
public class CommandViewModel
{
    private Command<Object> tappedCommand;
    public Command<object> TappedCommand
    {
        get { return tappedCommand; }
        set { tappedCommand = value; }
    }
    public CommandViewModel()
    {
        TappedCommand = new Command<object>(TappedCommandMethod);
    }
    private void TappedCommandMethod(object obj)
    {
        if ((obj as Syncfusion.ListView.XForms.ItemTappedEventArgs).ItemData ==
            viewModel.InboxInfo[0])
            viewModel.InboxInfo.Remove(e.ItemData as ListViewInboxInfo)
    }
}
```

#### Hold command

The [HoldCommand](#) will be triggered whenever long pressing the item and passing the [ItemHoldingEventArgs](#) as parameter.

#### C#

```
listView.HoldCommand = viewModel.HoldCommand;
public class CommandViewModel
{
    private Command<Object> holdingCommand;
    public Command<object> HoldingCommand
    {
        get { return holdingCommand; }
        set { holdingCommand = value; }
    }
    public CommandViewModel()
    {
        HoldingCommand = new Command<object>(HoldingCommandMethod);
    }
    private void HoldingCommandMethod(object obj)
    {
        if ((obj as Syncfusion.ListView.XForms.ItemHoldingEventArgs).ItemData ==
            viewModel.InboxInfo[3])
            viewModel.InboxInfo.Remove(e.ItemData as ListViewInboxInfo);
    }
}
```

---

**Note:** When Command is bound to ItemTemplate, it must also define **Source** property with its root element as reference. Only then it executes the property in the ViewModel of type Command.

---

### Event to command

The ListView event can be converted into commands using [Behaviors](#). To achieve this, create a command in the ViewModel class and associate it to the ListView event using **Behaviors**.

### XML

```
<listView:SfListView x:Name="listView" ItemsSource="{Binding contactsinfo}">
<listView:SfListView.Behaviors>
<local:EventToCommandBehavior EventName="SelectionChanged" Command="{Binding
SelectionChangedCommand}" />
</listView:SfListView.Behaviors>
</listView:SfListView>
```

### C#

```
public class ContactsViewModel
{
    public Command<ItemSelectionChangedEventArgs> selectionChangedCommand;
    public Command<ItemSelectionChangedEventArgs> SelectionChangedCommand
    {
        get { return selectionChangedCommand; }
        set { selectionChangedCommand = value; }
    }
    public ContactsViewModel()
    {
        SelectionChangedCommand = new
        Command<Syncfusion.ListView.XForms.ItemSelectionChangedEventArgs>(OnSelection
        Changed);
    }
    private void OnSelectionChanged(ItemSelectionChangedEventArgs obj)
    {
        App.Current.MainPage.DisplayAlert("Alert", (obj.AddedItems[0] as
        Contacts).ContactName + " is selected", "OK");
    }
}
```

Download the entire source code from GitHub [here](#).

For more information regarding the event to command behavior in Xamarin.Forms, you can refer [this](#) link.

### Binding command of inner ListView to Model Command?

You can bind command of Button inside ItemTemplate to the command in Model by specifying Source property with its root element as reference to execute the binded property of type command.

### XML

```
<listView:SfListView x:Name="listView" ItemsSource="{Binding ContactInfo}">
<listView:SfListView.ItemTemplate>
<DataTemplate>
<ViewCell>
```

```

<ViewCell.View>
<StackLayout>
<listView:SfListView x:Name="list1" ItemsSource="{Binding ContactDetails}"
TapCommand="{Binding NavigateToSelectModelsCommand}"
TapCommandParameter="{Binding}">
<listView:SfListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<StackLayout BackgroundColor="Teal" >
<Label Text="{Binding ContactName}" />
<Label Text="{Binding ContactNumber}" />
<StackLayout HeightRequest="1" BackgroundColor="Gray"/>
</StackLayout>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</listView:SfListView.ItemTemplate>
</listView:SfListView>
</StackLayout>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</listView:SfListView.ItemTemplate>
</listView:SfListView>

```

## C#

```

public class ContactInfo_NestedListView
{
    public Command<Object> NavigateToSelectModelsCommand { get; private set; }
    public ContactInfo_NestedListView()
    {
        NavigateToSelectModelsCommand = new Command<Object>(NavigateToSelectModels,
        CanNavigate);
    }
    private bool CanNavigate(object argument)
    {
        return true;
    }
    private void NavigateToSelectModels(object model)
    {
        var customer = model as ContactInfo_NestedListView;
        App.Current.MainPage.DisplayAlert("Message", "Tapped customer group value : "
        + customer.location, "OK");
    }
}

```

Binding command of Button inside the ItemTemplate of Xamarin.Forms ListView to ViewModel Command?

You can bind command of Button inside ItemTemplate to the command in ViewModel by specifying Source property with its root element as reference to execute the binded property of type command.

## XML

```
<Syncfusion:SfListView x:Name="listView" ItemsSource="{Binding
contactsinfo}" >
<Syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<Grid >
<Button Text="Delete" Command="{Binding Path=BindingContext.DeleteCommand,
Source={x:Reference listView}}" CommandParameter="{x:Reference listView}"/>
</Grid>
</ViewCell>
</DataTemplate>
</Syncfusion:SfListView.ItemTemplate>
</Syncfusion:SfListView>
```

## C#

```
namespace ListViewSample
{
    public class ContactsViewModel : INotifyPropertyChanged
    {
        public Command<object> DeleteCommand { get; set; }
        public ContactsViewModel()
        {
            DeleteCommand = new Command<object>(OnTapped);
        }
        private void OnTapped(object obj)
        {
            var contact = obj as Contacts;
            contactsInfo.Remove(contact);
            App.Current.MainPage.DisplayAlert("Message", "Item Deleted :",
            +contact.ContactName, "Ok");
        }
    }
}
```

You can download the sample from GitHub [here](#).

## ListView with Prism Framework

The SfListView allows the user to work with prism for MVVM Framework. Steps to be followed:

1. Replace your application to prism application in App.xaml file.
2. Inherit App.xaml.cs from prism application instead of your application.
3. Create a prism namespace library reference in xaml file of the ContentPage.
4. Connect view and view model instead of binding context by registering them.

## C#

```
public partial class App : PrismApplication
{
    public App(IPlatformInitializer initializer = null) : base(initializer) { }
    protected override void OnInitialized()
    {
        InitializeComponent();
    }
}
```

```
protected override void RegisterTypes ()
{
    Container.RegisterTypeForNavigation<MainPage, MainPageViewModel>();
}
}
```

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<prism:PrismApplication xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:prism="clr-namespace:Prism.Unity;assembly=Prism.Unity.Forms"
x:Class="ListViewPrism.App">
</prism:PrismApplication>
```

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"
xmlns:local="clr-namespace:ListViewPrism;assembly=ListViewPrism"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
x:Class="ListViewPrism.Views.MainPage"
Title="MainPage">
<StackLayout HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
<syncfusion:SfListView x:Name="listView" ItemSize="70" ItemSpacing="0,0,5,0"
AutoFitMode="Height"
ItemsSource="{Binding ContactsInfo}" IsStickyHeader="True"
AllowSwiping="True" IsStickyGroupHeader="True" GroupHeaderSize="50">
</syncfusion:SfListView>
</StackLayout>
</ContentPage>
```

For more details, refer to <https://xamgirl.com/prism-in-xamarin-forms-step-by-step-part-1>.

Download the entire source code from GitHub [here](#).

### ListView with MVVMCross

The SfListView allows users work with MVVMCross Framework. Follow the below steps to work with MVVMCross Framework:

1. Inherit App.cs from MvxApplication instead of your application.

**C#**

```
public class CoreApp : MvvmCross.Core.ViewModels.MvxApplication
{
    public override void Initialize ()
    {
    }
}
```

2. Inherit ViewModel from the MvxViewModel.

**C#**

```
public class MvxFormsViewModel : MvxViewModel
{
    public MvxFormsViewModel()
    {
    }
}
```

3. Connect view and view model instead of binding context by registering them.

**C#**

```
public class CoreApp : MvvmCross.Core.ViewModels.MvxApplication
{
    public override void Initialize()
    {
        CreatableTypes()
        .EndingWith("Service")
        .AsInterfaces()
        .RegisterAsLazySingleton();
        RegisterNavigationServiceAppStart<ViewModels.MvxFormsViewModel>();
    }
}
```

4. Derive MainActivity and AppDelegate from MvxFormsAppCompatActivity and MvxFormsApplicationDelegate for initializing renderer.

**C#**

```
public class MainActivity : MvxFormsAppCompatActivity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        TabLayoutResource = Resource.Layout.Tabbar;
        ToolbarResource = Resource.Layout.Toolbar;
    }
}

public partial class AppDelegate : MvxFormsApplicationDelegate
{
    public override UIWindow Window { get; set; }
    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        SfListViewRenderer.Init();
    }
}
```

For more details, refer to this [documentation](#).

You can download the entire source code of this demo in the following link: [Source code](#).

Binding properties in MVVM pattern

*Binding ItemsSource*

ListView support to bind the [ItemsSource](#) property to populate the list view items from view model.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:MVVM"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
x:Class="MVVM.MainPage">
<ContentPage.BindingContext>
<local:BookInfoRepository/>
</ContentPage.BindingContext>
</ContentPage>
<syncfusion:SfListView x:Name="listView" ItemsSource="{Binding
BookInfoCollection}"/>
```

### C#

```
listView.SetBinding(SfListView.ItemsSourceProperty, new
Binding("BookInfoCollection", BindingMode.OneWay));
```

### C#

```
//ViewModel.cs
public class BookInfoRepository : INotifyPropertyChanged
{
    private ObservableCollection<BookInfo> bookInfoCollection;
    public event PropertyChangedEventHandler PropertyChanged;
    public ObservableCollection<BookInfo> BookInfoCollection
    {
        get { return bookInfoCollection; }
        set
        {
            this.bookInfoCollection = value;
            this.OnPropertyChanged("BookInfoCollection");
        }
    }
    public void OnPropertyChanged(string name)
    {
        if (this.PropertyChanged != null)
            this.PropertyChanged(this, new PropertyChangedEventArgs(name));
    }
    public BookInfoRepository()
    {
        GenerateNewBookInfo();
    }
    private void GenerateNewBookInfo()
    {
        BookInfoCollection = new ObservableCollection<BookInfo>();
    }
}
```



```
BookInfoCollection.Add(new BookInfo() { BookName = "Machine Learning Using C#", BookDescription = "You'll learn several different approaches to applying machine learning" });
BookInfoCollection.Add(new BookInfo() { BookName = "Object-Oriented Programming in C#", BookDescription = "Object-oriented programming is the de facto programming paradigm" });
BookInfoCollection.Add(new BookInfo() { BookName = "C# Code Contracts", BookDescription = "Code Contracts provide a way to convey code assumptions" });
}
```

### Binding SelectedItem

ListView 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:SfListView x:Name="listView"
SelectedItem="{Binding SelectedItem}"
ItemsSource="{Binding BookInfoCollection}"/>
```

### C#

```
listView.SetBinding(SfListView.SelectedItemProperty, new
Binding("SelectedItem", BindingMode.TwoWay));
```

### C#

```
//ViewModel.cs
public class BookInfoRepository : INotifyPropertyChanged
{
    private object selectedItem;
    public object SelectedItem
    {
        get { return this.selectedItem; }
        set
        {
            this.selectedItem = value;
            this.OnPropertyChanged("SelectedItem");
        }
    }
    public BookInfoRepository()
    {
        SelectedItem = BookInfoCollection[2];
    }
}
```

Download the entire sample from GitHub [here](#).

### Binding SelectedItems

ListView support to select multiple items through binding the [SelectedItems](#) property from view model with [ObservableCollection<object>](#) type. Set the [SelectionMode](#) property as [Multiple](#).

**XML**

```
<syncfusion:SfListView x:Name="listView"
SelectionMode="Multiple"
SelectedItems="{Binding SelectedItems}"
ItemsSource="{Binding BookInfoCollection}"/>
```

**C#**

```
listView.SelectionMode = SelectionMode.Multiple;
listView.SetBinding(SfListView.SelectedItemsProperty, new
Binding("SelectedItems", BindingMode.TwoWay));
```

**C#**

```
//ViewModel.cs
public class BookInfoRepository : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    private ObservableCollection<object> selectedItems;
    public ObservableCollection<object> SelectedItems
    {
        get { return this.selectedItems; }
        set
        {
            this.selectedItems = value;
            this.OnPropertyChanged("SelectedItems");
        }
    }
    public void OnPropertyChanged(string name)
    {
        if (this.PropertyChanged != null)
            this.PropertyChanged(this, new PropertyChangedEventArgs(name));
    }
    public BookInfoRepository()
    {
        SelectedItems = new ObservableCollection<object>();
        SelectedItems.Add(BookInfoCollection[1]);
        SelectedItems.Add(BookInfoCollection[2]);
    }
}
```

Download the entire sample from GitHub [here](#).

*Binding SelectionChanged event*

In ListView, the [SelectionChanged](#) event is raised once the selection process has been completed. MVVM for the [SelectionChanged](#) event can be achieved by binding through the event to command converter.

Refer [event to command](#) knowledge base to create the command for event using behavior.

**XML**

```
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding BookInfoCollection}">
<syncfusion:SfListView.Behaviors>
```

```
<local:EventToCommandBehavior EventName="SelectionChanged" Command="{Binding
SelectedItem}"
Converter="{StaticResource EventArgs}"/>
</syncfusion:SfListView.Behaviors>
</syncfusion:SfListView>
```

## C#

```
//ViewModel.cs
public class BookInfoRepository : INotifyPropertyChanged
{
    private Command<ItemSelectionChangedEventArgs> selectedItem
    public Command<ItemSelectionChangedEventArgs> SelectedItem
    {
        get { return this.selectedItem; }
        set
        {
            this.selectedItem = value;
            this.OnPropertyChanged("SelectedItem");
        }
    }
    public BookInfoRepository()
    {
        selectedItem = new
        Command<ItemSelectionChangedEventArgs>(OnSelectionChanged);
    }
    ///<summary>
    ///Remove the selected item
    ///</summary>
    public void OnSelectionChanged(ItemSelectionChangedEventArgs obj)
    {
        var eventArgs = obj as ItemSelectionChangedEventArgs;
        var item= eventArgs.AddedItems[0];
        this.bookInfoCollection.Remove(this.BookInfoCollection.FirstOrDefault(x => x
        == item));
    }
}
```

Download the entire sample from GitHub [here](#).

### Binding SelectionChanging event

In ListView, the [SelectionChanging](#) event will be raised when selecting an item at the execution time. MVVM for the [SelectionChanging](#) event can be achieved by binding through the event to command converter.

Refer [event to command](#) knowledge base to create the command for event using behavior.

## XML

```
<syncfusion:SfListView x:Name="listView"
ItemsSource="{Binding BookInfoCollection}">
<syncfusion:SfListView.Behaviors>
<local:EventToCommandBehavior EventName="SelectionChanging"
Command="{Binding SelectedItem}"
Converter="{StaticResource EventArgs}"/>
</syncfusion:SfListView.Behaviors>
```

---

```
</syncfusion:SfListView>
```

---

**C#**

```
//ViewModel.cs
public class BookInfoRepository : INotifyPropertyChanged
{
    private Command<ItemSelectionChangingEventArgs> selectedItem
    public Command<ItemSelectionChangingEventArgs> SelectedItem
    {
        get { return this.selectedItem; }
        set
        {
            this.selectedItem = value;
            this.OnPropertyChanged("SelectedItem");
        }
    }
    public BookInfoRepository()
    {
        selectedItem = new
        Command<ItemSelectionChangingEventArgs>(OnSelectionChanging);
    }
    ///<summary>
    ///To disable the selection for particular item
    ///</summary>
    public void OnSelectionChanging(ItemSelectionChangingEventArgs obj)
    {
        var eventArgs = obj as ItemSelectionChangingEventArgs;
        if (eventArgs.AddedItems.Count > 0 && eventArgs.AddedItems[0] ==
        this.BookInfoCollection[0])
            eventArgs.Cancel = true;
    }
}
```

Download the entire sample from GitHub [here](#).

---

**Note:** Similarly, you can bind the [ItemTapped](#), [ItemDoubleTapped](#), and [ItemHolding](#) event.

---

*Handling ItemTapped action*

ListView supports binding the [TapCommand](#) property with the item tapped action from view model, where you can write navigation or any other action code in the execution. When defining the command, [ItemTappedEventArgs](#) will be passed as command parameter which has item information in execution.

You can define the command parameter for `TapCommand` using [TappedCommandParameter](#), where you can get the element reference passed in view model.

**XML**

```
<syncfusion:SfListView x:Name="listView"
    TapCommand="{Binding TapCommand}"
    ItemsSource="{Binding BookInfoCollection}"/>
```

**C#**

```
listView.SetBinding(SfListView.TapCommandProperty, new Binding("TapCommand",
BindingMode.OneWay));
```

**C#**

```
//ViewModel.cs
public class BookInfoRepository : INotifyPropertyChanged
{
    private Command tapCommand;
    public Command TapCommand
    {
        get { return tapCommand; }
        protected set { tapCommand = value; }
    }
    public BookInfoRepository()
    {
        tapCommand = new Command(OnItemTapped);
    }
    ///<summary>
    ///To display tapped item content
    ///</summary>
    public void OnItemTapped(object obj)
    {
        var eventArgs = obj as Syncfusion.ListView.XForms.ItemTappedEventArgs;
        var bookName = (eventArgs.ItemData as BookInfo).BookName;
        var bookDescription = (eventArgs.ItemData as BookInfo).BookDescription;
        var display = Application.Current.MainPage.DisplayAlert(bookName,
        "Description:" + bookDescription, "Ok");
    }
}
```

Download the entire sample from GitHub [here]<https://github.com/SyncfusionExamples/xamarin-forms-listview-itemtapped-mvvm>).

*Handling ItemHolding action*

ListView supports binding the [HoldCommand](#) property with the item holding action from view model, where you can write navigation or any other action code in the execution. When defining the command, [ItemHoldingEventArgs](#) will be passed as command parameter which has item information in execution.

You can define the command parameter for the [HoldCommand](#) using [HoldCommandParameter](#), where you can get the element reference passed in view model.

**XML**

```
<syncfusion:SfListView x:Name="listView"
HoldCommand="{Binding HoldCommand}"
ItemsSource="{Binding BookInfoCollection}"/>
```

**C#**

```
listView.SetBinding(SfListView.HoldCommandProperty, new
Binding("HoldCommand", BindingMode.OneWay));
```

**C#**

```
//ViewModel.cs
public class BookInfoRepository : INotifyPropertyChanged
{
    private Command holdCommand;
    public Command HoldCommand
    {
        get { return holdCommand; }
        protected set { holdCommand = value; }
    }
    public BookInfoRepository()
    {
        holdCommand = new Command(OnHold);
    }
    ///<summary>
    /// Displays the item holding content
    ///</summary>
    public void OnHold(object obj)
    {
        var eventArgs = obj as Syncfusion.ListView.XForms.ItemHoldingEventArgs;
        Application.Current.MainPage.DisplayAlert("Type Of Item :" +
        eventArgs.ItemType, "Item Tapped Position : + " +eventArgs.Position , "Ok");
    }
}
```

Download the entire sample from GitHub [here](#).

#### *Binding button command*

The contents loaded in the [ItemTemplate](#) can be bound from the view model using their commands or gestures, where you can customize the loaded content or any other action code needed in the call back. You will get the BindingContext of [ListViewItem](#) as the parameter in execution when defining the command button.

You can also get the reference of element bound as parameter by using command parameter of loaded elements.

#### **XML**

```
<syncfusion:SfListView x:Name="listView" AutoFitMode="Height"
SelectedItem="{Binding SelectedItem}"
ItemsSource="{Binding BookInfoCollection}">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Frame HasShadow="True" Margin="5,5,5,5" >
<Grid Padding="5">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="2*" />
</Grid.RowDefinitions>
<Button x:Name="bookName" Text="{Binding BookName}" Command="{Binding
Path=BindingContext.BackgroundColorCommand, Source={x:Reference listView}}"
CommandParameter="{x:Reference bookName}" BackgroundColor="Transparent"
FontAttributes="Bold" FontSize="19"/>
<Label Grid.Row="1" Text="{Binding BookDescription}" FontSize="15" />
</Grid>
</Frame>
</DataTemplate>
```

```
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
```

**C#**

```
listView.ItemTemplate = new DataTemplate(() =>
{
    var frame = new Frame();
    frame.HasShadow = true;
    frame.Margin = 5;
    var grid = new Grid();
    grid.Padding = 5;
    var button = new Button();
    Binding binding = new Binding();
    binding.Path = "BookName";
    button.SetBinding(Button.TextProperty, binding);
    button.Command = this.viewModel.BackgroundColorCommand;
    button.BackgroundColor = Color.Transparent;
    button.FontAttributes = FontAttributes.Bold;
    button.FontSize = 19;
    var label = new Label();
    Binding bind = new Binding();
    bind.Path = "BookDescription";
    label.SetBinding(Label.TextProperty, bind);
    label.FontSize = 15;
    grid.Children.Add(button);
    grid.Children.Add(label, 0, 1);
    frame.Content = grid;
    return frame;
});
```

**C#**

```
public class BookInfoRepository : INotifyPropertyChanged
{
    private Command backgroundColorCommand;
    public Command BackgroundColorCommand
    {
        get { return backgroundColorCommand; }
        protected set { backgroundColorCommand = value; }
    }
    public BookInfoRepository()
    {
        backgroundColorCommand = new Command(OnButtonTapped);
    }
    ///<summary>
    ///To display tapped item content
    ///</summary>
    public void OnButtonTapped(object obj)
    {
        var firstButton = obj as Button;
        firstButton.BackgroundColor = Color.AliceBlue;
    }
}
```

Download the entire sample from GitHub [here](#).

#### *Processing LoadMore*

ListView supports binding the [LoadMoreOption](#), [LoadMoreCommand](#), and [IsBusy](#) properties from view model to load more number of items at runtime. [LoadMoreOption](#) enables load more manually or automatically the items when loading the items at runtime. [LoadMoreCommand](#) executes to load the items form view model. The [IsBusy](#) property notifies that the items are populating from view model to show or hide the load more view.

The [IsBusy](#) property in view model shows the busy indicator when populating the [ItemsSource](#).

#### **XML**

```
<syncfusion:SfListView x:Name="listView"
    LoadMoreOption="Auto"
    LoadMoreCommand="{Binding LoadMoreItemsCommand}"
    LoadMoreCommandParameter="{Binding Source={x:Reference Name=listView}}"
    IsBusy="{Binding IsBusy}"
    ItemsSource="{Binding BookInfoCollection}">
```

#### **C#**

```
listView.LoadMoreOption = LoadMoreOption.Auto;
listView.SetBinding(SfListView.LoadMoreCommandProperty, new
Binding("LoadMoreItemsCommand", BindingMode.OneWay));
listView.LoadMoreCommandParameter = listView;
listView.SetBinding(SfListView.IsBusyProperty, new Binding("IsBusy",
BindingMode.OneWay));
```

#### **C#**

```
public class ViewModel:INotifyPropertyChanged
{
    private bool isBusy;
    public bool IsBusy
    {
        get { return isBusy; }
        set
        {
            this.isBusy = value;
            RaisePropertyChanged("IsBusy");
        }
    }
    private int totalItems = 22;
    public Command<object> LoadMoreItemsCommand { get; set; }
    public ViewModel()
    {
        BookInfoCollection = new ObservableCollection<BookInfo>();
        AddBookInfos(0, 3);
        LoadMoreItemsCommand = new Command<object>(LoadMoreItems);
    }
    private bool CanLoadMoreItems(object obj)
    {
        if (BookInfoCollection.Count >= totalItems)
            return false;
    }
}
```



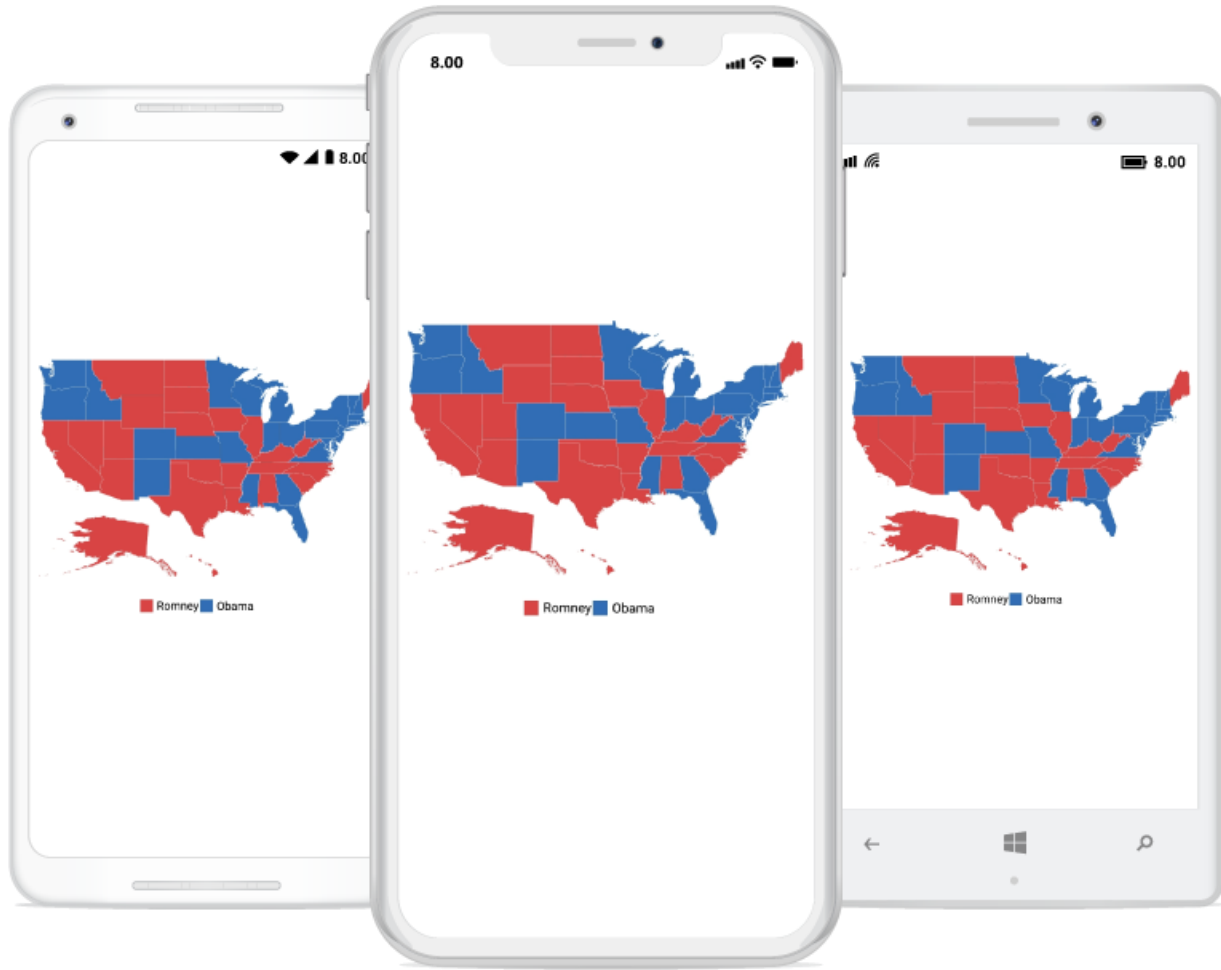
```
return true;
}
private async void LoadMoreItems(object obj)
{
    var listview = obj as Syncfusion.ListView.XForms.SfListView;
    try
    {
        IsBusy = true;
        await Task.Delay(1000);
        var index = BookInfoCollection.Count;
        var count = index + 3 >= totalItems ? totalItems - index : 3;
        AddBookInfos(index, 3);
    }
    catch
    {
    }
    finally
    {
        IsBusy = false;
    }
}
private void AddBookInfos(int index, int count)
{
    int bookNameCount = 0;
    int bookDescriptionCount = 0;
    for (int i = index; i < index + count; i++)
    {
        if (bookNameCount == 11)
            bookNameCount = 0;
        if (bookDescriptionCount == 11)
            bookDescriptionCount = 0;
        BookInfoCollection.Add(new BookInfo() { BookName = bookNames[bookNameCount]
            , BookDescription = bookDescription[bookDescriptionCount] });
        bookNameCount++;
        bookDescriptionCount++;
    }
}
}
```

You can download the entire sample from GitHub [here](#).

## SfMaps

### Overview

The maps control for Xamarin.Forms provides a graphical representation of geographical data. It is used to represent the statistical data of a particular geographical area on earth by panning and zooming. The maps control supports enhanced data visualization with bubbles and labels using data bound to map.



## Key features

- **Layers:** Visualize maps. A map can accommodate one or more layers.
- **Map elements:** Show additional information on map with customized appearance using various set of elements, shapes, bubbles, markers, legends, labels, and data items.
- **User interaction:** Enables options such as zooming, panning, and map selection to make the map elements more interactive.

## Getting Started

This section explains the steps required to configure the maps control and customize its elements.

### Adding SfMaps reference

You can add SfMaps reference using one of the following methods:

#### Method 1: Adding SfMaps reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfMaps to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfMaps](https://www.syncfusion.com/nuget/packages/syncfusion.xamarin.sfmmaps), and then install it.

![Adding SfMaps reference from NuGet](Images/Adding SfMaps reference.png)

---

**Note:** Install the same version of SfMaps NuGet in all the projects.

---

### Method 2: Adding SfMaps reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfMaps control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfMaps assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfMaps.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfMaps.Android.dll Syncfusion.SfMaps.XForms.Android.dll Syncfusion.SfMaps.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfMaps.iOS.dll Syncfusion.SfMaps.XForms.iOS.dll Syncfusion.SfMaps.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfMaps.UWP.dll Syncfusion.SfMaps.XForms.UWP.dll Syncfusion.SfMaps.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Launching an application on each platform with SfMaps.

To use the SfMaps control inside an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

### iOS

To launch the SfMaps in iOS, call the `SfMapsRenderer.Init()` in the `FinishedLaunching` overridden method of the AppDelegate class after the Xamarin.Forms Framework has been initialized and before the `LoadApplication` is called, as demonstrated in the following code example.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.SfMaps.XForms.iOS.SfMapsRenderer.Init ();
    LoadApplication (new App ());
    ...
}
```

### Universal Windows Platform (UWP)

You need to initialize the maps view assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with maps in Release mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    // Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.SfMaps.XForms.UWP.SfMapsRenderer).
        GetTypeInfo().Assembly);
    // Replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Android

The Android platform does not require any additional configuration to render the maps.

#### Adding namespace

Add the following namespace.

#### XML

```
xmlns:maps="clr-
namespace:Syncfusion.SfMaps.XForms;assembly=Syncfusion.SfMaps.XForms"
```

#### C#

```
using Syncfusion.SfMaps.XForms;
```

### Initializing maps

Create an instance for the maps control, and add it as content.

#### XML

```
<maps:SfMaps >  
</maps:SfMaps>
```

#### C#

```
SfMaps map = new SfMaps();  
this.Content = map;
```

### Adding layers

The maps control is maintained through [layers](#). It can accommodate one or more shape file layers.

#### XML

```
<maps:SfMaps>  
<maps:SfMaps.Layers>  
<maps:ShapeFileLayer/>  
</maps:SfMaps.Layers>  
</maps:SfMaps>
```

#### C#

```
SfMaps map = new SfMaps();  
ShapeFileLayer layer = new ShapeFileLayer();  
map.Layers.Add(layer);  
this.Content = map;
```

### Adding shape files

A shape file is a set of files that is stored in a non-topological geometry with the attribute information for the spatial features and records in a data set.

The maps control supports reading and loading the shape files. A shape file can be a set of files or a single file. Generally, a shape file contains the following files:

- Main file (.shp)
- dBase file (.dbf)

#### Android

- Add necessary shape files to the Assets folder of ProjectFileName.Droid.
- Right-click the added shape file, and navigate to properties.
- Choose the **AndroidAsset** option under BuildAction of respective shape file.

#### iOS

- Add necessary shape files to the Resources folder of ProjectFileName.iOS.
- Right-click the added shape file, and navigate to properties.

- Choose the **BundleResource** option under BuildAction of respective shape file.

#### UWP

- Add necessary shapes files in a folder, and name it as **ShapeFiles**. Add this **ShapeFiles** folder into the **Assets** folder of ProjectFileName.UWP.
- Right-click the added shape file, and navigate to properties.
- Choose the **EmbeddedResource** option under BuildAction of respective shape file.

The [Uri](#) property in shape file layer is used to retrieve the location of the shape file that is added.

#### XML

```
<maps:SfMaps x:Name="sfmap" BackgroundColor="White" >
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" >
  </maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

#### C#

```
SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
map.Layers.Add(layer);
this.Content = map;
```

The following screenshot illustrates the result of adding shape files.



#### GeoJSON support

The maps control supports reading and loading the GeoJSON files. The GeoJSON file contains attribute information for the spatial features and coordinates in a dataset.

#### XML

```
<maps:ShapeFileLayer Uri="usa_state.json">
</maps:ShapeFileLayer>
```

### C#

```
ShapeFileLayer layer = new ShapeFileLayer();  
layer.Uri = "usa_state.json";
```

#### Data binding

Data can be bound to the shape file layer using the [ItemsSource](#), [ShapeIDPath](#), and [ShapeIDTableField](#) properties.

The [Populate data](#) section gives the detailed explanation of data binding.

### XML

```
<maps:SfMaps.Layers >  
<maps:ShapeFileLayer Uri="usa_state.shp" ItemsSource="{Binding Data}"  
ShapeIDPath="State" ShapeIDTableField="STATE_NAME" >  
</maps:ShapeFileLayer>  
</maps:SfMaps.Layers>
```

### C#

```
ShapeFileLayer layer = new ShapeFileLayer();  
layer.Uri = "usa_state.shp";  
layer.ItemsSource =viewModel.Data;  
layer.ShapeIDTableField = "STATE_NAME";  
layer.ShapeIDPath = "State";  
map.Layers.Add(layer);
```

#### Adding markers

Markers are used to identify the shapes. They can be added to the shape file layers as demonstrated in the following code sample. Markers can be customized using the [MarkerSettings](#) property in the shape file layer.

The detailed explanation of marker and its customization are provided under [Markers](#) section.

### XML

```
<maps:ShapeFileLayer.Markers>  
<maps:MapMarker Label = "California" Latitude = "37" Longitude = "-120">  
</maps:MapMarker>  
</maps:ShapeFileLayer.Markers>
```

### C#

```
MapMarker marker = new MapMarker();  
marker.Label = "California";  
marker.Latitude = "37";  
marker.Longitude = "-120";  
layer.Markers.Add(marker);
```

## Color mapping

The color mapping support allows you customize the shape colors based on the underlying value of shape received from the bound data. Both the range color mapping and equal color mapping are supported in maps.

The detailed explanation of color mapping is provided in [colorMapping](#) section.

### XML

```
<maps:ShapeSetting.ColorMappings>
<maps:EqualColorMapping Color="#D84444" LegendLabel="Romney" Value =
"Romney"></maps:EqualColorMapping>
<maps:EqualColorMapping Color="#316DB5" LegendLabel="Obama"
Value="Obama"></maps:EqualColorMapping>
</maps:ShapeSetting.ColorMappings>
```

### C#

```
EqualColorMapping colorMapping = new EqualColorMapping();
colorMapping.Color = Color.FromHex("#D84444");
colorMapping.LegendLabel = "Romney";
colorMapping.Value = "Romney";
EqualColorMapping colorMapping1 = new EqualColorMapping();
colorMapping1.Color = Color.FromHex("#316DB5");
colorMapping1.LegendLabel = "Obama";
colorMapping1.Value = "Obama";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ColorMappings.Add(colorMapping);
shapeSetting.ColorMappings.Add(colorMapping1);
```

## Adding legends

Legends interpret what the map displays. They can be added to the shape file layer as demonstrated in the following code sample. Legends will be displayed based on the data bound to the layer, and color mapping plays a major role in enabling legends.

The detailed explanation of legend is provided under [Legend](#) section.

### XML

```
<maps:ShapeFileLayer.LegendSettings>
<maps:MapLegendSetting ShowLegend="true" LegendPosition="75,90"/>
</maps:ShapeFileLayer.LegendSettings>
```

### C#

```
MapLegendSetting setting = new MapLegendSetting();
setting.ShowLegend = true;
setting.LegendPosition = new Point(75, 90);
layer.LegendSettings = setting;
```

The following code sample gives you the complete code for map with markers and legends.

### XML

```
<maps:SfMaps x:Name="sfmap" BackgroundColor="White" >
```



```

<maps:SfMaps.Layers >
<maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
ShapeIDTableField="STATE_NAME"
ItemsSource="{Binding Data}"
>
<maps:ShapeFileLayer.Markers>
<maps:MapMarker Label = "California" Latitude = "37" Longitude = "-120">
</maps:MapMarker>
</maps:ShapeFileLayer.Markers>
<maps:ShapeFileLayer.MarkerSettings>
<maps:MapMarkerSetting IconColor="LimeGreen" IconSize="25"
LabelColor="White" LabelSize="20">
</maps:MapMarkerSetting>
</maps:ShapeFileLayer.MarkerSettings>
<maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeSetting ShapeColorValuePath="Candidate"
ShapeValuePath="Candidate" >
<maps:ShapeSetting.ColorMappings>
<maps:EqualColorMapping Color="#D84444" LegendLabel="Romney" Value =
"Romney"></maps:EqualColorMapping>
<maps:EqualColorMapping Color="#316DB5" LegendLabel="Obama"
Value="Obama"></maps:EqualColorMapping>
</maps:ShapeSetting.ColorMappings>
</maps:ShapeSetting>
</maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeFileLayer.LegendSettings>
<maps:MapLegendSetting ShowLegend="True"
LegendPosition="30,70">
<maps:MapLegendSetting.IconSize>
<Size Width="20" Height="20"></Size>
</maps:MapLegendSetting.IconSize>
</maps:MapLegendSetting>
</maps:ShapeFileLayer.LegendSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>

```

## C#

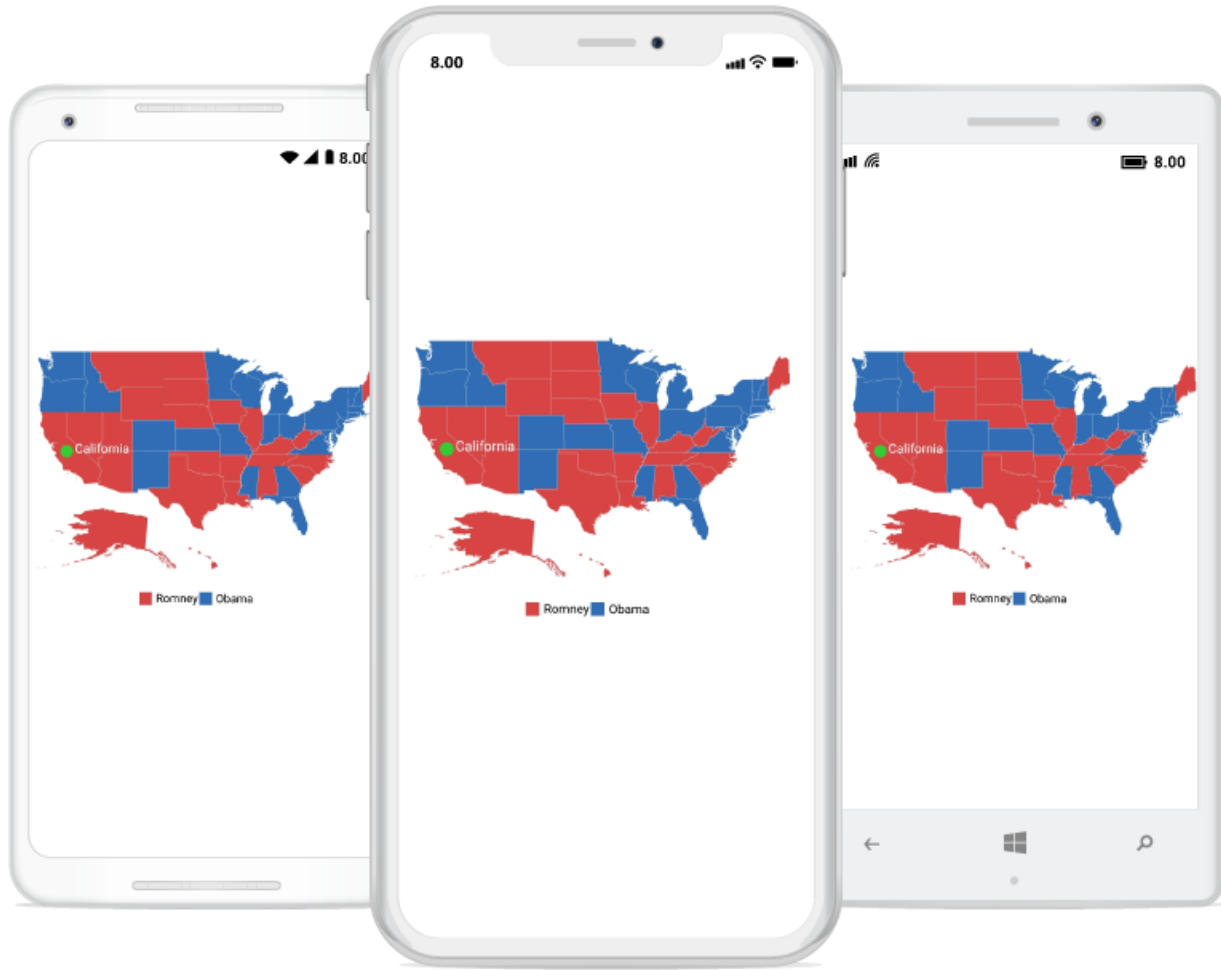
```

SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource = viewModel.Data;
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "State";
map.Layers.Add(layer);
MapLegendSetting legendSetting = new MapLegendSetting();
legendSetting.ShowLegend = true;
legendSetting.LegendPosition = new Point(30, 70);
legendSetting.IconSize = new Size(20, 20);
layer.LegendSettings = legendSetting;
MapMarker marker = new MapMarker();
marker.Label = "California";
marker.Latitude = "37";
marker.Longitude = "-120";

```

```
layer.Markers.Add(marker);
MapMarkerSetting markerSetting = new MapMarkerSetting();
markerSetting.IconColor = Color.LimeGreen;
markerSetting.IconSize = 25;
markerSetting.LabelColor = Color.White;
markerSetting.LabelSize = 20;
layer.MarkerSettings = markerSetting;
EqualColorMapping colorMapping = new EqualColorMapping();
colorMapping.Color = Color.FromHex("#D84444");
colorMapping.LegendLabel = "Romney";
colorMapping.Value = "Romney";
EqualColorMapping colorMapping1 = new EqualColorMapping();
colorMapping1.Color = Color.FromHex("#316DB5");
colorMapping1.LegendLabel = "Obama";
colorMapping1.Value = "Obama";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeValuePath = "Candidate";
shapeSetting.ShapeColorValuePath = "Candidate";
shapeSetting.ColorMappings.Add(colorMapping);
shapeSetting.ColorMappings.Add(colorMapping1);
layer.ShapeSettings = shapeSetting;
this.Content = map;
```

The following screenshot illustrates the result of the above code sample.



You can download the complete [Getting started](#) sample.

## Populate Data

This section explains how to populate shape data for providing data input to the maps control and usage of the `ItemsSource` property.

### Data binding

The maps control supports data binding using the `ItemsSource` property in the shape layers.

The following properties in shape layers are used for binding data in the maps control:

- `ItemsSource`
- `ShapeIDPath`
- `ShapeIDTableField`

### *ItemsSource*

The [ItemsSource](#) property accepts the collection values as input.

### *ShapeIDPath*

The [ShapeIDPath](#) property refers the data ID in `ItemsSource`.

### ShapeIDTableField

The [ShapeIDTableField](#) property is similar to the [ShapeIDPath](#); it refers to the column name in the data property of shape layers to identify the shape. When the values of the [ShapeIDPath](#) property in the [ItemsSource](#) property and the values of [ShapeIDTableField](#) in the data property match, the associated object from the ItemsSource will be bound to the corresponding shape.

### XML

```
<maps:SfMaps x:Name="sfmap" BackgroundColor="White" >
  <maps:SfMaps.BindingContext>
  <local:ViewModel></local:ViewModel>
</maps:SfMaps.BindingContext>
<maps:SfMaps.Layers >
  <maps:ShapeFileLayer Uri="usa_state.shp" ItemsSource="{Binding Data}"
  ShapeIDPath="State" ShapeIDTableField="STATE_NAME" >
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource =viewModel.Data;
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "State";
map.BackgroundColor = Color.White;
map.Layers.Add(layer);
```

The following code sample demonstrates referring the United States election data as items source.

### C#

```
public class ViewModel
{
  public ObservableCollection<ElectionData> Data { get; set; }
  public ViewModel()
  {
    Data = new ObservableCollection<ElectionData>();
    Data.Add(new ElectionData("Alabama", "Romney", 9));
    Data.Add(new ElectionData("Alaska", "Romney", 3));
    Data.Add(new ElectionData("Arizona", "Romney", 11));
    Data.Add(new ElectionData("Arkansas", "Romney", 6));
    Data.Add(new ElectionData("California", "Romney", 55));
    Data.Add(new ElectionData("Colorado", "Obama", 9));
    Data.Add(new ElectionData("Connecticut", "Obama", 7));
    Data.Add(new ElectionData("Delaware", "Obama", 3));
    Data.Add(new ElectionData("District of Columbia", "Obama", 3));
    Data.Add(new ElectionData("Florida", "Obama", 29));
    Data.Add(new ElectionData("Georgia", "Obama", 16));
    Data.Add(new ElectionData("Hawaii", "Romney", 4));
    Data.Add(new ElectionData("Idaho", "Obama", 4));
    Data.Add(new ElectionData("Illinois", "Romney", 20));
    Data.Add(new ElectionData("Indiana", "Obama", 11));
    Data.Add(new ElectionData("Iowa", "Romney", 6));
```

```
Data.Add(new ElectionData("Kansas", "Obama", 6));
Data.Add(new ElectionData("Kentucky", "Romney", 8));
Data.Add(new ElectionData("Louisiana", "Romney", 8));
Data.Add(new ElectionData("Maine", "Romney", 4));
Data.Add(new ElectionData("Maryland", "Obama", 10));
Data.Add(new ElectionData("Massachusetts", "Obama", 11));
Data.Add(new ElectionData("Michigan", "Obama", 16));
Data.Add(new ElectionData("Minnesota", "Obama", 10));
Data.Add(new ElectionData("Mississippi", "Obama", 6));
Data.Add(new ElectionData("Missouri", "Obama", 10));
Data.Add(new ElectionData("Montana", "Romney", 3));
Data.Add(new ElectionData("Nebraska", "Romney", 5));
Data.Add(new ElectionData("Nevada", "Romney", 6));
Data.Add(new ElectionData("New Hampshire", "Obama", 4));
Data.Add(new ElectionData("New Jersey", "Obama", 14));
Data.Add(new ElectionData("New Mexico", "Obama", 5));
Data.Add(new ElectionData("New York", "Obama", 29));
Data.Add(new ElectionData("North Carolina", "Romney", 15));
Data.Add(new ElectionData("North Dakota", "Romney", 3));
Data.Add(new ElectionData("Ohio", "Obama", 18));
Data.Add(new ElectionData("Oklahoma", "Romney", 7));
Data.Add(new ElectionData("Oregon", "Obama", 7));
Data.Add(new ElectionData("Pennsylvania", "Obama", 20));
Data.Add(new ElectionData("Rhode Island", "Obama", 4));
Data.Add(new ElectionData("South Carolina", "Romney", 9));
Data.Add(new ElectionData("South Dakota", "Romney", 3));
Data.Add(new ElectionData("Tennessee", "Romney", 11));
Data.Add(new ElectionData("Texas", "Romney", 38));
Data.Add(new ElectionData("Utah", "Romney", 6));
Data.Add(new ElectionData("Vermont", "Obama", 3));
Data.Add(new ElectionData("Virginia", "Obama", 13));
Data.Add(new ElectionData("Washington", "Obama", 12));
Data.Add(new ElectionData("West Virginia", "Romney", 5));
Data.Add(new ElectionData("Wisconsin", "Obama", 10));
Data.Add(new ElectionData("Wyoming", "Romney", 3));
}
}
public class ElectionData
{
    public ElectionData(string state, string candidate, int electors)
    {
        State = state;
        Candidate = candidate;
        Electors = electors;
    }
    public string State
    {
        get;
        set;
    }
    public string Candidate
    {
        get;
        set;
    }
    public int Electors
    {
```

```
get;
set;
}
}
```

## Layers

The maps control is maintained through [Layers](#); a map can accommodate one or more layers.

The maps control consists the following two layers:

- Imagery layer
- Shape file layer

### Imagery layer

The [MapsProvider](#) section explains about the imagery layer.

### Shape file layer

Using shape file layer, custom shape files can be rendered and the shapes can be customized.

### Shape settings

This section defines how to customize the shapes in a map.

You can customize a shape's fill, stroke, and stroke thickness using the [ShapeFill](#), [ShapeStroke](#), [ShapeStrokeThickness](#) properties.

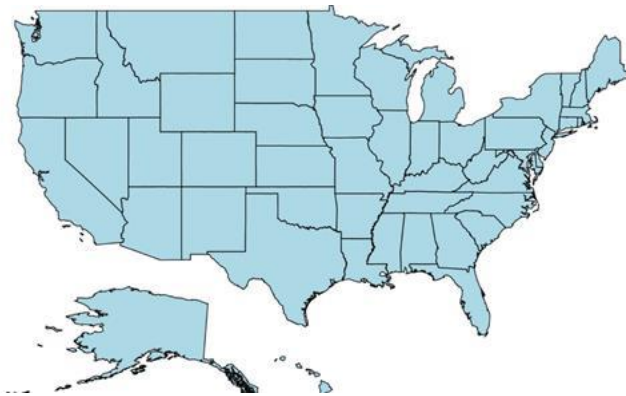
Refer to the following code sample for customizing shapes.

### XML

```
<maps:SfMaps>
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" >
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeFill="LightBlue" ShapeStroke="Black"
          ShapeStrokeThickness="2" >
        </maps:ShapeSetting>
      </maps:ShapeFileLayer.ShapeSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
SfMaps maps = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource = GetDataSource();
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "State";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeFill = Color.LightBlue;
shapeSetting.ShapeStroke = Color.Black;
shapeSetting.ShapeStrokeThickness = 2;
layer.ShapeSettings = shapeSetting;
maps.Layers.Add(layer);
```



To customize the shapes based on the bound values, use the following properties:

[ShapeValuePath](#): Field value that has to be bound for each shape.

[ShapeColorValuePath](#): Field value that has to be bound to determine the color.

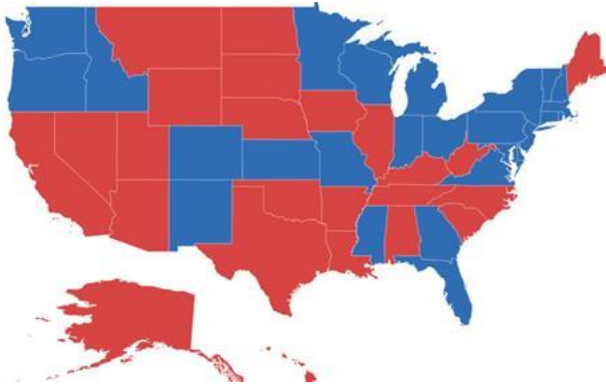
### XML

```
<maps:SfMaps>
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp"
      ShapeIDPath="State" ShapeIDTableField="STATE_NAME" >
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeColorValuePath="Candidate"
          ShapeValuePath="Candidate">
          <maps:ShapeSetting.ColorMappings>
            <maps:EqualColorMapping Color="#D84444" Value="Romney"
              LegendLabel="Romney"></maps:EqualColorMapping>
            <maps:EqualColorMapping Color="#316DB5" Value="Obama"
              LegendLabel="Obama"></maps:EqualColorMapping>
          </maps:ShapeSetting.ColorMappings>
        </maps:ShapeSetting>
      </maps:ShapeFileLayer.ShapeSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource = GetDataSource();
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "State";
map.Layers.Add(layer);
EqualColorMapping colorMapping = new EqualColorMapping();
colorMapping.Color = Color.FromHex("#D84444");
colorMapping.LegendLabel = "Romney";
colorMapping.Value = "Romney";
EqualColorMapping colorMapping1 = new EqualColorMapping();
colorMapping1.Color = Color.FromHex("#316DB5");
colorMapping1.LegendLabel = "Obama";
```

```
colorMapping1.Value = "Obama";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeValuePath = "Candidate";
shapeSetting.ShapeColorValuePath = "Candidate";
shapeSetting.ColorMappings.Add(colorMapping);
shapeSetting.ColorMappings.Add(colorMapping1);
layer.ShapeSettings = shapeSetting;
```



#### *Customize selected shapes*

To customize the selected shapes alone, use the following properties:

[SelectedShapeColor](#): Sets the color for selected shapes in a map.

[SelectedShapeStroke](#): Sets the border color for selected shapes in a map.

[SelectedShapeStrokeThickness](#): Sets the border thickness for selected shapes in a map.

#### **XML**

```
<maps:ShapeFileLayer.ShapeSettings>
  <maps:ShapeSetting ShapeFill="LightBlue" ShapeStroke="Black"
    ShapeStrokeThickness="1" >
  </maps:ShapeSetting>
</maps:ShapeFileLayer.ShapeSettings>
```

#### **C#**

```
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.SelectedShapeColor = Color.Green;
shapeSetting.SelectedShapeStroke = Color.Black;
shapeSetting.SelectedShapeStrokeThickness = 1;
layer.ShapeSettings = shapeSetting;
```





### Events

The [ShapeSelected](#) event will be triggered when a map shape is selected. A corresponding model data is passed as an argument. The ShapeSelected event has been deprecated.

The [ShapeSelectionChanged](#) event will be triggered when changing the the map shapes selection with corresponding argument as model [Data](#). Selection state of the shape will be changed using the [IsSelected](#) property

### XML

```
<maps:ShapeFileLayer ShapeSelectionChanged
="ShapeLayer_ShapeSelectionChanged" />
```

### C#

```
private void ShapeLayer_ShapeSelectionChanged(object sender,
ShapeSelectedEventArgs e)
{
    AgricultureData data = e.Data as AgricultureData;
    bool IsSelected = e.IsSelected;
}
```



### Sublayer

The shape layers are the core layers of the maps. Multiple layers can be added to the shape layers as sublayers within the shape layers.

### Adding multiple layers in the map

Multiple layers can be added to the maps using [Sublayer](#).

### Displaying layer in the view

The [BaseMapIndex](#) property allows drill-down from main layer to another layer.

In the ShapeSelected event, the *BaseMapIndex* property has been used to change the layer when Asia shape is selected.

### XML

```
<Grid>
  <Grid.BindingContext>
    <local:DrilldownViewModel/>
  </Grid.BindingContext>
  <maps:SfMaps x:Name="map">
    <maps:SfMaps.Layers>
      <maps:ShapeFileLayer EnableSelection="True" x:Name="layer1" Uri="world-
map.shp" ItemsSource="{Binding DataSource}" ShapeIDPath="Country"
ShapeIDTableField="admin"
ShapeSelectionChanged="ShapeFileLayer_ShapeSelected">
        <maps:ShapeFileLayer.ShapeSettings>
          <maps:ShapeSetting ShapeColorValuePath="Continent">
            <maps:ShapeSetting.ColorMappings>
              <maps:EqualColorMapping Color="#C13664" Value="North America"
            ></maps:EqualColorMapping>
              <maps:EqualColorMapping Color="#9C3367" Value="South
America"></maps:EqualColorMapping>
              <maps:EqualColorMapping Color="#80306A"
Value="Africa"></maps:EqualColorMapping>
              <maps:EqualColorMapping Color="#622D6C"
Value="Europe"></maps:EqualColorMapping>
              <maps:EqualColorMapping Color="#462A6D"
Value="Asia"></maps:EqualColorMapping>
              <maps:EqualColorMapping Color="#2A2870"
Value="Australia"></maps:EqualColorMapping>
            </maps:ShapeSetting.ColorMappings>
          </maps:ShapeSetting>
        </maps:ShapeFileLayer.ShapeSettings>
      </maps:ShapeFileLayer>
      <maps:ShapeFileLayer x:Name="layer2" Uri="asia.shp">
        <maps:ShapeFileLayer.ShapeSettings>
          <maps:ShapeSetting ShapeFill="#462A6D"/>
        </maps:ShapeFileLayer.ShapeSettings>
      </maps:ShapeFileLayer>
    </maps:SfMaps.Layers>
  </maps:SfMaps>
  <Label x:Name="label" Grid.Row="1" HorizontalOptions="Center"
VerticalOptions="Center" HorizontalTextAlignment="Center"
VerticalTextAlignment="Center" Text="Click on a Asia shape to drill down"/>
</Grid>
```

### C#

```
private void ShapeFileLayer_ShapeSelected(object sender,
Syncfusion.SfMaps.XForms.ShapeSelectedEventArgs e)
{
```

```
map.BaseMapIndex = 1;
label.IsVisible = false;
}
public class DrilldownViewModel
{
    public DrilldownViewModel()
    {
        DataSource = new ObservableCollection<DrilldownModel>();
        DataSource.Add(new DrilldownModel("Afghanistan", "Asia"));
        DataSource.Add(new DrilldownModel("Angola", "Africa"));
        DataSource.Add(new DrilldownModel("Albania", "Europe"));
        DataSource.Add(new DrilldownModel("United Arab Emirates", "Asia"));
        DataSource.Add(new DrilldownModel("Argentina", "South America"));
        DataSource.Add(new DrilldownModel("Armenia", "Asia"));
        DataSource.Add(new DrilldownModel("French Southern and Antarctic Lands",
            "Seven seas (open ocean)"));
        DataSource.Add(new DrilldownModel("Australia", "Australia"));
        //..
        //..
        DataSource.Add(new DrilldownModel("Zambia", "Africa"));
        DataSource.Add(new DrilldownModel("Zimbabwe", "Africa"));
    }
    public ObservableCollection<DrilldownModel> DataSource { get; set; }
}
public class DrilldownModel
{
    public DrilldownModel(string country, string con)
    {
        this.Country = country;
        this.Continent = con;
    }
    public string Continent
    {
        get;
        set;
    }
    public string Country
    {
        get;
        set;
    }
}
```



## Shape Type

[SfMaps](#) allows to provide various shape types in [ShapeFileLayer](#) such as Polygon, Polyline, and Points.

### Polygon

Polygon is a two-dimensional surface stored as a sequence of points defining its exterior bounding ring and 0 or more interior rings. Polygons are always simple. Mostly the polygon shape type defines a group of land, water bodies, and other features that have a spatial extent.



### Polyline

The polyline is a shape that has a dimension of 1. It is called a simple line if it does not intersect itself. The polylines are often used to define linear features such as roads, rivers, and power lines. Mostly the shape file layer with the polyline shape type is used as [sublayer](#).

The following code example demonstrates the sublayer with polyline shape type. The roads (sublayer) of Bahrain (base layer) is displayed using the shape file layer of maps.

### XML

```
<maps:SfMaps>
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="Bahrain.shp">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeFill="#bcbcbc"/>
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.Sublayers>
        <maps:ShapeFileLayer Uri="roads.shp">
          <maps:ShapeFileLayer.ShapeSettings>
            <maps:ShapeSetting ShapeFill="Black" ShapeStrokeThickness="2"/>
          </maps:ShapeFileLayer.ShapeSettings>
        </maps:ShapeFileLayer>
      </maps:ShapeFileLayer.Sublayers>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

```

</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>

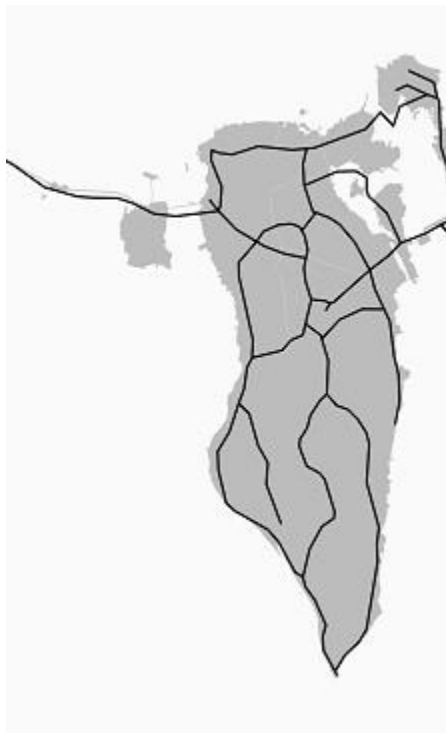
```

## C#

```

SfMaps maps = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "Bahrain.shp";
ShapeSetting layerSetting = new ShapeSetting();
layerSetting.ShapeFill = Color.FromHex("#bcbcbc");
layer.ShapeSettings = layerSetting;
ShapeFileLayer subLayer = new ShapeFileLayer();
subLayer.Uri = "roads.shp";
ShapeSetting subLayerSetting = new ShapeSetting();
subLayerSetting.ShapeFill = Color.Black;
subLayerSetting.ShapeStrokeThickness = 2;
subLayer.ShapeSettings = subLayerSetting;
layer.Sublayers.Add(subLayer);
maps.Layers.Add(layer);
this.Content = maps;

```



## Points

A point is shape with a dimension of 0 that occupies a single location in coordinate space. A point has a single x, y coordinate value. The points are often used to define features such as oil wells, landmarks, and elevations.

The following code example demonstrates the [sublayer](#) with the points shape type. The places (sublayer) of Australia (base layer) is displayed using the shape file layer of maps.

## XML

```
<maps:SfMaps>
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="australia.shp">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeFill="#bcbcbc"/>
      </maps:ShapeFileLayer.ShapeSettings>
    <maps:ShapeFileLayer.Sublayers>
      <maps:ShapeFileLayer Uri="points.shp">
        <maps:ShapeFileLayer.ShapeSettings>
          <maps:ShapeSetting ShapeFill="Black" />
        </maps:ShapeFileLayer.ShapeSettings>
      </maps:ShapeFileLayer>
    </maps:ShapeFileLayer.Sublayers>
  </maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

## C#

```
SfMaps maps = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "australia.shp";
ShapeSetting layerSetting = new ShapeSetting();
layerSetting.ShapeFill = Color.FromHex("#bcbcbc");
layer.ShapeSettings = layerSetting;
ShapeFileLayer subLayer = new ShapeFileLayer();
subLayer.Uri = "points.shp";
ShapeSetting subLayerSetting = new ShapeSetting();
subLayerSetting.ShapeFill = Color.Black;
subLayer.ShapeSettings = subLayerSetting;
layer.Sublayers.Add(subLayer);
maps.Layers.Add(layer);
this.Content = maps;
```



### Customization of Points

The size, shape, and position of the map points can be customized using the [MapPointIconSize], [MapPointIcon], [MapPointHorizontalAlignment] and [MapPointVerticalAlignment] properties of shape file layer.

### Map point icon

The shape of the map point is customized using the [MapPointIcon] property of ShapeFileLayer. SfMap supports the following map point icon types:

Circle, Rectangle, Square, Diamond, \* Star

### XML

```
<maps:SfMaps BackgroundColor="White">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="australia.shp">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeFill="#bcbcbc"/>
      </maps:ShapeFileLayer.ShapeSettings>
    <maps:ShapeFileLayer.Sublayers>
      <maps:ShapeFileLayer Uri="points.shp" MapPointIcon="Circle"
MapPointIconSize="10">
        <maps:ShapeFileLayer.ShapeSettings>
          <maps:ShapeSetting ShapeFill="Green" />
        </maps:ShapeFileLayer.ShapeSettings>
      </maps:ShapeFileLayer>
    </maps:ShapeFileLayer.Sublayers>
  </maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

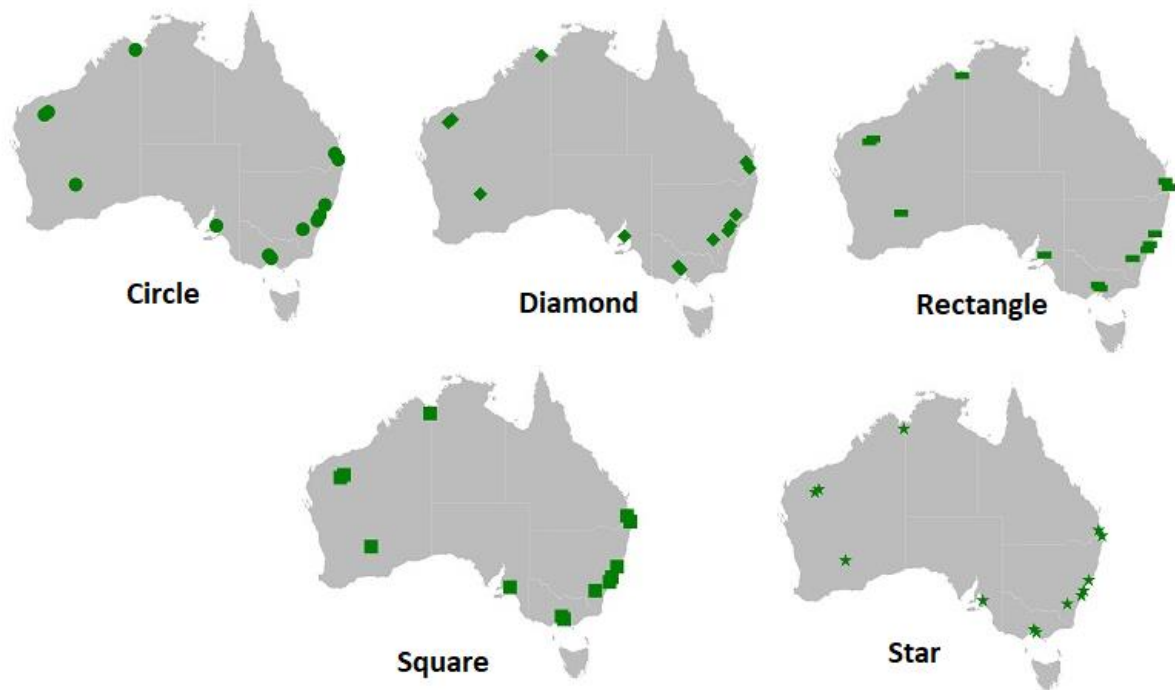


**C#**

```

SfMaps maps = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "australia.shp";
ShapeSetting layerSetting = new ShapeSetting();
layerSetting.ShapeFill = Color.FromHex("#bcbcbc");
layer.ShapeSettings = layerSetting;
ShapeFileLayer subLayer = new ShapeFileLayer();
subLayer.Uri = "points.shp";
subLayer.MapPointIcon = MapPointIcon.Circle;
subLayer.MapPointIconSize = 10;
ShapeSetting subLayerSetting = new ShapeSetting();
subLayerSetting.ShapeFill = Color.Green;
subLayer.ShapeSettings = subLayerSetting;
layer.Sublayers.Add(subLayer);
maps.Layers.Add(layer);

```

*Map point position*

The position of the map points can be customized using the `[MapPointHorizontalAlignment]` and `[MapPointVerticalAlignment]` properties of shape file layer. These properties allow to vary the position to Near, Far, and Center by considering the provided latitude, longitude, and the map point icon size.

**XML**

```

<maps:SfMaps BackgroundColor="White">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="australia.shp">
      <maps:ShapeFileLayer.ShapeSettings>

```

```

<maps:ShapeSetting ShapeFill="#bcbcbc"/>
</maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeFileLayer.Sublayers>
<maps:ShapeFileLayer Uri="points.shp" MapPointIcon="Circle"
MapPointIconSize="10"
MapPointHorizontalAlignment="Near" MapPointVerticalAlignment="Near">
<maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeSetting ShapeFill="Brown" />
</maps:ShapeFileLayer.ShapeSettings>
</maps:ShapeFileLayer>
</maps:ShapeFileLayer.Sublayers>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>

```

### C#

```

SfMaps maps = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "australia.shp";
ShapeSetting layerSetting = new ShapeSetting();
layerSetting.ShapeFill = Color.FromHex("#bcbcbc");
layer.ShapeSettings = layerSetting;
ShapeFileLayer subLayer = new ShapeFileLayer();
subLayer.Uri = "points.shp";
subLayer.MapPointIcon = MapPointIcon.Circle;
subLayer.MapPointIconSize = 10;
subLayer.MapPointHorizontalAlignment = MarkerAlignment.Near;
subLayer.MapPointVerticalAlignment = MarkerAlignment.Near;
ShapeSetting subLayerSetting = new ShapeSetting();
subLayerSetting.ShapeFill = Color.Green;
subLayer.ShapeSettings = subLayerSetting;
layer.Sublayers.Add(subLayer);
maps.Layers.Add(layer);

```



### Map point size

The size of the map points can be customized using the [MapPointIconSize] property of ShapeFileLayer.

### XML

```
<maps:SfMaps BackgroundColor="White">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="australia.shp">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeFill="#bcbcbc"/>
      </maps:ShapeFileLayer.ShapeSettings>
    <maps:ShapeFileLayer.Sublayers>
      <maps:ShapeFileLayer Uri="points.shp" MapPointIconSize="15">
        <maps:ShapeFileLayer.ShapeSettings>
          <maps:ShapeSetting ShapeFill="OrangeRed" />
        </maps:ShapeFileLayer.ShapeSettings>
      </maps:ShapeFileLayer>
    </maps:ShapeFileLayer.Sublayers>
  </maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
SfMaps maps = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "australia.shp";
ShapeSetting layerSetting = new ShapeSetting();
layerSetting.ShapeFill = Color.FromHex("#bcbcbc");
layer.ShapeSettings = layerSetting;
ShapeFileLayer subLayer = new ShapeFileLayer();
subLayer.Uri = "points.shp";
subLayer.MapPointIcon = MapPointIcon.Circle;
subLayer.MapPointIconSize = 15;
ShapeSetting subLayerSetting = new ShapeSetting();
subLayerSetting.ShapeFill = Color.OrangeRed;
subLayer.ShapeSettings = subLayerSetting;
layer.Sublayers.Add(subLayer);
maps.Layers.Add(layer);
```



## Markers

Markers are used to show some messages on maps.

Markers are set to the maps control using the following ways:

- Adding marker objects
- Defining custom markers

### Adding marker objects

Any number of markers can be added to the shape file layers using the [Markers](#) property. Each marker object contains the following properties:

[Label](#): Displays some messages on maps.

[Latitude](#): Specifies y-axis position of the marker.

[Longitude](#): Specifies x-axis position of the marker.

### XML

```
<maps:SfMaps x:Name="sfmap" BackgroundColor="White">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp">
      <maps:ShapeFileLayer.Markers>
        <maps:MapMarker Label = "California" Latitude = "37" Longitude = "-120"/>
      </maps:ShapeFileLayer.Markers>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
map.Layers.Add(layer);
MapMarker marker = new MapMarker();
marker.Label = "California";
marker.Latitude = "37";
marker.Longitude = "-120";
layer.Markers.Add(marker);
this.Content = map;
```



### Customizing markers

Markers can be customized using the [MarkerSettings](#) property in shape file layer.

#### Customizing marker icons

The size and color of marker icons can be customized using the [IconSize](#) and [IconColor](#) properties.

#### Icon types

The shape of a marker icon can be customized using the [MarkerIcon](#) property. The maps control supports the following types of marker icons:

- Circle
- Diamond
- Image
- Rectangle
- Square

### XML

```
<maps:SfMaps x:Name="sfmap">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp">
      <maps:ShapeFileLayer.Markers>
        <maps:MapMarker Label = "California" Latitude = "37" Longitude = "-120"/>
      </maps:ShapeFileLayer.Markers>
      <maps:ShapeFileLayer.MarkerSettings>
        <maps:MapMarkerSetting MarkerIcon="Square"/>
      </maps:ShapeFileLayer.MarkerSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
```

```
</maps:SfMaps>
```

## C#

```
SfMaps map = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
map.Layers.Add(layer);
MapMarker marker = new MapMarker();
marker.Label = "California";
marker.Latitude = "37";
marker.Longitude = "-120";
layer.Markers.Add(marker);
MapMarkerSetting markerSetting = new MapMarkerSetting();
markerSetting.MarkerIcon = MapMarkerIcon.Square;
layer.MarkerSettings = markerSetting;
this.Content = map;
```



### Setting contrast color

Based on the background color of the shapes, contrast color will be applied to marker icon.

## XML

```
<maps:SfMaps x:Name="sfmap" BackgroundColor="White">
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer x:Name="layer" Uri="usa_state.shp" ShapeIDPath="State"
      ShowMapItems="False"
      ShapeIDTableField="STATE_NAME">
      <maps:ShapeFileLayer.Markers>
        <maps:MapMarker Latitude = "37" Longitude = "-120">
        </maps:MapMarker>
```

```

<maps:MapMarker Latitude = "31" Longitude = "-97">
</maps:MapMarker>
<maps:MapMarker Latitude = "41" Longitude = "-92">
</maps:MapMarker>
<maps:MapMarker Latitude = "38" Longitude = "-98">
</maps:MapMarker>
<maps:MapMarker Latitude = "41" Longitude = "-99">
</maps:MapMarker>
</maps:ShapeFileLayer.Markers>
<maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeSetting ShapeColorValuePath="Candidate"
ShapeValuePath="Candidate">
<maps:ShapeSetting.ColorMappings>
<maps:EqualColorMapping Color="#FFD84F" LegendLabel="Romney"
Value = "Romney"></maps:EqualColorMapping>
<maps:EqualColorMapping Color="#316DB5" LegendLabel="Obama"
Value="Obama"></maps:EqualColorMapping>
</maps:ShapeSetting.ColorMappings>
</maps:ShapeSetting>
</maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeFileLayer.LegendSettings>
<maps:MapLegendSetting ShowLegend="True"
LegendPosition="30,70">
<maps:MapLegendSetting.IconSize>
<Size Width="20" Height="20"></Size>
</maps:MapLegendSetting.IconSize>
</maps:MapLegendSetting>
</maps:ShapeFileLayer.LegendSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>

```

## C#

```

SfMaps map = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.ItemsSource = viewModel.Data;
layer.Uri = "usa_state.shp";
map.Layers.Add(layer);
MapMarker marker = new MapMarker();
marker.Label = "California";
marker.Latitude = "37";
marker.Longitude = "-120";
layer.Markers.Add(marker);
MapLegendSetting legendSetting = new MapLegendSetting();
legendSetting.ShowLegend = true;
legendSetting.LegendPosition = new Point(30, 70);
legendSetting.IconSize = new Size(20, 20);
layer.LegendSettings = legendSetting;
MapMarker marker1 = new MapMarker();
marker1.Label = "California";
marker1.Latitude = "31";
marker1.Longitude = "-97";
layer.Markers.Add(marker1);
MapMarker marker2 = new MapMarker();
marker2.Label = "California";

```

```
marker2.Latitude = "41";
marker2.Longitude = "-92";
layer.Markers.Add(marker2);
MapMarker marker3 = new MapMarker();
marker3.Label = "California";
marker3.Latitude = "38";
marker3.Longitude = "-98";
layer.Markers.Add(marker3);
MapMarker marker4 = new MapMarker();
marker4.Label = "California";
marker4.Latitude = "41";
marker4.Longitude = "-99";
layer.Markers.Add(marker4);
EqualColorMapping colorMapping = new EqualColorMapping();
colorMapping.Color = Color.FromHex("#FFD84F");
colorMapping.LegendLabel = "Romney";
colorMapping.Value = "Romney";
EqualColorMapping colorMapping1 = new EqualColorMapping();
colorMapping1.Color = Color.FromHex("#316DB5");
colorMapping1.LegendLabel = "Obama";
colorMapping1.Value = "Obama";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeValuePath = "Candidate";
shapeSetting.ShapeColorValuePath = "Candidate";
shapeSetting.ColorMappings.Add(colorMapping);
shapeSetting.ColorMappings.Add(colorMapping1);
layer.ShapeSettings = shapeSetting;
this.Content = map;
```





### Setting image marker icon

You can pin an image as marker icon by setting the icon type as `Image`. Set [ImageSource](#) to get the image from local path.

### XML

```
<maps:SfMaps x:Name="sfmap">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp">
      <maps:ShapeFileLayer.Markers>
        <maps:MapMarker Label = "Texas" Latitude = "31.267153" Longitude = "-97.7430608"/>
        <maps:MapMarker Label = "California" Latitude = "37" Longitude = "-120"/>
      </maps:ShapeFileLayer.Markers>
      <maps:ShapeFileLayer.MarkerSettings>
        <maps:MapMarkerSetting MarkerIcon="Image" ImageSource="pin.png"/>
      </maps:ShapeFileLayer.MarkerSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
SfMaps map = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
map.Layers.Add(layer);
MapMarker marker = new MapMarker();
```

```

marker.Label = "Texas";
marker.Latitude = "31.267153";
marker.Longitude = "-97.7430608";
layer.Markers.Add(marker);
MapMarker marker1 = new MapMarker();
marker1.Label = "California";
marker1.Latitude = "37";
marker1.Longitude = "-120";
layer.Markers.Add(marker1);
MapMarkerSetting markerSetting = new MapMarkerSetting();
markerSetting.MarkerIcon = MapMarkerIcon.Image;
markerSetting.ImageSource = "pin.png";
layer.MarkerSettings = markerSetting;
this.Content = map;

```



### Customizing labels

The color and size of marker labels can be customized using the [LabelColor](#) and [LabelSize](#) properties.

The following code sample explains how to customize a marker.

### XML

```

<maps:SfMaps x:Name="sfmap" BackgroundColor="White">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp">
      <maps:ShapeFileLayer.Markers>
        <maps:MapMarker Label = "California" Latitude = "37" Longitude = "-120"/>
      </maps:ShapeFileLayer.Markers>
      <maps:ShapeFileLayer.MarkerSettings>
        <maps:MapMarkerSetting IconColor="Red" IconSize="25" MarkerIcon="Diamond"
          LabelColor="White" LabelSize="20"/>
      </maps:ShapeFileLayer.MarkerSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>

```

```

</maps:ShapeFileLayer.MarkerSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>

```

## C#

```

SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
map.Layers.Add(layer);
MapMarker marker = new MapMarker();
marker.Label = "California";
marker.Latitude = "37";
marker.Longitude = "-120";
layer.Markers.Add(marker);
MapMarkerSetting markerSetting = new MapMarkerSetting();
markerSetting.IconColor = Color.Red;
markerSetting.IconSize = 25;
markerSetting.MarkerIcon = MapMarkerIcon.Diamond;
markerSetting.LabelColor = Color.White;
markerSetting.LabelSize = 20;
layer.MarkerSettings = markerSetting;
this.Content = map;

```



## Custom marker

The maps control provides support for defining custom markers using the [MarkerTemplate](#) property.

## XML

```

<maps:SfMaps x:Name="sfmap" BackgroundColor="White">
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="world1.shp" >
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeFill="Gray" />
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.Markers>
        <local:CustomMarker Latitude="38.8833" Longitude= "-77.0167" />
        <local:CustomMarker Latitude="-15.7833" Longitude= "-47.8667" />
        <local:CustomMarker Latitude="21.0000" Longitude= "78.0000" />
        <local:CustomMarker Latitude="35.0000" Longitude= "103.0000" />
      </maps:ShapeFileLayer.Markers>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>

```

```

<local:CustomMarker Latitude="-6.1750" Longitude="106.8283" />
</maps:ShapeFileLayer.Markers>
<maps:ShapeFileLayer.MarkerTemplate>
<DataTemplate >
<StackLayout Padding="-12,-12,0,0" IsClippedToBounds="false"
HorizontalOptions="StartAndExpand" VerticalOptions="Center"
HeightRequest="60" WidthRequest="60" >
<Image Source="{Binding ImageName}" Scale="1" Aspect="AspectFit "
HorizontalOptions="StartAndExpand" VerticalOptions="Center"
HeightRequest="15" WidthRequest="23" />
</StackLayout>
</DataTemplate>
</maps:ShapeFileLayer.MarkerTemplate>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>

```

## C#

```

SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "world1.shp";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeFill = Color.Gray;
layer.ShapeSettings = shapeSetting;
map.Layers.Add(layer);
CustomMarker marker1 = new CustomMarker();
marker1.Latitude = "38.8833";
marker1.Longitude = "-77.0167";
CustomMarker marker2 = new CustomMarker();
marker2.Latitude = "-15.7833";
marker2.Longitude = "-47.866";
CustomMarker marker3 = new CustomMarker();
marker3.Latitude = "21.0000";
marker3.Longitude = "78.0000";
CustomMarker marker4 = new CustomMarker();
marker4.Latitude = "35.0000";
marker4.Longitude = "103.0000";
CustomMarker marker5 = new CustomMarker();
marker5.Latitude = "-6.1750";
marker5.Longitude = "106.8283";
layer.Markers.Add(marker1);
layer.Markers.Add(marker2);
layer.Markers.Add(marker3);
layer.Markers.Add(marker4);
layer.Markers.Add(marker5);
DataTemplate dataTemplate = new DataTemplate(() =>
{
    StackLayout stackLayout = new StackLayout();
    stackLayout.Padding = new Thickness(-12, -12, 0, 0);
    stackLayout.IsClippedToBounds = false;
    stackLayout.HorizontalOptions = LayoutOptions.StartAndExpand;
    stackLayout.VerticalOptions = LayoutOptions.Center;
    stackLayout.HeightRequest = 60;
    stackLayout.WidthRequest = 60;

```

```
Image image = new Image();
image.Source = marker1.ImageName;
image.Scale = 1;
image.Aspect = Aspect.AspectFit;
image.HorizontalOptions = LayoutOptions.StartAndExpand;
image.VerticalOptions = LayoutOptions.Center;
image.HeightRequest = 15;
image.WidthRequest = 23;
stackLayout.Children.Add(image);
return stackLayout;
});
layer.MarkerTemplate = dataTemplate;
grid.Children.Add(map);
```

The following code sample explains how to define a custom marker with image support.

### C#

```
public class CustomMarker : MapMarker
{
    public ImageSource ImageName { get; set; }
    public CustomMarker()
    {
        ImageName = ImageSource.FromResource("MapsSample.pin.png");
    }
}
```



### Marker Alignment

You can align the maps marker horizontally and vertically using the [HorizontalAlignment](#) and [VerticalAlignment](#) properties.

#### Setting horizontal alignment

The [HorizontalAlignment](#) property is used to position the marker icon in x-axis. The marker icon can be positioned using the following ways in x-axis:

- Near: Specifies the near position of the marker icon for the given latitude and longitude values in x-axis position.
- Center: Specifies the center position of the marker icon for the given latitude and longitude values in x-axis position.

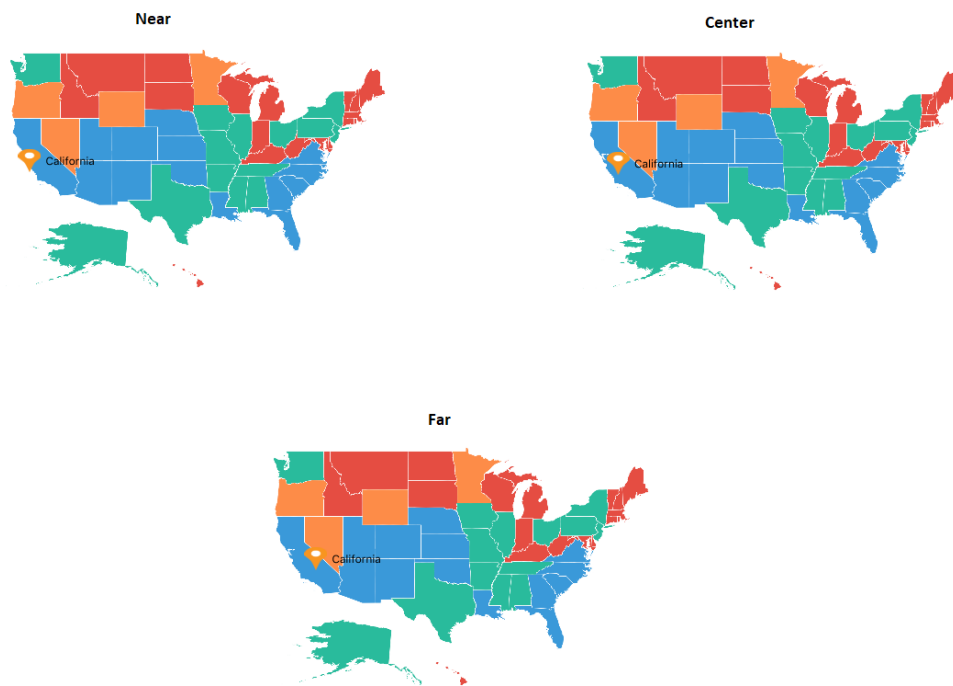
- Far: Specifies the far position of the marker icon for the given latitude and longitude values in x-axis position.

### XML

```
<maps:SfMaps Grid.Row="0" x:Name="Map" EnableZooming="False"
EnablePanning="False">
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="Name"
ShapeIDTableField="STATE_NAME"
ItemsSource="{Binding DataSource}">
      ....
    <maps:ShapeFileLayer.MarkerSettings>
      <maps:MapMarkerSetting x:Name="markerSetting" MarkerIcon="Image"
ImageSource="pin.png" HorizontalAlignment="Center" IconSize="25"/>
    </maps:ShapeFileLayer.MarkerSettings>
      ....
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
SfMaps map = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
CustomMarker customMarker = new CustomMarker();
layer.ItemsSource = customMarker.DataSource;
layer.Uri = "usa_state.shp";
layer.ShapeIDPath = "Name";
layer.ShapeIDTableField = "STATE_NAME";
map.Layers.Add(layer);
....
MapMarkerSetting mapMarkerSetting = new MapMarkerSetting();
mapMarkerSetting.MarkerIcon = MapMarkerIcon.Image;
mapMarkerSetting.ImageSource = "pin.png";
mapMarkerSetting.IconSize = 25;
mapMarkerSetting.HorizontalAlignment = MarkerAlignment.Center;
layer.MarkerSettings = mapMarkerSetting;
....
this.Content = map;
```



### Setting vertical alignment

The VerticalAlignment property is used to position the marker icon in y-axis. The marker icon can be positioned using the following ways in y-axis:

- Near: Specifies the near position of the marker icon for the given latitude and longitude values in y-axis position.
- Center: Specifies the center position of the marker icon for the given latitude and longitude values in y-axis position.
- Far: Specifies the far position of the marker icon for the given latitude and longitude values in y-axis position.

### XML

```
<maps:SfMaps Grid.Row="0" x:Name="Map" EnableZooming="False"
EnablePanning="False">
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="Name"
ShapeIDTableField="STATE_NAME"
ItemsSource="{Binding DataSource}">
      ....
      <maps:ShapeFileLayer.MarkerSettings>
        <maps:MapMarkerSetting x:Name="markerSetting" MarkerIcon="Image"
ImageSource="pin.png" VerticalAlignment="Center" IconSize="25"/>
      </maps:ShapeFileLayer.MarkerSettings>
      ....
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

```

</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>

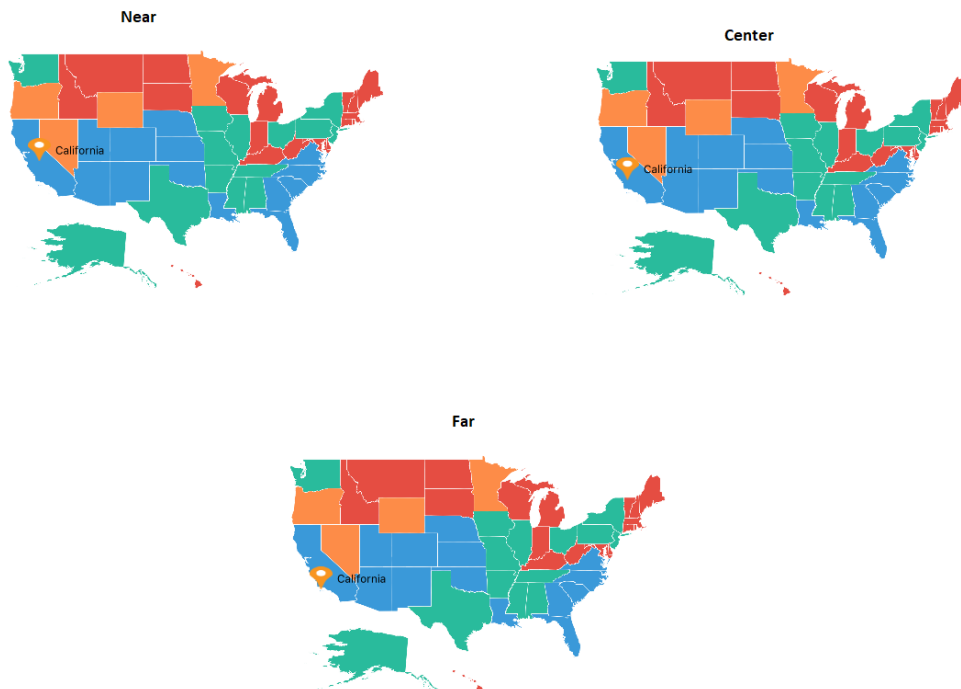
```

## C#

```

SfMaps map = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
CustomMarker customMarker = new CustomMarker();
layer.ItemsSource = customMarker.DataSource;
layer.Uri = "usa_state.shp";
layer.ShapeIDPath = "Name";
layer.ShapeIDTableField = "STATE_NAME";
map.Layers.Add(layer);
....
MapMarkerSetting mapMarkerSetting = new MapMarkerSetting();
mapMarkerSetting.MarkerIcon = MapMarkerIcon.Image;
mapMarkerSetting.ImageSource = "pin.png";
mapMarkerSetting.IconSize = 25;
mapMarkerSetting.VerticalAlignment = MarkerAlignment.Center;
layer.MarkerSettings = mapMarkerSetting;
....
this.Content = map;

```



**Note:** The default marker icon position for VerticalAlignment and HorizontalAlignment is Center.



## Events

The `MarkerSelected` event is fired when a marker is selected. The `CustomView` and `MapMarker` will be passed to `MarkerSelectedEventArgs`.

If you set any view for the `CustomView` property of `MarkerSelectedEventArgs`, then the corresponding view will be applied to the selected marker.

## XML

```
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="selectedMarker">
<StackLayout >
<Image Source="pin.png" Scale="1" Aspect="AspectFit "
HorizontalOptions="StartAndExpand" VerticalOptions="Center"
HeightRequest="15" WidthRequest="23" />
</StackLayout>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<maps:ImageryLayer MarkerSelected="Layer_MarkerSelected" >
<maps:ImageryLayer.MarkerSettings>
<maps:MapMarkerSetting IconColor="Red"
IconSize="13" MarkerIcon="Diamond"/>
</maps:ImageryLayer.MarkerSettings>
<maps:ImageryLayer.Markers>
<maps:MapMarker Label="United States"
Latitude="40" Longitude= "-101"/>
<maps:MapMarker Label="Brazil"
Latitude="-15.7833" Longitude= "-52" />
<maps:MapMarker Label="Congo"
Latitude="-1.6" Longitude= "24.4" />
<maps:MapMarker Label="Kazakhstan"
Latitude="49.9" Longitude= "72.23" />
<maps:MapMarker Label="Australia"
Latitude="-20.54" Longitude= "134.10" />
</maps:ImageryLayer.Markers>
</maps:ImageryLayer>
</maps:SfMaps.Layers>
```

## C#

```
private void Layer_MarkerSelected(object sender, MarkerSelectedEventArgs e)
{
    e.CustomView = this.Resources["selectedMarker"] as DataTemplate;
}
```



## Bubble Marker

Bubbles in the maps control represent the underlying data values of the maps. Bubbles are scattered throughout the map shapes that contain bound values.

### Bubble data

Bubbles are included when the data binding and [BubbleMarkerSettings](#) are set to the shape layers.

The following code sample explains data binding provided for bubble.

#### XML

```
<maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="Name"
ShapeIDTableField="STATE_NAME"
ItemsSource="{Binding DataSource}" />
```

#### C#

```
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource = viewModel.DataSource;
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "Name";
map.Layers.Add(layer);
```

### Adding bubbles

To add bubbles to maps, the bubble marker setting should be added to the shape file layer. The [ShowBubbles](#) property should be enabled for making the bubbles visible.

The [ValuePath](#) represents the field value to be fetched from the data for each bubble.

#### XML

```
<maps:ShapeFileLayer.BubbleMarkerSettings>
<maps:BubbleMarkerSetting ShowBubbles="True" ValuePath="index" />
</maps:ShapeFileLayer.BubbleMarkerSettings>
```

#### C#

```
BubbleMarkerSetting bubbleSetting = new BubbleMarkerSetting();
bubbleSetting.ShowBubbles = true;
bubbleSetting.ValuePath = "index";
layer.BubbleMarkerSettings = bubbleSetting;
```



### *Range color mapping*

It is used to differentiate the bubble fill based on its under-bound value and color ranges.

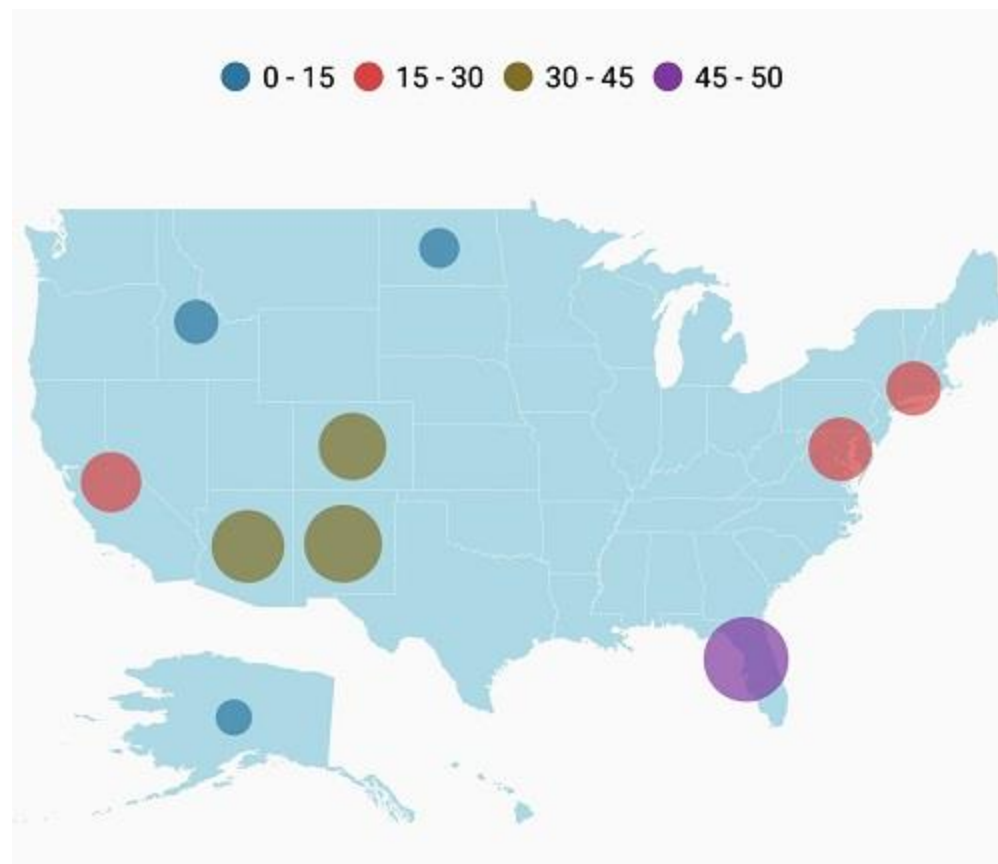
The [From](#) and [To](#) properties are used to define the color range and color for the range that can be specified using the Color property.

### **XML**

```
<maps:ShapeFileLayer.BubbleMarkerSettings>
  <maps:BubbleMarkerSetting ValuePath="index" ColorValuePath="index" >
    <maps:BubbleMarkerSetting.ColorMappings>
      <maps:RangeColorMapping Color="#2E769F" From="0" To="15" />
      <maps:RangeColorMapping Color="#D84444" To="30" From="15" />
      <maps:RangeColorMapping Color="#816F28" To="45" From="30" />
      <maps:RangeColorMapping Color="#7F38A0" To="50" From="45" />
    </maps:BubbleMarkerSetting.ColorMappings>
  </maps:BubbleMarkerSetting>
</maps:ShapeFileLayer.BubbleMarkerSettings>
```

### **C#**

```
BubbleMarkerSetting bubbleSetting = new BubbleMarkerSetting();
bubbleSetting.ValuePath = "index";
bubbleSetting.ColorValuePath = "index";
RangeColorMapping colorMapping1 = new RangeColorMapping() { Color =
Color.FromHex("#2E769F"), From = 0, To = 15 };
RangeColorMapping colorMapping2 = new RangeColorMapping() { Color =
Color.FromHex("#D84444"), From = 15, To = 30 };
RangeColorMapping colorMapping3 = new RangeColorMapping() { Color =
Color.FromHex("#816F28"), From = 30, To = 45 };
RangeColorMapping colorMapping4 = new RangeColorMapping() { Color =
Color.FromHex("#7F38A0"), From = 45, To = 50 };
bubbleSetting.ColorMappings.Add(colorMapping1);
bubbleSetting.ColorMappings.Add(colorMapping2);
bubbleSetting.ColorMappings.Add(colorMapping3);
bubbleSetting.ColorMappings.Add(colorMapping4);
layer.BubbleMarkerSettings = bubbleSetting;
```



### Equal color mapping

It is used to differentiate the bubble fill based on its underlying value and color using the [Value](#) and [Color](#) properties.

### XML

```
<maps:ShapeFileLayer.BubbleMarkerSettings>
  <maps:BubbleMarkerSetting ValuePath="index" ColorValuePath="Type" >
    <maps:BubbleMarkerSetting.ColorMappings>
      <maps:EqualColorMapping Color="#2E769F" Value="Vegetables" />
      <maps:EqualColorMapping Color="#D84444" Value="Rice" />
      <maps:EqualColorMapping Color="#816F28" Value="Wheat" />
      <maps:EqualColorMapping Color="#7F38A0" Value="Grains" />
    </maps:BubbleMarkerSetting.ColorMappings>
  </maps:BubbleMarkerSetting>
</maps:ShapeFileLayer.BubbleMarkerSettings>
```

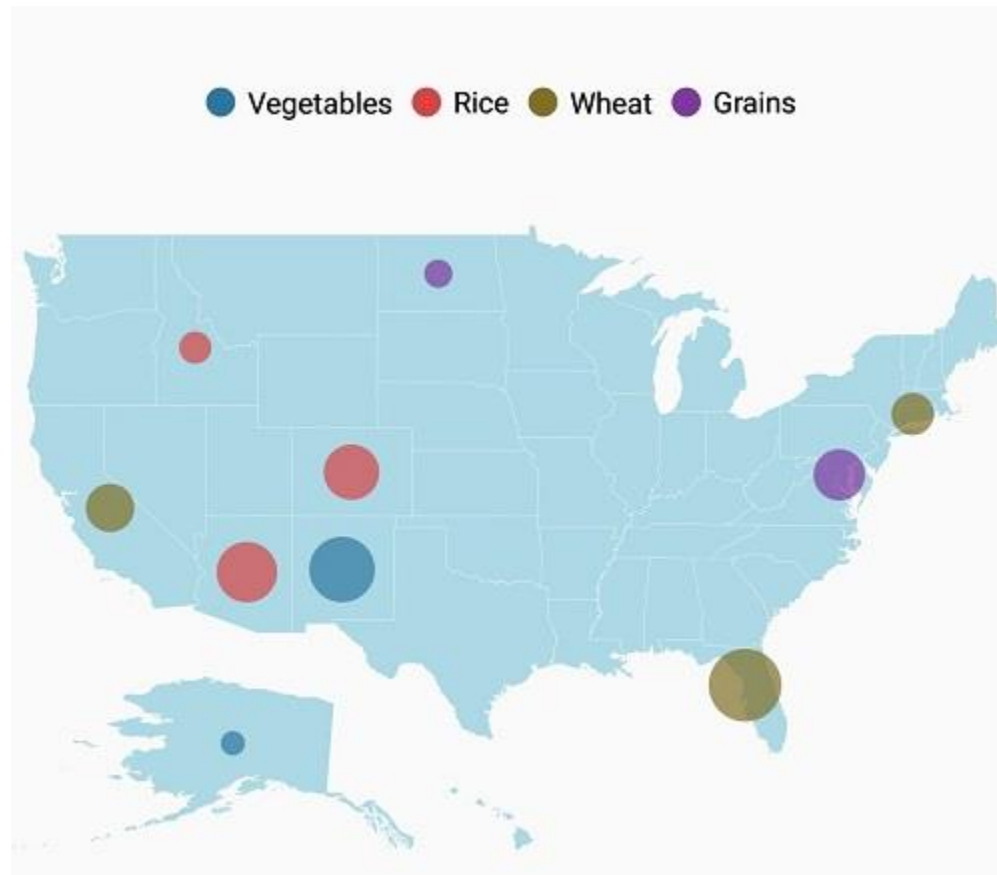
### C#

```
BubbleMarkerSetting bubbleSetting = new BubbleMarkerSetting();
bubbleSetting.ValuePath = "index";
bubbleSetting.ColorValuePath = "Type";
EqualColorMapping colorMapping1 = new EqualColorMapping() { Color =
Color.FromHex("#2E769F"), Value = "Vegetables" };
EqualColorMapping colorMapping2 = new EqualColorMapping() { Color =
Color.FromHex("#D84444"), Value = "Rice" };
```

```

EqualColorMapping colorMapping3 = new EqualColorMapping() { Color =
Color.FromHex("#816F28"), Value = "Wheat" };
EqualColorMapping colorMapping4 = new EqualColorMapping() { Color =
Color.FromHex("#7F38A0"), Value = "Grains" };
bubbleSetting.ColorMappings.Add(colorMapping1);
bubbleSetting.ColorMappings.Add(colorMapping2);
bubbleSetting.ColorMappings.Add(colorMapping3);
bubbleSetting.ColorMappings.Add(colorMapping4);
layer.BubbleMarkerSettings = bubbleSetting;

```



### Customizing bubble size

The size of the bubbles depends on the data bound to the [ValuePath](#). The maximum and minimum sizes of the bubbles can be customized using [MaxSize](#) and [MinSize](#) properties.

**Information:** The [ShowMapItems](#) should be enabled to display label on bubble marker.

### XML

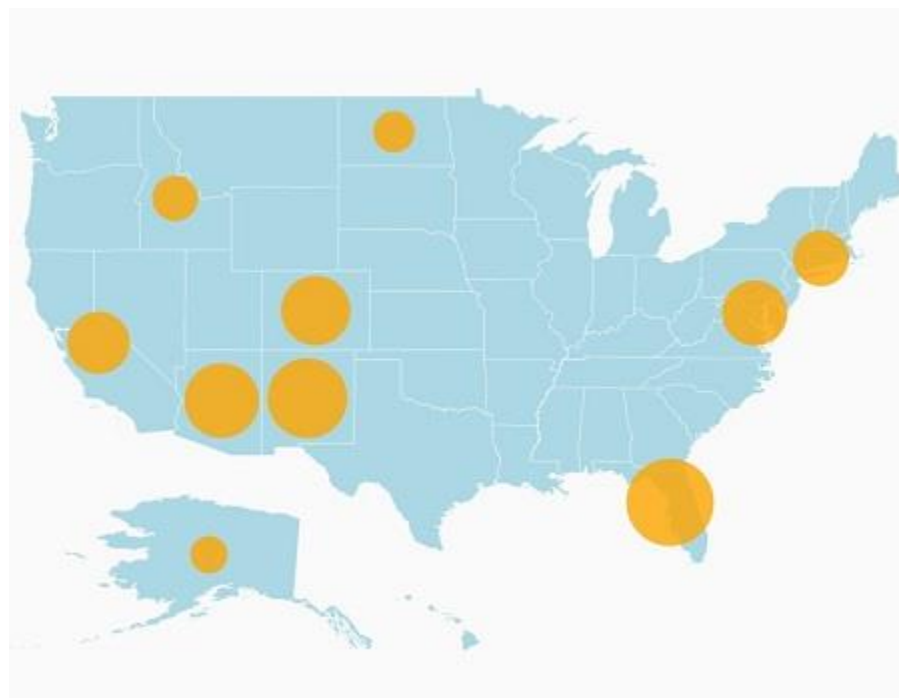
```

<maps:ShapeFileLayer.BubbleMarkerSettings>
  <maps:BubbleMarkerSetting ShowBubbles="True" ValuePath="Electors"
  Fill="Orange"
  MaxSize="25" MinSize="20" Opacity="0.8" />
</maps:ShapeFileLayer.BubbleMarkerSettings>

```

### C#

```
BubbleMarkerSetting bubbleSetting = new BubbleMarkerSetting();
bubbleSetting.ShowBubbles = true;
bubbleSetting.ValuePath = "Electors";
bubbleSetting.Fill = Color.Orange;
bubbleSetting.Opacity = 0.8;
bubbleSetting.MinSize = 20;
bubbleSetting.MaxSize = 25;
layer.BubbleMarkerSettings = bubbleSetting;
```



The following code sample demonstrates how to add bubbles to maps and customize them

#### XML

```
<maps:SfMaps>
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer x:Name="layer" Uri="usa_state.shp" ShapeIDPath="Name"
      ShapeIDTableField="STATE_NAME"
      ShowMapItems="True" ItemsSource="{Binding DataSource}">
      <maps:ShapeFileLayer.BubbleMarkerSettings>
        <maps:BubbleMarkerSetting ValuePath="index" ColorValuePath="index" >
          <maps:BubbleMarkerSetting.ColorMappings>
            <maps:RangeColorMapping Color="#2E769F" From="0" To="15" />
            <maps:RangeColorMapping Color="#D84444" To="30" From="15" />
            <maps:RangeColorMapping Color="#816F28" To="45" From="30" />
            <maps:RangeColorMapping Color="#7F38A0" To="50" From="45" />
          </maps:BubbleMarkerSetting.ColorMappings>
        </maps:BubbleMarkerSetting>
      </maps:ShapeFileLayer.BubbleMarkerSettings>
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeFill="LightBlue"/>
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.LegendSettings>
```

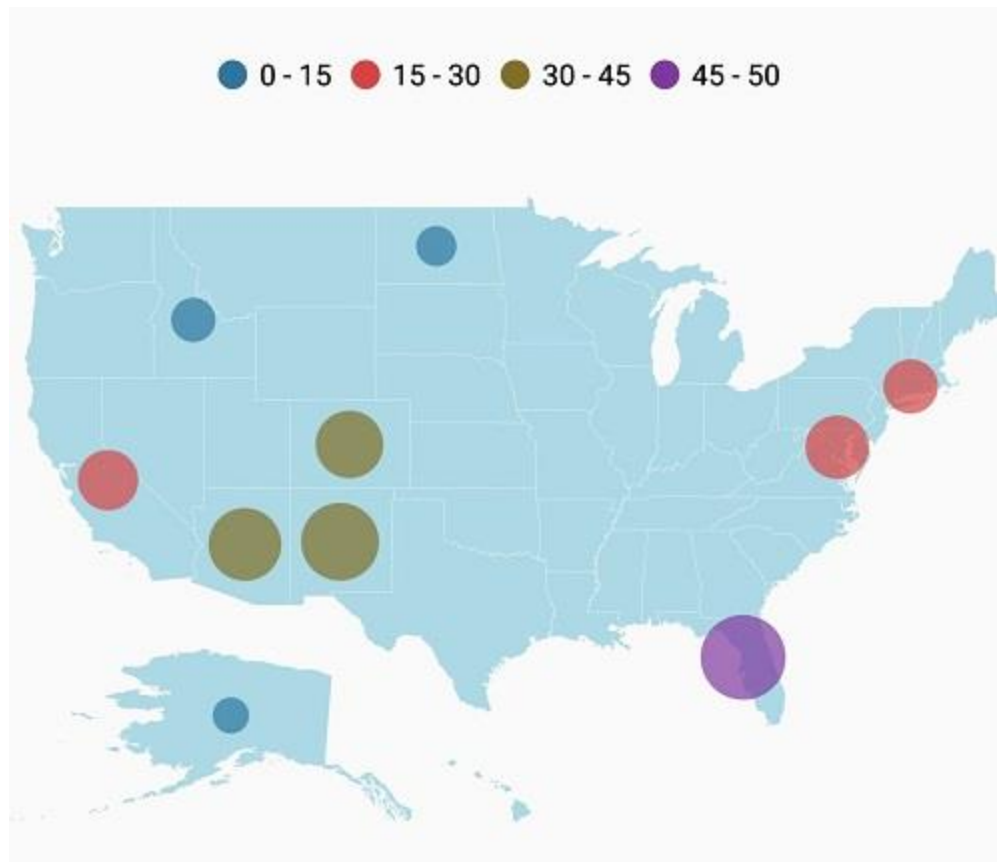
```
<maps:MapLegendSetting LegendType="Bubbles"
ShowLegend="True"></maps:MapLegendSetting>
</maps:ShapeFileLayer.LegendSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

**C#**

```
ViewModel viewModel = new ViewModel();
SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource = viewModel.DataSource;
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "Name";
layer.ShowMapItems = true;
BubbleMarkerSetting bubbleSetting = new BubbleMarkerSetting();
bubbleSetting.ValuePath = "index";
bubbleSetting.ColorValuePath = "index";
RangeColorMapping colorMapping1 = new RangeColorMapping() { Color =
Color.FromHex("#2E769F"), From = 0, To = 15 };
RangeColorMapping colorMapping2 = new RangeColorMapping() { Color =
Color.FromHex("#D84444"), From = 15, To = 30 };
RangeColorMapping colorMapping3 = new RangeColorMapping() { Color =
Color.FromHex("#816F28"), From = 30, To = 45 };
RangeColorMapping colorMapping4 = new RangeColorMapping() { Color =
Color.FromHex("#7F38A0"), From = 45, To = 50 };
bubbleSetting.ColorMappings.Add(colorMapping1);
bubbleSetting.ColorMappings.Add(colorMapping2);
bubbleSetting.ColorMappings.Add(colorMapping3);
bubbleSetting.ColorMappings.Add(colorMapping4);
layer.BubbleMarkerSettings = bubbleSetting;
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeFill = Color.LightBlue;
layer.ShapeSettings = shapeSetting;
MapLegendSetting legendSetting = new MapLegendSetting();
legendSetting.ShowLegend = true;
legendSetting.LegendType = LegendType.Bubbles;
layer.LegendSettings = legendSetting;
map.Layers.Add(layer);
this.Content = map;
public class AgricultureData
{
    public AgricultureData(string name, string type, int count)
    {
        Name = name;
        Type = type;
        index = count;
    }
    public string Name
    {
        get;
        set;
    }
}
```



```
public string Type
{
    get;
    set;
}
public int index
{
    get;
    set;
}
}
public class ViewModel
{
    public ViewModel()
    {
        DataSource = new ObservableCollection<AgricultureData>();
        DataSource.Add(new AgricultureData("Alaska", "Vegetables", 0));
        DataSource.Add(new AgricultureData("Arizona", "Rice", 36));
        DataSource.Add(new AgricultureData("California", "Wheat", 24));
        DataSource.Add(new AgricultureData("Colorado", "Rice", 31));
        DataSource.Add(new AgricultureData("North Dakota", "Grains", 4));
        DataSource.Add(new AgricultureData("Connecticut", "Wheat", 18));
        DataSource.Add(new AgricultureData("District of Columbia", "Grains", 27));
        DataSource.Add(new AgricultureData("Florida", "Wheat", 48));
        DataSource.Add(new AgricultureData("New Mexico", "Vegetables", 41));
        DataSource.Add(new AgricultureData("Idaho", "Rice", 8));
    }
    public ObservableCollection<AgricultureData> DataSource { get; set; }
}
```



### Color mapping

The color mapping support allows you to customize the shape colors based on the underlying value of shapes received from the bound data.

The maps control provides two types of color mapping:

- Equal color mapping
- Range Color mapping

### Equal color mapping

It is used to differentiate the shape's fill based on its underlying value and color using the [Value](#) and [Color](#) properties.

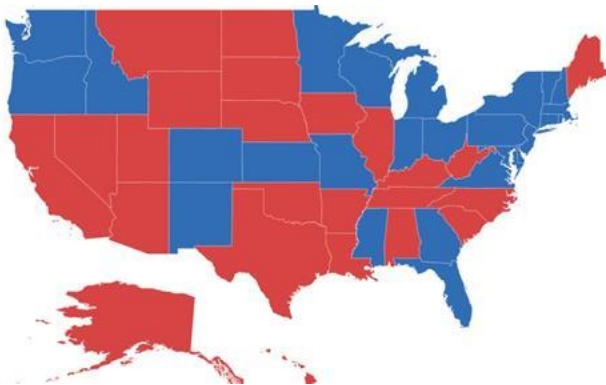
### XML

```
<maps:SfMaps x:Name="sfmap" BackgroundColor="White">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeValuePath="Candidate"
          ShapeColorValuePath="Candidate" >
          <maps:ShapeSetting.ColorMappings>
            <maps:EqualColorMapping Color="#D84444" Value = "Romney"/>
            <maps:EqualColorMapping Color="#316DB5" Value="Obama"/>
          </maps:ShapeSetting.ColorMappings>
        </maps:ShapeSetting>
      </maps:ShapeFileLayer.ShapeSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

```
</maps:ShapeSetting>
</maps:ShapeFileLayer.ShapeSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

## C#

```
SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource = viewModel.Data;
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "State";
map.Layers.Add(layer);
EqualColorMapping colorMapping = new EqualColorMapping();
colorMapping.Color = Color.FromHex("#D84444");
colorMapping.Value = "Romney";
EqualColorMapping colorMapping1 = new EqualColorMapping();
colorMapping1.Color = Color.FromHex("#316DB5");
colorMapping1.Value = "Obama";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeValuePath = "Candidate";
shapeSetting.ShapeColorValuePath = "Candidate";
shapeSetting.ColorMappings.Add(colorMapping);
shapeSetting.ColorMappings.Add(colorMapping1);
layer.ShapeSettings = shapeSetting;
this.Content = map;
```



## Range color mapping

It is used to differentiate the bubble's fill based on its under-bound value and color ranges.

The [From](#) and [To](#) properties are used to define the color range and color for the range that can be specified using the [Color](#) property.

## XML

```
<maps:SfMaps x:Name="sfmap" BackgroundColor="White">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
    <maps:ShapeFileLayer.ShapeSettings>
```

```
<maps:ShapeSetting ShapeColorValuePath="Electors" >
<maps:ShapeSetting.ColorMappings>
<maps:RangeColorMapping From="30" To="70" Color="#397D02"/>
<maps:RangeColorMapping From="15" To="30" Color="#316DB5"/>
<maps:RangeColorMapping From="0" To="15" Color="#D84444"/>
</maps:ShapeSetting.ColorMappings>
</maps:ShapeSetting>
</maps:ShapeFileLayer.ShapeSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

## C#

```
SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource = viewModel.Data;
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "State";
map.Layers.Add(layer);
RangeColorMapping rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 30;
rangeColorMapping.To = 70;
rangeColorMapping.Color = Color.FromHex("#397D02");
RangeColorMapping rangeColorMapping1 = new RangeColorMapping();
rangeColorMapping1.From = 15;
rangeColorMapping1.To = 30;
rangeColorMapping1.Color = Color.FromHex("#316DB5");
RangeColorMapping rangeColorMapping2 = new RangeColorMapping();
rangeColorMapping2.From = 0;
rangeColorMapping2.To = 15;
rangeColorMapping2.Color = Color.FromHex("#D84444");
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeColorValuePath = "Electors";
shapeSetting.ColorMappings.Add(rangeColorMapping);
shapeSetting.ColorMappings.Add(rangeColorMapping1);
shapeSetting.ColorMappings.Add(rangeColorMapping2);
layer.ShapeSettings = shapeSetting;
this.Content = map;
```



## Data Labels

Data labels are used to display the values of shapes.

### Adding data labels

The [ShowMapItems](#) property, which is a boolean property, displays or hides the data labels in shapes.

Set the [ShapeValuePath](#) property to get the data labels bound to each shape.

### XML

```
<ContentPage.BindingContext>
<local:USAShapeViewModel/>
</ContentPage.BindingContext>
<maps:SfMaps x:Name="sfmap" >
<maps:SfMaps.Layers>
<maps:ShapeFileLayer Uri="usa_state.shp" ShowMapItems="True"
ItemsSource="{Binding DataSource}" ShapeIDPath="Name"
ShapeIDTableField="STATE_NAME">
<maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeSetting ShapeValuePath="Type" ShapeFill="LightGray"/>
</maps:ShapeFileLayer.ShapeSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
SfMaps map = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ShowMapItems = true;
layer.ShapeSettings.ShapeFill = Color.LightGray;
USAShapeViewModel usaShapeViewModel = new USAShapeViewModel();
layer.ItemsSource = usaShapeViewModel.DataSource;
layer.ShapeIDPath = "Name";
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeSettings.ShapeValuePath = "Type";
map.Layers.Add(layer);
this.Content = map;
```



### Setting contrast color

Based on the background color of the shapes, contrast color will be applied to data labels.

### XML

```
<maps:SfMaps x:Name="sfmap">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp" ShowMapItems="True"
      ItemsSource="{Binding DataSource}" ShapeIDPath="Name"
      ShapeIDTableField="STATE_NAME">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeFill="#A9D9F7" ShapeValuePath="Type"
          ShapeColorValuePath="index" >
          <maps:ShapeSetting.ColorMappings>
            <maps:RangeColorMapping To="25" From="0" Color="#FFD84F"/>
            <maps:RangeColorMapping To="50" From="25" Color="#316DB5"/>
          </maps:ShapeSetting.ColorMappings>
        </maps:ShapeSetting>
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.DataLabelSettings>
        <maps:DataLabelSetting SmartLabelMode="Trim"/>
      </maps:ShapeFileLayer.DataLabelSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

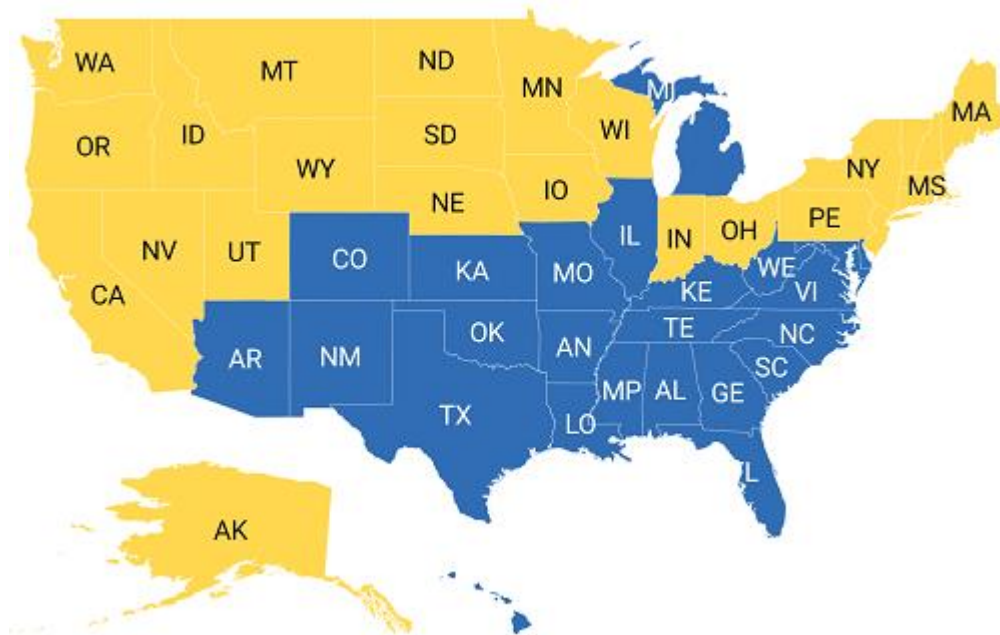
### C#

```
SfMaps map = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ShowMapItems = true;
layer.ShapeSettings.ShapeFill = Color.DarkBlue;
```

```

USAShapeViewModel usaShapeViewModel = new USAShapeViewModel();
layer.ItemsSource = usaShapeViewModel.DataSource;
layer.ShapeIDPath = "Name";
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeSettings.ShapeValuePath = "Type";
layer.ShapeSettings.ShapeColorValuePath = "index";
ObservableCollection<ColorMapping> colorMapping = new
ObservableCollection<ColorMapping>();
RangeColorMapping rangeColorMapping = new RangeColorMapping();
rangeColorMapping.From = 0;
rangeColorMapping.To = 25;
rangeColorMapping.Color = Color.FromHex("#FFD84F");
colorMapping.Add(rangeColorMapping);
RangeColorMapping rangeColorMapping1 = new RangeColorMapping();
rangeColorMapping1.From = 25;
rangeColorMapping1.To = 50;
rangeColorMapping1.Color = Color.FromHex("#316DB5");
colorMapping.Add(rangeColorMapping1);
layer.ShapeSettings.ColorMappings = colorMapping;
DataLabelSetting dataLabelSetting = new DataLabelSetting();
dataLabelSetting.SmartLabelMode = IntersectAction.Trim;
layer.DataLabelSettings = dataLabelSetting;
map.Layers.Add(layer);
this.Content = map;

```



### Customizing data labels

Data labels can be customized using the [DataLabelSetting](#) property in shape file layer. The font attribute, color, font size, and font family can be customized using the [FontAttributes](#), [TextColor](#), [FontSize](#), and [FontFamily](#) properties.

### XML

```

<maps:SfMaps x:Name="sfmap" >
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp" ShowMapItems="True"
      ItemsSource="{Binding DataSource}" ShapeIDPath="Name"
      ShapeIDTableField="STATE_NAME">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeValuePath="Type" ShapeFill="LightGray"/>
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.DataLabelSettings>
        <maps:DataLabelSetting TextColor="Blue" FontAttributes="Bold"
          FontFamily="cursive" FontSize="12" />
      </maps:ShapeFileLayer.DataLabelSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>

```

## C#

```

SfMaps map = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ShowMapItems = true;
USAStateViewModel usaStateViewModel = new USAStateViewModel();
layer.ItemsSource = usaStateViewModel.DataSource;
layer.ShapeIDPath = "Name";
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeSettings.ShapeValuePath = "Type";
layer.ShapeSettings.ShapeFill = Color.LightGray;
DataLabelSetting dataLabelSetting = new DataLabelSetting();
dataLabelSetting.TextColor = Color.Blue;
dataLabelSetting.FontAttributes = FontAttributes.Bold;
dataLabelSetting.FontFamily = "cursive";
dataLabelSetting.FontSize = 12;
layer.DataLabelSettings = dataLabelSetting;
map.Layers.Add(layer);
this.Content = map;

```





To smartly align data labels

The [SmartLabelMode](#) property aligns the labels smartly within shape boundaries and avoids labels overlapping. Labels can be customized using the `Hide`, `Trim`, and `None` options.

#### XML

```
<maps:SfMaps x:Name="sfmap">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp" ShowMapItems="True"
      ItemsSource="{Binding Data}" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeValuePath="State" ShapeFill="LightGray" />
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.DataLabelSettings>
        <maps:DataLabelSetting SmartLabelMode="Trim"/>
      </maps:ShapeFileLayer.DataLabelSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

#### C#

```
SfMaps map = new SfMaps();
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ShowMapItems = true;
USASateViewModel usaStateViewModel = new USASateViewModel();
layer.ItemsSource = usaStateViewModel.Data;
layer.ShapeIDPath = "State";
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeSettings.ShapeValuePath = "State";
```

```

layer.ShapeSettings.ShapeFill = Color.LightGray;
DataLabelSetting dataLabelSetting = new DataLabelSetting();
dataLabelSetting.SmartLabelMode = IntersectAction.Trim;
layer.DataLabelSettings = dataLabelSetting;
map.Layers.Add(layer);
this.Content = map;

```



To avoid overlap in data labels

The [IntersectionAction](#) property aligns labels that overlap. Labels can be customized using the `Hide`, `Trim`, and `None` options. First, set the [SmartLabelMode](#) property to `None`.

### XML

```

<maps:SfMaps x:Name="sfmap">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp" ShowMapItems="True"
      ItemsSource="{Binding Data}" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeValuePath="State" ShapeFill="LightGray" />
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.DataLabelSettings>
        <maps:DataLabelSetting IntersectionAction="Hide" SmartLabelMode="None"/>
      </maps:ShapeFileLayer.DataLabelSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>

```

### C#

```

SfMaps map = new SfMaps();

```

```

ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ShowMapItems = true;
USAStateViewModel usaStateViewModel = new USAStateViewModel();
layer.ItemsSource = usaStateViewModel.Data;
layer.ShapeIDPath = "State";
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeSettings.ShapeValuePath = "State";
layer.ShapeSettings.ShapeFill = Color.LightGray;
DataLabelSetting dataLabelSetting = new DataLabelSetting();
dataLabelSetting.IntersectionAction = IntersectAction.Hide;
dataLabelSetting.SmartLabelMode = IntersectAction.None;
layer.DataLabelSettings = dataLabelSetting;
map.Layers.Add(layer);
this.Content = map;

```



## Legend

Legends are keys used on maps; they contain swatches of symbols with descriptions. A legend interprets what the map displays; it can be represented in various colors, shapes, or other identifiers based on the data. It gives a breakdown on what each symbol represents throughout the map.

Legends can be added using the [LegendSettings](#) in the shape file layer.

## Visibility

The legends can be made visible by setting the [ShowLegend](#) property in [MapLegendSetting](#) to true.

## XML

```

<maps:ShapeFileLayer.LegendSettings>
  <maps:MapLegendSetting ShowLegend="True"></maps:MapLegendSetting>
</maps:ShapeFileLayer.LegendSettings>

```

## C#

```
MapLegendSetting legendSetting = new MapLegendSetting();  
legendSetting.ShowLegend = true;  
layer.LegendSettings = legendSetting;
```

### Legend type

The [LegendType](#) property is used to display the shapes and bubble legends in maps.

- Layers
- Bubbles

### Legend position

Based on the values of x (in the range of 0 to 100) and y (in the range of 0 to 100), the legends can be positioned using the [LegendPosition](#) property of the [MapLegendSetting](#) class. Legends will be positioned in the range of 0 to 100 (screen size ratio). By default, the *LegendPosition* is (50,10).

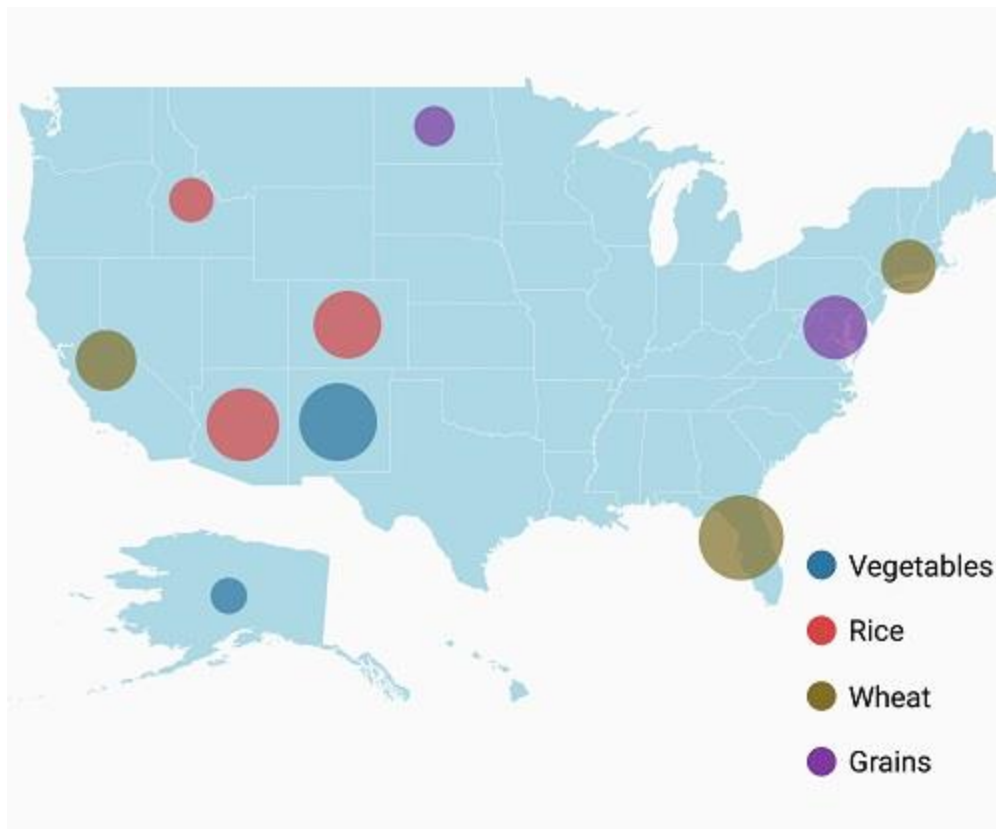
The legend items will be placed in multiple rows if size of the total legend exceeds the available size.

## XML

```
<maps:ShapeFileLayer.LegendSettings>  
  <maps:MapLegendSetting LegendType="Bubbles" HorizontalAlignment="Start"  
    LegendPosition="80,60" ShowLegend="True"></maps:MapLegendSetting>  
</maps:ShapeFileLayer.LegendSettings>
```

## C#

```
MapLegendSetting legendSetting = new MapLegendSetting();  
legendSetting.ShowLegend = true;  
legendSetting.LegendType = LegendType.Bubbles;  
legendSetting.LegendPosition = new Point(80, 60);  
legendSetting.HorizontalAlignment = HorizontalAlignment.Start;  
layer.LegendSettings = legendSetting;
```



### Legend alignment

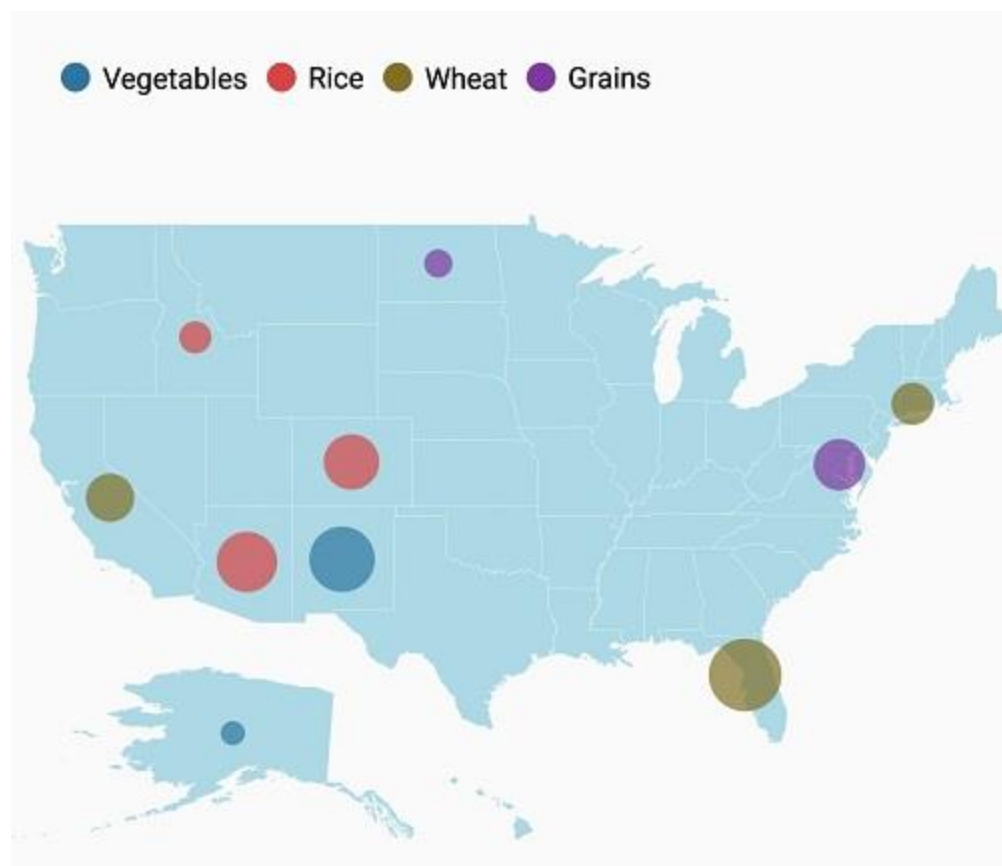
Legends can be aligned using the [HorizontalAlignment](#) and [VerticalAlignment](#) Properties. By default, the legends will be aligned to the center.

### XML

```
<maps:ShapeFileLayer.LegendSettings>  
<maps:MapLegendSetting LegendType="Bubbles" HorizontalAlignment="Start"  
LegendPosition="5,20" VerticalAlignment="Center"  
ShowLegend="True"></maps:MapLegendSetting>  
</maps:ShapeFileLayer.LegendSettings>
```

### C#

```
MapLegendSetting legendSetting = new MapLegendSetting();  
legendSetting.ShowLegend = true;  
legendSetting.LegendType = LegendType.Bubbles;  
legendSetting.LegendPosition = new Point(5, 20);  
legendSetting.HorizontalAlignment = HorizontalAlignment.Start;  
legendSetting.VerticalAlignment = VerticalAlignment.Center;  
layer.LegendSettings = legendSetting;
```



### Icon customization

The icon size of a legend can be customized using the [IconSize](#) property.

#### XML

```
<maps:ShapeFileLayer.LegendSettings>
  <maps:MapLegendSetting ShowLegend="True" LegendPosition="75,90">
    <maps:MapLegendSetting.IconSize>
      <Size Width="20" Height="20"/>
    </maps:MapLegendSetting.IconSize>
  </maps:MapLegendSetting>
</maps:ShapeFileLayer.LegendSettings>
```

#### C#

```
MapLegendSetting legendSetting = new MapLegendSetting();
legendSetting.ShowLegend = true;
legendSetting.LegendPosition = new Point(75, 90);
legendSetting.IconSize = new Size(20, 20);
layer.LegendSettings = legendSetting;
```

The icon shape can be customized using the [LegendIcon](#) property. By default, this property is set to circle for bubbles and rectangle for shapes.

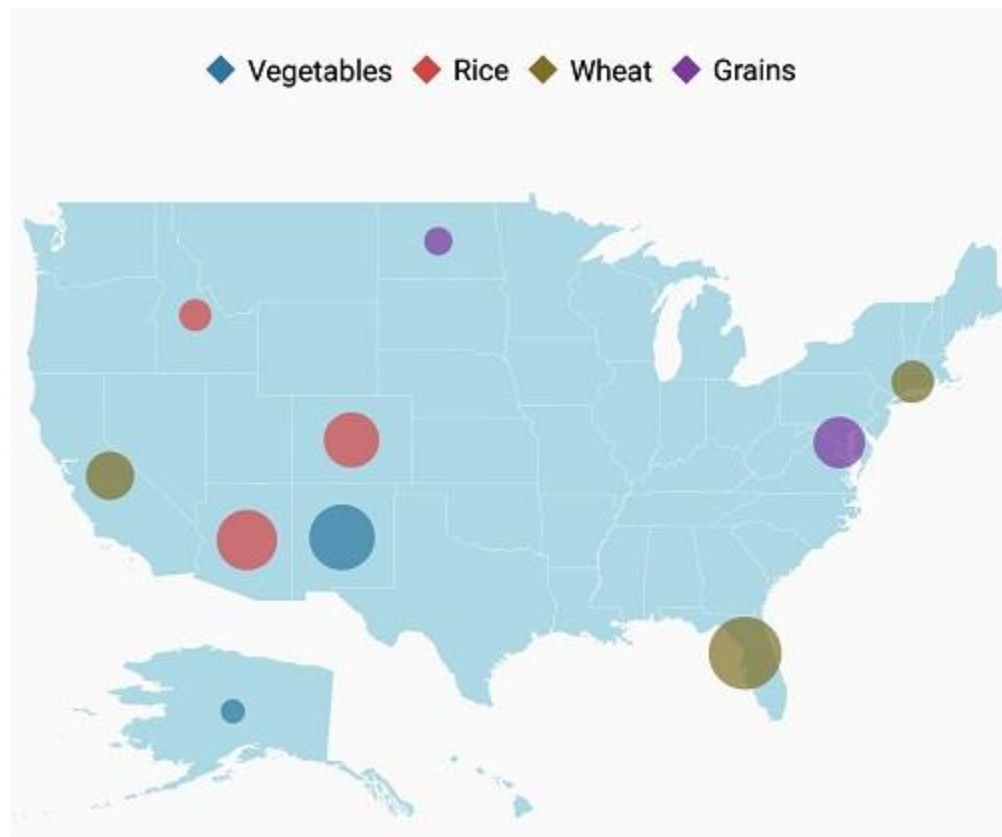
#### XML

```
<maps:ShapeFileLayer.LegendSettings>
```

```
<maps:MapLegendSetting ShowLegend="True" LegendType="Bubbles"
LegendIcon="Diamond">
</maps:MapLegendSetting>
</maps:ShapeFileLayer.LegendSettings>
```

**C#**

```
MapLegendSetting legendSetting = new MapLegendSetting();
legendSetting.ShowLegend = true;
legendSetting.LegendType = LegendType.Bubbles;
legendSetting.LegendIcon = LegendIcon.Diamond;
layer.LegendSettings = legendSetting;
```



Item margin

The [ItemMargin](#) property is used to set spacing between the legend items.

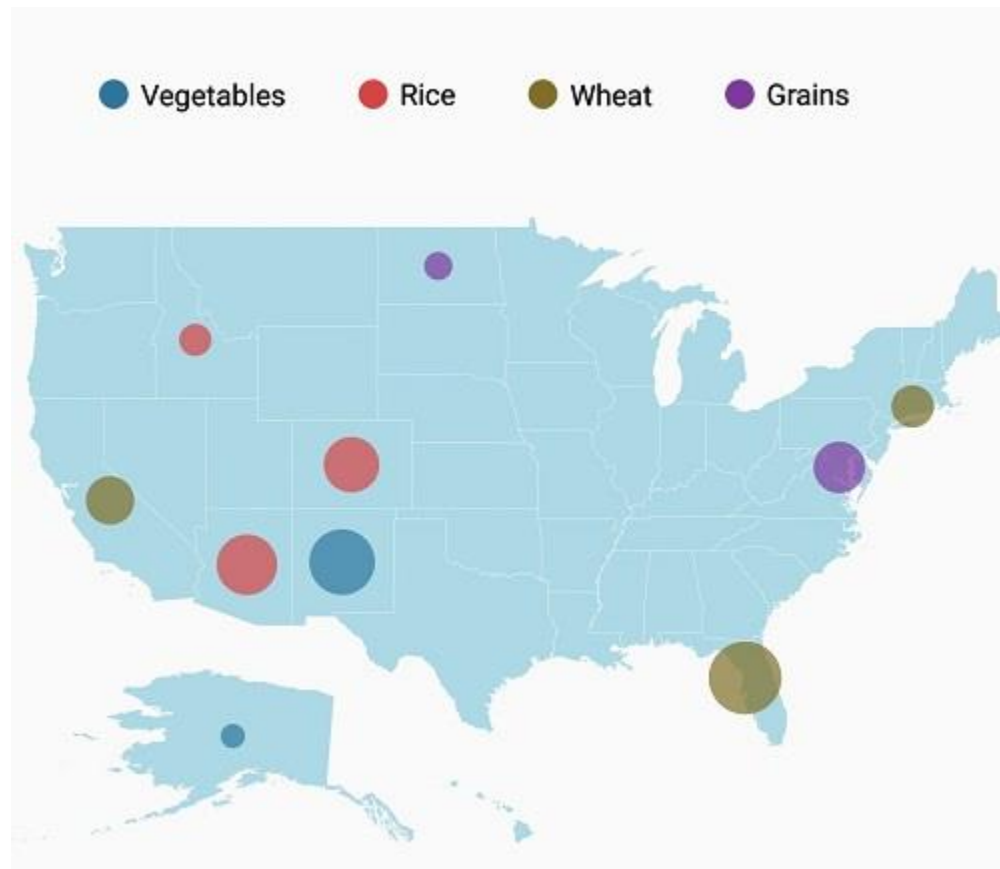
**XML**

```
<maps:ShapeFileLayer.LegendSettings>
<maps:MapLegendSetting LegendType="Bubbles" ShowLegend="True"
ItemMargin="30" />
</maps:ShapeFileLayer.LegendSettings>
```

**C#**

```
MapLegendSetting legendSetting = new MapLegendSetting();
legendSetting.ShowLegend = true;
```

```
legendSetting.LegendType = LegendType.Bubbles;
legendSetting.ItemMargin = 30;
layer.LegendSettings = legendSetting;
```



### Legend label

The [LegendLabel](#) provides information about the maps. It is specified under color mapping. If *LegendLabel* is not specified, ColorMapping values will be applied as legend label.

The following properties are used to customize the label of the legends:

- [TextColor](#) : Changes the color of the legend text.
- [FontAttributes](#) : Changes the font weight of the legend label.
- [FontFamily](#) : Changes the font family of the legend label.
- [FontSize](#) : Changes the text size of the legend label.

### XML

```
<maps:ShapeFileLayer.LegendSettings>
  <maps:MapLegendSetting LegendType="Bubbles" ShowLegend="True"
    FontFamily="algerian.ttf" TextColor="Maroon" />
</maps:ShapeFileLayer.LegendSettings>
```

### C#

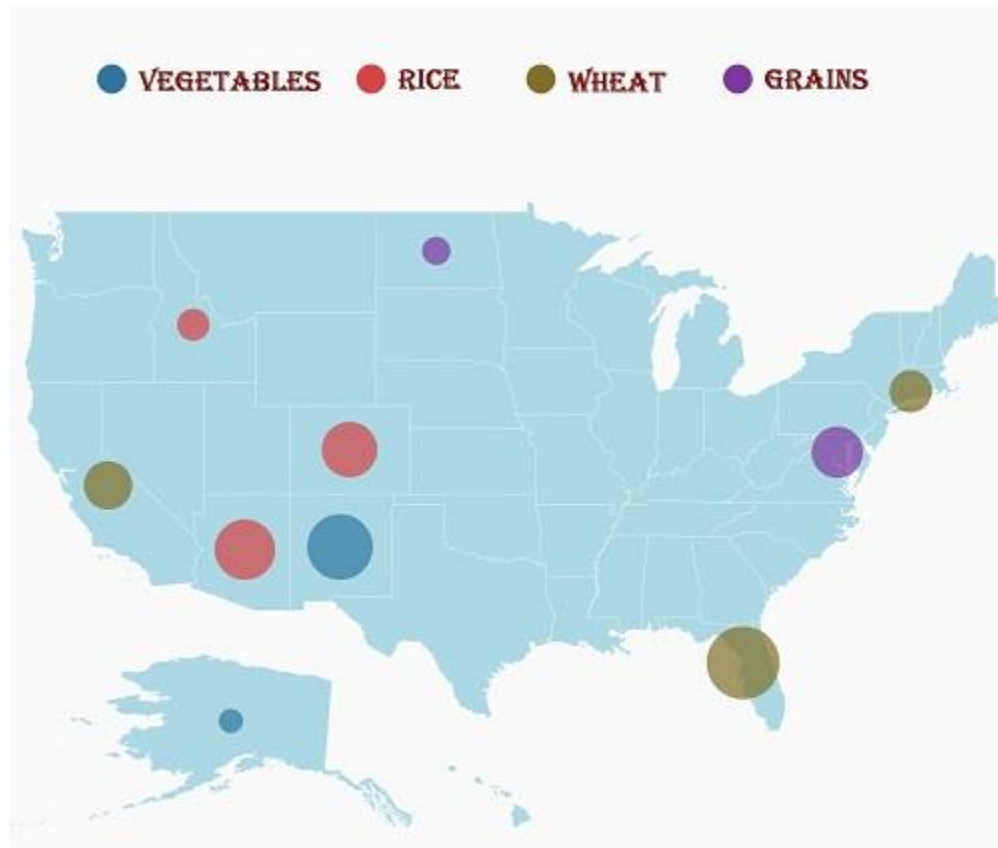
```
MapLegendSetting legendSetting = new MapLegendSetting();
```



```

legendSetting.ShowLegend = true;
legendSetting.LegendType = LegendType.Bubbles;
legendSetting.FontFamily = "algerian.ttf";
legendSetting.TextColor = Color.Maroon;
layer.LegendSettings = legendSetting;

```



The following code sample demonstrates how to add a legend to maps and customize it.

*Legend for bubbles*

**XML**

```

<Grid>
<Grid.BindingContext>
<local:ViewModel />
</Grid.BindingContext>
<maps:SfMaps>
<maps:SfMaps.Layers>
<maps:ShapeFileLayer x:Name="layer" Uri="usa_state.shp" ShapeIDPath="Name"
ShapeIDTableField="STATE_NAME"
ShowMapItems="True" ItemsSource="{Binding DataSource}">
<maps:ShapeFileLayer.BubbleMarkerSettings>
<maps:BubbleMarkerSetting ValuePath="index" ColorValuePath="index" >
<maps:BubbleMarkerSetting.ColorMappings>
<maps:RangeColorMapping Color="#2E769F" From="0" To="15" LegendLabel="0 - 15" />
<maps:RangeColorMapping Color="#D84444" To="30" From="15" LegendLabel="15-30" />

```

```

<maps:RangeColorMapping Color="#816F28" To="45" From="30" LegendLabel="30 -
45" />
<maps:RangeColorMapping Color="#7F38A0" To="60" From="45" LegendLabel="45 -
60" />
</maps:BubbleMarkerSetting.ColorMappings>
</maps:BubbleMarkerSetting>
</maps:ShapeFileLayer.BubbleMarkerSettings>
<maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeSetting ShapeFill="LightBlue"/>
</maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeFileLayer.LegendSettings>
<maps:MapLegendSetting LegendType="Bubbles" ItemMargin="30"
LegendIcon="Diamond" LegendPosition="5,20" HorizontalAlignment="Start"
VerticalAlignment="Bottom" FontFamily="algerian.ttf" FontSize="14"
TextColor="Maroon" ShowLegend="True"></maps:MapLegendSetting>
</maps:ShapeFileLayer.LegendSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
</Grid>

```

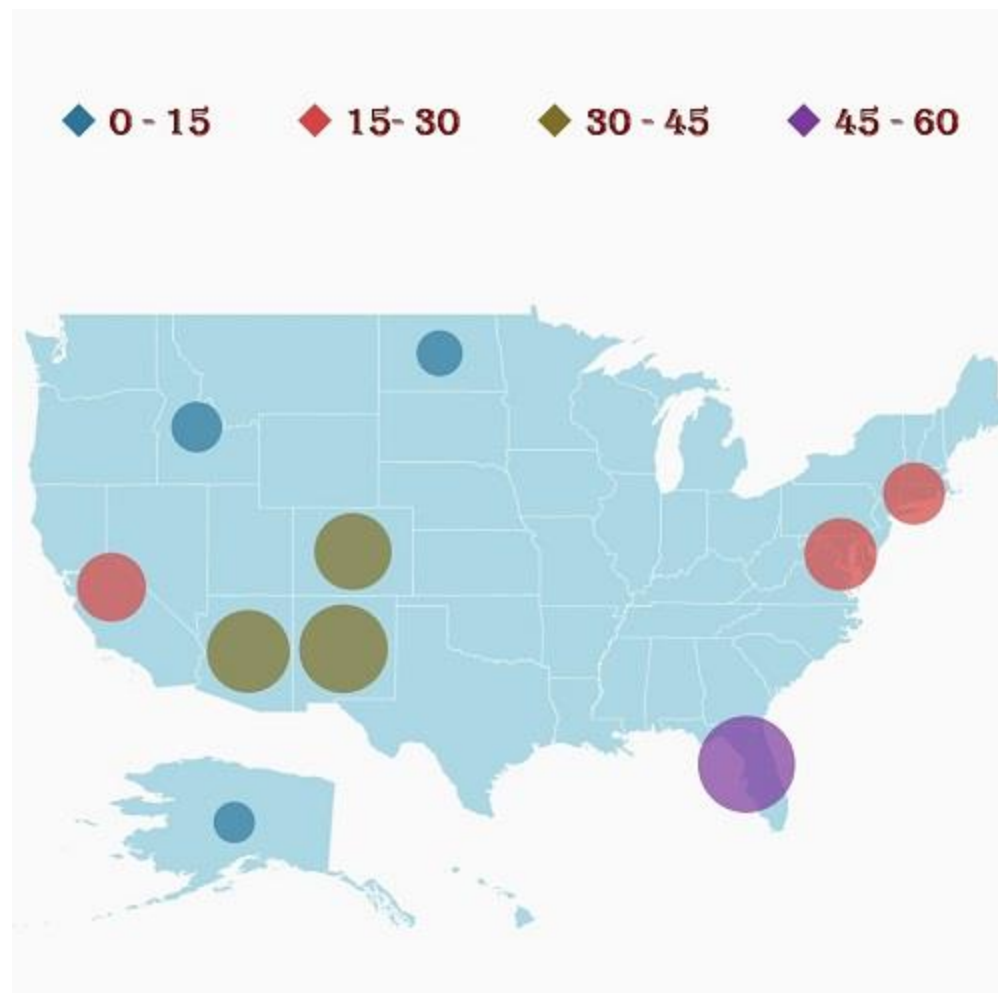
## C#

```

ViewModel viewModel = new ViewModel();
SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource = viewModel.DataSource;
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "Name";
layer.ShowMapItems = true;
BubbleMarkerSetting bubbleSetting = new BubbleMarkerSetting();
bubbleSetting.ValuePath = "index";
bubbleSetting.ColorValuePath = "index";
RangeColorMapping colorMapping1 = new RangeColorMapping() { Color =
Color.FromHex("#2E769F"), From = 0, To = 15 };
RangeColorMapping colorMapping2 = new RangeColorMapping() { Color =
Color.FromHex("#D84444"), From = 15, To = 30 };
RangeColorMapping colorMapping3 = new RangeColorMapping() { Color =
Color.FromHex("#816F28"), From = 30, To = 45 };
RangeColorMapping colorMapping4 = new RangeColorMapping() { Color =
Color.FromHex("#7F38A0"), From = 45, To = 50 };
bubbleSetting.ColorMappings.Add(colorMapping1);
bubbleSetting.ColorMappings.Add(colorMapping2);
bubbleSetting.ColorMappings.Add(colorMapping3);
bubbleSetting.ColorMappings.Add(colorMapping4);
layer.BubbleMarkerSettings = bubbleSetting;
MapLegendSetting legendSetting = new MapLegendSetting();
legendSetting.ShowLegend = true;
legendSetting.LegendPosition = new Point(5, 20);
legendSetting.LegendType = LegendType.Bubbles;
legendSetting.FontFamily = "algerian.ttf";
legendSetting.TextColor = Color.Maroon;
legendSetting.ItemMargin = 30;
legendSetting.LegendIcon = LegendIcon.Diamond;

```

```
legendSetting.HorizontalAlignment = HorizontalAlignment.Start;
layer.LegendSettings = legendSetting;
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeFill = Color.LightBlue;
layer.ShapeSettings = shapeSetting;
map.Layers.Add(layer);
this.Content = map;
public class AgricultureData
{
    public AgricultureData(string name, string type, int count)
    {
        Name = name;
        Type = type;
        index = count;
    }
    public string Name
    {
        get;
        set;
    }
    public string Type
    {
        get;
        set;
    }
    public int index
    {
        get;
        set;
    }
}
public class ViewModel
{
    public ViewModel()
    {
        DataSource = new ObservableCollection<AgricultureData>();
        DataSource.Add(new AgricultureData("Alaska", "Vegetables", 0));
        DataSource.Add(new AgricultureData("Arizona", "Rice", 36));
        DataSource.Add(new AgricultureData("California", "Wheat", 24));
        DataSource.Add(new AgricultureData("Colorado", "Rice", 31));
        DataSource.Add(new AgricultureData("North Dakota", "Grains", 4));
        DataSource.Add(new AgricultureData("Connecticut", "Wheat", 18));
        DataSource.Add(new AgricultureData("District of Columbia", "Grains", 27));
        DataSource.Add(new AgricultureData("Florida", "Wheat", 48));
        DataSource.Add(new AgricultureData("New Mexico", "Vegetables", 41));
        DataSource.Add(new AgricultureData("Idaho", "Rice", 8));
    }
    public ObservableCollection<AgricultureData> DataSource { get; set; }
}
```



*Legend for shapes*

### XML

```
<Grid>
  <Grid.BindingContext>
  <local:ViewModel/>
  </Grid.BindingContext>
  <maps:SfMaps x:Name="sfmap" BackgroundColor="White">
    <maps:SfMaps.Layers>
      <maps:ShapeFileLayer Uri="usa_state.shp" ItemsSource="{Binding Data}"
        ShapeIDPath="State" ShapeIDTableField="STATE_NAME" >
        <maps:ShapeFileLayer.LegendSettings>
          <maps:MapLegendSetting ItemMargin="30" LegendIcon="Diamond"
            LegendPosition="50,20"
            FontFamily="algerian.ttf" FontSize="14" TextColor="Maroon"
            ShowLegend="True">
          </maps:MapLegendSetting>
        </maps:ShapeFileLayer.LegendSettings>
        <maps:ShapeFileLayer.ShapeSettings>
          <maps:ShapeSetting ShapeColorValuePath="Candidate"
            ShapeValuePath="Candidate">
          <maps:ShapeSetting.ColorMappings>
            <maps:EqualColorMapping Color="#D84444" Value="Romney"
```

```

LegendLabel="Romney"/>
<maps:EqualColorMapping Color="#316DB5" Value="Obama"
LegendLabel="Obama"/>
</maps:ShapeSetting.ColorMappings>
</maps:ShapeSetting>
</maps:ShapeFileLayer.ShapeSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
</Grid>

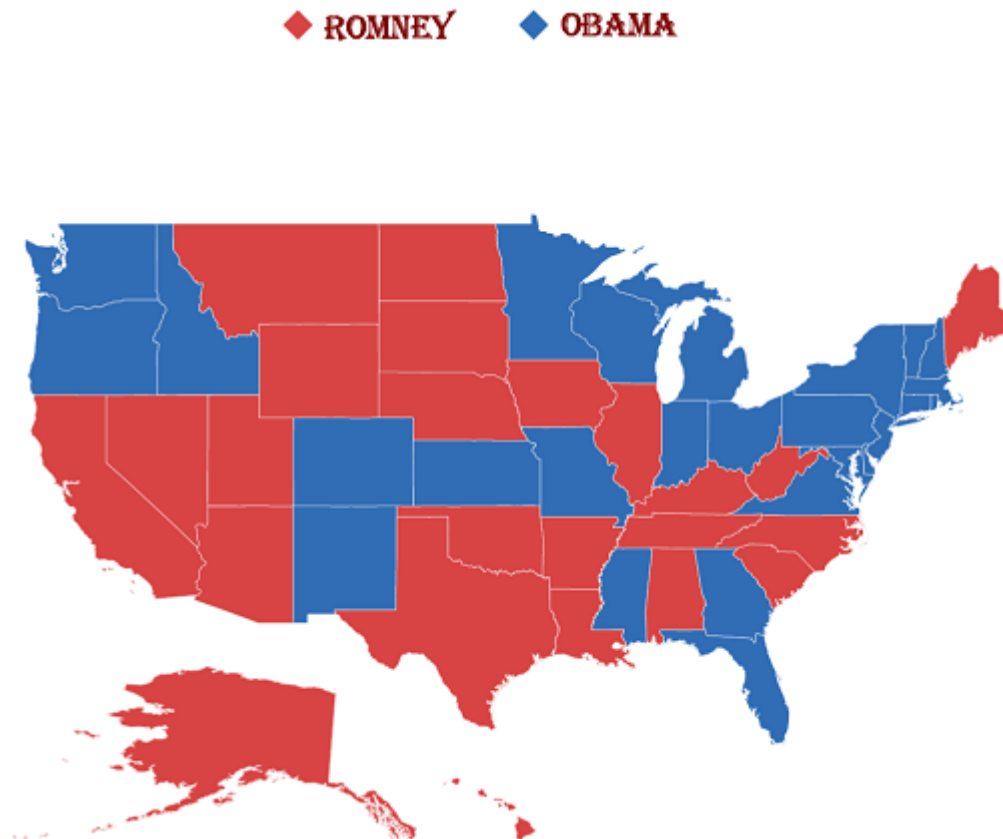
```

**C#**

```

ViewModel viewModel = new ViewModel();
SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.ItemsSource = viewModel.Data;
layer.ShapeIDTableField = "STATE_NAME";
layer.ShapeIDPath = "State";
map.Layers.Add(layer);
MapLegendSetting legendSetting = new MapLegendSetting();
legendSetting.ShowLegend = true;
legendSetting.LegendPosition = new Point(50, 20);
legendSetting.LegendType = LegendType.Layers;
legendSetting.FontFamily = "algerian.ttf";
legendSetting.TextColor = Color.Maroon;
legendSetting.ItemMargin = 30;
legendSetting.LegendIcon = LegendIcon.Diamond;
layer.LegendSettings = legendSetting;
EqualColorMapping colorMapping = new EqualColorMapping();
colorMapping.Color = Color.FromHex("#D84444");
colorMapping.LegendLabel = "Romney";
colorMapping.Value = "Romney";
EqualColorMapping colorMapping1 = new EqualColorMapping();
colorMapping1.Color = Color.FromHex("#316DB5");
colorMapping1.LegendLabel = "Obama";
colorMapping1.Value = "Obama";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeValuePath = "Candidate";
shapeSetting.ShapeColorValuePath = "Candidate";
shapeSetting.ColorMappings.Add(colorMapping);
shapeSetting.ColorMappings.Add(colorMapping1);
layer.ShapeSettings = shapeSetting;
this.Content = map;

```



## Tooltip

Tooltip provides additional information about the shapes in the maps. To enable tooltip, set the [ShowTooltip](#) property to true, and set the [ValuePath](#) property of tooltip.

Tooltip is displayed by tapping the following elements:

- Shapes
- Bubbles
- Markers

The following code sample shows how to enable tooltip by tapping the shapes.

### XML

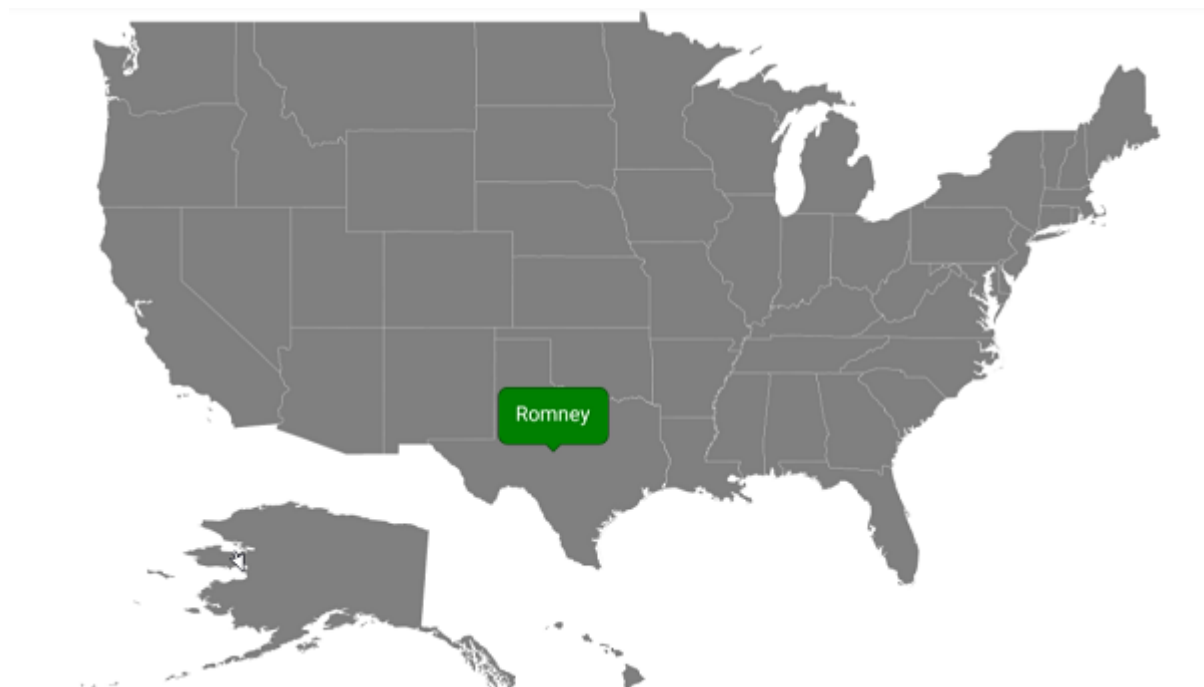
```
<maps:SfMaps>
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME"
      ItemsSource="{Binding Data}">
      <maps:ShapeFileLayer.TooltipSettings>
        <maps:TooltipSetting ShowTooltip="True" ValuePath="Candidate"/>
      </maps:ShapeFileLayer.TooltipSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

**C#**

```
SfMaps sfMaps = new SfMaps();  
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();  
shapeFileLayer.Uri = "usa_state.shp";  
ViewModel model = new ViewModel();  
shapeFileLayer.ItemsSource = model.Data;  
shapeFileLayer.ShapeIDTableField = "STATE_NAME";  
shapeFileLayer.ShapeIDPath = "State";  
shapeFileLayer.TooltipSettings.ShowTooltip = true;  
shapeFileLayer.TooltipSettings.ValuePath = "Candidate";  
sfMaps.Layers.Add(shapeFileLayer);  
Content = sfMaps;
```

[Tooltip customization](#)

The appearance of the tooltip can be customized using the following properties:



*[TextColor](#): Customizes the text color of tooltip. \* [BackgroundColor](#): Customizes the background color of tooltip. \* [StrokeColor](#): Customizes the stroke color of tooltip. \* [StrokeWidth](#): Customizes the stroke width of tooltip. \* [Duration](#): Specifies the duration of tooltip to be displayed. \* [Margin](#): Sets the margin for tooltip.*

The following code sample shows all the above customizations.

#### XML

```
<maps:SfMaps>
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME"
      ItemsSource="{Binding Data}">
      <maps:ShapeFileLayer.TooltipSettings>
        <maps:TooltipSetting ShowTooltip="True" ValuePath="Candidate"
          TextColor="White" Margin="10"
          BackgroundColor="Green" StrokeColor="Black" StrokeWidth="2"
          Duration="2000"/>
      </maps:ShapeFileLayer.TooltipSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

#### C#

```
SfMaps sfMaps = new SfMaps();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "usa_state.shp";
ViewModel model = new ViewModel();
shapeFileLayer.ItemsSource = model.Data;
shapeFileLayer.ShapeIDTableField = "STATE_NAME";
shapeFileLayer.ShapeIDPath = "State";
shapeFileLayer.TooltipSettings.ShowTooltip = true;
```



```

shapeFileLayer.TooltipSettings.ValuePath = "Candidate";
shapeFileLayer.TooltipSettings.TextColor = Color.White;
shapeFileLayer.TooltipSettings.BackgroundColor = Color.Green;
shapeFileLayer.TooltipSettings.StrokeColor = Color.Black;
shapeFileLayer.TooltipSettings.StrokeWidth = 2;
shapeFileLayer.TooltipSettings.Margin = 10;
shapeFileLayer.TooltipSettings.Duration = 2000;
sfMaps.Layers.Add(shapeFileLayer);
Content = sfMaps;

```

![Xamarin Tooltip Customization Image](Images/tooltip\_customization.png)

### Tooltip font customization

The font can be customized using the [FontSize](#), [FontAttributes](#), and [FontFamily](#) properties of tooltip.

### XML

```

<maps:SfMaps>
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
      <maps:ShapeFileLayer.TooltipSettings>
        <maps:TooltipSetting ShowTooltip="True" ValuePath="Candidate" FontSize="15"
          FontAttributes="Bold">
          <maps:TooltipSetting.FontFamily>
            <OnPlatform x:TypeArguments="x:String" iOS="Chalkduster" Android="monospace"
              WinPhone="Chiller" />
          </maps:TooltipSetting.FontFamily>
        </maps:TooltipSetting>
      </maps:ShapeFileLayer.TooltipSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>

```

### C#

```

SfMaps sfMaps = new SfMaps();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "usa_state.shp";
ViewModel model = new ViewModel();
shapeFileLayer.ItemsSource = model.Data;
shapeFileLayer.ShapeIDTableField = "STATE_NAME";
shapeFileLayer.ShapeIDPath = "State";
shapeFileLayer.TooltipSettings.ShowTooltip = true;
shapeFileLayer.TooltipSettings.ValuePath = "Candidate";
shapeFileLayer.TooltipSettings.FontSize = 15;
shapeFileLayer.TooltipSettings.FontAttributes = FontAttributes.Bold;
shapeFileLayer.TooltipSettings.FontFamily = Device.RuntimePlatform ==
Device.iOS ? "Chalkduster" : Device.RuntimePlatform == Device.Android ?
"monospace" : "Chiller";
sfMaps.Layers.Add(shapeFileLayer);
Content = sfMaps;

```



### Custom template for tooltip

The maps control provides options to design your own template for tooltip using the [TooltipTemplate](#) property.

### XML

```
<maps:SfMaps Margin="20" BackgroundColor="White">
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeValuePath="Candidate"
          ShapeColorValuePath="Candidate" >
          <maps:ShapeSetting.ColorMappings>
            <maps:EqualColorMapping Color="#D84444" Value = "Romney"/>
            <maps:EqualColorMapping Color="#316DB5" Value="Obama"/>
          </maps:ShapeSetting.ColorMappings>
        </maps:ShapeSetting>
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.TooltipSettings>
        <maps:TooltipSetting ShowTooltip="True" ValuePath="State" Duration="2000">
          <maps:TooltipSetting.TooltipTemplate>
            <DataTemplate>
              <StackLayout>
                <Label Text="{Binding State}" HorizontalTextAlignment="Center"
                  VerticalTextAlignment="Center" TextColor="#FFFFFF" FontAttributes="Bold"
                  FontFamily="Helvetica" Margin="0" FontSize="12" Grid.Row="0" />
                <BoxView Color="#888C91" HeightRequest="1" VerticalOptions="FillAndExpand"
                  HorizontalOptions="FillAndExpand" />
                <Label Text="{Binding Candidate}" VerticalTextAlignment="Center"
                  HorizontalOptions="Center" TextColor="#FFFFFF" FontAttributes="Bold"
                  FontFamily="Helvetica" FontSize="12" />
              </StackLayout>
            </DataTemplate>
          </maps:TooltipSetting.TooltipTemplate>
        </maps:TooltipSetting>
      </maps:ShapeFileLayer.TooltipSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

```

</maps:TooltipSetting>
</maps:ShapeFileLayer.TooltipSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>

```

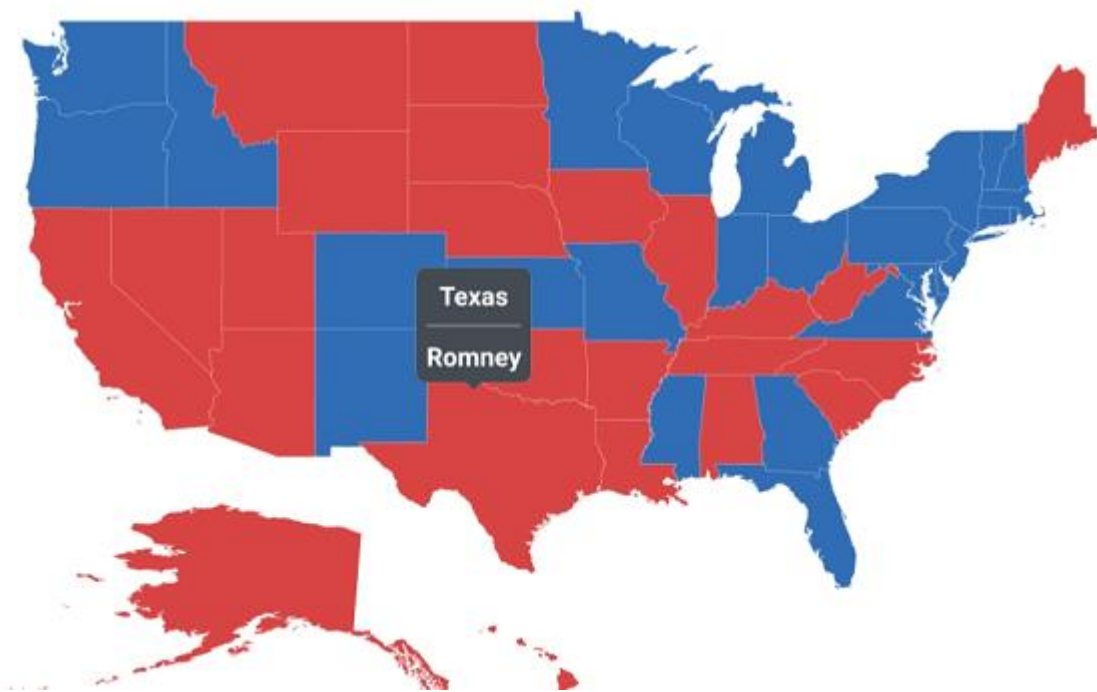
## C#

```

SfMaps sfMaps = new SfMaps();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "usa_state.shp";
ViewModel model = new ViewModel();
shapeFileLayer.ItemsSource = model.Data;
shapeFileLayer.ShapeIDTableField = "STATE_NAME";
shapeFileLayer.ShapeIDPath = "State";
EqualColorMapping colorMapping = new EqualColorMapping();
colorMapping.Color = Color.FromHex("#D84444");
colorMapping.Value = "Romney";
EqualColorMapping colorMapping1 = new EqualColorMapping();
colorMapping1.Color = Color.FromHex("#316DB5");
colorMapping1.Value = "Obama";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeValuePath = "Candidate";
shapeSetting.ShapeColorValuePath = "Candidate";
shapeSetting.ColorMappings.Add(colorMapping);
shapeSetting.ColorMappings.Add(colorMapping1);
shapeFileLayer.ShapeSettings = shapeSetting;
shapeFileLayer.TooltipSettings.ShowTooltip = true;
shapeFileLayer.TooltipSettings.ValuePath = "State";
DataTemplate template = new DataTemplate(() =>
{
    StackLayout categoryLayout = new StackLayout();
    Label category = new Label();
    category.HorizontalTextAlignment = TextAlignment.Center;
    category.VerticalTextAlignment = TextAlignment.Center;
    category.TextColor = Color.FromHex("#FFFFFF");
    category.FontAttributes = FontAttributes.Bold;
    category.FontFamily = "Helvetica";
    category.Margin = 0;
    category.FontSize = 12;
    category.SetBinding(Label.TextProperty, "State");
    categoryLayout.Children.Add(category);
    BoxView boxView = new BoxView();
    boxView.Color = Color.FromHex("#888C91");
    boxView.HeightRequest = 1;
    boxView.VerticalOptions = LayoutOptions.FillAndExpand;
    boxView.HorizontalOptions = LayoutOptions.FillAndExpand;
    categoryLayout.Children.Add(boxView);
    Label category1 = new Label();
    category1.HorizontalTextAlignment = TextAlignment.Center;
    category1.VerticalTextAlignment = TextAlignment.Center;
    category1.TextColor = Color.FromHex("#FFFFFF");
    category1.FontAttributes = FontAttributes.Bold;
    category1.FontFamily = "Helvetica";
    category1.Margin = 0;
    category1.FontSize = 12;

```

```
category1.SetBinding(Label.TextProperty, "Candidate");
categoryLayout.Children.Add(category1);
return categoryLayout;
});
shapeFileLayer.TooltipSettings.TooltipTemplate = template;
sfMaps.Layers.Add(shapeFileLayer);
Content = sfMaps;
```



### Setting animation for tooltip

You can enable or disable the animation of tooltip using the [EnableAnimation](#) property.

#### XML

```
<maps:ShapeFileLayer.TooltipSettings>
  <maps:TooltipSetting ShowTooltip="True" ValuePath="State"
  EnableAnimation="False"/>
</maps:ShapeFileLayer.TooltipSettings>
```

#### C#

```
shapeFileLayer.TooltipSettings.ShowTooltip = true;
shapeFileLayer.TooltipSettings.ValuePath = "State";
shapeFileLayer.TooltipSettings.EnableAnimation = false;
sfMaps.Layers.Add(shapeFileLayer);
```

### Setting pointer length for tooltip

You can customize the pointer length of the tooltip using the [PointerLength](#) property.

#### XML

```
<maps:SfMaps>
  <maps:SfMaps.Layers >
```

```

<maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
  <maps:ShapeFileLayer.TooltipSettings>
    <maps:TooltipSetting ShowTooltip="True" ValuePath="Candidate"
    PointerLength="18"/>
  </maps:ShapeFileLayer.TooltipSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>

```

## C#

```

SfMaps sfMaps = new SfMaps();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "usa_state.shp";
ViewModel model = new ViewModel();
shapeFileLayer.ItemsSource = model.Data;
shapeFileLayer.ShapeIDTableField = "STATE_NAME";
shapeFileLayer.ShapeIDPath = "State";
shapeFileLayer.TooltipSettings.ShowTooltip = true;
shapeFileLayer.TooltipSettings.ValuePath = "Candidate";
shapeFileLayer.TooltipSettings.PointerLength = 18;
sfMaps.Layers.Add(shapeFileLayer);
Content = sfMaps;

```



## Events

### Tooltip opening event

This event occurs whenever you select a shape, bubble, or marker. You will get the [Data](#) and [TooltipType](#) properties as arguments from [TooltipOpeningEventArgs](#) handler, and you can cancel the event for a particular shape using the [Cancel](#) property.

**XML**

```
<maps:SfMaps TooltipOpening="SfMaps_TooltipOpening">
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
      <maps:ShapeFileLayer.TooltipSettings>
        <maps:TooltipSetting ShowTooltip="True" ValuePath="State" />
      </maps:ShapeFileLayer.TooltipSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

**C#**

```
private void SfMaps_TooltipOpening(object sender, TooltipOpeningEventArgs e)
{
    if ((e.Data is ElectionData) && (e.Data as ElectionData).State == "Montana")
    {
        e.Cancel = true;
    }
}
```

## Tooltip for bubbles

You can get tooltip by tapping the bubbles in the shapes.

**XML**

```
<maps:SfMaps>
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
      <maps:ShapeFileLayer.BubbleMarkerSettings>
        <maps:BubbleMarkerSetting ValuePath="Electors" ColorValuePath="Electors"
          Fill="#7F38A0">
          <maps:BubbleMarkerSetting.TooltipSettings>
            <maps:TooltipSetting ShowTooltip="True" ValuePath="Electors"/>
          </maps:BubbleMarkerSetting.TooltipSettings>
        </maps:BubbleMarkerSetting>
      </maps:ShapeFileLayer.BubbleMarkerSettings>
    <maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeSetting ShapeFill="LightGray"/>
    </maps:ShapeFileLayer.ShapeSettings>
  </maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

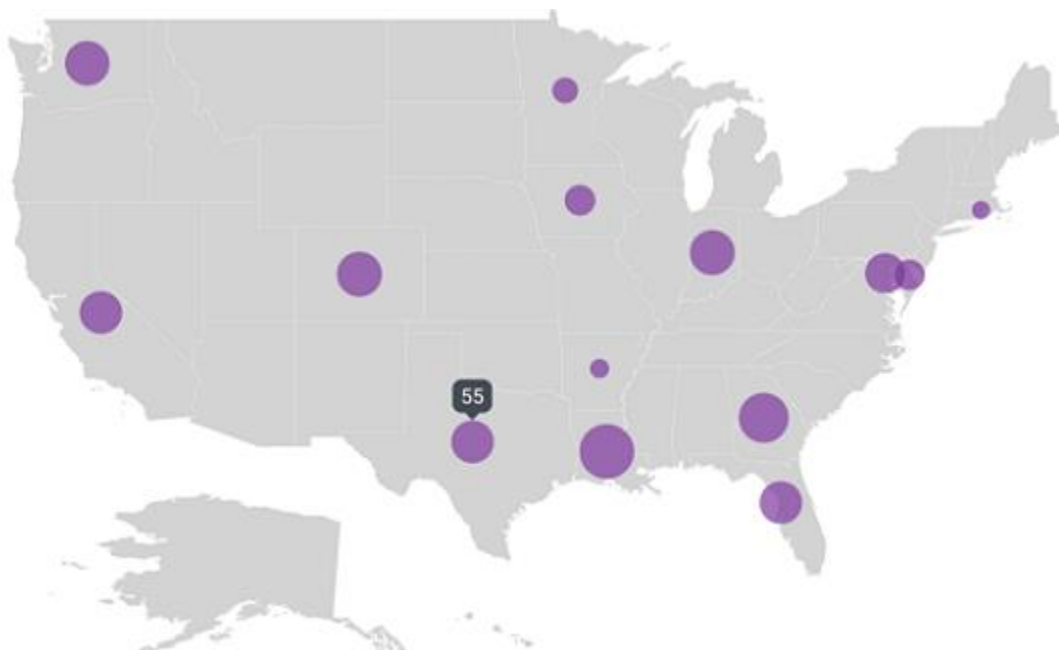
**C#**

```
SfMaps sfMaps = new SfMaps();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "usa_state.shp";
ViewModel model = new ViewModel();
shapeFileLayer.ItemsSource = model.Data;
shapeFileLayer.ShapeIDTableField = "STATE_NAME";
```

```

shapeFileLayer.ShapeIDPath = "State";
BubbleMarkerSetting bubbleMarkerSetting = new BubbleMarkerSetting();
bubbleMarkerSetting.ColorValuePath = "Electors";
bubbleMarkerSetting.ValuePath = "Electors";
bubbleMarkerSetting.Fill = Color.FromHex("#7F38A0");
bubbleMarkerSetting.TooltipSettings.ShowTooltip = true;
bubbleMarkerSetting.TooltipSettings.ValuePath = "Electors";
shapeFileLayer.BubbleMarkerSettings = bubbleMarkerSetting;
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeFill = Color.LightGray;
shapeFileLayer.ShapeSettings = shapeSetting;
sfMaps.Layers.Add(shapeFileLayer);
Content = sfMaps;

```



### Bubble tooltip customization

To customize the tooltip in the bubbles, refer to this link.

### XML

```

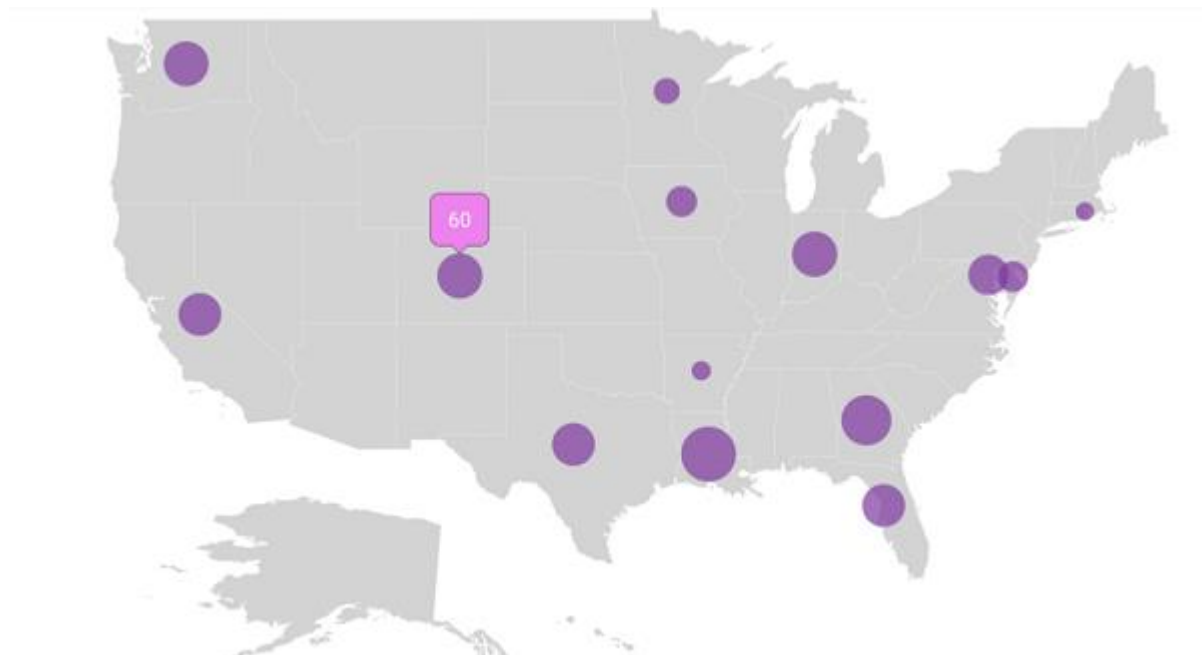
<maps:SfMaps>
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
      <maps:ShapeFileLayer.BubbleMarkerSettings>
        <maps:BubbleMarkerSetting ValuePath="Electors" ColorValuePath="Electors"
          Fill="#7F38A0">
          <maps:BubbleMarkerSetting.TooltipSettings>
            <maps:TooltipSetting ShowTooltip="True" ValuePath="Electors"
              TextColor="White" Margin="10" BackgroundColor="Violet" StrokeColor="Black"
              StrokeWidth="2" Duration="2000"/>
          </maps:BubbleMarkerSetting.TooltipSettings>
        </maps:BubbleMarkerSetting>
      </maps:ShapeFileLayer.BubbleMarkerSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>

```

```
<maps:ShapeSetting ShapeFill="LightGray"/>
</maps:ShapeFileLayer.ShapeSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

## C#

```
SfMaps sfMaps = new SfMaps();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "usa_state.shp";
ViewModel model = new ViewModel();
shapeFileLayer.ItemsSource = model.Data;
shapeFileLayer.ShapeIDTableField = "STATE_NAME";
shapeFileLayer.ShapeIDPath = "State";
BubbleMarkerSetting bubbleMarkerSetting = new BubbleMarkerSetting();
bubbleMarkerSetting.ColorValuePath = "Electors";
bubbleMarkerSetting.ValuePath = "Electors";
bubbleMarkerSetting.Fill = Color.FromHex("#7F38A0");
bubbleMarkerSetting.TooltipSettings.ShowTooltip = true;
bubbleMarkerSetting.TooltipSettings.TextColor = Color.White;
bubbleMarkerSetting.TooltipSettings.BackgroundColor = Color.Violet;
bubbleMarkerSetting.TooltipSettings.StrokeColor = Color.Black;
bubbleMarkerSetting.TooltipSettings.StrokeWidth = 2;
bubbleMarkerSetting.TooltipSettings.Margin = 10;
bubbleMarkerSetting.TooltipSettings.Duration = 2000;
bubbleMarkerSetting.TooltipSettings.ValuePath = "Electors";
shapeFileLayer.BubbleMarkerSettings = bubbleMarkerSetting;
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeFill = Color.LightGray;
shapeFileLayer.ShapeSettings = shapeSetting;
sfMaps.Layers.Add(shapeFileLayer);
Content = sfMaps;
```





## Tooltip for markers

You can get the tooltip by tapping the markers in the shapes and markers in the imagery layer.

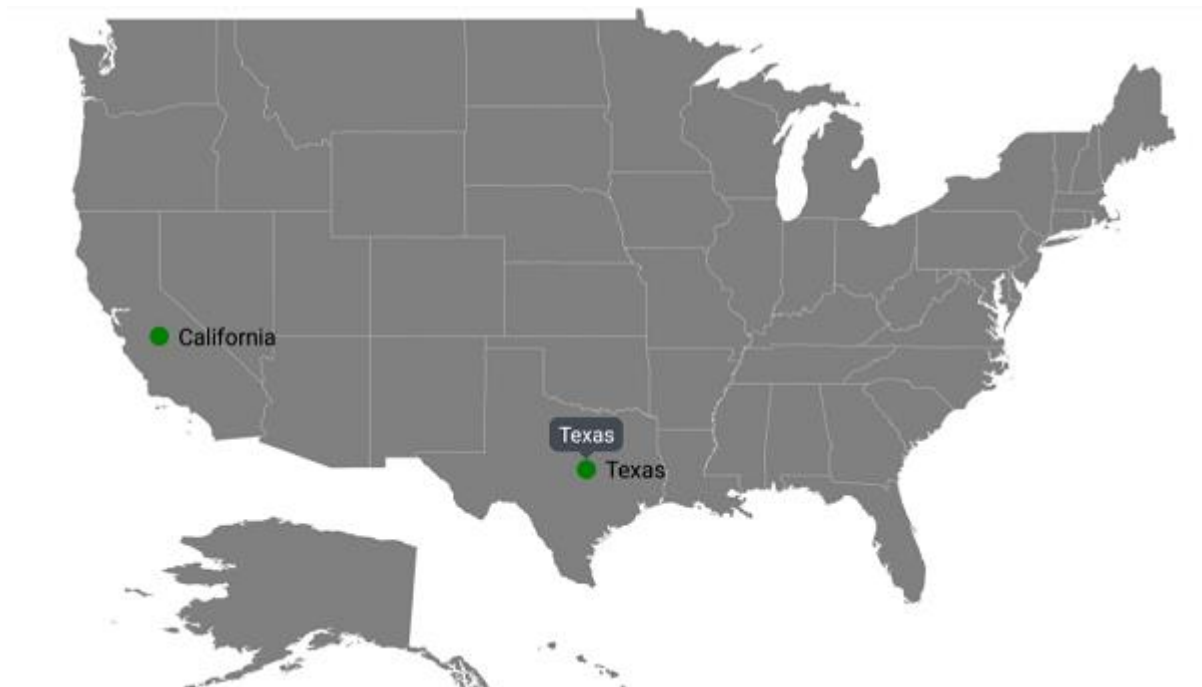
### Marker tooltip in shape layer

#### XML

```
<maps:SfMaps>
<maps:SfMaps.Layers >
<maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
<maps:ShapeFileLayer.Markers>
<maps:MapMarker Label="Texas" Latitude="31.267153" Longitude="-97.7430608"/>
<maps:MapMarker Label="California" Latitude="37" Longitude="-120"/>
</maps:ShapeFileLayer.Markers>
<maps:ShapeFileLayer.MarkerSettings>
<maps:MapMarkerSetting>
<maps:MapMarkerSetting.TooltipSettings>
<maps:TooltipSetting ShowTooltip="True" ValuePath="Label" />
</maps:MapMarkerSetting.TooltipSettings>
</maps:MapMarkerSetting>
</maps:ShapeFileLayer.MarkerSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

#### C#

```
SfMaps sfMaps = new SfMaps();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "usa_state.shp";
ViewModel model = new ViewModel();
shapeFileLayer.ItemsSource = model.Data;
shapeFileLayer.ShapeIDTableField = "STATE_NAME";
shapeFileLayer.ShapeIDPath = "State";
MapMarker mapMarker = new MapMarker();
mapMarker.Label = "Texas";
mapMarker.Latitude = "31.267153";
mapMarker.Longitude = "-97.7430608";
shapeFileLayer.Markers.Add(mapMarker);
MapMarker mapMarker1 = new MapMarker();
mapMarker1.Label = "California";
mapMarker1.Latitude = "37";
mapMarker1.Longitude = "-120";
shapeFileLayer.Markers.Add(mapMarker1);
MapMarkerSetting mapMarkerSetting = new MapMarkerSetting();
mapMarkerSetting.TooltipSettings.ShowTooltip = true;
mapMarkerSetting.TooltipSettings.ValuePath = "Label";
mapMarkerSetting.TooltipSettings.Duration = 2000;
shapeFileLayer.MarkerSettings = mapMarkerSetting;
sfMaps.Layers.Add(shapeFileLayer);
Content = sfMaps;
```



*Marker tooltip in imagery layer*

#### **XML**

```
<maps:SfMaps BackgroundColor="White" Margin="10">
  <maps:SfMaps.Layers >
    <maps:ImageryLayer>
      <maps:ImageryLayer.Markers>
        <maps:MapMarker Label="India" Latitude="20.593683" Longitude="78.962883" />
      </maps:ImageryLayer.Markers>
      <maps:ImageryLayer.MarkerSettings>
        <maps:MapMarkerSetting MarkerIcon="Image" ImageSource="pin.png"
          IconSize="15">
          <maps:MapMarkerSetting.TooltipSettings>
            <maps:TooltipSetting ShowTooltip="True" ValuePath="Label" TextColor="White"
              Margin="10" BackgroundColor="Navy"
              StrokeColor="Black" StrokeWidth="2" />
          </maps:MapMarkerSetting.TooltipSettings>
        </maps:MapMarkerSetting>
      </maps:ImageryLayer.MarkerSettings>
    </maps:ImageryLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

#### **C#**

```
SfMaps sfMaps = new SfMaps();
ImageryLayer imageryLayer = new ImageryLayer();
MapMarker mapMarker = new MapMarker();
mapMarker.Label = "India";
mapMarker.Latitude = "20.593683";
mapMarker.Longitude = "78.962883";
imageryLayer.Markers.Add(mapMarker);
MapMarkerSetting mapMarkerSetting = new MapMarkerSetting();
```

```
mapMarkerSetting.MarkerIcon = MapMarkerIcon.Image;
mapMarkerSetting.ImageSource = "pin.png";
mapMarkerSetting.IconSize = 15;
mapMarkerSetting.TooltipSettings.ShowTooltip = true;
mapMarkerSetting.TooltipSettings.ValuePath = "Label";
mapMarkerSetting.TooltipSettings.TextColor = Color.White;
mapMarkerSetting.TooltipSettings.BackgroundColor = Color.Navy;
mapMarkerSetting.TooltipSettings.StrokeColor = Color.Black;
mapMarkerSetting.TooltipSettings.StrokeWidth = 2;
mapMarkerSetting.TooltipSettings.Margin = 10;
imageryLayer.MarkerSettings = mapMarkerSetting;
sfMaps.Layers.Add(imageryLayer);
Content = sfMaps;
```



### Marker tooltip customization

To customize the tooltip in the marker, refer to this link.

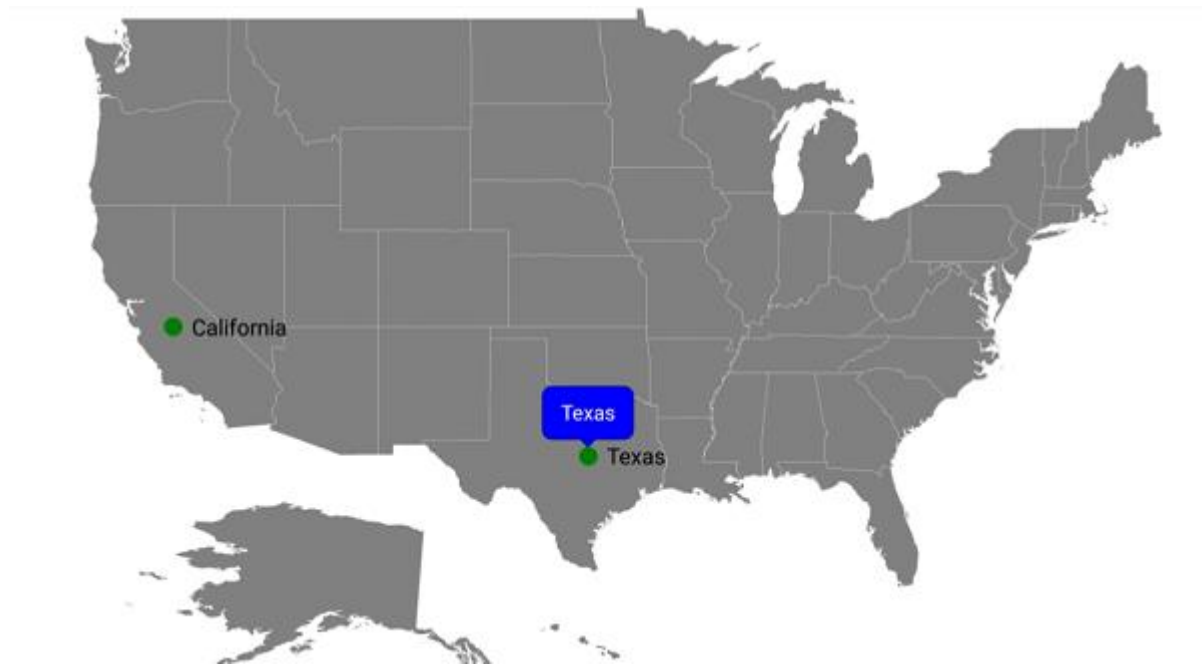
### XML

```
<maps:SfMaps BackgroundColor="White" Margin="10">
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" ShapeIDPath="State"
      ShapeIDTableField="STATE_NAME" ItemsSource="{Binding Data}">
      <maps:ShapeFileLayer.Markers>
        <maps:MapMarker Label="Texas" Latitude="31.267153" Longitude="-97.7430608"/>
        <maps:MapMarker Label="California" Latitude="37" Longitude="-120"/>
      </maps:ShapeFileLayer.Markers>
      <maps:ShapeFileLayer.MarkerSettings>
        <maps:MapMarkerSetting>
          <maps:MapMarkerSetting.TooltipSettings>
            <maps:TooltipSetting ShowTooltip="True" ValuePath="Label" TextColor="White"
```

```
Margin="10" BackgroundColor="Blue"
StrokeColor="Black" StrokeWidth="2" Duration="2000"/>
</maps:MapMarkerSetting.TooltipSettings>
</maps:MapMarkerSetting>
</maps:ShapeFileLayer.MarkerSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

## C#

```
SfMaps sfMaps = new SfMaps();
ShapeFileLayer shapeFileLayer = new ShapeFileLayer();
shapeFileLayer.Uri = "usa_state.shp";
ViewModel model = new ViewModel();
shapeFileLayer.ItemsSource = model.Data;
shapeFileLayer.ShapeIDTableField = "STATE_NAME";
shapeFileLayer.ShapeIDPath = "State";
MapMarker mapMarker = new MapMarker();
mapMarker.Label = "Texas";
mapMarker.Latitude = "31.267153";
mapMarker.Longitude = "-97.7430608";
shapeFileLayer.Markers.Add(mapMarker);
MapMarker mapMarker1 = new MapMarker();
mapMarker1.Label = "California";
mapMarker1.Latitude = "37";
mapMarker1.Longitude = "-120";
shapeFileLayer.Markers.Add(mapMarker1);
MapMarkerSetting mapMarkerSetting = new MapMarkerSetting();
mapMarkerSetting.TooltipSettings.ShowTooltip = true;
mapMarkerSetting.TooltipSettings.ValuePath = "Label";
mapMarkerSetting.TooltipSettings.TextColor = Color.White;
mapMarkerSetting.TooltipSettings.BackgroundColor = Color.Blue;
mapMarkerSetting.TooltipSettings.StrokeColor = Color.Black;
mapMarkerSetting.TooltipSettings.StrokeWidth = 2;
mapMarkerSetting.TooltipSettings.Margin = 10;
mapMarkerSetting.TooltipSettings.Duration = 2000;
shapeFileLayer.MarkerSettings = mapMarkerSetting;
sfMaps.Layers.Add(shapeFileLayer);
Content = sfMaps;
```



## Sublayer

Sublayer in the maps control allows to load multiple shape files in a single container and enables maps to display more information.

### Adding sublayers in ShapeFileLayer

You can add multiple shape files in the [ShapeFileLayer](#) using the [Sublayers](#) property.

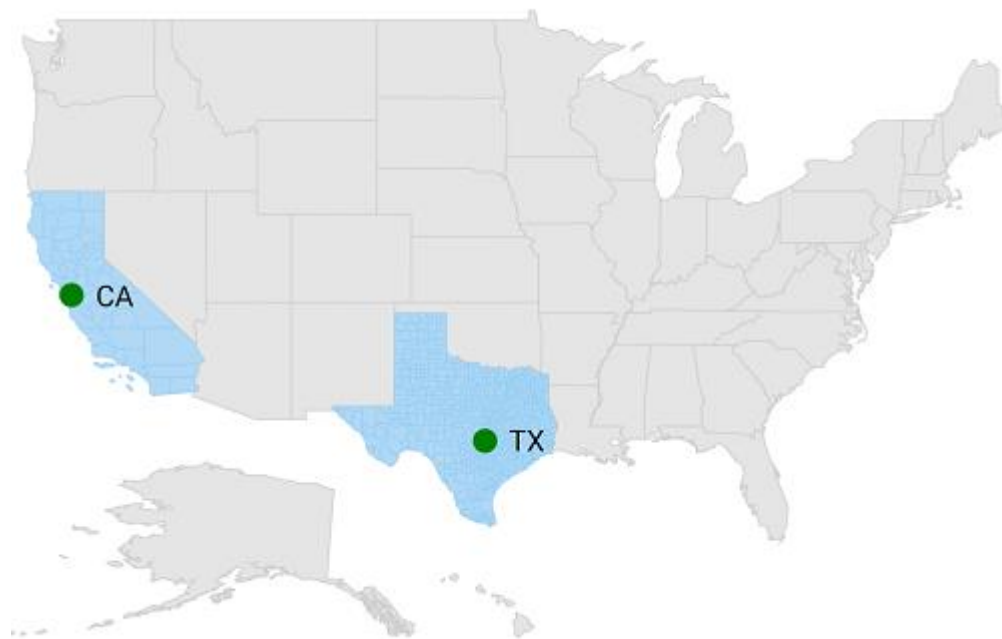
### XML

```
<maps:SfMaps x:Name="sfmap">
  <maps:SfMaps.Layers>
    <maps:ShapeFileLayer Uri="usa_state.shp">
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting ShapeStroke="#D0D0D0" ShapeStrokeThickness="2"
          ShapeFill="#E5E5E5" />
      </maps:ShapeFileLayer.ShapeSettings>
      <maps:ShapeFileLayer.Sublayers>
        <maps:ShapeFileLayer Uri="Texas.shp">
          <maps:ShapeFileLayer.ShapeSettings>
            <maps:ShapeSetting ShapeFill="#B1D8F5" ShapeStroke="#8DCCF4"
              ShapeStrokeThickness="1"/>
          </maps:ShapeFileLayer.ShapeSettings>
          <maps:ShapeFileLayer.Markers>
            <maps:MapMarker Label="TX" Latitude="30.267153" Longitude="-97.7430608"/>
          </maps:ShapeFileLayer.Markers>
        </maps:ShapeFileLayer>
        <maps:ShapeFileLayer Uri="California.shp">
          <maps:ShapeFileLayer.ShapeSettings>
            <maps:ShapeSetting ShapeFill="#B1D8F5" ShapeStroke="#8DCCF4"
              ShapeStrokeThickness="1"/>
          </maps:ShapeFileLayer.ShapeSettings>
          <maps:ShapeFileLayer.Markers>
            <maps:MapMarker Label="CA" Latitude="37.3382082" Longitude="-121.8863286"/>
          </maps:ShapeFileLayer.Markers>
        </maps:ShapeFileLayer>
      </maps:ShapeFileLayer.Sublayers>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

```
</maps:ShapeFileLayer>  
</maps:ShapeFileLayer.Sublayers>  
</maps:ShapeFileLayer>  
</maps:SfMaps.Layers>  
</maps:SfMaps>
```

## C#

```
SfMaps map = new SfMaps();  
ShapeFileLayer layer = new ShapeFileLayer();  
layer.Uri = "usa_state.shp";  
ShapeSetting shapeSetting = new ShapeSetting();  
shapeSetting.ShapeStroke = Color.FromHex("#D0D0D0");  
shapeSetting.ShapeStrokeThickness = 2;  
shapeSetting.ShapeFill = Color.FromHex("#E5E5E5");  
layer.ShapeSettings = shapeSetting;  
ShapeFileLayer subShapeLayer = new ShapeFileLayer();  
subShapeLayer.Uri = "Texas.shp";  
ShapeSetting shapeSetting1 = new ShapeSetting();  
shapeSetting1.ShapeFill = Color.FromHex("#B1D8F5");  
shapeSetting1.ShapeStrokeThickness = 1;  
shapeSetting1.ShapeStroke = Color.FromHex("#8DCCF4");  
subShapeLayer.ShapeSettings = shapeSetting1;  
MapMarker mapMarker = new MapMarker();  
mapMarker.Label = "TX";  
mapMarker.Latitude = "30.267153";  
mapMarker.Longitude = "-97.7430608";  
subShapeLayer.Markers.Add(mapMarker);  
layer.Sublayers.Add(subShapeLayer);  
ShapeFileLayer subShapeLayer1 = new ShapeFileLayer();  
subShapeLayer1.Uri = "California.shp";  
ShapeSetting shapeSetting2 = new ShapeSetting();  
shapeSetting2.ShapeFill = Color.FromHex("#B1D8F5");  
shapeSetting2.ShapeStrokeThickness = 1;  
shapeSetting2.ShapeStroke = Color.FromHex("#8DCCF4");  
subShapeLayer1.ShapeSettings = shapeSetting2;  
MapMarker mapMarker1 = new MapMarker();  
mapMarker1.Label = "CA";  
mapMarker1.Latitude = "37.3382082";  
mapMarker1.Longitude = "-121.8863286";  
subShapeLayer1.Markers.Add(mapMarker1);  
layer.Sublayers.Add(subShapeLayer1);  
map.Layers.Add(layer);  
this.Content = map;
```



### Adding sublayers in ImageryLayer

You can add multiple shape files in the `ImageryLayer` using the `Sublayers` property.

#### XML

```
<maps:SfMaps x:Name="sfmap">
  <maps:SfMaps.Layers>
    <maps:ImageryLayer>
      <maps:ImageryLayer.Sublayers>
        <maps:ShapeFileLayer Uri="africa.shp">
          <maps:ShapeFileLayer.ShapeSettings>
            <maps:ShapeSetting ShapeFill="#FD8C48" ShapeStrokeThickness="1"/>
          </maps:ShapeFileLayer.ShapeSettings>
        </maps:ShapeFileLayer>
      </maps:ImageryLayer.Sublayers>
    </maps:ImageryLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

#### C#

```
SfMaps map = new SfMaps();
ImageryLayer imageryLayer = new ImageryLayer();
ShapeFileLayer subShapeLayer = new ShapeFileLayer();
subShapeLayer.Uri = "africa.shp";
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.ShapeFill = Color.FromHex("#FD8C48");
shapeSetting.ShapeStrokeThickness = 1;
subShapeLayer.ShapeSettings = shapeSetting;
imageryLayer.Sublayers.Add(subShapeLayer);
map.Layers.Add(imageryLayer);
```

```
Content = map;
```



#### *Customizing sublayer*

Sublayer is a type of shapefile layer. You can add all the elements such as markers, bubbles, color mapping, and legends to sublayer. Please refer to the following links to add the sublayer properties.

- [Adding markers](#)
- [Color mapping](#)
- [Adding legend](#)
- [Adding bubbles](#)
- [Adding data labels](#)

#### User interaction

Options such as zooming, panning, and selection enable effective interaction on map elements.

##### Selection

Each shape in a map can be selected or deselected when interacting with shapes. Map shapes can be selected using the following ways:

- Single selection
- Multiple selection

The selected map shapes are differentiated by their fill. The [SelectedShapeColor](#) property of [ShapeSettings](#) gets or sets the selected shape color. The [SelectedShapeStroke](#) property is used to customize the selected shape border.



### Single selection

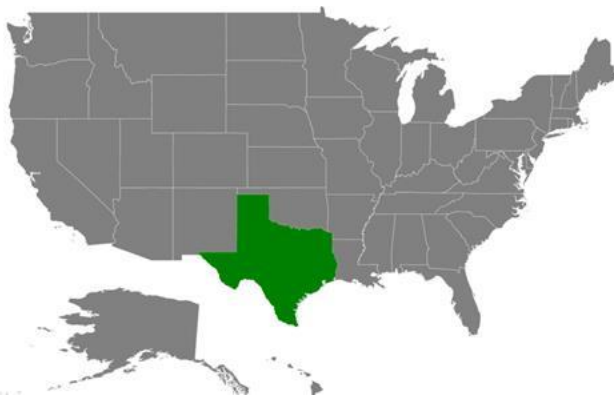
Single selection allows you select only one shape at a time. You can select a shape by tapping it. By default, the single selection is enabled when the [EnableSelection](#) property is set to true. You can also enable the single selection by setting the [SelectionMode](#) property of ShapeFileLayer to “Single”. When selecting or tapping the rest of the area, the selected shape will be deselected.

#### XML

```
<maps:SfMaps x:Name="sfmap" BackgroundColor="White" >
  <maps:SfMaps.Layers >
    <maps:ShapeFileLayer Uri="usa_state.shp" EnableSelection="True"
      SelectionMode="Single" >
      <maps:ShapeFileLayer.ShapeSettings>
        <maps:ShapeSetting SelectedShapeColor="Green" SelectedShapeStroke="Black"
          SelectedShapeStrokeThickness="1" />
      </maps:ShapeFileLayer.ShapeSettings>
    </maps:ShapeFileLayer>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

#### C#

```
SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.EnableSelection = true;
layer.SelectionMode = SelectionMode.Single;
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.SelectedShapeColor = Color.Green;
shapeSetting.SelectedShapeStroke = Color.Black;
shapeSetting.SelectedShapeStrokeThickness = 1;
layer.ShapeSettings = shapeSetting;
map.Layers.Add(layer);
this.Content = map;
```



### Multiple selection

Multiple selection allows you select multiple shapes at a time. You can select many shapes by tapping them. To enable this feature, set the [SelectionMode](#) property to “Multiple” along with the [EnableSelection](#) property.

---

**Information:** Shapes cannot be selected when the [EnableSelection](#) property is set to false.

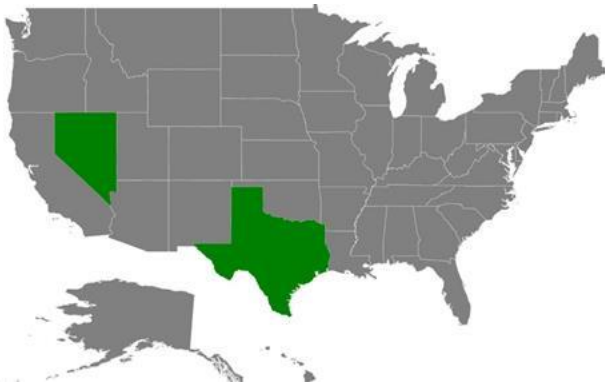
---

**XML**

```
<maps:SfMaps x:Name="sfmap" BackgroundColor="White" >
<maps:SfMaps.Layers >
<maps:ShapeFileLayer Uri="usa_state.shp" EnableSelection="True"
SelectionMode="Multiple" >
<maps:ShapeFileLayer.ShapeSettings>
<maps:ShapeSetting SelectedShapeColor="Green" SelectedShapeStroke="Black"
SelectedShapeStrokeThickness="1" />
</maps:ShapeFileLayer.ShapeSettings>
</maps:ShapeFileLayer>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

**C#**

```
SfMaps map = new SfMaps();
map.BackgroundColor = Color.White;
ShapeFileLayer layer = new ShapeFileLayer();
layer.Uri = "usa_state.shp";
layer.EnableSelection = true;
layer.SelectionMode = SelectionMode.Multiple;
ShapeSetting shapeSetting = new ShapeSetting();
shapeSetting.SelectedShapeColor = Color.Green;
shapeSetting.SelectedShapeStroke = Color.Black;
shapeSetting.SelectedShapeStrokeThickness = 1;
layer.ShapeSettings = shapeSetting;
map.Layers.Add(layer);
this.Content = map;
```

*Selected items*

The **SelectedItems** property allows you select the shapes without tapping or touching them.

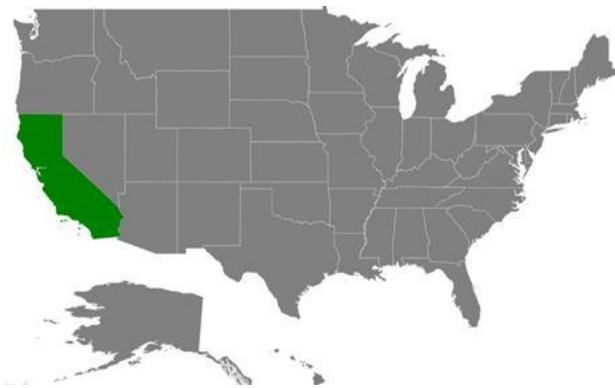
To select a shape and deselect it from the same collection without tapping or touching, just add the shape that is to be selected to the selected items collection.

The following code sample demonstrates how to select and deselect a shape.

**C#**

```
var model = GetDataSource();
ShapeFileLayer layer=new ShapeFileLayer();
layer.ItemsSource=model;
```

```
SelectedItemButton.Clicked += (sender, e) =>
{
    layer.SelectedItems.Add(model[4]);
};
RemoveItemButton.Clicked += (sender, e) =>
{
    layer.SelectedItems.Remove(model[4]);
};
```



### Zooming

The zooming feature enables you to zoom in and zoom out the maps to show the in-depth information. The following properties are used to zoom in and zoom out maps:

[EnableZooming](#): Controls whether to perform zooming or not.

[MinZoom](#): Sets the minimum level of zooming.

[MaxZoom](#): Sets the maximum level of zooming.

[ZoomLevel](#): Sets zooming level to shapes.

### XML

```
<maps:SfMaps EnableZooming="True" MinZoom="1"
MaxZoom="10" ZoomLevel="2"/>
```

### C#

```
SfMaps map = new SfMaps();
map.EnableZooming = true;
map.MinZoom = 1;
map.MaxZoom = 10;
map.ZoomLevel = 2;
```

### Panning

Panning feature allows you to move the visible area of the maps when it is zoomed in. To enable panning, set the [EnablePanning](#) property to true.

### XML

```
<maps:SfMaps x:Name="sfmap" EnablePanning="True" EnableZooming="True"
MinZoom="1"
MaxZoom="10" BackgroundColor="White" />
```

## C#

```
SfMaps map = new SfMaps();  
map.EnablePanning = true;  
map.EnableZooming = true;  
map.MinZoom = 1;  
map.MaxZoom = 10;
```

## Map Providers

The maps control supports map providers such as OpenStreetMap and Bing Maps that can be added to an imagery layer in maps.

### OpenStreetMap

The OpenStreetMap (OSM) is a world map; it was built by a community of mappers. It is free to use under an open license. This allows you view geographical data in a collaborative way from anywhere on the earth. The OSM provides small tile images based on your requests and combines them into a single image to display the map area in the maps control.

#### *Adding OSM in maps*

The maps control uses **imagery layer** to display the tile images from the OSM service. To use **OSM**, add an imagery layer in maps' layers collection.

## XML

```
<maps:SfMaps.Layers>  
<maps:ImageryLayer/>  
</maps:SfMaps.Layers>
```

## C#

```
SfMaps maps = new SfMaps();  
ImageryLayer layer = new ImageryLayer();  
maps.Layers.Add(layer);
```



**Note:** Both the [ShapeFileLayer](#) and **ImageryLayer** have been derived commonly from MapsLayer.

## Bing Maps

The Bing Maps is a world map owned by Microsoft. As OSM, Bing Maps also provides map tile images based on your requests and combines them into a single image to display the map area. To use Bing maps, set the `LayerType` property of `ImageryLayer` to "Bing". Then, set the Bing Maps key, which is obtained from [Bing Maps Key](#).

### XML

```
<maps:SfMaps.Layers>
  <maps:ImageryLayer LayerType="Bing" BingMapKey="Your bing map key"/>
</maps:SfMaps.Layers>
```

### C#

```
SfMaps maps = new SfMaps();
ImageryLayer layer = new ImageryLayer();
layer.LayerType = LayerType.Bing;
layer.BingMapKey = "Your bing map key";
maps.Layers.Add(layer);
```



**Note:** The `LayerType` property of `ImageryLayer` provides support to `OSM` and `Bing Maps`. The default value of the `LayerType` property is `OSM`.

### Set different bing map style

The `ImageryLayer` provides support to the following types of Bing Maps:

- `Road`
- `Aerial`
- `AerialWithLabels`

The desired style for the Bing Maps can be set using the `BingMapStyle` property of `ImageryLayer`. The default value of `BingMapStyle` is "Road".

### Road

The Road view displays the default map view of roads, buildings, and geography. The default value of the `BingMapStyle` property of imagery layer is "Road".

### Aerial

The Aerial view displays the satellite images to highlight the roads and major landmarks for easy identification. The aerial view can be applied to maps by setting the `BingMapStyle` to "Aerial".

#### XML

```
<maps:SfMaps.Layers>
  <maps:ImageryLayer LayerType="Bing"
    BingMapStyle="Aerial"
    BingMapKey=" Your bing map key "/>
</maps:SfMaps.Layers>
```

#### C#

```
SfMaps maps = new SfMaps();
ImageryLayer layer = new ImageryLayer();
layer.LayerType = LayerType.Bing;
layer.BingMapStyle = BingMapStyle.Aerial;
layer.BingMapKey = "Your bing map key ";
maps.Layers.Add(layer);
```



### AerialWithLabel

The AerialWithLabel view displays the Aerial map with labels for continent, country, ocean, etc. This view can be applied to maps by setting the `BingMapStyle` to "AerialWithLabel".

#### XML

```
<maps:SfMaps.Layers>
  <maps:ImageryLayer LayerType="Bing"
    BingMapStyle="AerialWithLabels"
    BingMapKey=" Your bing map key "/>
</maps:SfMaps.Layers>
```

#### C#

```
SfMaps maps = new SfMaps();  
ImageryLayer layer = new ImageryLayer();  
layer.LayerType = LayerType.Bing;  
layer.BingMapStyle = BingMapStyle.AerialWithLabels;  
layer.BingMapKey = "Your bing map key ";  
maps.Layers.Add(layer);
```



### Zooming and panning

The maps control provides interactive zooming and panning supports to OSM and Bing Maps.

Zooming helps you get a closer look of an area on maps for in-depth analysis. Panning helps you move a map around to focus the targeted area. You can perform zooming and panning with the pinching gesture in a map area.



### Reset zooming

The ImageryLayer provides support to reset the maps to the default view when you double tap the imagery layer by setting the `ResetOnDoubleTap` property to true. The default value of this property is true. This behavior can be restricted by setting the `ResetOnDoubleTap` property to false.

#### XML

```
<maps:SfMaps>
  <maps:SfMaps.Layers>
    <maps:ImageryLayer ResetOnDoubleTap="True"/>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

#### C#

```
SfMaps maps = new SfMaps();
ImageryLayer layer = new ImageryLayer();
layer.ResetOnDoubleTap = true;
maps.Layers.Add(layer);
```

### Set Geo coordinates points(center position)

The `GeoCoordinates` property allows you view the desired area at the center on loading. By default, the `GeoCoordinates` value is (0,0). So, the latitude value "0" and longitude value "0" are shown at the center.

#### XML

```
<maps:SfMaps ZoomLevel="2">
  <maps:SfMaps.Layers>
    <maps:ImageryLayer GeoCoordinates="69.07,-37.08"/>
  </maps:SfMaps.Layers>
</maps:SfMaps>
```

#### C#

```
SfMaps maps = new SfMaps();
maps.ZoomLevel = 2;
ImageryLayer layer = new ImageryLayer();
layer.GeoCoordinates = new Point(69.07, -37.08);
maps.Layers.Add(layer);
```





### XML

```
<maps:SfMaps ZoomLevel="2">  
  <maps:SfMaps.Layers>  
    <maps:ImageryLayer GeoCoordinates="0,0"/>  
  </maps:SfMaps.Layers>  
</maps:SfMaps>
```

### C#

```
SfMaps maps = new SfMaps();  
maps.ZoomLevel = 2;  
ImageryLayer layer = new ImageryLayer();  
layer.GeoCoordinates = new Point(0,0);  
maps.Layers.Add(layer);
```



Set markers in imagery layer

As [ShapeFileLayer](#), markers also can be added to imagery layer. Markers can be customized using the `MarkerSettings` property in imagery layer.

The detailed explanation of marker and its customization have been provided in Markers section.

### XML

```
<maps:ImageryLayer >
  <maps:ImageryLayer.MarkerSettings>
    <maps:MapMarkerSetting IconColor="Red"
      IconSize="13" MarkerIcon="Diamond"/>
  </maps:ImageryLayer.MarkerSettings>
  <maps:ImageryLayer.Markers>
    <maps:MapMarker Label="United States" Latitude="40"
      Longitude= "-101"/>
    <maps:MapMarker Label="Brazil" Latitude="-15.7833"
      Longitude= "-52" />
    <maps:MapMarker Label="Congo" Latitude="-1.6"
      Longitude= "24.4" />
    <maps:MapMarker Label="Kazakhstan" Latitude="49.9"
      Longitude= "72.23" />
    <maps:MapMarker Label="Australia" Latitude="-20.54"
      Longitude= "134.10" />
  </maps:ImageryLayer.Markers>
</maps:ImageryLayer>
```

### C#

```
ImageryLayer layer = new ImageryLayer();
layer.MarkerSettings = new MapMarkerSetting();
layer.MarkerSettings.IconColor = Color.Red;
layer.MarkerSettings.MarkerIcon = MapMarkerIcon.Diamond;
layer.MarkerSettings.IconSize = 13;
MapMarker marker1 = new MapMarker();
marker1.Label = "United States";
marker1.Latitude = "40";
marker1.Longitude = "-101";
layer.Markers.Add(marker1);
MapMarker marker2 = new MapMarker();
marker2.Label = "Brazil";
marker2.Latitude = "-15.7833";
marker2.Longitude = "-52";
layer.Markers.Add(marker2);
MapMarker marker3 = new MapMarker();
marker3.Label = "Congo";
marker3.Latitude = "-1.6";
marker3.Longitude = "24.4";
layer.Markers.Add(marker3);
MapMarker marker4 = new MapMarker();
marker4.Label = "Kazakhstan";
marker4.Latitude = "49.9";
marker4.Longitude = "72.23";
layer.Markers.Add(marker4);
MapMarker marker5 = new MapMarker();
marker5.Label = "Australia";
```

```
marker5.Latitude = "-20.54";
marker5.Longitude = "134.10";
layer.Markers.Add(marker5);
```



### Custom map providers

You can show the other map providers maps using imagery layer. First, initialize the map extension class, override the GetUri method of imagery layer extension class, and then pass the Map providers tile image Uri link like Google uri with corresponding x, y, and zoom level. Finally, add the imagery layer extension class to layers collection of native map control by overriding the OnElementChanged method of each platform's (Xamarin.Android, Xamarin.iOS, and UWP) custom map renderer. For more information to add custom map provider, refer to this [KB article](#).

### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        MapExt maps = new MapExt();
        ImageryLayer layer = new ImageryLayer();
        maps.Layers.Add(layer);
        this.Content = maps;
    }
}

public class MapExt : SfMaps
{
}

public class CustomMapRenderer : SfMapsRenderer
{
    protected override void OnElementChanged(ElementChangedEventArgs<SfMaps> e)
    {
        base.OnElementChanged(e);
        if (Control != null)
            AddLayer();
    }

    void AddLayer()
    {
        ImageryLayerExt layer = new ImageryLayerExt();
        (Control as NativeMap.SfMaps).Layers.Clear();
        (Control as NativeMap.SfMaps).Layers.Add(layer as NativeMap.ImageryLayer);
    }
}
```

```
}  
}  
public class ImageryLayerExt : NativeMap.ImageryLayer  
{  
    protected override string GetUri(int X, int Y, int Scale)  
    {  
        var link = "http://mt1.google.com/vt/lyrs=y&x=" + X.ToString()  
        + "&y=" + Y.ToString() + "&z=" + Scale.ToString();  
        return link;  
    }  
}
```



You can download the demo sample in this [link](#).

Cache tile images in application memory

The [CanCacheTiles](#) property used to decide whether the tile images should be cached in application memory or not.

### C#

```
ImageryLayer imageryLayer = new ImageryLayer();  
imageryLayer.CanCacheTiles = true;
```

Clear cached tile images from application memory

The [DeleteTilesFromCache](#) method used to clear the cached tile images from application cache memory.

### XML

```
<maps:SfMaps>  
  <maps:SfMaps.Layers>  
    <maps:ImageryLayer x:Name="imageryLayer" />  
  </maps:SfMaps.Layers>  
</maps:SfMaps>
```

```
</maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
imageryLayer.DeleteTilesFromCache();
```

### Events

The [ZoomLevelChanging](#) event triggers when zoom level changed. Following arguments can be get from the ZoomLevelChanging event .

- [Cancel](#) : Used to cancel the zooming.
- [PreviousLevel](#) : Returns the previous level after the zooming.
- [CurrentLevel](#) : Returns the current level to be zoomed.

### XML

```
<maps:SfMaps>
<maps:SfMaps.Layers>
<maps:ImageryLayer ZoomLevelChanging="Layer_ZoomLevelChanging" />
</maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
private void Layer_ZoomLevelChanging(object sender,
ZoomLevelChangingEventArgs e)
{
    if(e.PreviousLevel == 10) // Returns the previous zoom level
    {
        e.Cancel = true; // Cancels the zooming event
        var CurrentLevel = e.CurrentLevel; // Returns the current zoomed level
    }
}
```

The [GeoCoordinateChanged](#) event is triggered while zooming and panning the maps.

The following arguments can be gotten from the `ImageryLayer_GeoCoordinateChanged` event:

- [Center](#): Returns the center Geo coordinate point of the visual tiles while zooming and panning.
- [TopLeft](#): Returns the top-left Geo coordinate point of the visual tiles while zooming and panning.
- [TopRight](#): Returns the top-right Geo coordinate point of the visual tiles while zooming and panning.
- [BottomLeft](#): Returns the bottom-left Geo coordinate point of the visual tiles while zooming and panning.
- [BottomRight](#): Returns the bottom-right Geo coordinate point of the visual tiles while zooming and panning.

### XML

```
<maps:SfMaps >
```

```
<maps:SfMaps.Layers >
<maps:ImageryLayer
GeoCoordinateChanged="ImageryLayer_GeoCoordinateChanged"/>
</maps:SfMaps.Layers>
</maps:SfMaps>
```

### C#

```
private void ImageryLayer_GeoCoordinateChanged(object sender,
GeoCoordinateChangedEventArgs e)
{
var topLeft = e.TopLeft;
var topRight = e.TopRight;
var bottomLeft = e.BottomLeft;
var bottomRight = e.BottomRight;
var center = e.Center;
}
```

### How to

#### Transform latitude and longitude value to pixel value and vice-versa

SfMaps offers two utility methods to transform the pixel values to longitude and latitude values and vice-versa. This method is used for both ShapeFileLayer and ImageryLayer.

- **GeopointToViewPoint(double latitude, double longitude)** - Converts the latitude and longitude values to screen point. Here, pass the parameters as latitude and longitude values, from that values we can get screen points x and y.
- **GetLatLonFromPoint(Point point)** - Converts the screen point to longitude and latitude values. Here, pass the parameters as screen points x and y, from that points we can get longitude(Point.X) and latitude(Point.Y) values.

### C#

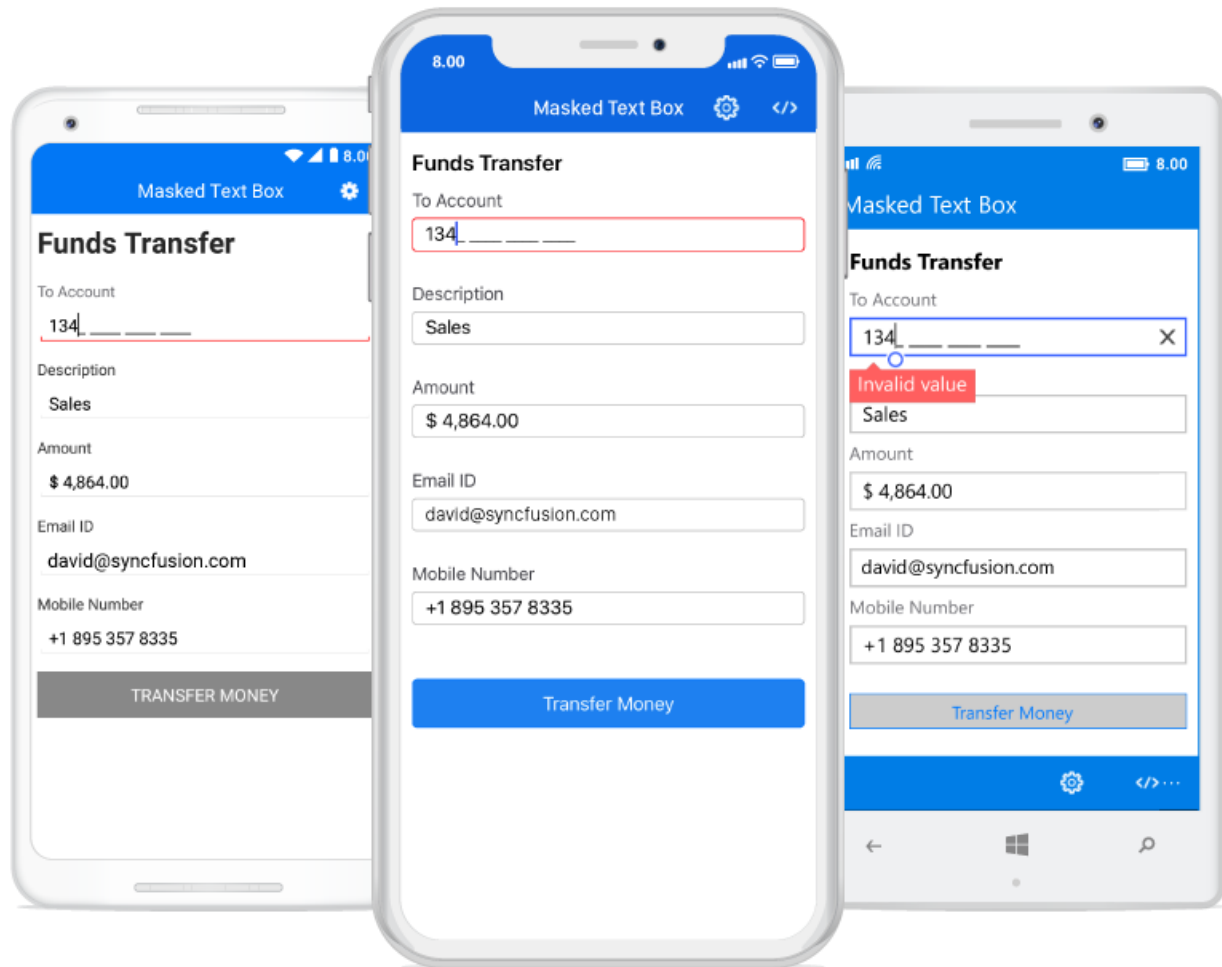
```
Point pixelPoint = layer.GeopointToViewPoint(21.00, 78.00);
Point longitudeLatitude = layer.GetLatLonFromPoint(pixelPoint);
marker.Latitude = longitudeLatitude.Y.ToString();
marker.Longitude = longitudeLatitude.X.ToString();
```



## SfMaskedEdit

### Overview

The Masked text box is an advanced version of the Entry control that restricts your input to certain types of characters, text, and numbers using a mask pattern. This control is used to create a template for providing information such as telephone numbers, IP addresses, product IDs, and so on.



## Key features

- The input can be masked with a fixed or variable length by setting the **MaskType** to **Text** or **Regex**, respectively.
- Custom prompt characters can be set.
- Special symbols such as currency symbol, date separator, decimal separator, etc., can be localized based on the culture.
- Input validation can be done either during each key press or when the control loses its focus.
- Values can be accessed with or without literal and prompt characters.
- Clipboard operations can be used with or without literal and prompt characters.
- Watermark text can be used to display an instruction or important information.
- The UI of the masked text box is completely customizable.

## Getting Started

This section explains you the steps required to configure a **SfMaskedEdit** control in a real-time scenario and provides a walk-through on some of the customization features available in **SfMaskedEdit** control.

### Adding SfMaskedEdit reference

You can add SfMaskedEdit reference using one of the following methods:



### Method 1: Adding SfMaskedEdit reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.syncfusion.com/nuget-packages). To add SfMaskedEdit to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfMaskedEdit](https://www.syncfusion.com/nuget-packages), and then install it.

!{Adding SfMaskedEdit reference from NuGet}(SfMaskedEditImages/Adding SfMaskedEdit reference.png)

---

**Note:** Install the same version of SfMaskedEdit NuGet in all the projects.

---

### Method 2: Adding SfMaskedEdit reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfMaskedEdit control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfMaskedEdit assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfMaskedEdit.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfMaskedEdit.XForms.Android.dll Syncfusion.SfMaskedEdit.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfMaskedEdit.XForms.iOS.dll Syncfusion.SfMaskedEdit.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfMaskedEdit.XForms.UWP.dll Syncfusion.SfMaskedEdit.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

**Note:** After adding the reference, an additional step is required for iOS and UWP projects. If you are adding the references from toolbox, this step is not needed.

#### *Additional step for iOS*

To launch SfMaskedEdit in iOS, call the `SfMaskedEditRenderer.Init()` in `FinishedLaunching` overridden method of `AppDelegate` class in iOS Project, as demonstrated in the following code example.

#### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    LoadApplication(new App());
    SfMaskedEditRenderer.Init();
    return base.FinishedLaunching(app, options);
}
```

#### *Additional step for UWP*

This step is required only if the application is deployed in Release mode with .NET native tool chain enabled and it is for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the SfMaskedEdit assembly at `OnLaunched` overridden method of the `App` class in UWP project is the suggested work around, as demonstrated in the following code example.

#### **C#**

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    .....
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfMaskedEditRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    .....
}
```

#### Create a Simple SfMaskedEdit

The SfMaskedEdit control is configured entirely in C# code or by using XAML markup. The following steps explain how to create a SfMaskedEdit and configure its elements:

#### *Add namespace for referred assemblies*

#### **XML**

```
xmlns:syncfusion="clr-namespace:Syncfusion.XForms.MaskedEdit;assembly=Syncfusion.SfMaskedEdit.XForms"
```

**C#**

```
using Syncfusion.XForms.MaskedEdit;
```

Refer SfMaskedEdit control with declared suffix name for Namespace

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:syncmaskededit="clr-
namespace:Syncfusion.XForms.MaskedEdit;assembly=Syncfusion.SfMaskedEdit.XFor
ms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<StackLayout Margin="0,100,0,0">
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" />
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

**C#**

```
using Syncfusion.XForms.MaskedEdit;
using Xamarin.Forms;
namespace GettingStarted
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout();
stackLayout.Margin = new Thickness(0, 100, 0, 0);
SfMaskedEdit maskedEdit = new SfMaskedEdit();
stackLayout.Children.Add(maskedEdit);
this.Content = stackLayout;
}
}
}
```

## Masking the input

To mask the input, set the **Mask** property as follows:

**XML**

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" Mask="00/00/0000"/>
```

**C#**

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.Mask = "00/00/0000";
```

This mask expression allows only numeric inputs in the places of 0.

Refer to this [link](#) to know more about the Mask characters and Mask Types available in SfMaskedEdit control.

Run the project and check if you get the following output to make sure that you have configured your project properly to add SfMaskedEdit.



You can find the complete getting started sample from this [link](#).

## Basic Features

### Setting Value

The SfMaskedEdit control displays the value that can be set using the **Value** property:

#### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" Mask="00/00/0000"
Value="14/11/2014"/>
```

#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.Mask = "00/00/0000";
maskedEdit.Value =@"14/11/2014";
```



### Setting Prompt Character

Displays prompt character for the absence of your input in Mask and its default value is '\_'. You can set the custom prompt character using **PromptChar** property.

#### XML

```
< syncmaskededit:SfMaskedEdit x:Name="maskedEdit" Mask="000000"
MaskType="Text" PromptChar="*" />
```

#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.Mask = "00/00/0000";
maskedEdit.PromptChar = '*';
```



### Setting Watermark

The watermark will prompt you with instructions or important information when it is not on focus and any valid characters are not entered. The `Watermark` property of `SfMaskedEdit` is used to set the watermark text for the control.

The following properties are used to customize its appearance:

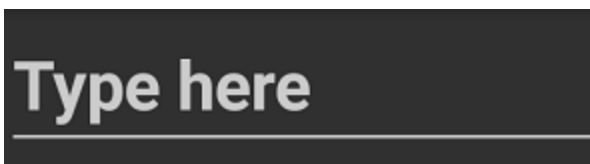
- `WatermarkColor`: Sets text color for the watermark.
- `WatermarkFontFamily`: Represents the font to be used in the watermark.
- `WatermarkFontAttributes`: Sets font attributes(bold/italic/none) for the watermark.
- `WatermarkFontSize`: Sets font size for the watermark.

### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" Mask="00/00/0000"
Watermark="Type here" WatermarkColor="LightGray" WatermarkFontFamily="Arial"
WatermarkFontAttributes="Bold" WatermarkFontSize="20"/>
```

### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.Mask = "00/00/0000";
maskedEdit.Watermark = "Type here";
maskedEdit.WatermarkColor = Color.LightGray;
maskedEdit.WatermarkFontFamily = "Arial";
maskedEdit.WatermarkFontAttributes = FontAttributes.Bold;
maskedEdit.WatermarkFontSize = 20;
```



You can find the complete basic features sample from this [link](#).

### MaskType

Each `MaskType` has different set of mask characters that are combined to form a mask expression. Based on the complexity and usage, mask types are classified as:

- Text
- RegEx

#### Text

The expressions that are generated with letters, digits, and special characters come under this group. This is mainly used for fixed length inputs. For example: phone number, zip code, and so on.

*Text Mask characters*

Characters	Description
A	Alphanumeric, required.
a	Alphanumeric, optional.
L	Letter, required. Restricts input to the ASCII letters a-z and A-Z.
l	Letter, optional. Restricts input to the ASCII letters a-z and A-Z.
0	Digit, required. This character accepts any single digit between 0 and 9
9	Digit or space, optional.
#	Digit or space, optional. Plus (+) and minus (-) signs are allowed.
C	Character, optional.
\	Escapes a mask character, turning it into a literal. "\" is the escape sequence for a backslash.
Any other characters	Considered as literals. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted.

**XML**

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="+1 (000) 000000"/>
```

**C#**

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "+1 (000) 000000";
```

This mask expression allows only numeric inputs in the places of 0.

*Regex*

The expressions that are generated with regular expressions come under this group, preferable for variable length inputs and input in range. For example: hexadecimal values [0-9A-C].

The regular expressions provide significant advantages when creating masks as compared with other mask modes.

**Advantages**

- Allows you to enter the value of indeterminate length.
- Restricts with specific pattern. Example email, password, and more.
- Restricts you to enter specific range at specific position.

*Regex Mask Characters*

Characters	Description
\w	Accepts any alphabet, number, including the _(Underscore) character.
\d	Accepts any digit.
{n}	Accepts the input for $\hat{\sim}$ number of times.
{n,m}	Accepts the input for $\hat{\sim}$ minimum number of times and $\hat{\sim}$ maximum number of times.
?	Optional input.
+	Accepts the input for one or more times.
*	Accepts the input for zero or more times.
[aeiou]	Accepts any single character included in the specified set of characters.
[0-9a-fA-F]	Accepts any character between[A-F]/[a-f] and numbers between [0-9].

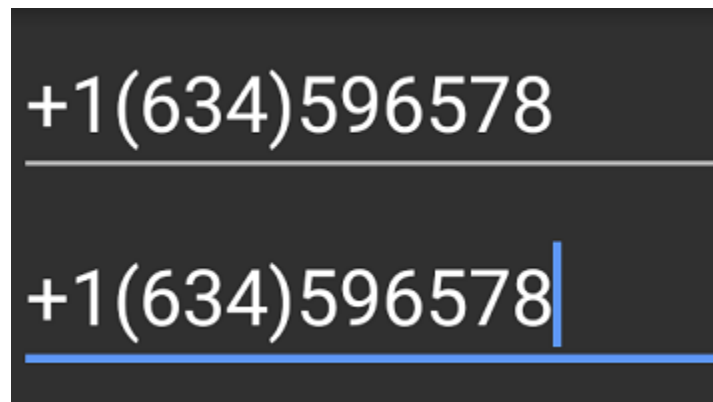
**XML**

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Regex"
Mask="+1(\d{3})\d{5}"/>
```

**C#**

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.RegEx;
maskedEdit.Mask = @"+1(\d{3})\d{5}";
```

This mask expression '\d{3}' and '\d{5}' allows only numeric, where {n} is the count that the input should be accepted.



You can find the complete mask type sample from this [link](#).

## Localization

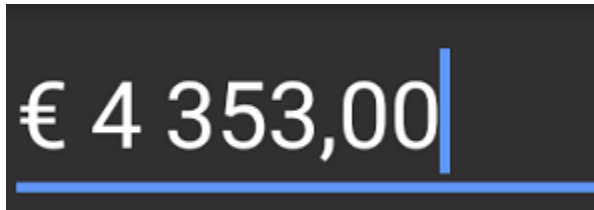
The special symbols such as currency symbol, date separator, decimal separator etc., can be localized to any specific culture using the `Culture` property.

Characters	Description
.	Decimal separator determined by current culture.
,	Group separator determined by current culture.
/	Date separator determined by current culture.
:	Time separator determined by current culture.
\$	Currency symbol determined by current culture.

### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.Mask = "$ 0,000.00";
maskedEdit.Culture = new CultureInfo("fr-FR");
```

Now the '\$' will be localized to '€'; '.' will be localized to ',' and ' ' will be localized to ' '(single white space).



You can find the runnable localization sample from this [link](#).

### Using Mask Characters as Literals

To use mask character as a literal, precede the mask character with a backslash (\\). For example, to display the dollar sign (\$), then set the mask as follows:

### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text" Mask="\$ 0000"/>
```

### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = @"\$ 0000";
```

This will produce a mask that displays a dollar sign (\$) followed by the prompt characters for entering numbers.





This demo can be downloaded from this [link](#).

### Hiding Prompt Characters

When the `HidePromptOnLeave` property is set to true, prompt characters are ignored when control loses focus. Again, the prompt characters are restored when the control is focused.

#### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" HidePromptOnLeave="True" />
```

#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.HidePromptOnLeave = true;
```



This demo can be downloaded from this [link](#).

### Validation

#### Validation Mode

Input validation happens based on the value of the `ValidationMode` property. The enum values of this property are:

- KeyPress
- LostFocus

The default value for validation mode is `LostFocus`.

#### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" ValidationMode="KeyPress" />
```

#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
```

```
maskedEdit.ValidationMode = InputValidationMode.KeyPress;
```

When the ValidationMode is LostFocus, the validation takes place when the control lost its focus. For KeyPress, the validation triggers for each key press.

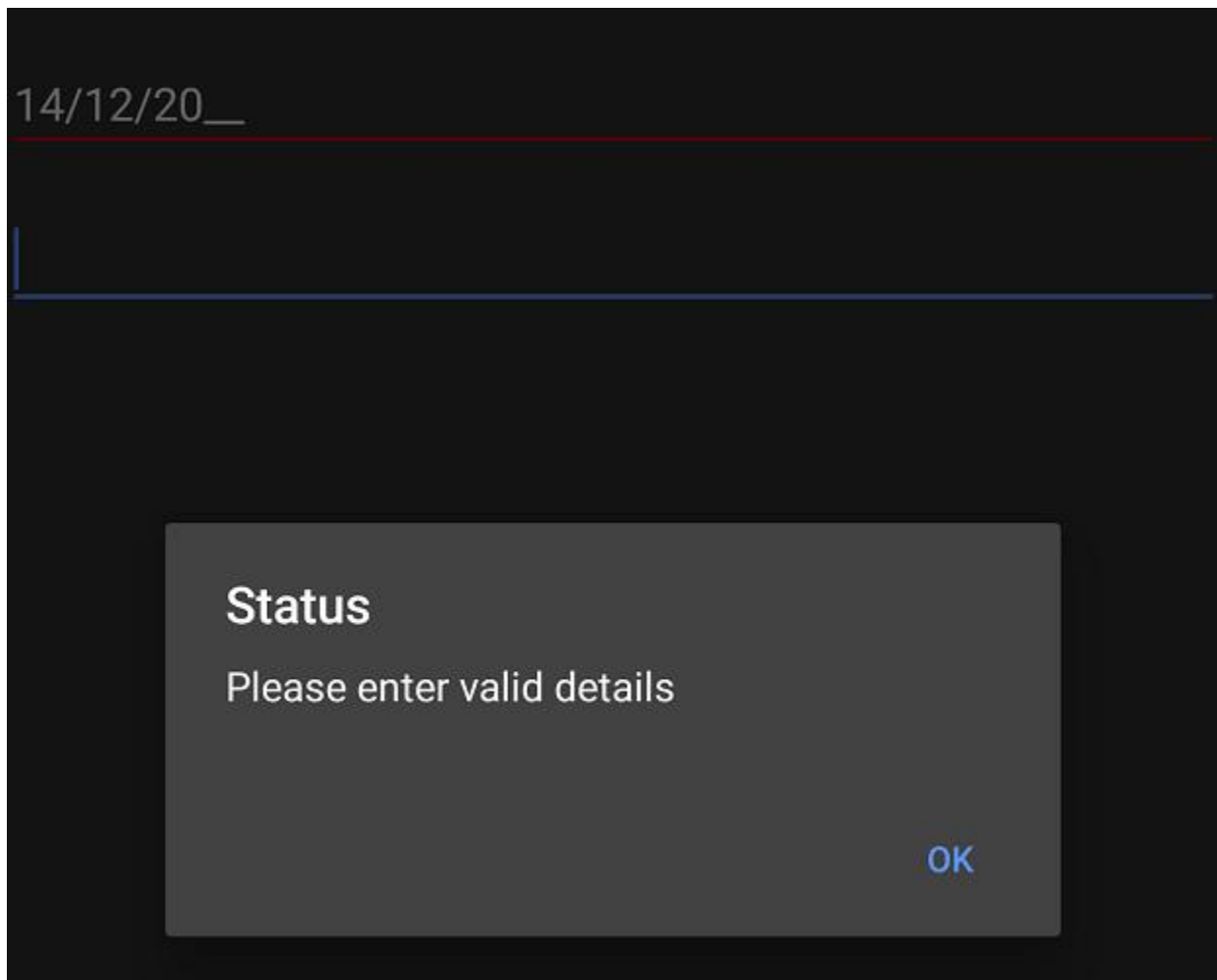
### HasError

This read only property is used to check whether the validation succeeds or not. It returns true once validation succeeds or else returns false. The following code example shows the usage of **HasError** property.

### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.ValidationMode = InputValidationMode.LostFocus;
maskedEdit.ValueChanged += MaskedEdit_ValueChanged;
private void MaskedEdit_ValueChanged(object sender,
Syncfusion.XForms.MaskedEdit.ValueChangedEventArgs e)
{
    SfMaskedEdit maskedEdit = sender as SfMaskedEdit;
    if(maskedEdit.HasError)
    {
        DisplayAlert("Alert", "Please enter valid details", "OK");
    }
}
```

Refer this [link](#) to know more about the **ValueChanged** event of SfMaskedEdit control.



This demo can be downloaded from this [link](#).

## Events

The SfMaskedEdit exposes the following events:

- **ValueChanged**: Occurs when the value of **Value** property is changed.
- **MaskInputRejected**: Occurs when a character is rejected by the input mask.

### ValueChanged event

Occurs when the value of the **Value** property is changed by either entering the valid input character or setting the value to the **Value** property through XAML or C# code. The event arguments are of type **ValueChangedEventArgs** and expose the following property:

- **Value**: The read only property contains the updated value of the **Value** property of SfMaskedEdit.

---

Your valid input character is updated to **Value** property based on the **ValidationMode** property.

Refer to this [link](#) to know more about the **ValidationMode** property of SfMaskedEdit control.

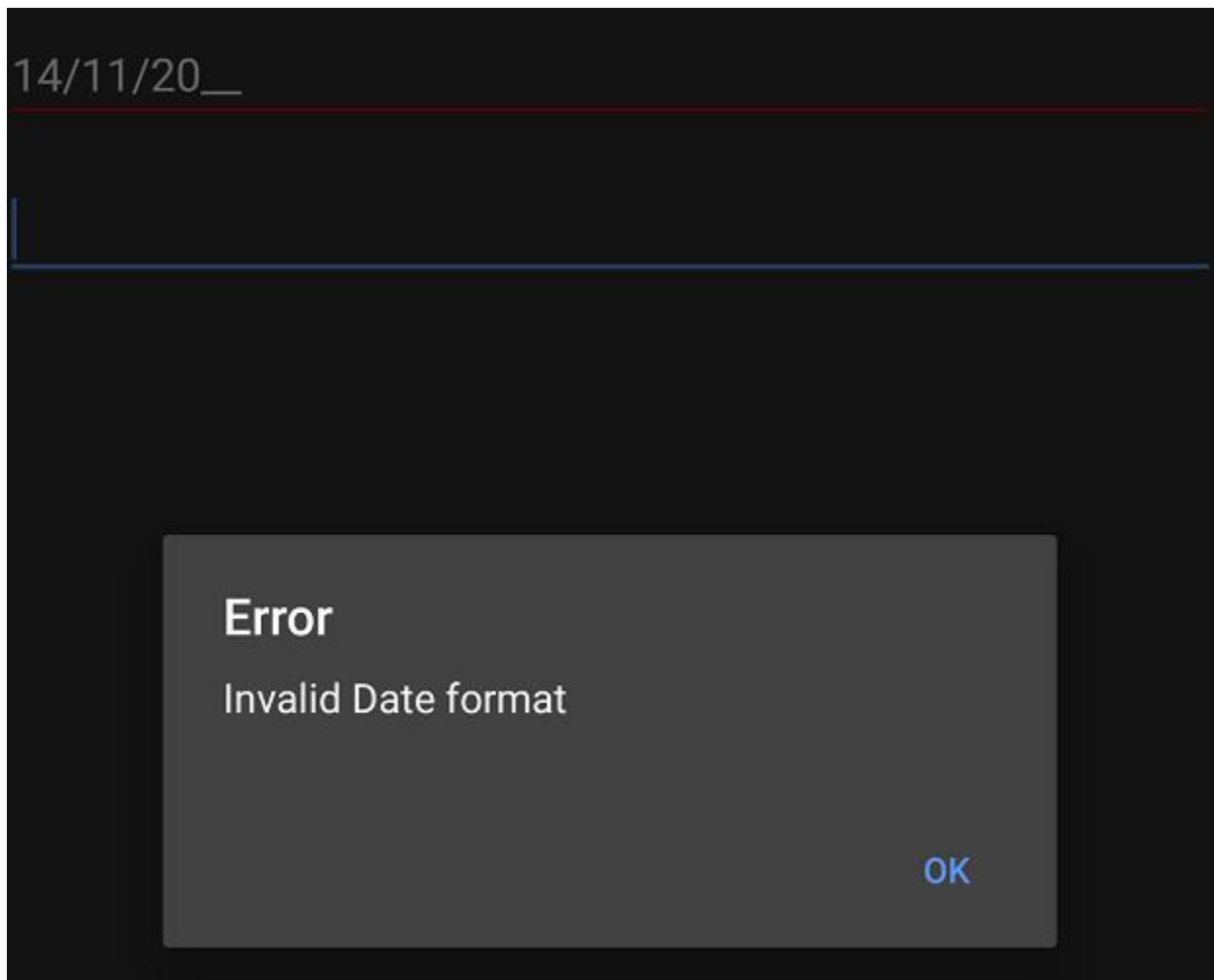
---

## XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" Watermark="dd/MM/YYYY" ValidationMode="LostFocus"
ValueChanged="MaskedEdit_OnValueChanged" />
private void MaskedEdit_OnValueChanged(object sender,
Syncfusion.XForms.MaskedEdit.ValueChangedEventArgs e)
{
    string date= e.Value.ToString();
    if (!string.IsNullOrEmpty(date))
    {
        try
        {
            DateTime datetime =
            DateTime.ParseExact(date,CultureInfo.CurrentCulture.DateTimeFormat.ShortDate
            Pattern, CultureInfo.InvariantCulture);
        }
        catch (Exception exception)
        {
            DisplayAlert("Error", "Invalid Date format", "Ok");
        }
    }
}
```

## C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.Watermark = "dd/MM/YYYY";
maskedEdit.ValidationMode = InputValidationMode.LostFocus;
maskedEdit.ValueChanged += MaskedEdit_OnValueChanged;
private void MaskedEdit_OnValueChanged(object sender,
Syncfusion.XForms.MaskedEdit.ValueChangedEventArgs e)
{
    string date= e.Value.ToString();
    if (!string.IsNullOrEmpty(date))
    {
        try
        {
            DateTime datetime =
            DateTime.ParseExact(date,CultureInfo.CurrentCulture.DateTimeFormat.ShortDate
            Pattern, CultureInfo.InvariantCulture);
        }
        catch (Exception exception)
        {
            DisplayAlert("Error", "Invalid Date format", "Ok");
        }
    }
}
```



### [MaskInputRejected event](#)

Occurs when your input or assigned character does not match the corresponding format element of the input mask. The event arguments are type of `MaskInputRejectedEventArgs` and expose the following properties:

- **Position**: The position in the mask corresponding to the invalid input character.
- **RejectionHint**: The enumerated value that describes why the input character was rejected.

`MaskInputRejected` is raised in the following situations:

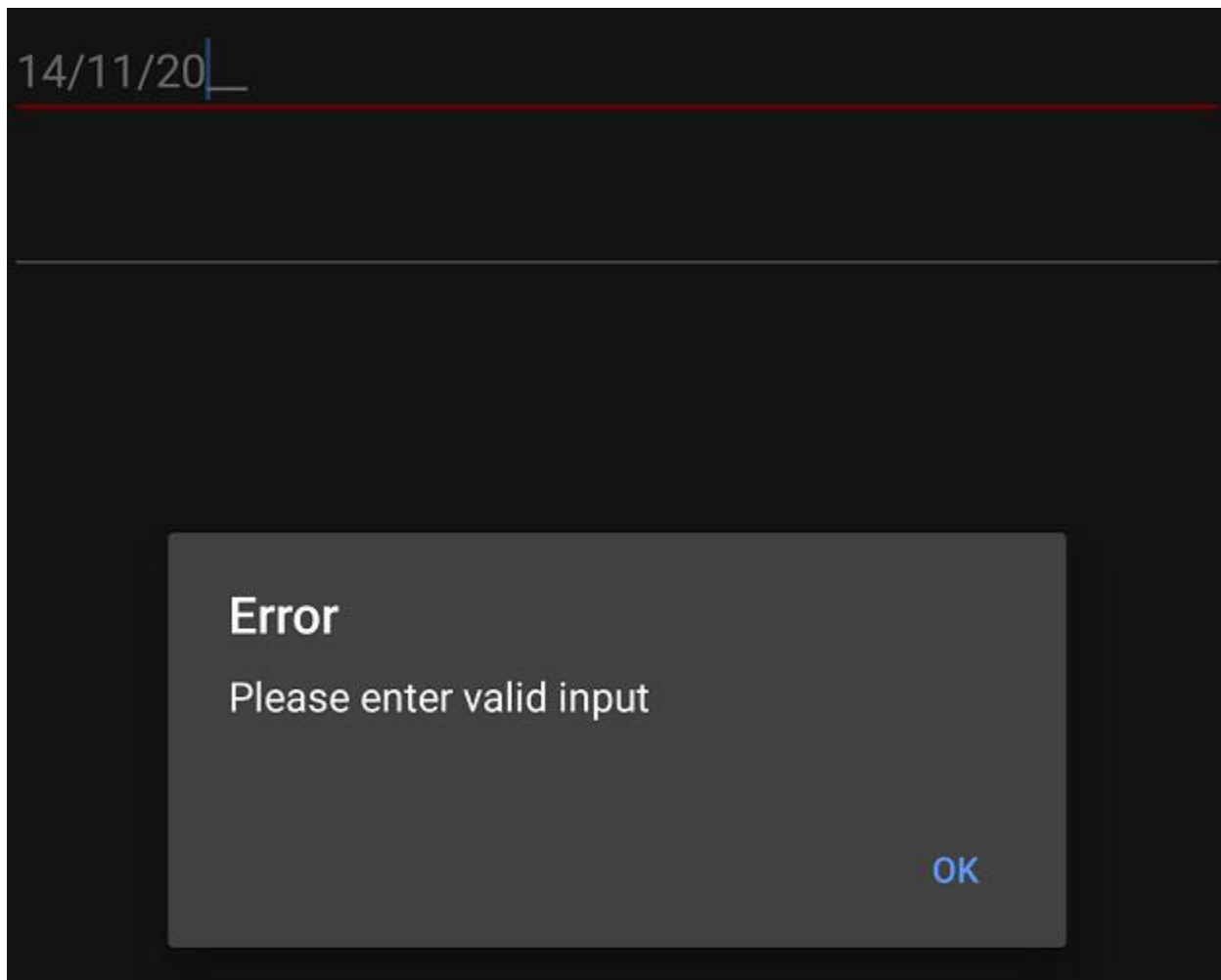
- An input character does not match the corresponding format element. For example, if you enter an alphabetic character when a digit is required. This is probably the most common reason why this event is raised.
- When you try to input extraneous characters beyond the end of the mask.
- A paste operation inserts a character that does not match with its associated format element.

### **XML**

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" Watermark="dd/MM/YYYY" ValidationMode="LostFocus"
MaskInputRejected="MaskedEdit_OnMaskInputRejected"/>
private void MaskedEdit_OnMaskInputRejected(object sender,
MaskInputRejectedEventArgs e)
{
if(e.RejectionHint!= MaskedTextResultHint.UnavailableEditPosition)
DisplayAlert("Error", "Please enter valid input", "Ok");
}
```

## C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.Watermark = "dd/MM/YYYY";
maskedEdit.ValidationMode = InputValidationMode.LostFocus;
maskedEdit.MaskInputRejected += MaskedEdit_OnMaskInputRejected;
private void MaskedEdit_OnMaskInputRejected(object sender,
MaskInputRejectedEventArgs e)
{
if(e.RejectionHint!= MaskedTextResultHint.UnavailableEditPosition)
DisplayAlert("Error", "Please enter valid input", "Ok");
}
```



This demo can be downloaded from this [link](#).

### Formatting Value

SfMaskedEdit allows you to format the characters in the `Value` property in a mask scenario (when the `Mask` property is set). By default, the `Value` property holds your input characters, prompt characters and the literals defined in the mask. You can modify this and allow the `Value` property to hold the characters without prompt and literals by setting the `ValueMaskFormat` property of the control. The `Value` in the `SfMaskedEdit` is formatted by any one of the following formatting enum values:

- `ExcludePromptAndLiterals`
- `IncludePrompt`
- `IncludeLiterals`
- `IncludePromptAndLiterals`

#### `ExcludePromptAndLiterals`

Value contains only the typed characters, the prompt characters and literals are excluded.

#### `IncludePrompt`

Value contains the typed characters and prompt characters, literals are excluded.

### IncludeLiterals

Value contains the typed characters and literals, prompt characters are excluded.

### IncludePromptAndLiterals

Value contains typed characters, prompt characters, and literals.

### XML

```

<syncmaskededit:SfMaskedEdit x:Name="maskedEdit1" MaskType="Text"
Mask="00/00/0000" ValueMaskFormat="ExcludePromptAndLiterals"/>
<Label BindingContext="{x:Reference maskedEdit1}" Text="{Binding Value}" />
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit2" MaskType="Text"
Mask="00/00/0000" ValueMaskFormat="IncludeLiterals"/>
<Label BindingContext="{x:Reference maskedEdit2}" Text="{Binding Value}" />
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit3" MaskType="Text"
Mask="00/00/0000" ValueMaskFormat="IncludePrompt"/>
<Label BindingContext="{x:Reference maskedEdit3}" Text="{Binding Value}" />
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit4" MaskType="Text"
Mask="00/00/0000" ValueMaskFormat="IncludePromptAndLiterals"/>
<Label BindingContext="{x:Reference maskedEdit4}" Text="{Binding Value}" />

```

### C#

```

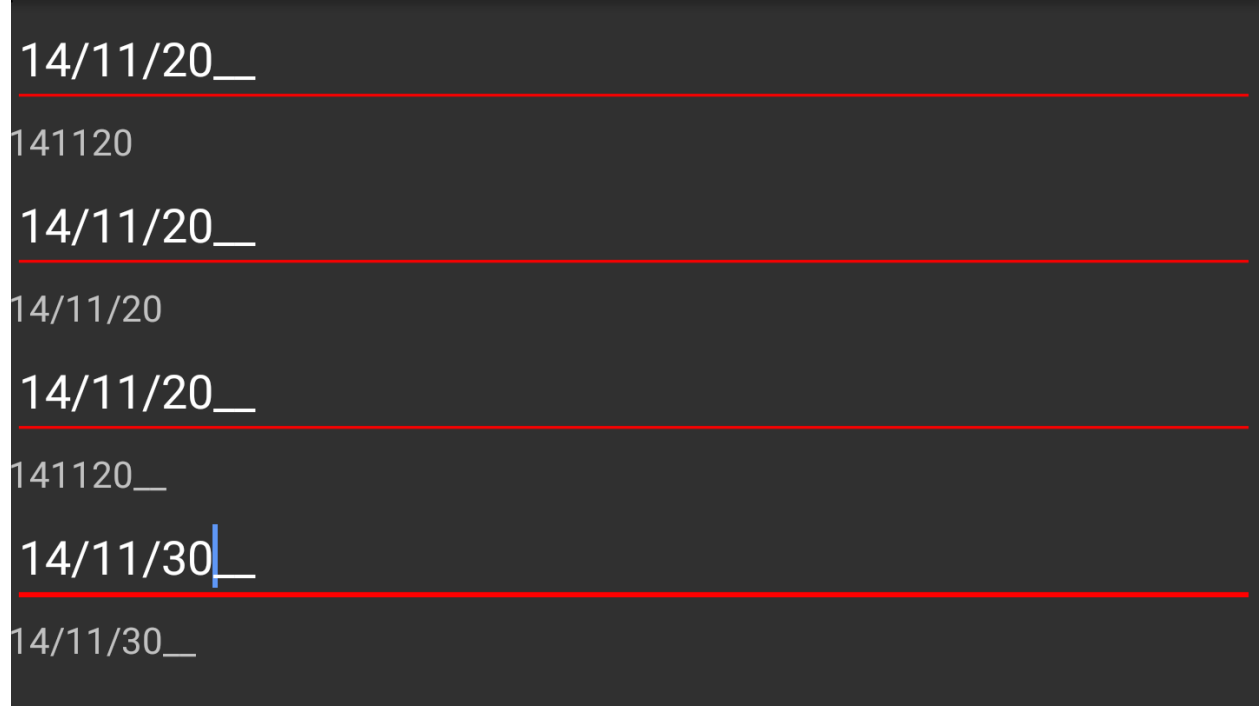
Label label1;
Label label2;
Label label3;
Label label4;
SfMaskedEdit maskedEdit1;
SfMaskedEdit maskedEdit2;
SfMaskedEdit maskedEdit3;
SfMaskedEdit maskedEdit4;
StackLayout stackLayout = new StackLayout();
maskedEdit1 = new SfMaskedEdit();
maskedEdit1.MaskType = MaskType.Text;
maskedEdit1.Mask = "00/00/0000";
maskedEdit1.ValueMaskFormat = MaskFormat.ExcludePromptAndLiterals;
maskedEdit1.ValueChanged += MaskedEdit1_ValueChanged;
label1 = new Label();
stackLayout.Children.Add(maskedEdit1);
stackLayout.Children.Add(label1);
maskedEdit2 = new SfMaskedEdit();
maskedEdit2.MaskType = MaskType.Text;
maskedEdit2.Mask = "00/00/0000";
maskedEdit2.ValueMaskFormat = MaskFormat.IncludeLiterals;
maskedEdit2.ValueChanged += MaskedEdit2_ValueChanged;
label2 = new Label();
stackLayout.Children.Add(maskedEdit2);
stackLayout.Children.Add(label2);
maskedEdit3 = new SfMaskedEdit();
maskedEdit3.MaskType = MaskType.Text;
maskedEdit3.Mask = "00/00/0000";
maskedEdit3.ValueMaskFormat = MaskFormat.IncludePrompt;
maskedEdit3.ValueChanged += MaskedEdit3_ValueChanged;
label3 = new Label();
stackLayout.Children.Add(maskedEdit3);
stackLayout.Children.Add(label3);

```



```
maskedEdit4 = new SfMaskedEdit();
maskedEdit4.MaskType = MaskType.Text;
maskedEdit4.Mask = "00/00/0000";
maskedEdit4.ValueMaskFormat = MaskFormat.IncludePromptAndLiterals;
maskedEdit4.ValueChanged += MaskedEdit4_ValueChanged;
label4 = new Label();
stackLayout.Children.Add(maskedEdit4);
stackLayout.Children.Add(label4);
private void MaskedEdit1_ValueChanged(object sender,
Syncfusion.XForms.MaskedEdit.ValueChangedEventArgs e)
{
    label1.Text = e.Value as string;
}
private void MaskedEdit2_ValueChanged(object sender,
Syncfusion.XForms.MaskedEdit.ValueChangedEventArgs e)
{
    label2.Text = e.Value as string;
}
private void MaskedEdit3_ValueChanged(object sender,
Syncfusion.XForms.MaskedEdit.ValueChangedEventArgs e)
{
    label3.Text = e.Value as string;
}
private void MaskedEdit4_ValueChanged(object sender,
Syncfusion.XForms.MaskedEdit.ValueChangedEventArgs e)
{
    label4.Text = e.Value as string;
}
```

Refer this [link](#) to know more about the `ValueChanged` event of SfMaskedEdit control.



14/11/20\_\_

141120

14/11/20\_\_

14/11/20

14/11/20\_\_

141120\_\_

14/11/30|\_\_

14/11/30\_\_

This demo can be downloaded from this [link](#).

## Formatting clipboard text

SfMaskedEdit allows you to format the clipboard text in a mask scenario (when the Mask property is set). When you perform the cut or copy operation, the clipboard text will be formatted with your input characters and the literals defined in the mask. You can modify this and allow the clipboard to hold the characters with or without prompt and literals by setting the `CutCopyMaskFormat` property of the control. The clipboard text is formatted by any one of the following formatting enum values:

- ExcludePromptAndLiterals
- IncludePrompt
- IncludeLiterals
- IncludePromptAndLiterals

### ExcludePromptAndLiterals

Clipboard text will contain only the typed characters, the prompt characters and literals are excluded.

#### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" CutCopyMaskFormat ="ExcludePromptAndLiterals"/>
```

#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.CutCopyMaskFormat = MaskFormat.ExcludePromptAndLiterals;
```

### IncludePrompt

Clipboard text contains the typed characters and prompt characters, literals are excluded.

#### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" CutCopyMaskFormat ="IncludePrompt"/>
```

#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.CutCopyMaskFormat = MaskFormat.IncludePrompt;
```

### IncludeLiterals

Clipboard text contains the typed characters and literals, prompt characters are excluded.

#### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" CutCopyMaskFormat ="IncludeLiterals"/>
```

#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.CutCopyMaskFormat = MaskFormat.IncludeLiterals;
```

### IncludePromptAndLiterals

Clipboard text contains typed characters, prompt characters, and literals.

#### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" CutCopyMaskFormat="IncludePromptAndLiterals"/>
```

#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.CutCopyMaskFormat = MaskFormat.IncludePromptAndLiterals;
```

### Visual Customization

The appearance of **SfMaskedEdit** can be customized using the following properties:

#### BorderColor

Sets the custom border color to SfMaskedEdit.

#### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" BorderColor="Green"/>
```

#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.BorderColor = Color.Green;
```



#### ErrorBorderColor

Sets the custom error border color to SfMaskedEdit. Error border color indicates the color to be used when the validation fails for your input with respect to the mask used.

Validation triggers based on **ValidationMode** property.

Refer this [link](#) to know more about the **ValidationMode** property of SfMaskedEdit control.

#### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" BorderColor="Green" ErrorBorderColor="Yellow"/>
```

### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.ErrorBorderColor = Color.Yellow;
```



### Setting Appearance of Text

You can customize the display text appearance of SfMaskedEdit control using the following properties:

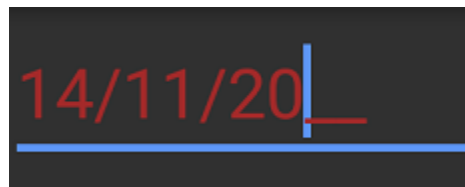
- **TextColor**: Changes the color of the text.
- **HorizontalTextAlignment**: Changes the horizontal alignment of the text.
- **FontFamily**: Changes the font family of the text.
- **FontAttributes**: Sets font attributes(bold/italic/none) of the text.
- **FontSize**: Sets font size of the text.

### XML

```
<syncmaskededit:SfMaskedEdit x:Name="maskedEdit" MaskType="Text"
Mask="00/00/0000" TextColor="Brown" HorizontalTextAlignment="Center"
FontFamily="Arial" FontAttributes="Bold" FontSize="20"/>
```

### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.MaskType = MaskType.Text;
maskedEdit.Mask = "00/00/0000";
maskedEdit.TextColor = Color.Brown;
maskedEdit.HorizontalTextAlignment = TextAlignment.Center;
maskedEdit.FontFamily = "Arial";
maskedEdit.FontAttributes = FontAttributes.Bold;
maskedEdit.FontSize = 20;
```



This demo can be downloaded from this [link](#).

## Show password character

The SfMaskedEdit control supports to work as a password text box when setting a character to the PasswordChar property.

### XML

```
<syncfusion:SfMaskedEdit x:Name="maskedEdit" Mask="\w+" MaskType="Regex" PasswordChar="*" />
```

### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();  
maskedEdit.Mask = @"\w+";  
maskedEdit.MaskType = MaskType.RegEx;  
maskedEdit.PasswordChar = '*';
```

### Password Character



## Password Delay

When providing password character, you can show the typed character with some delay using the EnablePasswordDelay property. When enabling the EnablePasswordDelay property, the typed character will be displayed for a few seconds before it is converted to the password character.

### XML

```
<syncfusion:SfMaskedEdit x:Name="maskedEdit" Mask="\w+" MaskType="Regex" PasswordChar="*" EnablePasswordDelay="True" />
```

### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();  
maskedEdit.Mask = @"\w+";  
maskedEdit.MaskType = MaskType.RegEx;  
maskedEdit.PasswordChar = '*';  
maskedEdit.EnablePasswordDelay = true;
```

## Password Character



---

**Note:** The default value of the EnablePasswordDelay property is false.

---

### Password Delay Duration

When “PasswordDelay” is enabled, you can handle the duration of the displaying typed character using the `PasswordDelayDuration` property.

#### XML

```
<syncfusion:SfMaskedEdit x:Name="maskedEdit" Mask="\w+" MaskType="Regex"
PasswordChar="*" EnablePasswordDelay="True" PasswordDelayDuration="2" />
```

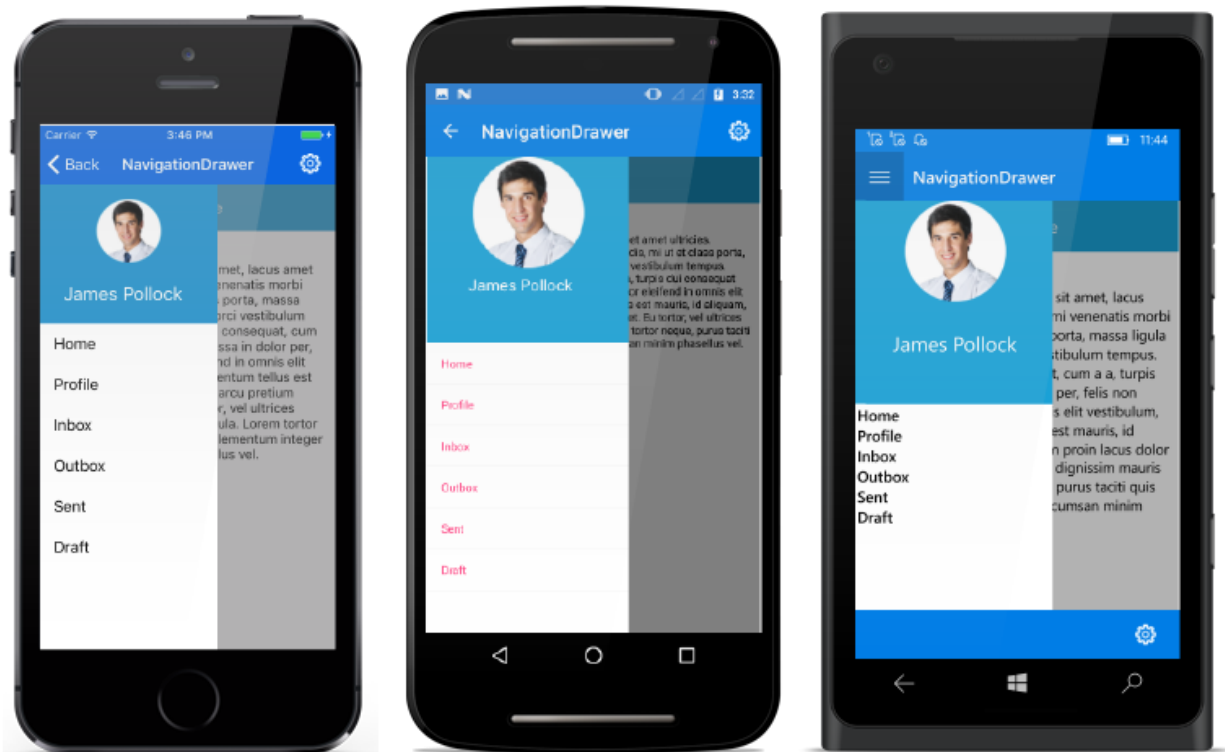
#### C#

```
SfMaskedEdit maskedEdit = new SfMaskedEdit();
maskedEdit.Mask = @"\w+";
maskedEdit.MaskType = MaskType.RegEx;
maskedEdit.PasswordChar = '*';
maskedEdit.EnablePasswordDelay = true;
maskedEdit.PasswordDelayDuration = 2;
```

## SfNavigationDrawer

### Overview

Essential NavigationDrawer for Xamarin.Forms is a simpler component to create navigation pane in application. It has a content area and a sliding pane that slides out from the edge of the page. The pane can be opened by swiping the edges of the screen or programmatically.



## Key Features

- Pane positions – Supports pane position in all four directions such as Left, Right, Top and Bottom.
- Animated transitions – Supports opening/closing of pane due to the transition SlideOnTop, Push and Reveal.
- Changeable swipe sensitivity – Flexible for the users to update touch threshold based on their device screen size.

## Getting Started

This section explains you the steps required to create a navigation DrawerPanel with content area and data filled drawer and it covers only the minimal features that you need to know to get started with the NavigationDrawer.

### Adding SfNavigationDrawer reference

You can add SfNavigationDrawer reference using one of the following methods:

#### Method 1: Adding SfNavigationDrawer reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfNavigationDrawer). To add SfNavigationDrawer to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfNavigationDrawer](https://www.nuget.org/packages/Syncfusion.Xamarin.SfNavigationDrawer), and then install it.

![[Adding SfNavigationDrawer reference from NuGet]](Images/Adding SfNavigationDrawer reference.png)

**Note:** Install the same version of SfNavigationDrawer NuGet in all the projects.

#### Method 2: Adding SfNavigationDrawer reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfNavigationDrawer control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfNavigationDrawer assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfNavigationDrawer.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfNavigationDrawer.Android.dll Syncfusion.SfNavigationDrawer.XForms.Android.dll Syncfusion.SfNavigationDrawer.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfNavigationDrawer.iOS.dll Syncfusion.SfNavigationDrawer.XForms.iOS.dll Syncfusion.SfNavigationDrawer.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfNavigationDrawer.UWP.dll Syncfusion.SfNavigationDrawer.XForms.UWP.dll Syncfusion.SfNavigationDrawer.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** After adding the reference, an additional step is required for iOS and UWP projects. If you are adding the references from toolbox, this step is not needed.

#### *Additional Step for iOS*

Currently an additional step is required for iOS project. We need to create an instance of the NavigationDrawer custom renderer as shown below.

Create an instance of SfNavigationDrawerRenderer in FinishedLaunching overridden method of AppDelegate class in iOS project as shown below:



**C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    new Syncfusion.SfNavigationDrawer.XForms.iOS.SfNavigationDrawerRenderer();
    global::Xamarin.Forms.Forms.Init(); LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

*Additional Step for UWP*

This step is required only if application is deployed in Release mode with .NET native tool chain enabled and it is for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the SfNavigationDrawer assembly at OnLaunched overridden method of App class in UWP project is the suggested workaround. And the code example is shown below:

**C#**

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    #if DEBUG
    if (System.Diagnostics.Debugger.IsAttached)
    {
        this.DebugSettings.EnableFrameRateCounter = true;
    }
    #endif
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame == null)
    {
        rootFrame = new Frame();
        rootFrame.NavigationFailed += OnNavigationFailed;
        List<System.Reflection.Assembly> assembliesToInclude = new
        List<System.Reflection.Assembly>();
        // Add all the renderer assemblies your app uses
        assembliesToInclude.Add(typeof(Syncfusion.SfNavigationDrawer.XForms.UWP.SfNa
        vigationDrawerRenderer).GetTypeInfo().Assembly);
        // Replace the Xamarin.Forms.Forms.Init(e);
        Xamarin.Forms.Forms.Init(e, assembliesToInclude);
        if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
        {
            //TODO: Load state from previously suspended application
        }
        // Place the frame in the current Window
        Window.Current.Content = rootFrame;
    }
    if (rootFrame.Content == null)
    {
        // When the navigation stack isn't restored navigate to the first page,
        // configuring the new page by passing required information as a navigation
        // parameter rootFrame.Navigate(typeof(MainPage), e.Arguments);
    }
    // Ensure the current window is active
    Window.Current.Activate();
}
```

## Initialize SfNavigationDrawer

Import the SfNavigationDrawer namespace in respective Page as shown below:

### XML

```
<xmlns:navigationdrawer="clr-  
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi  
gationDrawer.XForms"/>
```

### C#

```
using Syncfusion.SfNavigationDrawer.XForms;
```

Then initialize an empty navigation drawer as shown below,

### XML

```
<?xml version="1.0" encoding="utf-8"?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:local="clr-namespace:NaviSample"  
xmlns:navigationdrawer="clr-  
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi  
gationDrawer.XForms"  
x:Class="NaviSample.MainPage">  
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer">  
<navigationdrawer:SfNavigationDrawer.ContentView>  
<Grid/>  
</navigationdrawer:SfNavigationDrawer.ContentView>  
</navigationdrawer:SfNavigationDrawer>  
</ContentPage>
```

### C#

```
using Syncfusion.SfNavigationDrawer.XForms;  
using Xamarin.Forms;  
namespace NaviSample  
{  
    public partial class MainPage : ContentPage  
    {  
        public MainPage()  
        {  
            InitializeComponent();  
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();  
            Grid grid = new Grid();  
            navigationDrawer.ContentView = grid;  
            this.Content = navigationDrawer;  
        }  
    }  
}
```

**Note:** It is mandatory to set ContentView for SfNavigationDrawer on initializing.

## Adjust Drawer Size

The default position of navigation pane is left so let us change the drawer width to 200.

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
DrawerWidth="200">
<navigationdrawer:SfNavigationDrawer.ContentView>
<Grid/>
</navigationdrawer:SfNavigationDrawer.ContentView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

### C#

```
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                DrawerWidth = 200
            };
            Grid grid = new Grid();
            navigationDrawer.ContentView = grid;
            this.Content = navigationDrawer;
        }
    }
}
```

**Note:** For changing the side of navigation pane use Position property. Use DrawerHeight property to change the drawer height in Top and Bottom positions.

### Add Hamburger Menu for Toggling Drawer

Create a button and set required image to the Image property of Button. Subscribe Clicked event of the button and invoke ToggleDrawer() method in it to toggle the drawer. Set this button as ContentView property of SfNavigationDrawer. Align the layout of ContentView properly to get the hamburger icon at top left as shown in following code:

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
```

```

xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
DrawerWidth ="200">
<navigationdrawer:SfNavigationDrawer.ContentView>
<Grid x:Name="mainContentView"
BackgroundColor="White">
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
<RowDefinition/>
</Grid.RowDefinitions>
<StackLayout BackgroundColor="#1aa1d6"
Orientation="Horizontal">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
<Label x:Name="headerLabel"
HeightRequest="50"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"
Text="Home" FontSize="16"
TextColor="White"
BackgroundColor="#1aa1d6"/>
</StackLayout>
<Label Grid.Row="1"
x:Name="contentLabel"
VerticalOptions="Center"
HorizontalOptions="Center"
Text="Content View"
FontSize="14"
TextColor="Black"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.ContentView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>

```

**C#**

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage ()
        {
            InitializeComponent();
            hamburgerButton.Image =
            (FileImageSource) ImageSource.FromFile("hamburger_icon.png");

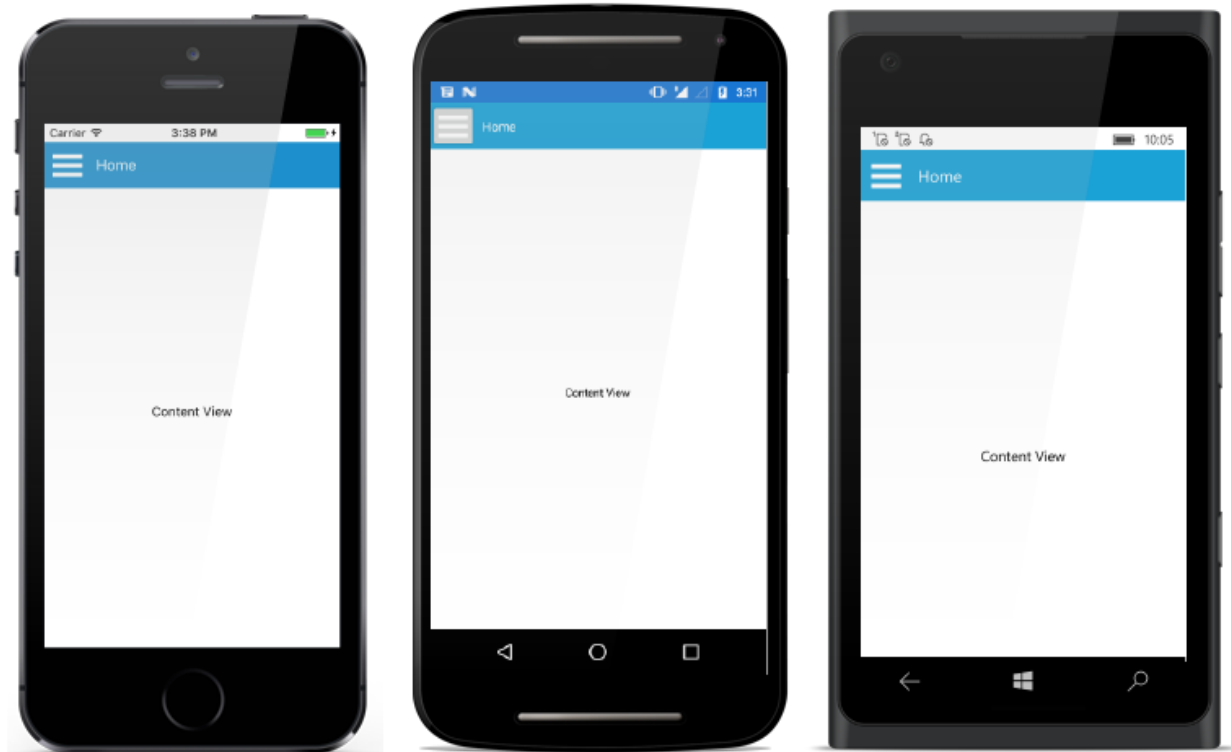
```

```

}
void hamburgerButton_Clicked(object sender, EventArgs e)
{
    navigationDrawer.ToggleDrawer();
}
}
}

```

**Note:** Add the required images in drawable folder of Android project, Assets folder of iOS project and add it directly to the UWP project.



### Set ListView as Drawer Content

Create a ListView with five items and set it as DrawerContentView.

### XML

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
DrawerWidth ="200"
DrawerHeaderHeight="160">
<navigationdrawer:SfNavigationDrawer.ContentView>
<Grid x:Name="mainContentView"
BackgroundColor="White">

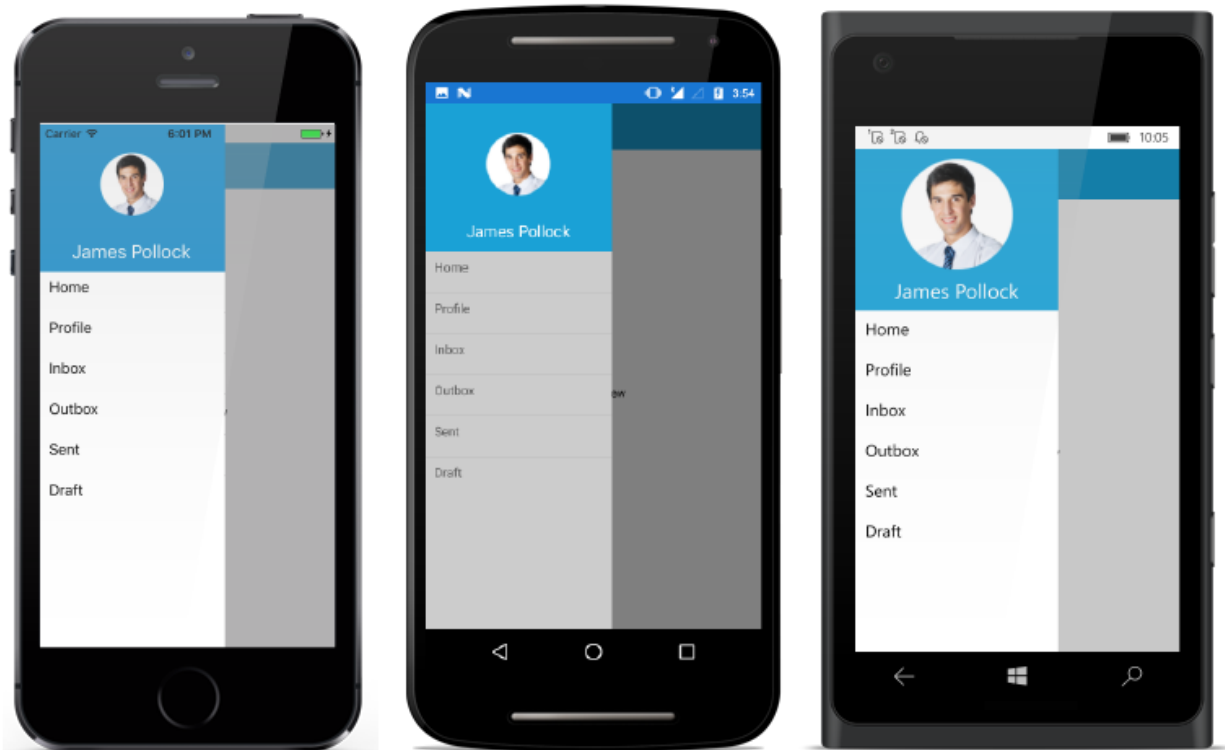
```

```
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
</Grid.RowDefinitions>
<StackLayout BackgroundColor="#1aa1d6"
Orientation="Horizontal">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
<Label x:Name="headerLabel"
HeightRequest="50"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"
Text="Home"
FontSize="16"
TextColor="White"
BackgroundColor="#1aa1d6"/>
</StackLayout>
<Label Grid.Row="1"
x:Name="contentLabel"
VerticalOptions="Center"
HorizontalOptions="Center"
Text="Content View"
FontSize="14"
TextColor="Black"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1aa1d6">
<Grid.RowDefinitions>
<RowDefinition Height="120"/>
<RowDefinition Height="40"/>
</Grid.RowDefinitions>
<Image Source="user.png"
HeightRequest="110"
Margin="0,10,0,0"
BackgroundColor="#1aa1d6"
VerticalOptions="Center"
HorizontalOptions="Center"/>
<Label Text="James Pollock"
Grid.Row="1"
HorizontalTextAlignment="Center"
HorizontalOptions="Center"
FontSize="20"
TextColor="White"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<navigationdrawer:SfNavigationDrawer.DrawerContentView>
<ListView x:Name="listView"
ItemSelected="listView_ItemSelected">
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
```

```
<StackLayout HeightRequest="40">
<Label Margin="10,7,0,0"
Text="{Binding}"
FontSize="16"/>
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</navigationdrawer:SfNavigationDrawer.DrawerContentView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
using System;
using System.Collections.Generic;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
hamburgerButton.Image =
(FileImageSource) ImageSource.FromFile("hamburger_icon.png");
List<string> list = new List<string>();
list.Add("Home");
list.Add("Profile");
list.Add("Inbox");
list.Add("Out box");
list.Add("Sent");
list.Add("Draft");
listView.ItemsSource = list;
}
void hamburgerButton_Clicked(object sender, EventArgs e)
{
navigationDrawer.ToggleDrawer();
}
private void listView_ItemSelected(object sender,
SelectedItemChangedEventArgs e)
{
// Your codes here
navigationDrawer.ToggleDrawer();
}
}
}
```



We have created knowledge base document by creating SfNavigationDrawer sample fully in code behind. Please refer the same in this [link](#).

You can find the Getting Started Sample from this [link](#).

## Main Content

Main content of NavigationDrawer is always visible and it can be set using **ContentView** property. In the following code example, ContentView is switched when selection changes in ListView.

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
DrawerHeaderHeight="160">
<navigationdrawer:SfNavigationDrawer.ContentView>
<Grid x:Name="mainContentView" BackgroundColor="White">
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
<RowDefinition/>
</Grid.RowDefinitions>
<StackLayout BackgroundColor="#1a1d6" Orientation="Horizontal">
<Button x:Name="hamburgerButton" HeightRequest="50" WidthRequest="50"
HorizontalOptions="Start" FontSize="20" BackgroundColor="#1a1d6"
Clicked="hamburgerButton_Clicked"/>
```



```

<Label x:Name="headerLabel" HeightRequest="50"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center" Text="Home"
FontSize="16" TextColor="White" BackgroundColor="#1a1d6"/>
</StackLayout>
<Label Grid.Row="1" x:Name="contentLabel" VerticalOptions="Center"
HorizontalOptions="Center" Text="Content View" FontSize="14"
TextColor="Black"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1a1d6">
<Grid.RowDefinitions>
<RowDefinition Height="120"/>
<RowDefinition Height="40"/>
</Grid.RowDefinitions>
<Image Source="user.png" HeightRequest="110" Margin="0,10,0,0"
BackgroundColor="#1a1d6" VerticalOptions="Center"
HorizontalOptions="Center"/>
<Label Text="James Pollock" Grid.Row="1" HorizontalTextAlignment="Center"
HorizontalOptions="Center" FontSize="20" TextColor="White"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<navigationdrawer:SfNavigationDrawer.DrawerContentView>
<ListView x:Name="listView" ItemSelected="listView_ItemSelected">
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout HeightRequest="40">
<Label Margin="10,7,0,0" Text="{Binding}" FontSize="16"/>
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</navigationdrawer:SfNavigationDrawer.DrawerContentView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>

```

**C#**

```

using System;
using System.Collections.Generic;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
public partial class MainPage : ContentPage
{
SfNavigationDrawer navigationDrawer = new SfNavigationDrawer() {
DrawerHeaderHeight = 160 };
Label label1;
public MainPage()
{
InitializeComponent();
Grid grid = new Grid();
grid.BackgroundColor = Color.White;

```

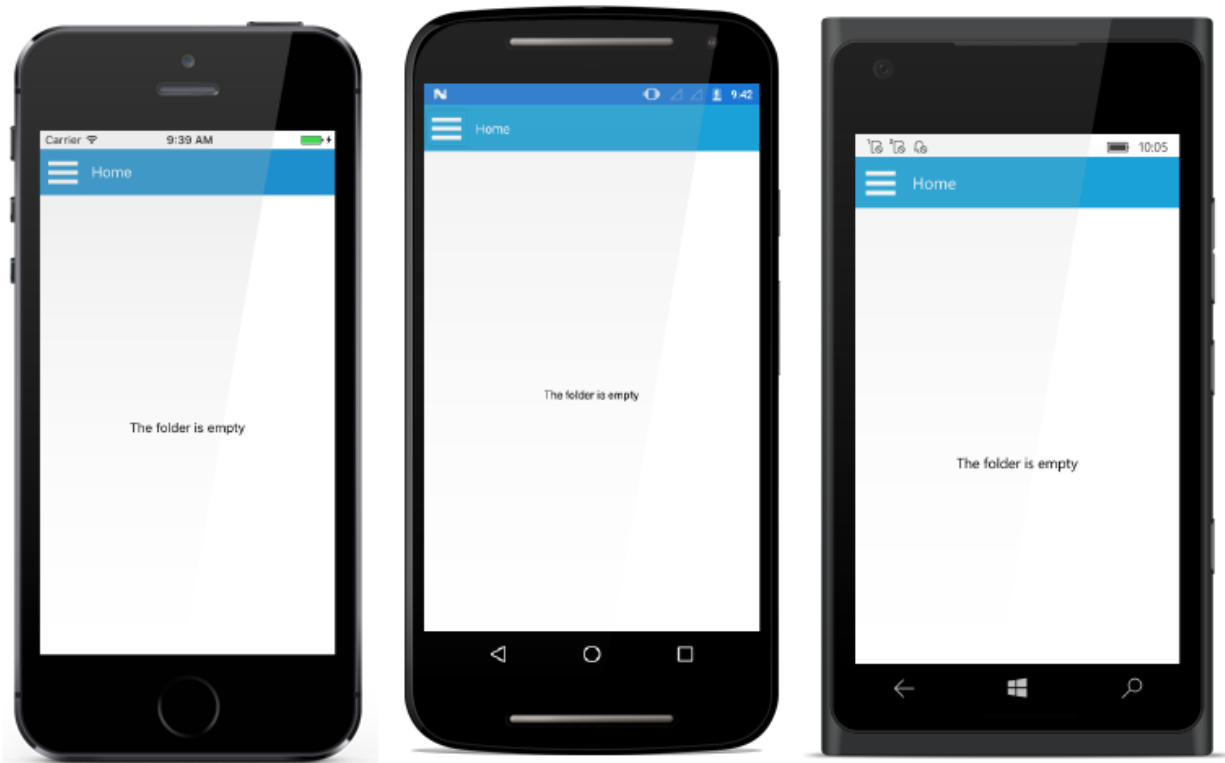
```
RowDefinition rowDef1 = new RowDefinition() { Height = GridLength.Auto };
RowDefinition rowDef2 = new RowDefinition() { Height = GridLength.Star };
grid.RowDefinitions.Add(rowDef1);
grid.RowDefinitions.Add(rowDef2);
StackLayout layout = new StackLayout() { BackgroundColor =
Color.FromHex("#1aa1d6"), Orientation = StackOrientation.Horizontal };
var hamburgerButton = new Button
{
    Text = "StackLayout",
    HeightRequest = 50,
    WidthRequest = 50,
    HorizontalOptions = LayoutOptions.Start,
    FontSize = 20,
    BackgroundColor = Color.FromHex("#1aa1d6")
};
hamburgerButton.Clicked += hamburgerButton_Clicked;
var label = new Label
{
    HeightRequest = 50,
    HorizontalTextAlignment = TextAlignment.Center,
    VerticalTextAlignment = TextAlignment.Center,
    Text = "Home",
    FontSize = 16,
    TextColor = Color.White,
    BackgroundColor = Color.FromHex("#1aa1d6")
};
layout.Children.Add(hamburgerButton);
layout.Children.Add(label);
labell = new Label
{
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    Text = "Content View",
    FontSize = 14,
    TextColor = Color.Black
};
grid.Children.Add(layout, 0, 0);
grid.Children.Add(labell, 0, 1);
navigationDrawer.ContentView = grid;
Grid grid1 = new Grid();
grid1.BackgroundColor = Color.FromHex("#1aa1d6");
RowDefinition rowDef3 = new RowDefinition() { Height = 120 };
RowDefinition rowDef4 = new RowDefinition() { Height = 40 };
grid1.RowDefinitions.Add(rowDef3);
grid1.RowDefinitions.Add(rowDef4);
var image = new Image
{
    Source = (FileImageSource)ImageSource.FromFile("user.png"),
    HeightRequest = 110,
    Margin = new Thickness(0, 10, 0, 0),
    BackgroundColor = Color.FromHex("#1aa1d6"),
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center
};
var label2 = new Label
{
    Text = "James Pollock",
```

```
HorizontalTextAlignment = TextAlignment.Center,
HorizontalOptions = LayoutOptions.Center,
FontSize = 20,
TextColor = Color.White
};
grid1.Children.Add(image, 0, 0);
grid1.Children.Add(label2, 0, 1);
navigationDrawer.DrawerHeaderView = grid1;
ListView listView = new ListView();
listView.ItemSelected += listView_ItemSelected;
StackLayout layout1 = new StackLayout() { HeightRequest = 40 };
var label3 = new Label
{
    Margin = new Thickness(10, 7, 0, 0),
    TextColor = Color.Black,
    FontSize = 16,
};
label3.SetBinding(Label.TextProperty, "Text");
navigationDrawer.DrawerContentView = listView;
this.Content = navigationDrawer;
navigationDrawer.DrawerWidth = 200;
hamburgerButton.Image =
    (FileImageSource) ImageSource.FromFile("hamburger_icon.png");
List<string> list = new List<string>();
list.Add("Home");
list.Add("Profile");
list.Add("Inbox");
list.Add("Out box");
list.Add("Sent");
list.Add("Draft");
listView.ItemsSource = list;
}
void hamburgerButton_Clicked(object sender, EventArgs e)
{
    navigationDrawer.ToggleDrawer();
}
private void listView_ItemSelected(object sender,
    SelectedItemChangedEventArgs e)
{
    if (e.SelectedItem.ToString() == "Home")
        label1.Text = "Home";
    else if (e.SelectedItem.ToString() == "Profile")
        label1.Text = "Profile";
    else if (e.SelectedItem.ToString() == "Inbox")
        label1.Text = "Inbox";
    else if (e.SelectedItem.ToString() == "Out box")
        label1.Text = "Out box";
    else if (e.SelectedItem.ToString() == "Sent")
        label1.Text = "Sent";
    else if (e.SelectedItem.ToString() == "Draft")
        label1.Text = "The folder is empty";
    navigationDrawer.ToggleDrawer();
}
}
```

---

**Note:** It is mandatory to set ContentView for SfNavigationDrawer on initializing.

---



## Multi Drawer

The navigation drawer allows users to open the drawer on multiple sides with different toggle methods. The DrawerSettings class and its properties need to be used when users need to provide multiple drawer. The multiple drawers can be implemented using the following drawer settings.

- Default drawer settings
- Secondary drawer settings

---

**Note:** The header and footer content are optional, but the drawer content is mandatory to allocate space for the drawer.

---

## Default drawer settings

Implement the default drawer using the default drawer settings class. The following code sample demonstrates how to set the default drawer settings's properties inside the DrawerSettings class.

## XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
x:Class="NaviSample.MainPage">
  <navigationdrawer:SfNavigationDrawer.DefaultDrawerSettings >
  <navigationdrawer:DrawerSettings
```

```

DrawerWidth="150"
DrawerHeight="150"
Transition ="SlideOnTop"
DrawerHeaderHeight="150"
DrawerFooterHeight="150"
ContentBackgroundColor ="Red"
Position="Left">
</navigationdrawer:DrawerSettings>
</navigationdrawer:SfNavigationDrawer.DefaultDrawerSettings>
</ContentPage>

```

**C#**

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
            DrawerSettings defaultDrawerSettings = new DrawerSettings();
            defaultDrawerSettings.DrawerHeight = 150;
            defaultDrawerSettings.Position = Position.Left;
            defaultDrawerSettings.Transition = Transition.SlideOnTop;
            defaultDrawerSettings.ContentBackgroundColor = Color.Red;
            defaultDrawerSettings.DrawerWidth = 150;
            navigationDrawer.DrawerHeaderHeight = 150;
            navigationDrawer.DrawerFooterHeight = 150;
            navigationDrawer.DefaultDrawerSettings = defaultDrawerSettings;
            this.Content = navigationDrawer;
        }
    }
}

```

**Note:** The navigation drawer works with the value given for the properties inside the DrawerSettings class when using the default drawer settings.

*Default drawer header view*

The header content can be provided to the default drawer using the `DrawerHeaderView` property inside the DrawerSettings class of DefaultDrawerSettings. The following code demonstrates how to set header content for the default drawer.

**XML**

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"

```

```

x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer.DefaultDrawerSettings>
<Grid BackgroundColor="#1aa1d6">
<Grid.RowDefinitions>
<RowDefinition Height="120"/>
</Grid.RowDefinitions>
<navigationdrawer:DrawerSettings.DrawerHeaderView>
<Label Text="Syncfusion Xamarin Products" HorizontalTextAlignment="Center"
HorizontalOptions="Center" FontSize="20" TextColor="White"/>
</navigationdrawer:DrawerSettings.DrawerHeaderView>
</Grid>
</navigationdrawer:SfNavigationDrawer.DefaultDrawerSettings>
</ContentPage>

```

### C#

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
DrawerSettings defaultDrawerSettings = new DrawerSettings();
Grid headerLayout = new Grid();
headerLayout.BackgroundColor = Color.FromHex("#1aa1d6");
Label header = new Label();
header.Text = "Syncfusion Xamarin Products";
header.FontSize = 20;
header.TextColor = Color.White;
header.HorizontalTextAlignment = TextAlignment.Center;
headerLayout.Children.Add(header);
defaultDrawerSettings.DrawerHeaderView = headerLayout;
this.Content = navigationDrawer;
}
}
}

```

#### Default drawer content view

The drawer content can be provided to the default drawer using the `DrawerContentView` property inside the `DrawerSettings` class. The following code demonstrates how to set drawer content to the default drawer.

### XML

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"

```

```

xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer.DefaultDrawerSettings>
<navigationdrawer:DrawerSettings>
<navigationdrawer:DrawerSettings.DrawerContentView>
<Label Text="DrawerContent "
HorizontalTextAlignment="Center"
HorizontalOptions="Center"
FontSize="20"
TextColor="White"/>
</navigationdrawer:DrawerSettings.DrawerContentView>
</navigationdrawer:DrawerSettings>
</navigationdrawer:SfNavigationDrawer.DefaultDrawerSettings>
</ContentPage>

```

## C#

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
DrawerSettings defaultDrawerSettings = new DrawerSettings();
Grid contentLayout = new Grid();
contentLayout.BackgroundColor = Color.FromHex("#1aa1d6");
Label content = new Label();
content.Text = "DrawerContent";
content.FontSize = 20;
content.TextColor = Color.White;
content.HorizontalTextAlignment = TextAlignment.Center;
contentLayout.Children.Add(content);
defaultDrawerSettings.DrawerContentView = contentLayout;
this.Content = navigationDrawer;
}
}
}

```

### Default drawer footer view

The footer content can be provided to the default drawer using the **DrawerFooterView** property inside the **DrawerSettings** class of **DefaultDrawerSettings**. The following code demonstrates how to set footer content to the default drawer.

## XML

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

```

```

xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer.DefaultDrawerSettings>
<navigationdrawer:DrawerSettings.DrawerFooterView>
<Grid BackgroundColor="#1aa1d6">
<Grid.RowDefinitions>
<RowDefinition Height="120"/>
</Grid.RowDefinitions>
<Label Text="Close"
Grid.Row="0"
HorizontalTextAlignment="Center"
HorizontalOptions="Center"
FontSize="20"
TextColor="White"/>
</Grid>
</navigationdrawer:DrawerSettings.DrawerFooterView>
</navigationdrawer:SfNavigationDrawer.DefaultDrawerSettings>
</ContentPage>

```

## C#

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
            DrawerSettings defaultDrawerSettings = new DrawerSettings();
            Grid footerLayout = new Grid();
            footerLayout.BackgroundColor = Color.FromHex("#1aa1d6");
            Label footer = new Label();
            footer.Text = "Close";
            footer.FontSize = 20;
            footer.TextColor = Color.White;
            footer.HorizontalTextAlignment = TextAlignment.Center;
            footerLayout.Children.Add(footer);
            defaultDrawerSettings.DrawerFooterView = footerLayout;
            this.Content = navigationDrawer;
        }
    }
}

```

### Secondary drawer settings

Implement the secondary drawer using the secondary drawer settings class. Its properties and functionalities are same as the default drawer. The secondary drawer can be set to different positions



similar to the default drawer. The following code demonstrates how to set the secondary drawer settings's properties inside the DrawerSettings class.

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer.SecondaryDrawerSettings >
<navigationdrawer:DrawerSettings x:Name="secondaryDrawer"
DrawerHeaderHeight="40"
DrawerFooterHeight="40"
ContentBackgroundColor="Blue"
DrawerHeight="300"
DrawerWidth="150"
Duration="400"
Position="Right"
Transition="SlideOnTop">
</navigationdrawer:DrawerSettings>
</navigationdrawer:SfNavigationDrawer.SecondaryDrawerSettings>
</ContentPage>
```

### C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
            DrawerSettings secondaryDrawer = new DrawerSettings();
            secondaryDrawer.DrawerHeight = 300;
            secondaryDrawer.Position = Position.Right;
            secondaryDrawer.Transition = Transition.SlideOnTop;
            secondaryDrawer.ContentBackgroundColor = Color.Blue;
            secondaryDrawer.DrawerWidth = 150;
            secondaryDrawer.DrawerHeaderHeight = 40;
            secondaryDrawer.DrawerFooterHeight = 40;
            navigationDrawer.SecondaryDrawerSettings = secondaryDrawer;
            this.Content = navigationDrawer;
        }
    }
}
```

**Note:** When the default drawer and the secondary drawer are set to the same position, the default drawer will open on swiping.

#### Secondary drawer header view

The header content can be provided to the secondary drawer using the `DrawerHeaderView` property inside the `DrawerSettings` class of `SecondaryDrawerSettings`. The following code demonstrates how to set the header content to the secondary drawer.

#### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:NaviSample"
  xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
  x:Class="NaviSample.MainPage">
  <navigationdrawer:SfNavigationDrawer.SecondaryDrawerSettings>
  <navigationdrawer:DrawerSettings.DrawerHeaderView>
  <Grid BackgroundColor="#1aald6">
  <Grid.RowDefinitions>
  <RowDefinition Height="120"/>
  </Grid.RowDefinitions>
  <Label Text="Syncfusion Enterprise solution"
  Grid.Row="0"
  HorizontalTextAlignment="Center"
  HorizontalOptions="Center"
  FontSize="20"
  TextColor="White"/>
  </Grid>
  </navigationdrawer:DrawerSettings.DrawerHeaderView>
  </navigationdrawer:SfNavigationDrawer.SecondaryDrawerSettings>
</ContentPage>
```

#### C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
  public partial class MainPage : ContentPage
  {
    public MainPage()
    {
      InitializeComponent();
      SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
      DrawerSettings secondaryDrawer = new DrawerSettings();
      Grid secondary = new Grid();
      secondary.BackgroundColor = Color.FromHex("#1aald6");
      Label header = new Label();
      header.Text = "Syncfusion Enterprise solution";
      header.FontSize = 20;
      header.TextColor = Color.White;
      header.HorizontalTextAlignment = TextAlignment.Center;
```

```

secondary.Children.Add(header);
secondaryDrawer.DrawerHeaderView = secondary;
this.Content = navigationDrawer;
}
}
}

```

### Secondary drawer content view

The drawer content can be provided to the default drawer using the `DrawerContentView` property inside the `DrawerSettings` class of `SecondaryDrawerSettings`. The following code demonstrates how to set the drawer content to the secondary drawer.

### XML

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer.SecondaryDrawerSettings>
<navigationdrawer:DrawerSettings>
<navigationdrawer:DrawerSettings.DrawerContentView>
<Label Text="DrawerContent "
HorizontalTextAlignment="Center"
HorizontalOptions="Center"
FontSize="20"
TextColor="White"/>
</navigationdrawer:DrawerSettings.DrawerContentView>
</navigationdrawer:DrawerSettings>
</navigationdrawer:SfNavigationDrawer.SecondaryDrawerSettings>
</ContentPage>

```

### C#

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
DrawerSettings secondaryDrawer = new DrawerSettings();
Grid Layout = new Grid();
Layout.BackgroundColor = Color.FromHex("#1aa1d6");
Label content = new Label();
content.Text = "DrawerContent";
content.FontSize = 20;
content.TextColor = Color.White;
}
}
}

```

```

content.HorizontalTextAlignment = TextAlignment.Center;
Layout.Children.Add(content);
secondaryDrawer.DrawerContentView = Layout;
this.Content = navigationDrawer;
}
}
}

```

### Secondary drawer footer view

The footer content can be provided to the secondary drawer using the **DrawerFooterView** property inside the **DrawerSettings** class of **SecondaryDrawerSettings**. The following code demonstrates how to set footer content to the secondary drawer.

### XML

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer.SecondaryDrawerSettings>
<navigationdrawer:DrawerSettings.DrawerFooterView>
<Grid BackgroundColor="#1aald6">
<Grid.RowDefinitions>
<RowDefinition Height="120"/>
</Grid.RowDefinitions>
<Label Text="Close"
Grid.Row="0"
HorizontalTextAlignment="Center"
HorizontalOptions="Center"
FontSize="20"
TextColor="White"/>
</Grid>
</navigationdrawer:DrawerSettings.DrawerFooterView>
</navigationdrawer:SfNavigationDrawer.SecondaryDrawerSettings>
</ContentPage>

```

### C#

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer();
            DrawerSettings secondaryDrawer = new DrawerSettings();
            Grid Layout = new Grid();

```

```

Layout.BackgroundColor = Color.FromHex("#1aa1d6");
Label footer = new Label();
footer.Text = "Close";
footer.FontSize = 20;
footer.TextColor = Color.White;
footer.HorizontalTextAlignment = TextAlignment.Center;
Layout.Children.Add(footer);
secondaryDrawer.DrawerFooterView = Layout;
this.Content = navigationDrawer;
}
}
}

```

### Toggle method

Users can toggle the secondary drawer using the `ToggleSecondaryDrawer` method.

### C#

```
navigationDrawer.ToggleSecondaryDrawer();
```

### Opening the drawer programmatically

The `IsOpen` property in the `DrawerSettings` of `SecondaryDrawerSettings` is used to open or close the drawer programmatically.

---

**Note:** Users can open only one drawer at a time.

---

Note: The sample for implementing multiple drawers can be downloaded from this [link](#).

### Navigation Pane Sides

The supplemental pane can be drawn in and out from all four sides. `Position` property is used to change the side of pane and the values are

*Left Right Top Bottom*

---

**Note:** The default position is Left.

---

### Left

The navigation pane draws in and out from Left side. It can be set as shown below:

### XML

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
Position="Left"
DrawerHeaderHeight="0">
<navigationdrawer:SfNavigationDrawer.ContentView>
<StackLayout>
<Grid BackgroundColor="#1aa1d6"
HeightRequest="50"

```

```

VerticalOptions="Start">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerContentView>
<Grid >
<ListView x:Name="listView"
ItemSelected="listView_ItemSelected"
SeparatorColor="Transparent">
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout HeightRequest="40">
<Label Margin="10,7,0,0"
Text="{Binding}"
TextColor="Black"
FontSize="16"/>
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerContentView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>

```

**C#**

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
navigationDrawer.DrawerWidth = 200;
navigationDrawer.Position =
Syncfusion.SfNavigationDrawer.XForms.Position.Left;
hamburgerButton.Image =
(FileImageSource) ImageSource.FromFile("hamburger_icon.png");
List<string> list = new List<string>();
list.Add("Web");
list.Add("Images");
list.Add("Videos");
list.Add("Maps");
}
}

```

```
list.Add("News");
listView.ItemsSource = list;
}
void hamburgerButton_Clicked(object sender, EventArgs e)
{
    navigationDrawer.ToggleDrawer();
}
private void listView_ItemSelected(object sender,
SelectedItemChangedEventArgs e)
{
    //    Your codes here
    navigationDrawer.ToggleDrawer();
}
}
}
```



### Right

The navigation pane draws in and out from Right side. It can be set as shown below:

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigat
ionDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
```

```

Position="Right"
DrawerHeaderHeight="0">
<navigationdrawer:SfNavigationDrawer.ContentView>
<StackLayout>
<Grid BackgroundColor="#1aa1d6"
HeightRequest="50"
VerticalOptions="Start">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerContentView>
<Grid >
<ListView x:Name="listView"
ItemSelected="listView_ItemSelected"
SeparatorColor="Transparent" >
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout HeightRequest="40">
<Label Margin="10,7,0,0"
Text="{Binding}"
TextColor="Black"
FontSize="16"/>
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerContentView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>

```

**C#**

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            navigationDrawer.DrawerWidth = 200;
            navigationDrawer.Position =
            Syncfusion.SfNavigationDrawer.XForms.Position.Right;

```



```

hamburgerButton.Image =
(FileImageSource) ImageSource.FromFile("hamburger_icon.png");
List<string> list = new List<string>();
list.Add("Web");
list.Add("Images");
list.Add("Videos");
list.Add("Maps");
list.Add("News");
listView.ItemsSource = list;
}
void hamburgerButton_Clicked(object sender, EventArgs e)
{
navigationDrawer.ToggleDrawer();
}
private void listView_ItemSelected(object sender,
SelectedItemChangedEventArgs e)
{
//Your codes here
navigationDrawer.ToggleDrawer();
}
}
}

```



[Top](#)

The navigation pane draws in and out from Top side. It can be set as shown below:

#### XML

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

```

```

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
Position="Top"
DrawerFooterHeight="0"
DrawerHeaderHeight="0">
<navigationdrawer:SfNavigationDrawer.ContentView>
<StackLayout>
<Grid BackgroundColor="#1aa1d6"
HeightRequest="50"
VerticalOptions="Start">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerContentView>
<Grid >
<ListView x:Name="listView"
ItemSelected="listView_ItemSelected"
SeparatorColor="Transparent" >
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout HeightRequest="40">
<Label Margin="10,7,0,0"
Text="{Binding}"
TextColor="Black"
FontSize="16"/>
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerContentView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>

```

## C#

```

using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
public partial class MainPage : ContentPage

```

```
{  
public MainPage()  
{  
InitializeComponent();  
navigationDrawer.DrawerWidth = 200;  
navigationDrawer.Position =  
Syncfusion.SfNavigationDrawer.XForms.Position.Top;  
hamburgerButton.Image =  
(FileImageSource) ImageSource.FromFile("hamburger_icon.png");  
List<string> list = new List<string>();  
list.Add("Web");  
list.Add("Images");  
list.Add("Videos");  
list.Add("Maps");  
list.Add("News");  
listView.ItemsSource = list;  
}  
void hamburgerButton_Clicked(object sender, EventArgs e)  
{  
navigationDrawer.ToggleDrawer();  
}  
private void listView_ItemSelected(object sender,  
SelectedItemChangedEventArgs e)  
{  
// Your codes here  
navigationDrawer.ToggleDrawer();  
}  
}  
}
```



## Bottom

The navigation pane draws in and out from Bottom side. It can be set as shown below:

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
Position="Bottom"
DrawerFooterHeight="0"
DrawerHeaderHeight="0">
<navigationdrawer:SfNavigationDrawer.ContentView>
<StackLayout>
<Grid BackgroundColor="#1aa1d6"
HeightRequest="50"
VerticalOptions="Start">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerContentView>
<Grid >
<ListView x:Name="listView"
ItemSelected="listView_ItemSelected"
SeparatorColor="Transparent" >
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout HeightRequest="40">
<Label Margin="10,7,0,0"
Text="{Binding}"
TextColor="Black"
FontSize="16"/>
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerContentView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

### C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            navigationDrawer.DrawerWidth = 200;
            navigationDrawer.Position =
            Syncfusion.SfNavigationDrawer.XForms.Position.Bottom;
            hamburgerButton.Image =
            (FileImageSource)ImageSource.FromFile("hamburger_icon.png");
            List<string> list = new List<string>();
            list.Add("Web");
            list.Add("Images");
            list.Add("Videos");
            list.Add("Maps");
            list.Add("News");
            listView.ItemsSource = list;
        }
        void hamburgerButton_Clicked(object sender, EventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
        private void listView_ItemSelected(object sender,
        SelectedItemChangedEventArgs e)
        {
            //    Your codes here
            navigationDrawer.ToggleDrawer();
        }
    }
}
```



## Side Pane Content

The drawer pane content is viewable only if drawer is in open state. Its content can be set as

*Header Content* *Drawer Content* \* *Footer Content*

**Note:** Header and Footer content are optional but Drawer content is mandatory to allocate space for the drawer.

## Header Content

As the name suggests it is displayed at the top of drawer. `DrawerHeaderView` property is used to set the header content of drawer.

## XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
x:Class="NaviSample.MainPage">
  <navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer">
    <navigationdrawer:SfNavigationDrawer.ContentView>
      <StackLayout>
        <Grid BackgroundColor="#1aa1d6"
HeightRequest="50"
VerticalOptions="Start">
          <Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"/>
```

```
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1a1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1a1d6">
<StackLayout VerticalOptions="Center"
HorizontalOptions="Center">
<Label Text="Header View"/>
</StackLayout>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            navigationDrawer.DrawerWidth = 200;
            hamburgerButton.Image =
                (ImageSource)ImageSource.FromFile("hamburger_icon.png");
        }
        void hamburgerButton_Clicked(object sender, EventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
}
```



### Header Height

Height of the drawer header content can be adjusted using `DrawerHeaderHeight` property.

---

**Note:** DrawerHeaderView will be disabled by setting DrawerHeaderHeight to zero

---

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
DrawerHeaderHeight="50">
<navigationdrawer:SfNavigationDrawer.ContentView>
<StackLayout>
<Grid BackgroundColor="#1aa1d6"
HeightRequest="50"
VerticalOptions="Start">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>
```



```
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1aald6" >
<StackLayout VerticalOptions="Center"
HorizontalOptions="Center">
<Label Text="Header View"/>
</StackLayout>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            navigationDrawer.DrawerHeaderHeight = 50;
            hamburgerButton.Image =
                (FileImageSource) ImageSource.FromFile("hamburger_icon.png");
        }
        void hamburgerButton_Clicked(object sender, EventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
}
```



### Footer Content

As the name suggests it is displayed at the bottom of drawer. `DrawerFooterView` property is used to set the footer content of drawer.

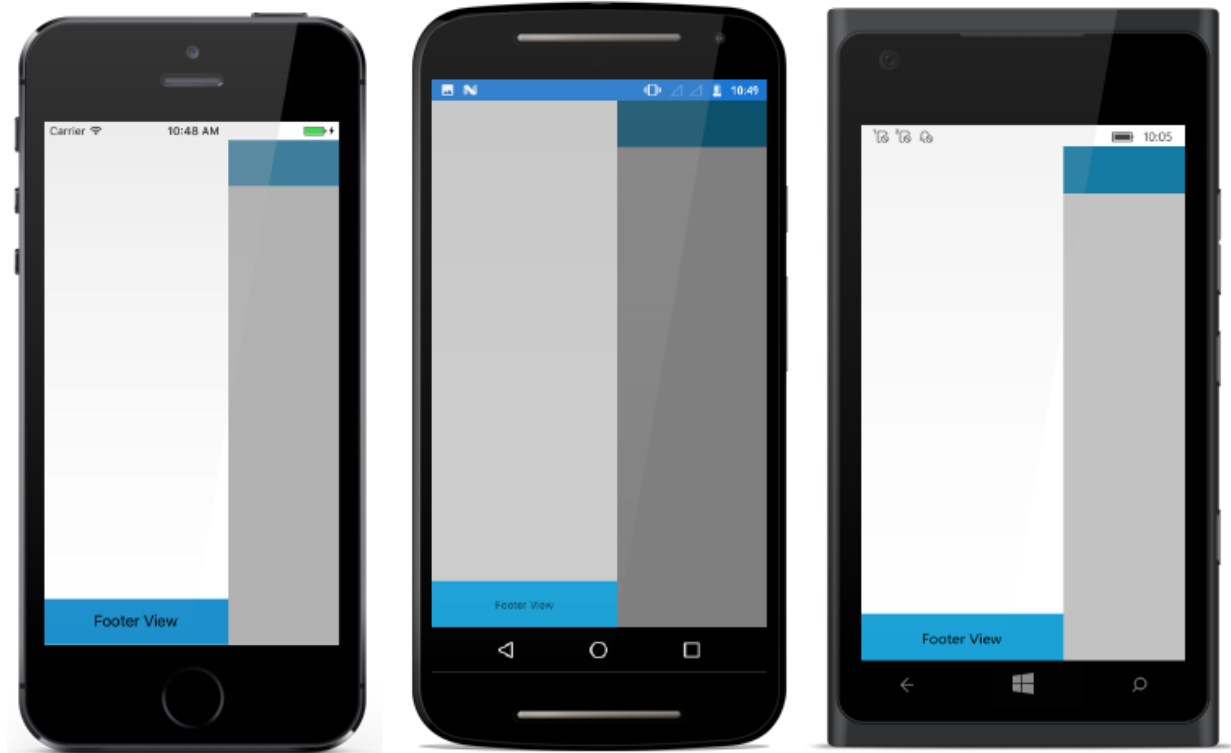
### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
x:Class="NaviSample.MainPage">
  <navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer">
    <navigationdrawer:SfNavigationDrawer.ContentView>
      <StackLayout>
        <Grid BackgroundColor="#1aa1d6"
HeightRequest="50"
VerticalOptions="Start">
          <Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
        </Grid>
      </StackLayout>
    </navigationdrawer:SfNavigationDrawer.ContentView>
  </navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

```
<navigationdrawer:SfNavigationDrawer.DrawerFooterView>
<Grid BackgroundColor="#1aald6" >
<StackLayout VerticalOptions="Center"
HorizontalOptions="Center">
<Label Text="Footer View"/>
</StackLayout>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerFooterView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            navigationDrawer.DrawerFooterHeight = 50;
            hamburgerButton.Image =
                (FileImageSource) ImageSource.FromFile("hamburger_icon.png");
        }
        void hamburgerButton_Clicked(object sender, EventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
}
```



### Footer Height

Height of the drawer footer content can be adjusted using `DrawerFooterHeight` property.

**Note:** `DrawerFooterView` will be disabled by setting `DrawerFooterHeight` to zero

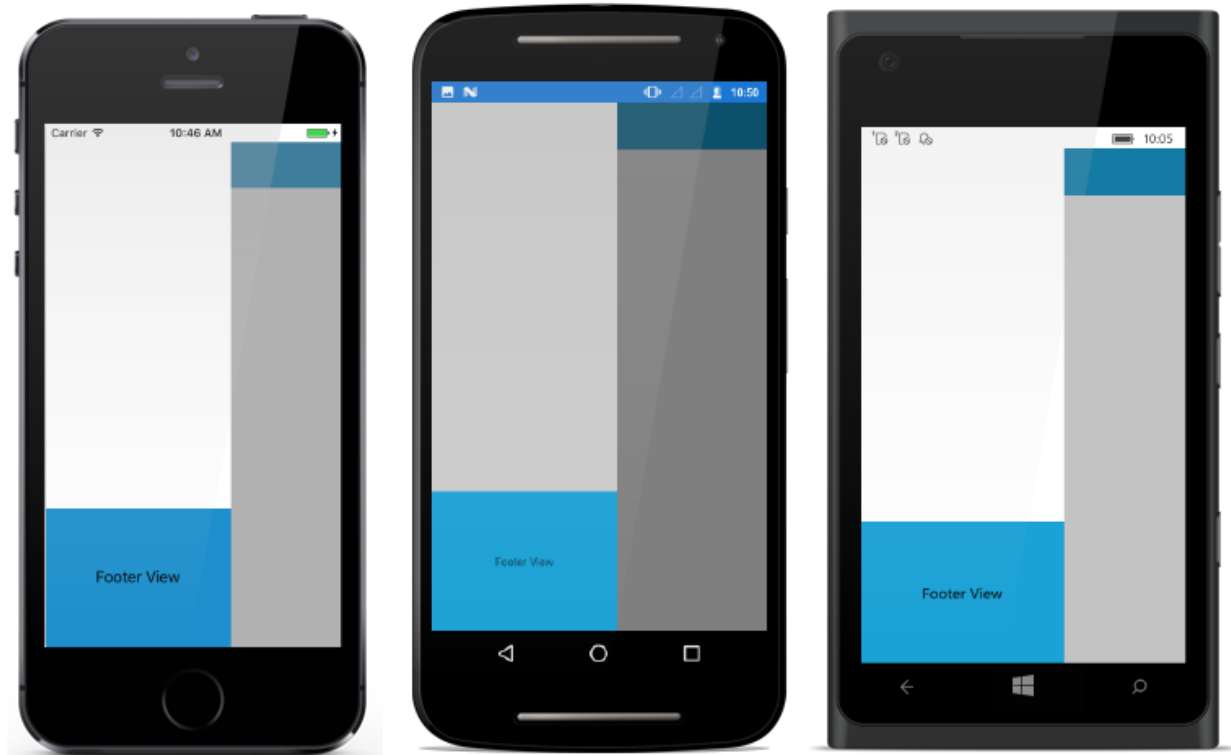
### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
DrawerFooterHeight="50">
<navigationdrawer:SfNavigationDrawer.ContentView>
<StackLayout>
<Grid BackgroundColor="#1aa1d6"
HeightRequest="50"
VerticalOptions="Start">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>
```

```
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerFooterView>
<Grid BackgroundColor="#1aald6" >
<StackLayout VerticalOptions="Center"
HorizontalOptions="Center">
<Label Text="Footer View"/>
</StackLayout>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerFooterView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            navigationDrawer.DrawerFooterHeight = 50;
            hamburgerButton.Image =
                (FileImageSource) ImageSource.FromFile("hamburger_icon.png");
        }
        void hamburgerButton_Clicked(object sender, EventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
}
```



### Main Content

Drawer main content is displayed in between header and footer content. It can be set using `DrawerContentView` property. Content view occupies the remaining space left by header and footer content.

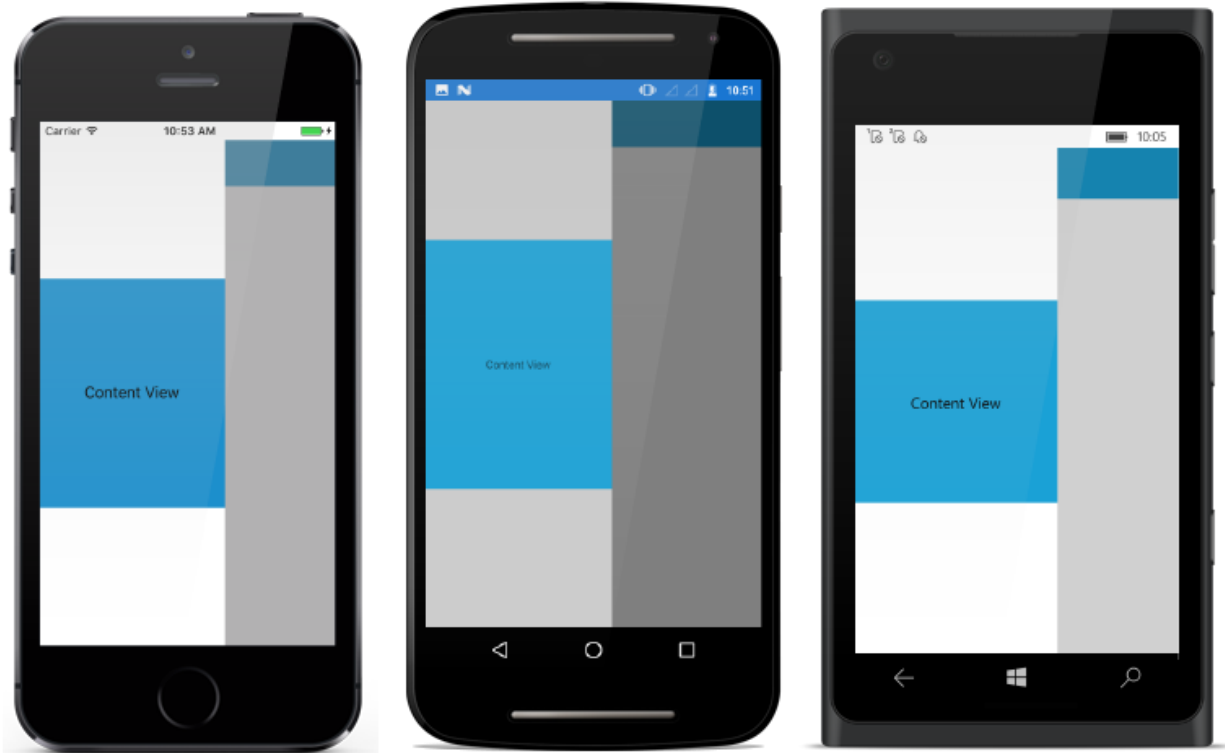
### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:NaviSample"
  xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
  x:Class="NaviSample.MainPage">
  <navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer">
    <navigationdrawer:SfNavigationDrawer.ContentView>
      <StackLayout>
        <Grid BackgroundColor="#1aa1d6"
          HeightRequest="50"
          VerticalOptions="Start">
          <Button x:Name="hamburgerButton"
            HeightRequest="50"
            WidthRequest="50"
            HorizontalOptions="Start"
            FontSize="20"
            BackgroundColor="#1aa1d6"
            Clicked="hamburgerButton_Clicked"/>
        </Grid>
      </StackLayout>
    </navigationdrawer:SfNavigationDrawer.ContentView>
  </navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

```
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerContentView>
<Grid BackgroundColor="#1aald6">
<Label Text="Content View"
VerticalOptions="Center"
HorizontalOptions="Center"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerContentView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            hamburgerButton.Image =
                (FileImageSource) ImageSource.FromFile("hamburger_icon.png");
        }
        void hamburgerButton_Clicked(object sender, EventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
}
```



## Side Pane Sizing

The size of side pane can be adjusted using `DrawerHeight` and `DrawerWidth` properties.

### Drawer Height

`DrawerHeight` property is used to change the height of side pane when the Position is Top or Bottom.

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationView.XForms;assembly=Syncfusion.SfNavi
gationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationView x:Name="navigationDrawer"
DrawerHeight="40"
Position="Top">
<navigationdrawer:SfNavigationView.DrawerHeaderView>
<Label Text="This is a very short content used to demonstrate the
DrawerHeight property "/>
</navigationdrawer:SfNavigationView.DrawerHeaderView>
</navigationdrawer:SfNavigationView>
</ContentPage>
```

### C#

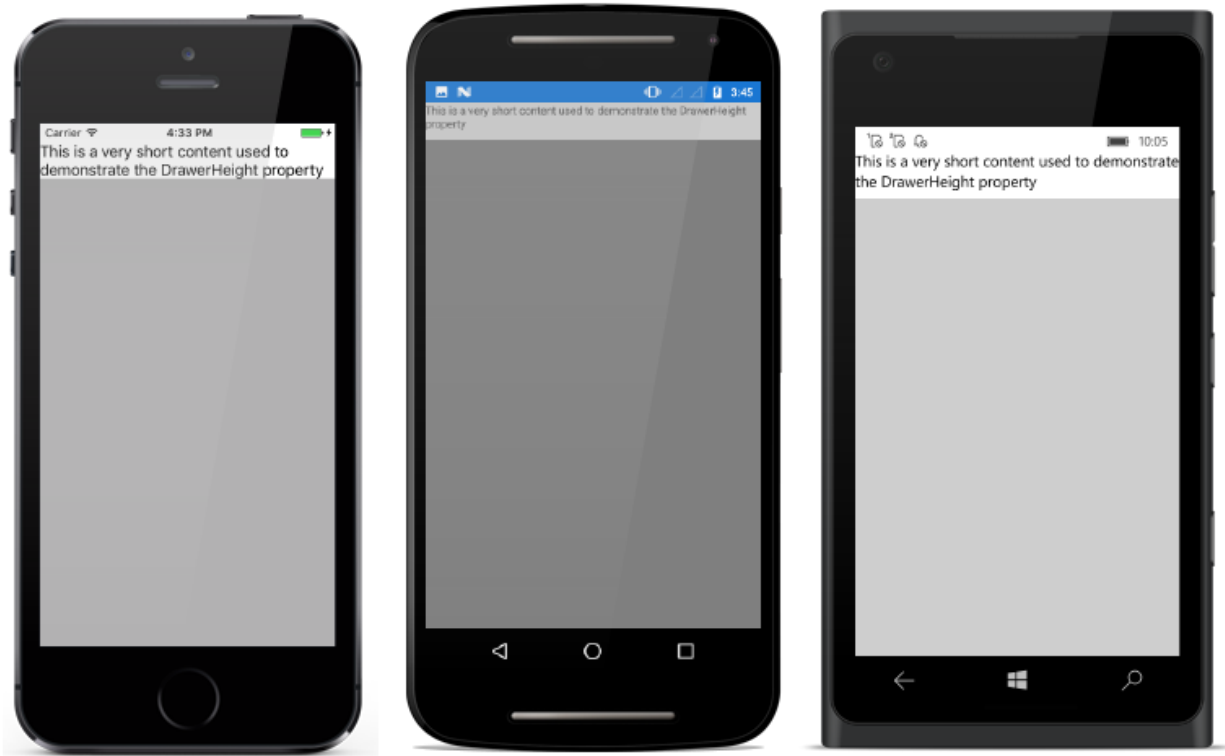
```
Syncfusion.SfNavigationView.XForms.SfNavigationView navigationDrawer =
new Syncfusion.SfNavigationView.XForms.SfNavigationView();
```



```

navigationDrawer.Position = Position.Top;
navigationDrawer.DrawerHeight = 40;
navigationDrawer.DrawerHeaderView = new Label()
{
    Text = "This is a very short content used to demonstrate the DrawerHeight property"
};
Content = navigationDrawer;

```



### Drawer Width

**DrawerWidth** property is used to change the width of side pane when the Position is Left or Right.

### XML

```

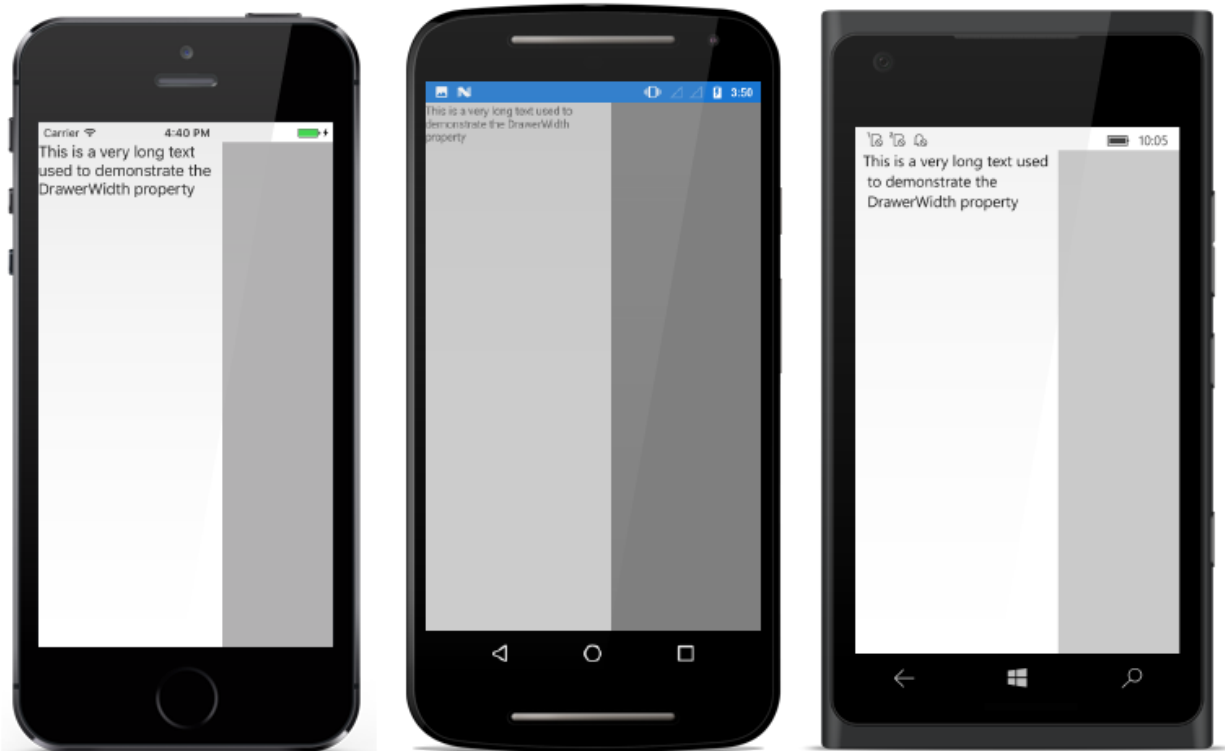
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:NaviSample"
    xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
    x:Class="NaviSample.MainPage">
    <navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
        DrawerWidth="200"
        Position="Left">
        <navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
        <Label Text="This is a very short content used to demonstrate the
        DrawerHeight property" />
        </navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
    </navigationdrawer:SfNavigationDrawer>

```

```
</ContentPage>
```

## C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfNavigationDrawer navigationDrawer = new SfNavigationDrawer()
            {
                Position = Position.Left,
                DrawerWidth = 200,
                DrawerHeaderView = new Label()
                {
                    Text = "This is a very short content used to demonstrate the DrawerHeight property"
                }
            };
            Content = navigationDrawer;
        }
    }
}
```



## Swipe Gesture and Sensitivity

NavigationDrawer supports swipe gesture for opening and closing the drawer.

### Enabling Swipe Gesture

It can be enabled/disabled using `EnableSwipeGesture` property.

#### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigat
ionDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
EnableSwipeGesture="True">
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1aald6"
VerticalOptions="Center"
HorizontalOptions="Center">
<Label Text="Header view"
FontSize="16"
VerticalOptions="Center"
HorizontalOptions="Center"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

#### C#

```
navigationDrawer.EnableSwipeGesture = true;
```

### Swipe Sensitivity

In smaller screens user may find it difficult to swipe the drawer from the edge in such cases we can increase the swipe region using `TouchThreshold` property. It can be set as below:

#### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigat
ionDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
EnableSwipeGesture="True"
TouchThreshold="70">
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1aald6"
VerticalOptions="Center"
```

```

HorizontalOptions="Center">
<Label Text="Header view"
FontSize="16"
VerticalOptions="Center"
HorizontalOptions="Center"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>

```

**C#**

```
navigationDrawer.TouchThreshold = 70;
```

**Toggle Animations**

The drawer toggling animation can be changed using Transition property and it can be set to three different values. They are

- SlideOnTop
- Push
- Reveal

---

**Note:** The default animation is SlideOnTop.

---

**SlideOnTop**

The navigation pane overlays the main content area when opened. It can be set as shown below:

**XML**

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
Transition="SlideOnTop"
DrawerHeaderHeight="50">
<navigationdrawer:SfNavigationDrawer.ContentView>
<StackLayout>
<Grid HeightRequest="50"
VerticalOptions="Start"
BackgroundColor="#1aa1d6">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1aa1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>

```

```
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1aald6"
VerticalOptions="Center"
HorizontalOptions="Center">
<Label Text="Header view"
FontSize="16"
VerticalOptions="Center"
HorizontalOptions="Center"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            navigationDrawer.DrawerHeight = 200;
            navigationDrawer.Transition =
            Syncfusion.SfNavigationDrawer.XForms.Transition.SlideOnTop;
            hamburgerButton.Image =
            (FileImageSource) ImageSource.FromFile("hamburger_icon.png");
        }
        void hamburgerButton_Clicked(object sender, EventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
}
```



### Push

The navigation pane is hidden. It pushes the main content area in opposite side up to drawer width when opened. It can be set as shown below:

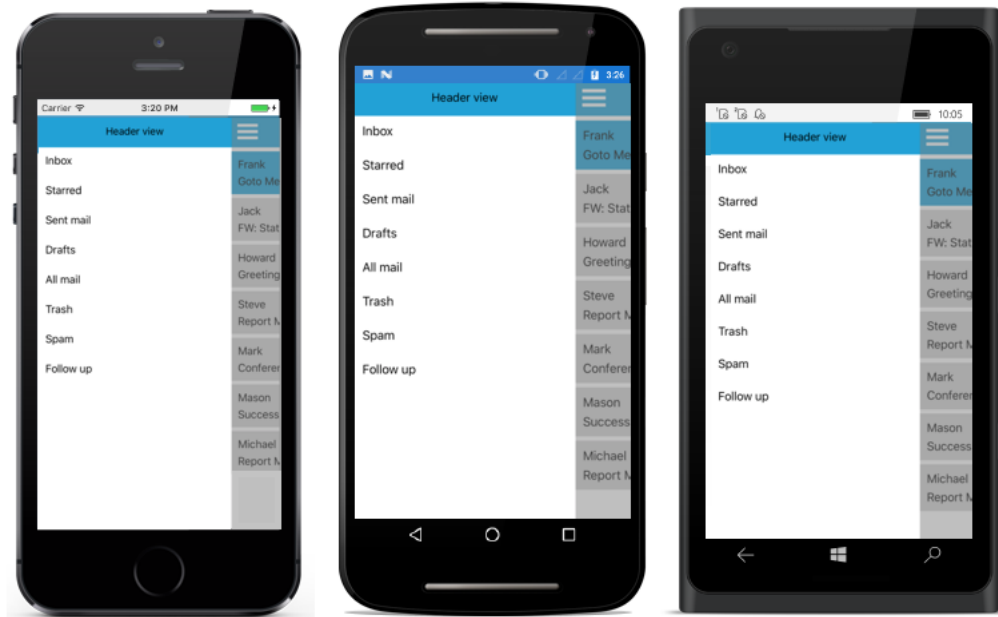
### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
Transition="Push"
DrawerHeaderHeight="50">
<navigationdrawer:SfNavigationDrawer.ContentView>
<StackLayout>
<Grid HeightRequest="50"
VerticalOptions="Start"
BackgroundColor="#1a1d6">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1a1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1a1d6"
VerticalOptions="Center"
```

```
HorizontalOptions="Center">
<Label Text="Header view"
FontSize="16"
VerticalOptions="Center"
HorizontalOptions="Center"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            navigationDrawer.DrawerHeight = 200;
            navigationDrawer.Transition =
            Syncfusion.SfNavigationDrawer.XForms.Transition.Push;
            hamburgerButton.Image =
            (FileImageSource) ImageSource.FromFile("hamburger_icon.png");
        }
        void hamburgerButton_Clicked(object sender, EventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
}
```



### Reveal

The navigation pane is hidden behind the main content. Main content moves away in opposite side up to drawer width to show the drawer content. It can be set as shown below:

### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-
namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigat
ionDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
Transition="Reveal"
DrawerHeaderHeight="50">
<navigationdrawer:SfNavigationDrawer.ContentView>
<StackLayout>
<Grid HeightRequest="50"
VerticalOptions="Start"
BackgroundColor="#1a1d6">
<Button x:Name="hamburgerButton"
HeightRequest="50"
WidthRequest="50"
HorizontalOptions="Start"
FontSize="20"
BackgroundColor="#1a1d6"
Clicked="hamburgerButton_Clicked"/>
</Grid>
</StackLayout>
</navigationdrawer:SfNavigationDrawer.ContentView>
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1a1d6"
VerticalOptions="Center"
```



```
HorizontalOptions="Center">
<Label Text="Header view"
FontSize="16"
VerticalOptions="Center"
HorizontalOptions="Center"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
using System;
using Syncfusion.SfNavigationDrawer.XForms;
using Xamarin.Forms;
namespace NaviSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            navigationDrawer.DrawerHeight = 200;
            navigationDrawer.Transition =
            Syncfusion.SfNavigationDrawer.XForms.Transition.Reveal;
            hamburgerButton.Image =
            (FileImageSource) ImageSource.FromFile("hamburger_icon.png");
        }
        void hamburgerButton_Clicked(object sender, EventArgs e)
        {
            navigationDrawer.ToggleDrawer();
        }
    }
}
```



## Toggling Drawer

Drawer can be toggled using

*IsOpen* property ToggleDrawer method \* Swipe gesture

## Opening Drawer Programmatically

**IsOpen** property and ToggleDrawer method can be used to open or close the drawer programmatically.

## XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:NaviSample"
xmlns:navigationdrawer="clr-namespace:Syncfusion.SfNavigationDrawer.XForms;assembly=Syncfusion.SfNavigationDrawer.XForms"
x:Class="NaviSample.MainPage">
<navigationdrawer:SfNavigationDrawer x:Name="navigationDrawer"
IsOpen="True">
<navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
<Grid BackgroundColor="#1aald6"
VerticalOptions="Center"
HorizontalOptions="Center">
<Label Text="Header view"
FontSize="16"
VerticalOptions="Center"
HorizontalOptions="Center"/>
</Grid>
</navigationdrawer:SfNavigationDrawer.DrawerHeaderView>
</navigationdrawer:SfNavigationDrawer>
</ContentPage>
```

## C#

```
navigationDrawer.IsOpen = true;
```

Using **ToggleDrawer** method,

### C#

```
navigationDrawer.ToggleDrawer();
```

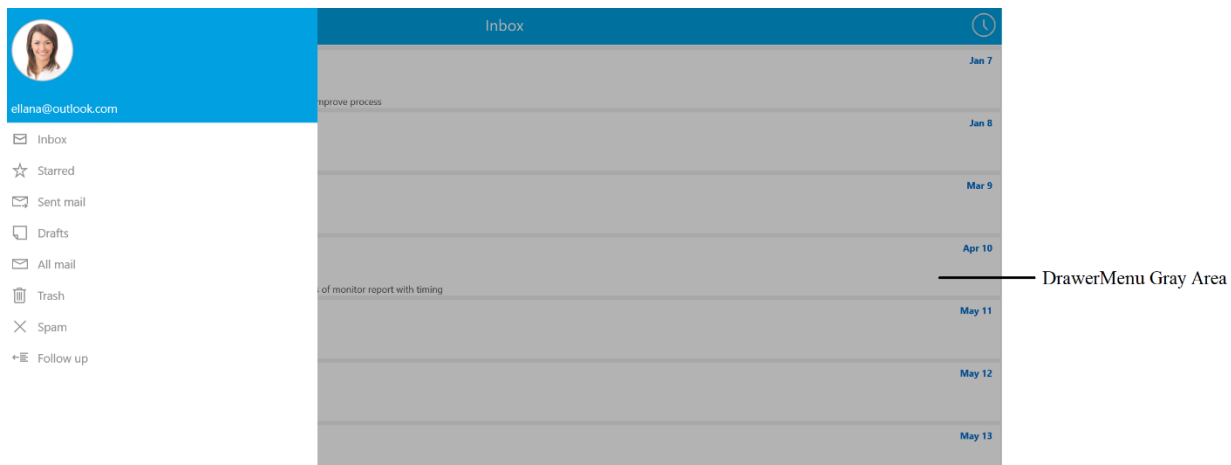
Toggling drawer through swipe gesture is explained in **Swipe Gesture and Sensitivity** section.

### AutomationId

The SfNavigationDrawer control has built-in **AutomationId** for inner elements. The **AutomationId** API allows the automation framework to find and interact with the inner elements of the **NavigationDrawer** control. To keep unique AutomationId, these inner elements' AutomationIds are updated based on the control's **AutomationId**.

For example, if you set SfNavigationDrawer's **AutomationId** as "DrawerMenu", then the automation framework will interact with the gray area as "DrawerMenu Gray Area".

The following screenshot illustrates the AutomationIds of inner elements. You can also interact with the elements inside the ContentView with the element's AutomationId.

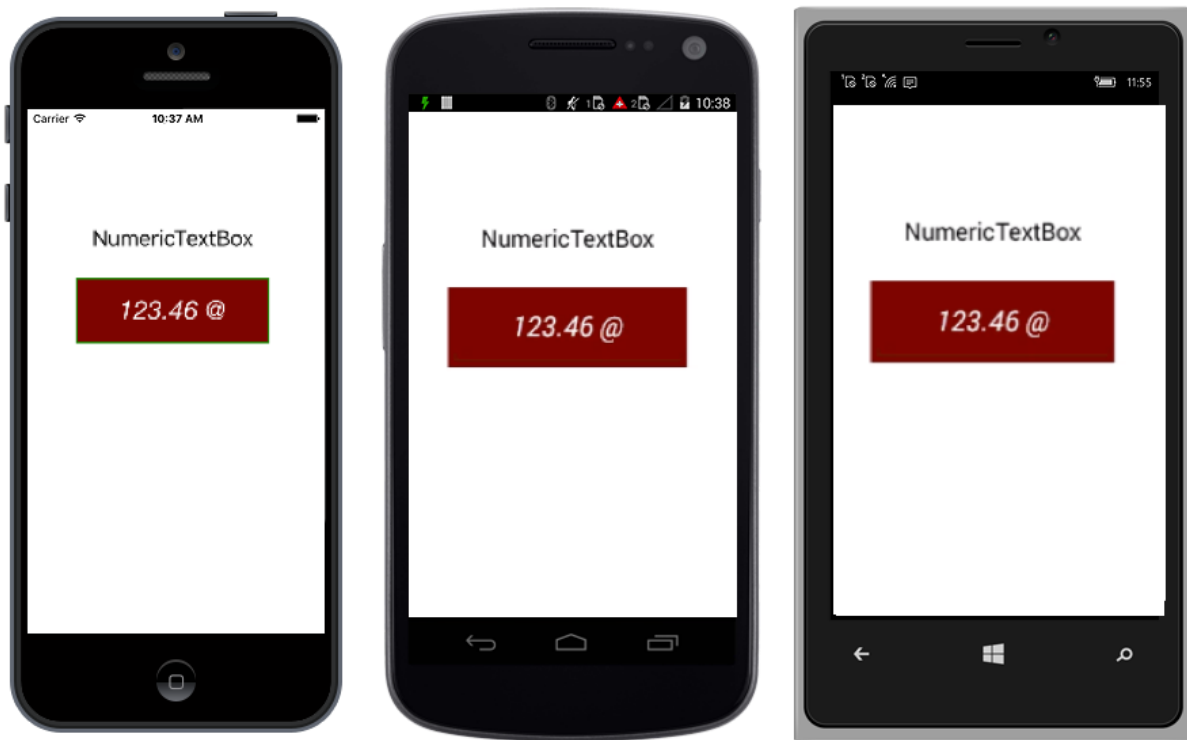


## SfNumericTextBox

### Overview

The Essential Xamarin.Forms NumericTextBox is an advanced version of the Entry control that restricts input to numeric values.

Also provides, a gesture friendly UI culture and can be configured to display different formats like currency format, scientific format etc.



## Key Features

- **FormatString** - Input string can be formatted by using the format strings.
- **Culture** - Number format can be localized to any specific culture.
- **AllowNull** - Allows the user to set null value.
- **ParserMode** - Value gets parsed based on parser mode.

## Getting Started

This section explains you the steps to configure a SfNumericTextBox control in a real-time scenario and also provides a walk-through on some of the customization features available in SfNumericTextBox control.

### Adding SfNumericTextBox reference

You can add SfNumericTextBox reference using one of the following methods:

#### Method 1: Adding SfNumericTextBox reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfNumericTextBox). To add SfNumericTextBox to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfNumericTextBox](https://www.nuget.org/packages/Syncfusion.Xamarin.SfNumericTextBox), and then install it.

![Adding SfNumericTextBox reference from NuGet](images/Adding SfNumericTextBox reference.png)

---

**Note:** Install the same version of SfNumericTextBox NuGet in all the projects.

---

#### Method 2: Adding SfNumericTextBox reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfNumericTextBox control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfNumericTextBox assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfNumericTextBox.Android.dll Syncfusion.SfNumericTextBox.XForms.Android.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfNumericTextBox.iOS.dll Syncfusion.SfNumericTextBox.XForms.iOS.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfInput.UWP.dll Syncfusion.SfShared.UWP.dll Syncfusion.SfNumericTextBox.XForms.UWP.dll Syncfusion.SfNumericTextBox.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the SfNumericTextBox on each platform

To use SfNumericTextBox inside an application, each platform application must initialize the SfNumericTextBox renderer. This initialization step varies from platform to platform and is discussed in the following sections.

**Note:** If you are adding the references from toolbox, below steps are not needed.

### *Android and UWP*

The Android and UWP launches the SfNumericTextBox without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application

### *iOS*

To launch SfNumericTextBox in iOS, need to create an instance of SfNumericTextBoxRenderer in FinishedLaunching overridden method of AppDelegate class in iOS Project as shown below.

### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    new SfNumericTextBoxRenderer();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

### *ReleaseMode issue in UWP platform*

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in Release Mode.

The above problem can be resolved by initializing the SfNumericTextBox assemblies in Main.xaml.cs in UWP project as like in below code snippet.

### **C#**

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfNumericTextBoxRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Create a Simple SfNumericTextBox

The SfNumericTextBox control is configured entirely in C# code or by using XAML markup. The following steps explain on how to create a SfNumericTextBox and configure its elements,

- Adding namespace for the added assemblies.

### **XML**

```
xmlns:syncfusion="clr-namespace:Syncfusion.SfNumericTextBox.XForms;assembly=Syncfusion.SfNumericTextBox.XForms"
```

## C#

```
using Syncfusion.SfNumericTextBox.XForms;
```

- Now add the SfNumericTextBox control with a required optimal name by using the included namespace.

## XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:GettingStarted"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfNumericTextBox.XForms;assembly=Syncfusion.SfNumericTe
xtBox.XForms"
x:Class="GettingStarted.NumericControlPage">
<ContentPage.Content>
<syncfusion:SfNumericTextBox x:Name="numericTextBox" />
</ContentPage.Content>
</ContentPage>
```

## C#

```
using Syncfusion.SfNumericTextBox.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class NumericControlPage : ContentPage
    {
        public NumericControlPage()
        {
            InitializeComponent();
            SfNumericTextBox numericTextBox = new SfNumericTextBox();
            this.Content = numericTextBox;
        }
    }
}
```

## Display Customization

### *Setting and Reading Value*

**Value** property is used to set and read the value presented by the SfNumericTextBox.

## XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="123.45" />
```

## C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.Value = 123.45;
this.Content = numericTextBox;
```



## Parsing the Value

Value of the SfNumericTextBox gets parsed based on **ParserMode** property. ParsingMode is of type Parsers which is enum of Double and Decimal. Hence we have option to display the value in double or decimal.

Following code shows the Decimal parsing mode which can be set through **ParserMode** property.

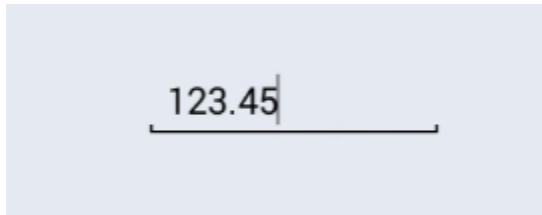
### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="123.45"
ParserMode="Decimal" />
```

### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.ParserMode=Parsers.Decimal;
numericTextBox.Value = 123.45;
this.Content = numericTextBox;
```

**Note:** The default Value for **ParserMode** is Double.



## Colors

SfNumericTextBox can be set to use a custom background, text and border colors via the following bindable properties:

- **TextColor** - sets the color of the NumericTextBox's value.
- **BackgroundColor** - sets the background color of NumericTextBox's frame.
- **BorderColor** - sets the border custom color of NumericTextBox
- **WatermarkColor** - sets the watermark custom color of NumericTextBox's watermark Text.

### TextColor

To set the TextColor color in XAML as well as in C#:

### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="123"
TextColor="Green" />
```



**C#**

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();  
numericTextBox.TextColor = Color.Green;  
numericTextBox.Value = 123;  
this.Content = numericTextBox;
```

*BackgroundColor*

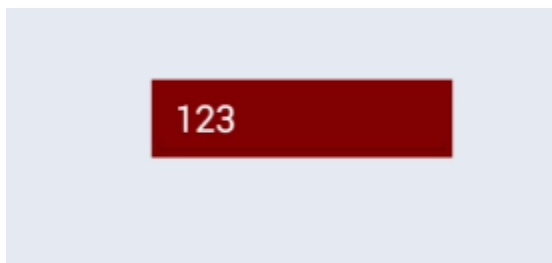
To set the BackgroundColor color in XAML as well as in C#:

**XML**

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="123"  
BackgroundColor="Maroon" TextColor="White"/>
```

**C#**

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();  
numericTextBox.BackgroundColor = Color.Maroon;  
numericTextBox.TextColor = Color.White;  
numericTextBox.Value = 123;  
this.Content = numericTextBox;
```

*BorderColor*

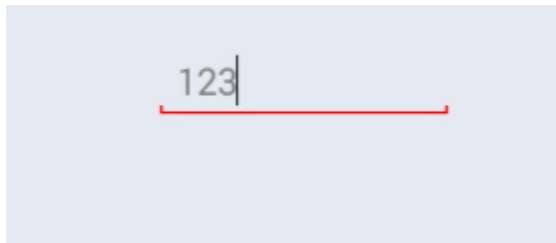
To set the BorderColor color in XAML as well as in C#:

**XML**

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="123"  
BorderColor="Red" />
```

**C#**

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();  
numericTextBox.BorderColor = Color.Red;  
numericTextBox.Value = 123;  
this.Content = numericTextBox;
```



### WatermarkColor

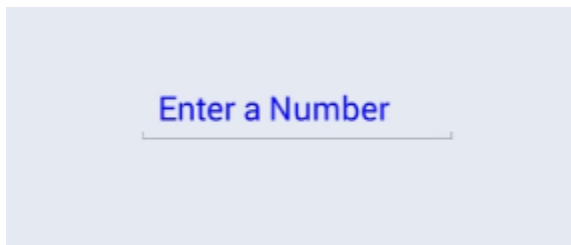
To set the WatermarkColor color in XAML as well as in C#:

#### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" AllowNull="true"
WatermarkColor="Blue" Watermark="Enter a Number"/>
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.AllowNull=true;
numericTextBox.WatermarkColor = Color.Blue;
numericTextBox.Watermark = "Enter a Number"
this.Content = numericTextBox;
```



### Number Formatting

The Values of the SfNumericTextBox can be configured to display different formats like currency format, percent format etc.

#### Format String

The **FormatString** property determines the format specifier by which the display text has to be formatted.

---

**Note:** The control displays the formatted text on lost focus. Default Value of **FormatString** is "n".

---

#### Display Currency Notation

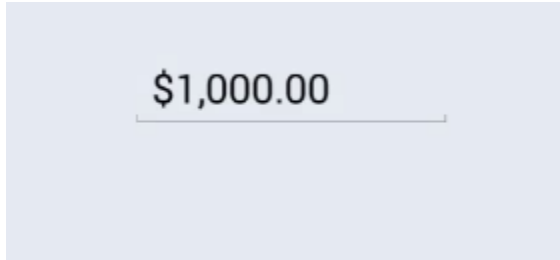
c - Displays the value with currency notation.

#### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="1000"
FormatString="c" />
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();  
numericTextBox.FormatString="c";  
numericTextBox.Value=1000;  
this.content=numericTextBox;
```



#### *Display Number Notation*

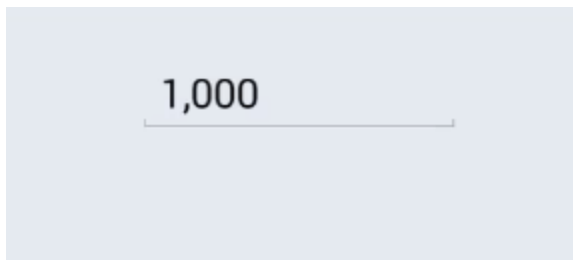
**n** – Displays the value in number format.

#### **XML**

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="1000"  
FormatString="n" />
```

#### **C#**

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();  
numericTextBox.FormatString="n";  
numericTextBox.Value=1000;  
this.content=numericTextBox;
```



#### *Display Percentage Notation*

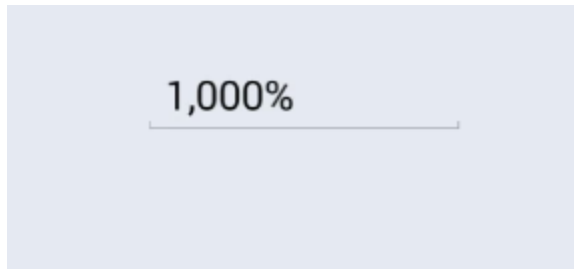
**p** – Displays the value in percentage.

#### **XML**

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="1000"  
FormatString="p" />
```

#### **C#**

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();  
numericTextBox.FormatString="p";  
numericTextBox.Value=1000;  
this.content=numericTextBox;
```



**Note:** Instead of using above `FormatString` types, we can provide any symbol or value as string in `FormatString` property which will be appended with the value in `SfNumericTextBox`.

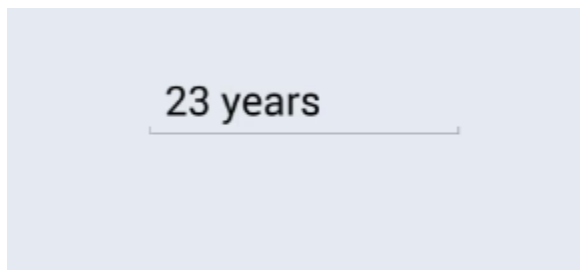
By passing any string , we can get the same as appended with the value of `NumericTextBox`

#### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="1000"
FormatString="years" />
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.FormatString="years";
numericTextBox.Value=23;
this.content=numericTextBox;
```



#### Compute to Percentage

When the `NumericTextBox` is in percentage format, the value can be displayed in two ways as follows

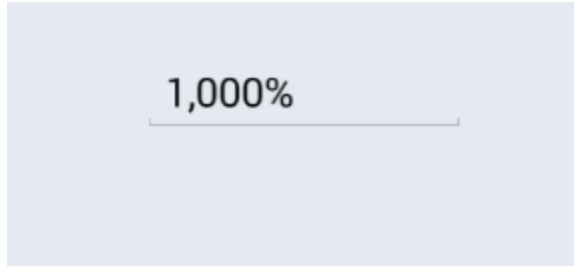
- **Value:** Displays the actual value with percentage symbol.

#### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" FormatString="p"
Value="1000" PercentDisplayMode="Value" />
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.PercentDisplayMode=PercentDisplayMode.Value;
numericTextBox.FormatString="p";
numericTextBox.Value=1000;
this.content=numericTextBox;
```



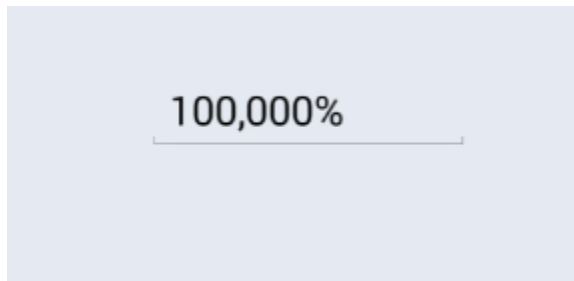
- **Compute**: Displays the computed value with percentage symbol.

#### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" FormatString="p"
Value="1000" PercentDisplayMode="Compute" />
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.PercentDisplayMode=PercentDisplayMode.Compute;
numericTextBox.FormatString="p";
numericTextBox.Value=1000;
this.Content=numericTextBox;
```



#### Set EnableGroupSeparator

**EnableGroupSeparator** property is used to get rid of the comma in the Value of SfNumericTextBox.

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.Value=12345;
numericTextBox.EnableGroupSeparator = true;
this.Content = numericTextBox;
```



#### Font Settings

NumericTextBox has the following two font-related properties that display the value's text:

We can customize the font style of NumericTextBox by using the following properties.

- **FontSize** : Sets the font size for NumericTextBox's text.

- **FontAttributes** : Sets the style of NumericTextBox's text. We can give three types of style on it. It is specifying style information like Italic and Bold (using the FontAttributes enumeration in C#)
  1. Bold- The font is bold
  2. Italic – The font is Italic
  3. None – The font is unmodified.

---

**Note:** Default value is None.

---

- **FontFamily** : Customizes the font family of the NumericTextBox's text.
- **TextAlignment** : Sets the style of NumericTextBox's text. We can give three types of style on it. It is specifying style information like Start, End and Center (using the TextAlignment enumeration in C#)

---

**Note:** Default value is Start.

---

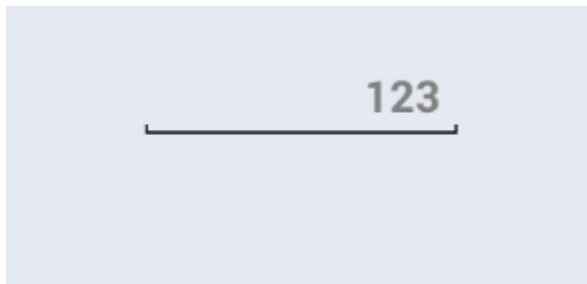
To set the font size and attributes in XAML as well as in C#:

#### XAML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" FontSize="27"
FontAttributes="Bold" Value="123" TextAlignment="End" />
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.FontSize = 27;
numericTextBox.Value = 123;
numericTextBox.TextAlignment=TextAlignment.End;
numericTextBox.FontAttributes = FontAttributes.Bold;
this.Content = numericTextBox;
```



#### Localization

The SfNumericTextBox value can be localized to any specific culture. It can be specified by setting the **Culture** property with **System.Globalization.CultureInfo** object instance.

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.Value=123.45;
numericTextBox.Culture = new System.Globalization.CultureInfo("fr-FR");
this.Content = numericTextBox;
```



Currency Format – French Culture

### Change Localization of Return key text

The SfNumericTextBox provides the Localization support for the Return Key in soft keypad of iOS. We have provided the knowledge base document for the same. Please refer the Knowledge Base document from this [link](#).

### Set Maximum Number of Decimal Digits

The maximum number of digits to be displayed after the decimal point can be specified by using `MaximumNumberDecimalDigits` property.

**Note:** The `MaximumNumberDecimalDigits` property can be provided with positive value only.

#### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Value="123.456"
MaximumNumberDecimalDigits="2" />
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.Value = 123.456;
numericTextBox.MaximumNumberDecimalDigits=2;
this.Content = numericTextBox;
```

123.45

### Assign Nullable Value

The null values can be set in SfNumericTextBox `Value` property, by setting `AllowNull` property value to true.

**Note:** By default, the property value is false.

#### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" AllowNull="true" />
```

#### C#

```
SfNumericTextBox numericTextBox = new SfNumericTextBox();
numericTextBox.AllowNull=true;
this.Content = numericTextBox;
```



### Set Hint Text

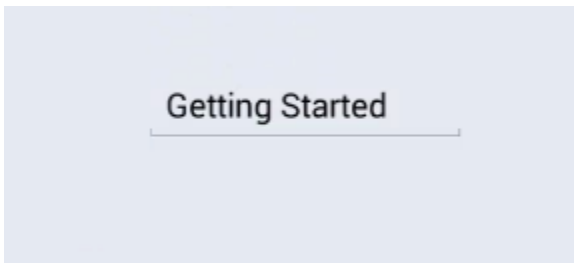
The **WaterMark** property can be used to provide a hint that helps the user to get started with their input. The watermark text is visible when value is empty or null.

#### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" AllowNull = "true"
Watermark="Getting Started" />
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.Watermark = "Getting Started";
numericTextBox.AllowNull = true;
this.Content=numericTextBox;
```



For customizing the color of NumericTextBox's Watermark [refer](#)

## Events and Interactivity

### Events

#### ValueChanged

We can perform operation while the changing the value of NumericTextBox's Value using ValueChanged event. ValueChanged event returns the changed value in NumericTextBox.

Members	Description
Value	Displays changed value in NumericTextBox

#### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox"
ValueChanged="Handle_ValueChanged" Value="123" />
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
numericTextBox.Value = 123;
numericTextBox.ValueChanged += Handle_ValueChanged;
this.Content=numericTextBox;
void Handle_ValueChanged(object sender,
Syncfusion.SfNumericTextBox.XForms.ValueEventArgs e)
```



```
{  
    System.Diagnostics.Debug.WriteLine(e.Value.ToString());  
}
```

### Interactivity : ValueChangeMode

The ValueChangeMode property is used to mention when the validation need to take place, either in key pressed or in focus lost. When ValueChangeMode is set to OnKeyFocus, the validation will be carried out for each key press. When ValueChangeMode is OnLostFocus, the validation occur when the control lost the focus or the focus move to next control. ValueChangeMode includes the following options:

1. OnKeyFocus
2. OnLostFocus

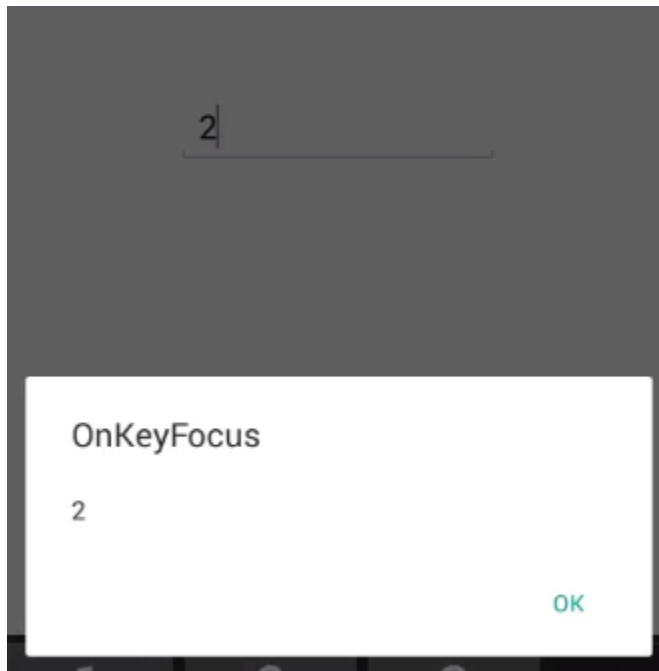
### OnKeyFocus

#### XML

```
<syncfusion:SfNumericTextBox ValueChangeMode="OnKeyFocus"  
x:Name="numericTextBox" Value="123" />
```

#### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();  
numericTextBox.Value = 123;  
numericTextBox.ValueChangeMode = ValueChangeMode.OnKeyFocus;  
this.Content=numericTextBox;
```



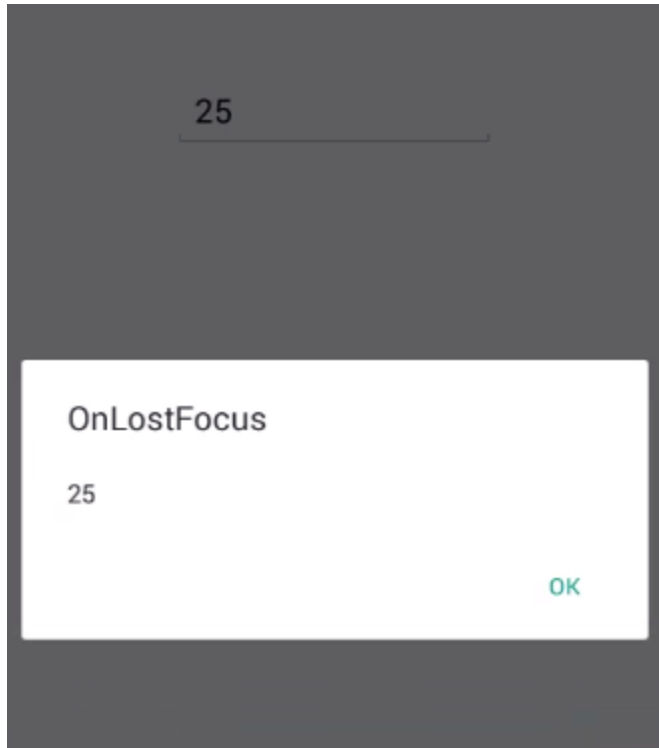
### OnLostFocus

#### XML

```
<syncfusion:SfNumericTextBox ValueChangeMode="OnLostFocus"  
x:Name="numericTextBox" Value="123" />
```

### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();  
numericTextBox.Value = 123;  
numericTextBox.ValueChangeMode = ValueChangeMode.OnLostFocus;  
this.Content=numericTextBox;
```



### Completed

Raised when the user finalizes the text in the NumericTextBox by pressing return key on the keyboard.

### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox"  
Completed="Handle_Completed"/>
```

### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();  
numericTextBox.Completed += Handle_Completed;  
this.Content=numericTextBox;  
void Handle_Completed(object sender, System.EventArgs e)  
{  
    System.Diagnostics.Debug.WriteLine("Completed");  
}
```

## Range Support

Restrict the Values within a specific range by setting the Maximum and Minimum property values.

### XML

```
<syncfusion:SfNumericTextBox x:Name="numericTextBox" Maximum="1000"  
Minimum="50" Value="10"/>
```

### C#

```
StackLayout stack = new StackLayout();  
SfNumericTextBox numericTextBox = new SfNumericTextBox();  
numericTextBox.Minimum = 50;  
numericTextBox.Maximum = 1000;  
numericTextBox.Value = 10;  
stack.Children.Add(numericTextBox);  
this.Content = stack;
```

**Note:** Default Value of Maximum and Minimum is "null".



## Set SelectAllOnFocus

SelectAllOnFocus the property is used to specify whether the text should be selected when the control gets focus.

### XML

```
<numeric:SfNumericTextBox SelectAllOnFocus="True" Value="12345"/>
```

### C#

```
SfNumericTextBox numericTextBox=new SfNumericTextBox();
```

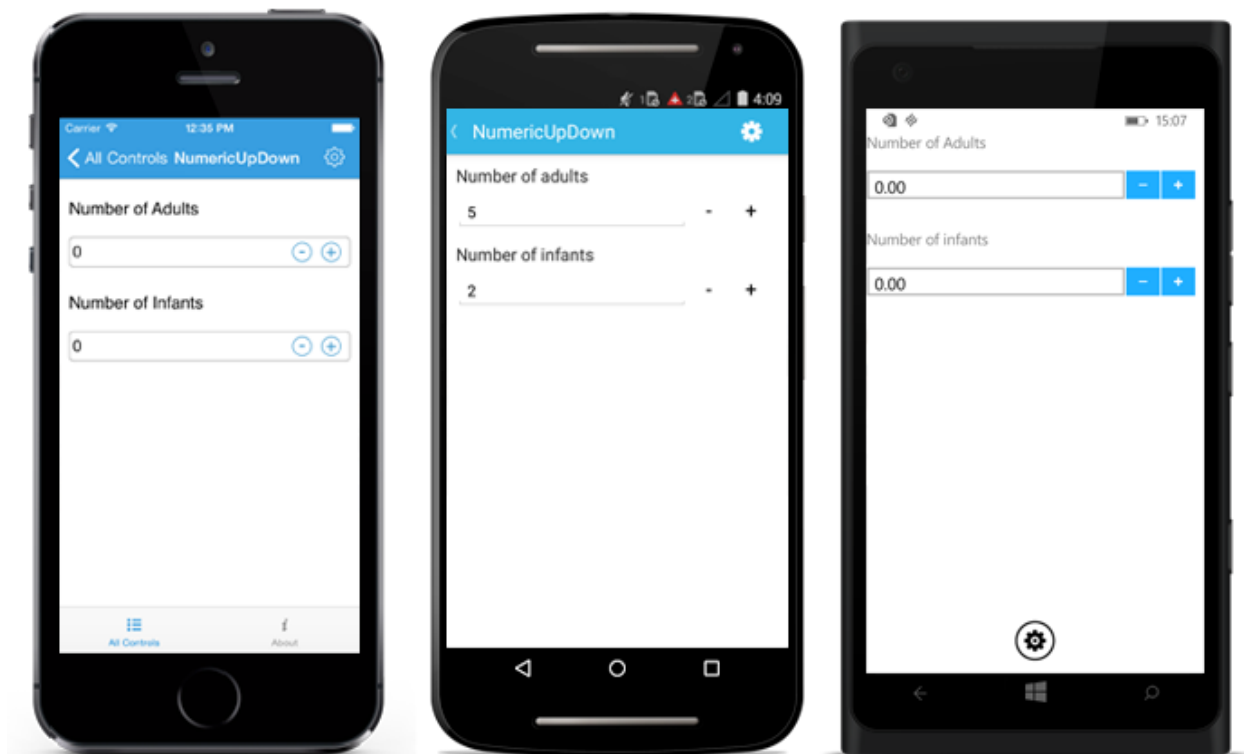
```
numericTextBox.Value=12345;  
numericTextBox.SelectAllOnFocus = true;  
this.Content = numericTextBox;
```



## SfNumericUpDown

### Overview

The Essential Xamarin.Forms NumericUpDown control provides up and down repeat buttons to increase and decrease values. The control respects UI culture and can be configured to display different formats like currency, scientific, etc.



### Key Features

- **FormatString** - Input string can be formatted by using the format strings.
- **Culture** - Number format can be localized to any specific culture.
- **AllowNull** - Allows the user to set null value.
- **ParserMode** - Value gets parsed based on parser mode.

### Getting Started

This section provides overview for working with Essential SfNumericUpDown for Xamarin.Forms. You can walk through the entire process of creating a SfNumericUpDown.

### Adding SfNumericUpDown reference

You can add SfNumericUpDown reference using one of the following methods:

#### Method 1: Adding SfNumericUpDown reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfNumericUpDown). To add SfNumericUpDown to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfNumericUpDown](https://www.nuget.org/packages/Syncfusion.Xamarin.SfNumericUpDown), and then install it.

![Adding SfNumericUpDown reference from NuGet](images/Adding SfNumericUpDown reference.png)

**Note:** Install the same version of SfNumericUpDown NuGet in all the projects.

#### Method 2: Adding SfNumericUpDown reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfNumericUpDown control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfNumericUpDown assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfNumericUpDown.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfNumericUpDown.XForms.Android.dll Syncfusion.SfNumericUpDown.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfNumericUpDown.XForms.iOS.dll Syncfusion.SfNumericUpDown.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfInput.UWP.dll Syncfusion.SfShared.UWP.dll Syncfusion.SfNumericUpDown.XForms.UWP.dll Syncfusion.SfNumericUpDown.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license](#)

[key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the SfNumericUpDown on each platform

To use SfNumericUpDown inside an application, each platform application must initialize the SfNumericUpDown renderer. This initialization step varies from platform to platform and is discussed in the following sections.

**Note:** If you are adding the references from toolbox, below steps are not needed.

#### Android and UWP

The Android and UWP launches the SfNumericUpDown without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

#### iOS

To launch SfNumericUpDown in iOS, need to create an instance of SfNumericUpDownRenderer in FinishedLaunching overridden method of AppDelegate class in iOS Project as shown below.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    SfNumericUpDownRenderer.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### ReleaseMode issue in UWP platform

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the SfNumericUpDown assemblies in **App.xaml.cs** in UWP project as like in below code snippet.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfNumericUpDownRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Create a Simple SfNumericUpDown

The SfNumericUpDown control configured entirely in C# code or by using XAML markup. The following steps explains how to create a SfNumericUpDown and configure its elements.

- Adding namespace for the added assemblies.

#### XML

```
<xmlns:numeric="clr-namespace:Syncfusion.SfNumericUpDown.XForms;assembly=Syncfusion.SfNumericUpDown.XForms"/>
```

#### C#

```
using Syncfusion.SfNumericUpDown.XForms;
```

- Now add the SfNumericUpDown control with a required optimal name by using the included namespace.

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown"/>
```

#### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
this.Content = numericUpDown;
```

### Set Value

The SfNumericUpDown control display value can be set using Value property.

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" Value="5"/>
```

#### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
numericUpDown.Value= 5;  
this.Content = numericUpDown;
```

### Parsing the Value

The value of NumericUpDown is parsed based on the ParsingMode property. ParsingMode is a type of Parsers; it can be enum of Double and Decimal. You have options to update the value in double or decimal.

The following code demonstrates the decimal parsing mode, which can be set using the ParsingMode property.

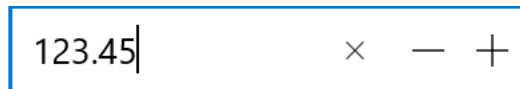
#### XML

```
<numeric:SfNumericUpDown x:Name="NumericUpDown" Value="123.45"
ParsingMode="Decimal" />
```

**C#**

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();
NumericUpDown.ParsingMode=Parsers.Decimal;
NumericUpDown.Value = 123.45;
this.Content = NumericUpDown;
```

**Note:** The default value of ParsingMode is Double.

**Colors**

SfNumericUpDown is used to set custom background, text, and border colors through the following bindable properties:

- **TextColor** - Sets the color of NumericUpDown's value
- **BackgroundColor** - Sets the background color of NumericUpDown.
- **BorderColor** - Sets the border custom color of NumericUpDown
- **WatermarkColor** - Sets the watermark custom color of NumericUpDown's watermark Text.

**TextColor**

The following code sample demonstrates how to set the TextColor in XAML and in C#:

**XML**

```
<numeric:SfNumericUpDown x:Name="NumericUpDown" Value="123"
TextColor="Green" />
```

**C#**

```
SfNumericUpDown NumericUpDown = new SfNumericUpDown();
NumericUpDown.TextColor = Color.Green;
NumericUpDown.Value = 123;
this.Content = NumericUpDown;
```

**BackgroundColor**

The following code sample demonstrates how to set the BackgroundColor in XAML and in C#:

**XML**

```
<numeric:SfNumericUpDown x:Name="NumericUpDown" Value="123"
BackgroundColor="Maroon" TextColor="White"/>
```



**C#**

```
SfNumericUpDown NumericUpDown = new SfNumericUpDown();  
NumericUpDown.BackgroundColor = Color.Maroon;  
NumericUpDown.TextColor = Color.White;  
NumericUpDown.Value = 123;  
this.Content = NumericUpDown;
```

*BorderColor*

The following code sample demonstrates how to set the BorderColor color in XAML and in C#:

**XML**

```
<numeric:SfNumericUpDown x:Name="NumericUpDown" Value="123"  
BorderColor="Red" />
```

**C#**

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();  
NumericUpDown.BorderColor = Color.Red;  
NumericUpDown.Value = 123;  
this.Content = NumericUpDown;
```

*WatermarkColor*

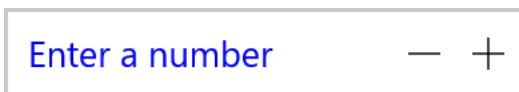
The following code sample demonstrates how to set the WatermarkColor in XAML and in C#:

**XML**

```
<numeric:SfNumericUpDown x:Name="NumericUpDown" AllowNull="true"  
WatermarkColor="Blue" />
```

**C#**

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();  
NumericUpDown.WatermarkColor = Color.Blue;  
NumericUpDown.AllowNull=true;  
this.Content = NumericUpDown;
```

*Number Formatting*

The Values of the SfNumericUpDown can be configured to display different formats like currency format, percent format etc.

### Format String

The `FormatString` property determines the format specifier by which the display text has to be formatted.

---

**Note:** The control displays the formatted text on lost focus. Default Value of `FormatString` is "n".

---

#### Display Currency Notation

**c** - Displays the value with currency notation.

#### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
numericUpDown.FormatString="c";
```

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" FormatString="c"/>
```

#### Display Percentage Notation

**p** – Displays the value in percentage.

#### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
numericUpDown.FormatString="p";
```

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" FormatString="p"/>
```

#### Display Number Notation

**n** – Displays the value in number format.

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" FormatString="n"/>
```

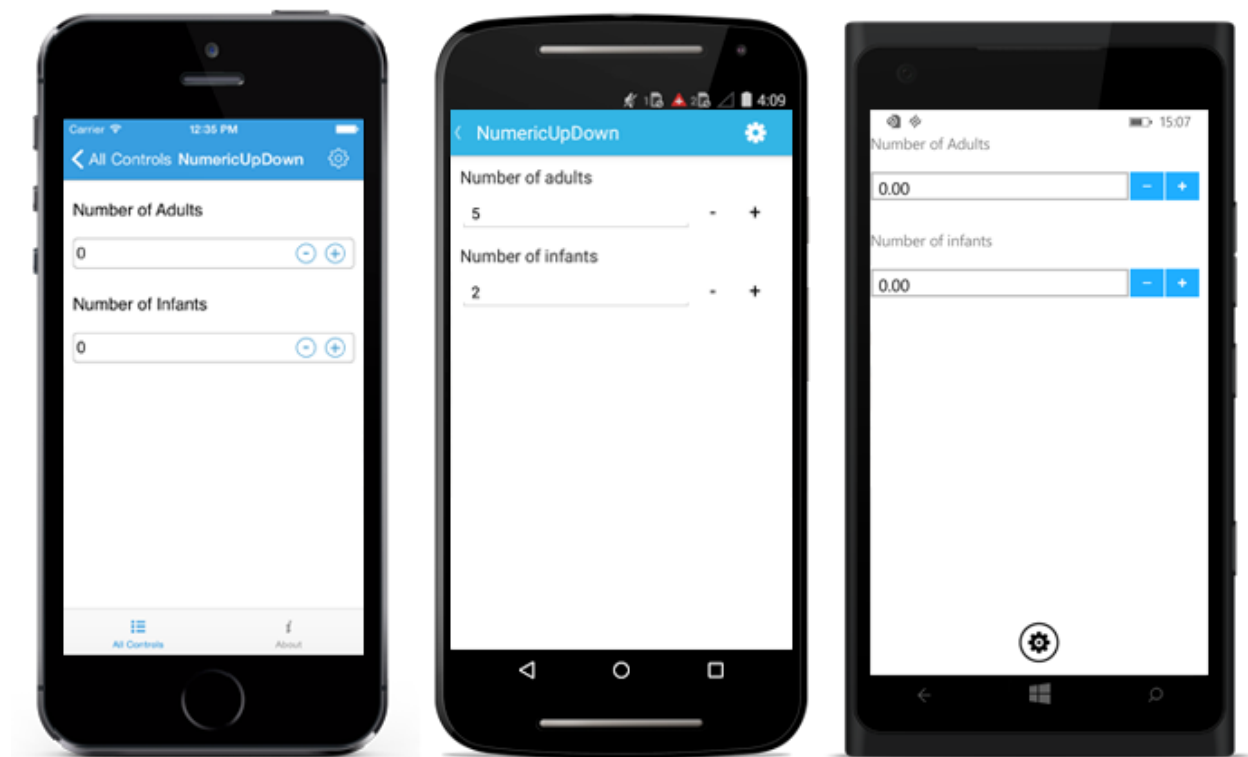
#### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
numericUpDown.FormatString="n";
```

---

**Note:** Instead of using above `FormatString` types, we can provide any symbol or value as string in `FormatString` property which will be appended with the value in `SfNumericUpDown`.

---



### Compute to Percentage

When the control is in percentage format, the value can be displayed in two ways as follows

- **Value**: Displays the actual value with percentage symbol.

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" Value="5" FormatString="p" PercentDisplayMode="Value"/>
```

#### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
numericUpDown.Value = 5;  
numericUpDown.FormatString="p";  
numericUpDown.PercentDisplayMode=PercentDisplayMode.Value;
```



- **Compute**: Displays the value computed by 100 with percentage symbol.

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" Value="5" FormatString="p"
PercentDisplayMode="Compute"/>
```

### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();
numericUpDown.Value = "5";
numericUpDown.FormatString="p";
numericUpDown.PercentDisplayMode=PercentDisplayMode.Compute;
```

**Note:** The control displays the percent value on lost focus.



### Set EnableGroupSeparator

The `EnableGroupSeparator` property is used to get rid of the comma in the Value of `SfNumericUpDown`.

### XML

```
<numeric:SfNumericUpDown x:Name="NumericUpDown" Value="12345"
EnableGroupSeparator="True"/>
```

### C#

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();
NumericUpDown.Value=12345;
NumericUpDown.EnableGroupSeparator = true;
this.Content = NumericUpDown;
```



### Font Customization

You can customize the font style of `NumericUpDown` using the following font-related properties that display the value's text:

- `FontSize` : Sets the font size for `NumericUpDown`'s text.
- `FontAttributes` : Sets the style of `NumericUpDown`'s text. You can add three types of styles to it. It specifies style information like Italic and Bold (using the `FontAttributes` enumeration in C#).
  1. Bold: Sets the style of `NumericUpDown`'s text to bold.
  2. Italic: Sets the style of `NumericUpDown`'s text to italic.
  3. None: Keeps the style of `NumericUpDown`'s text as same. It will not modify the style.

**Note:** The default value of this property is None.

- `FontFamily` : Customizes the font family of the `NumericUpDown`'s text.

- **TextAlignment** : Sets the style of NumericUpDown's text. We can give three types of style on it. It is specifying style information like Start, End and Center (using the TextAlignment enumeration in C#)

**Note:** The default value of this property is Start.

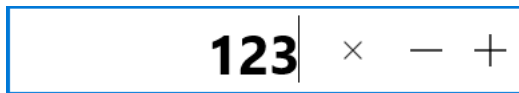
The following code sample demonstrates how to set the font size and attributes in XAML and in C#.

#### XAML

```
<numeric:SfNumericUpDown x:Name="NumericUpDown"
FontSize="27"
FontAttribute="Bold"
Value="123"
TextAlignment="End" />
```

#### C#

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();
NumericUpDown.FontSize = 27;
NumericUpDown.Value = 123;
NumericUpDown.TextAlignment=TextAlignment.End;
NumericUpDown.FontAttribute = FontAttributes.Bold;
this.Content = NumericUpDown;
```



#### Localization

The SfNumericUpDown value can be localized to any specific culture. It can be specified by setting the Culture property with System.Globalization.CultureInfo object instance.

**Note:** You cannot set value to the Culture property in XAML.

#### C#

```
numericUpDown.Culture = new System.Globalization.CultureInfo("hi-IN");
```



#### Change Localization of Return key text

The SfNumericUpDown provides the Localization support for the Return Key in soft keypad of iOS. We have provided the knowledge base document for the same. Please refer the Knowledge Base document from this [link](#).

#### Set Maximum Number of Decimal Digits

The maximum number of digits to be displayed after the decimal point can be specified by using [MaximumDecimalDigits](#) property.

**Note:** The [MaximumDecimalDigits](#). The MaximumDecimalDigits property is provided with positive value only. By default, the value of this property is 2.

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" MaximumDecimalDigits="3"/>
```

#### C#

```
numericUpDown.MaximumDecimalDigits = 3;
```

A screenshot of the SfNumericUpDown control. It displays the value "3.234" in a text box. To the right of the text box are two circular buttons: a minus sign (-) and a plus sign (+).

#### Assign Nullable Value

The null values can be set in SfNumericUpDown **Value** property, by setting **AllowNull** property value to true.

**Note:** By default, the property value is false.

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" AllowNull="true"/>
```

#### C#

```
numericUpDown.AllowNull=true;
```

A screenshot of the SfNumericUpDown control. The input field is empty. To the right of the input field are two circular buttons: a minus sign (-) and a plus sign (+).

#### Set Hint Text

The **WaterMark** property can be used to provide a hint that helps the user to get started with their input. The watermark text is visible when value is empty or null.

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" Watermark="NumericUpDown"/>
```

#### C#

```
numericUpDown.Watermark = "NumericUpDown";
```



### Auto Reverse

While incrementing, the control will start from Minimum value once it reaches the Maximum value and vice-versa.

**Note:** By default the property value is false.

### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
numericUpDown.AutoReverse = true;
```

### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" AutoReverse="true"/>
```

### Continuous Spinning Between Ranges

User can restrict the Values between a specific range by setting **Maximum** and **Minimum** property value.

**Note:** By default, the value of minimum property is Double.MinValue and the value of maximum property is Double.MaxValue.

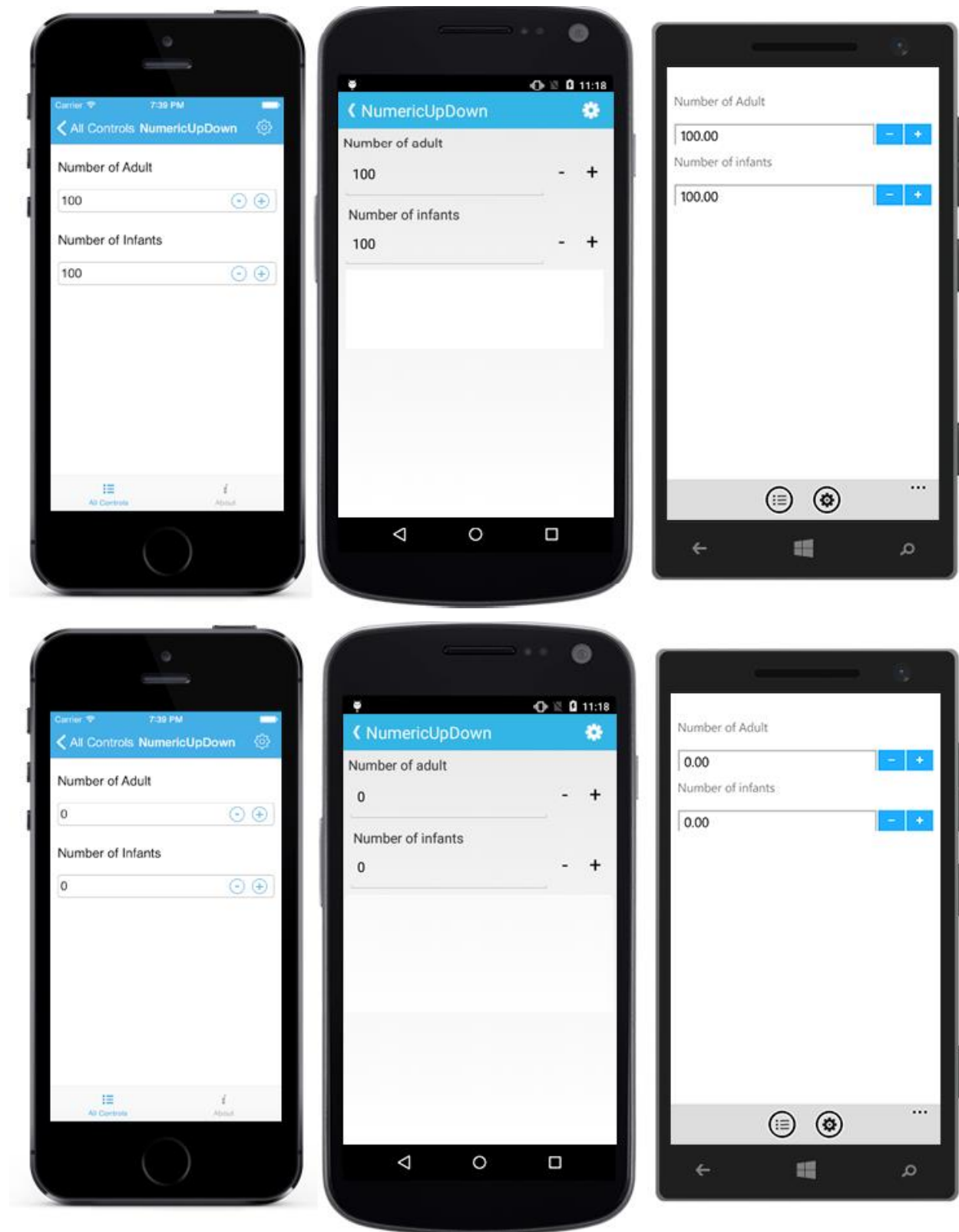
In iOS, if typed value is less than minimum value, the minimum value will be validated.

### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" Minimum="10" Maximum="100"/>
```

### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
numericUpDown.Minimum = 0;  
numericUpDown.Maximum = 100;  
this.Content = numericUpDown;
```



### Set Increment

Frequency in which values gets incremented can be decided using `StepValue` property.



---

**Note:** By default the property value is 1.

---

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" StepValue="6"/>
```

#### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
numericUpDown.StepValue = 6;  
this.Content = numericUpDown;
```

#### Set IsEditable

This property is used to decide whether users need to perform edit operation in input field.

---

**Note:** By default, the value of IsEditable property is true.

---

#### XML

```
<numeric:SfNumericUpDown Value="123" IsEditable="True"/>
```

#### C#

```
SfNumericUpDown numericUpDown=new SfNumericUpDown();  
numericUpDown.IsEditable = true;
```

#### Set SelectAllOnFocus

The **SelectAllOnFocus** property is used to specify whether the text should be selected when the control gets the focus.

#### XML

```
<numeric:SfNumericUpDown SelectAllOnFocus="True" Value="12345"/>
```

#### C#

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();  
NumericUpDown.Value=12345;  
NumericUpDown.SelectAllOnFocus = true;  
this.Content = NumericUpDown;
```



#### Events and Interactivity

##### Events

##### ValueChanged

You can perform any operation when changing the value of NumericUpDown using the ValueChanged event. The ValueChanged event returns the changed value in NumericUpDown.

For example you can restrict the NumericUpDown value if it exceed's greater than 3 digits using following code.

#### XML

```
<syncfusion:SfNumericUpDown x:Name="NumericUpDown"
ValueChangeMode="OnKeyFocus" ValueChanged="Handle_ValueChanged" Value="123"
/>
```

#### C#

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();
NumericUpDown.Value = 123;
NumericUpDown.ValueChangeMode = ValueChangeMode.OnKeyFocus;
NumericUpDown.ValueChanged += Handle_ValueChanged;
this.Content=NumericUpDown;
string updateValue = "";
void Handle_ValueChanged(object sender,
Syncfusion.SfNumericUpDown.XForms.ValueEventArgs e)
{
if (e.Value.ToString().Length <= 3 && e.Value != null)
{
updateValue = e.Value.ToString();
}
else
{
updown.Value = updateValue.ToString();
}
}
```

#### Interactivity : ValueChangeMode

The ValueChangeMode property is used to mention when value needs to update, either in key pressed or focus lost state. When ValueChangeMode is set to OnKeyFocus, the value will be updated on each key press. When ValueChangeMode is set to OnLostFocus, the value is updated when the control lose the focus or the focus is moved to the next control. ValueChangeMode includes the following options:

1. OnKeyFocus
2. OnLostFocus

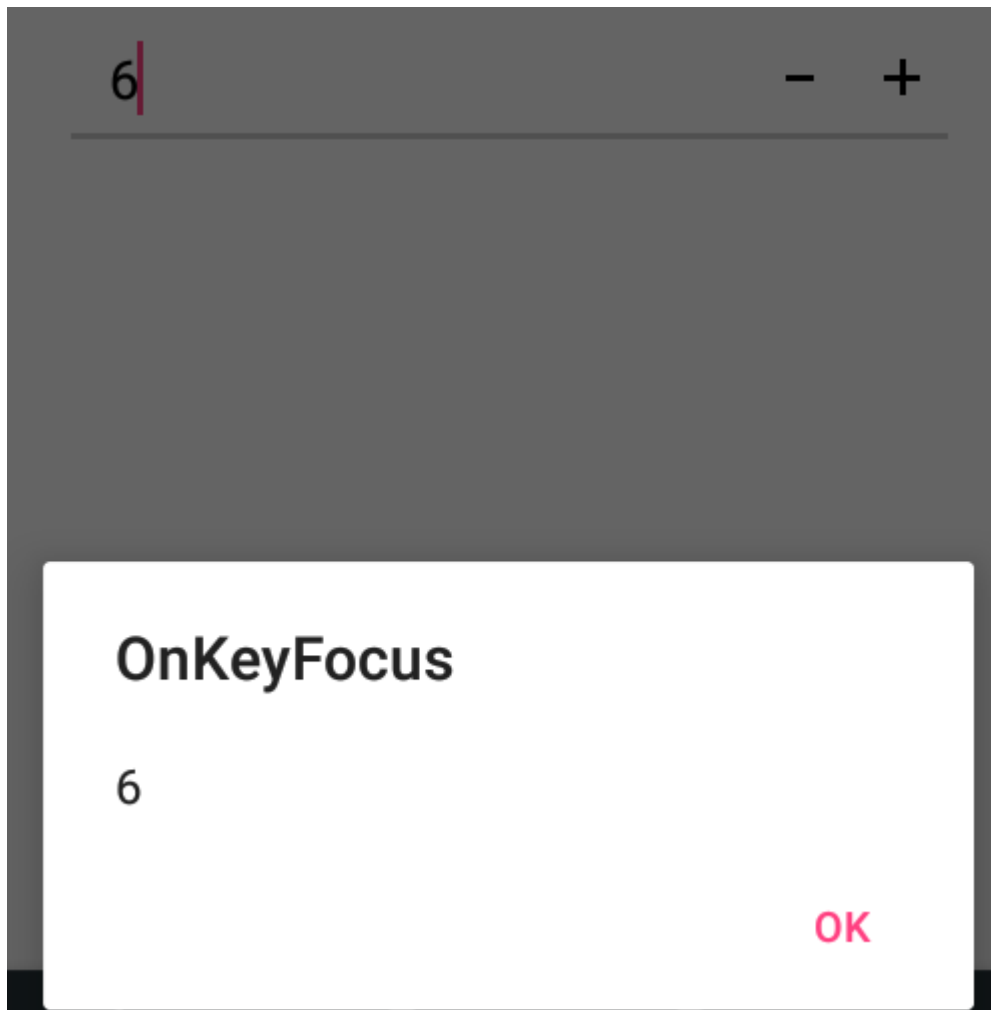
#### OnKeyFocus

##### XML

```
<syncfusion:SfNumericUpDown ValueChangeMode="OnKeyFocus"
x:Name="NumericUpDown" Value="123" />
```

##### C#

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();
NumericUpDown.Value = 123;
NumericUpDown.ValueChangeMode = ValueChangeMode.OnKeyFocus;
this.Content=NumericUpDown;
```



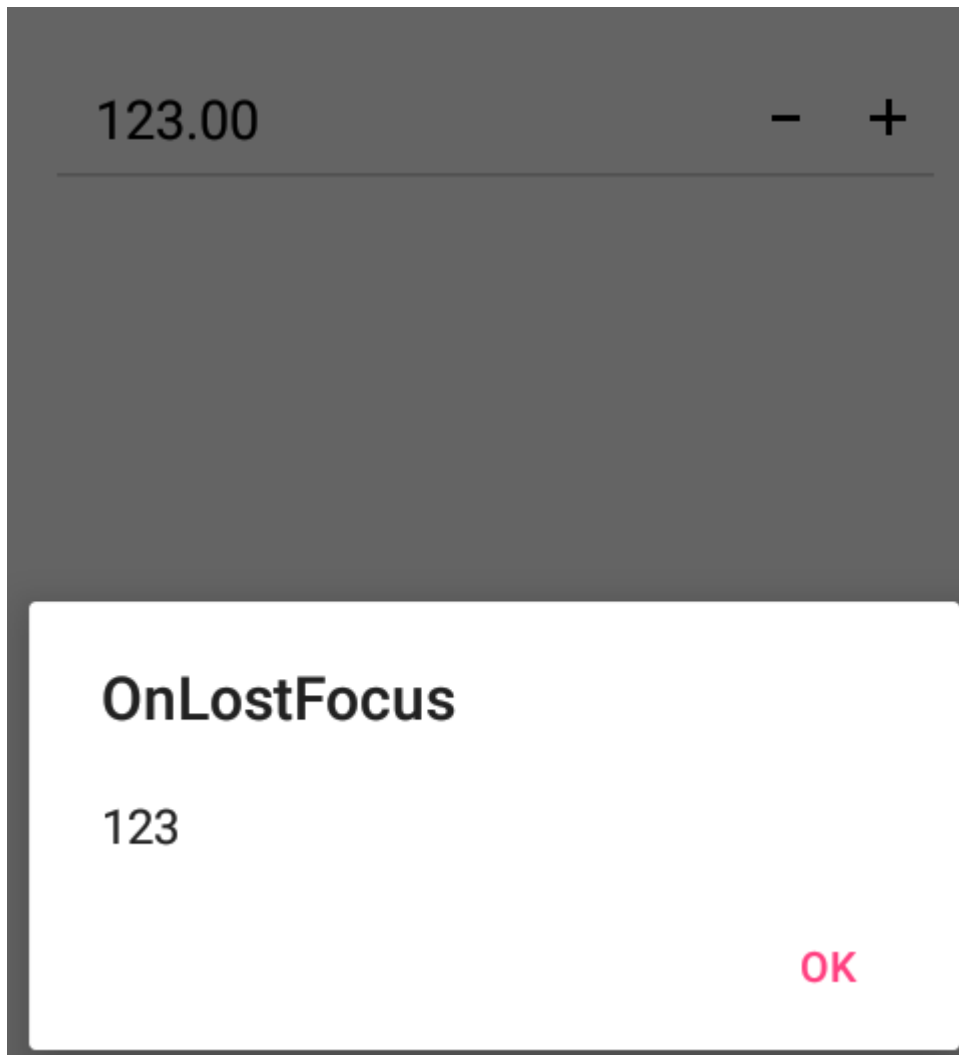
*OnLostFocus*

#### **XML**

```
<syncfusion:SfNumericUpDown ValueChangeMode="OnLostFocus"  
x:Name="NumericUpDown" Value="123" />
```

#### **C#**

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();  
NumericUpDown.Value = 123;  
NumericUpDown.ValueChangeMode = ValueChangeMode.OnLostFocus;  
this.Content=NumericUpDown;
```



#### Completed

This event occurs when users finalize the text in the NumericUpDown by pressing the return key(enter, ok) on keyboard.

#### XML

```
<syncfusion:SfNumericUpDown x:Name="NumericUpDown"
    Completed="Handle_Completed"/>
```

#### C#

```
SfNumericUpDown NumericUpDown=new SfNumericUpDown();
NumericUpDown.Completed += Handle_Completed;
this.Content=NumericUpDown;
void Handle_Completed(object sender, System.EventArgs e)
{
    System.Diagnostics.Debug.WriteLine("Completed");
}
```

## Spin Button Alignment

Spin Button position in the SfNumericUpDown control can be changed relative to the TextBox based on `SpinButtonAlignment` property.

There are three built-in modes.

### Right

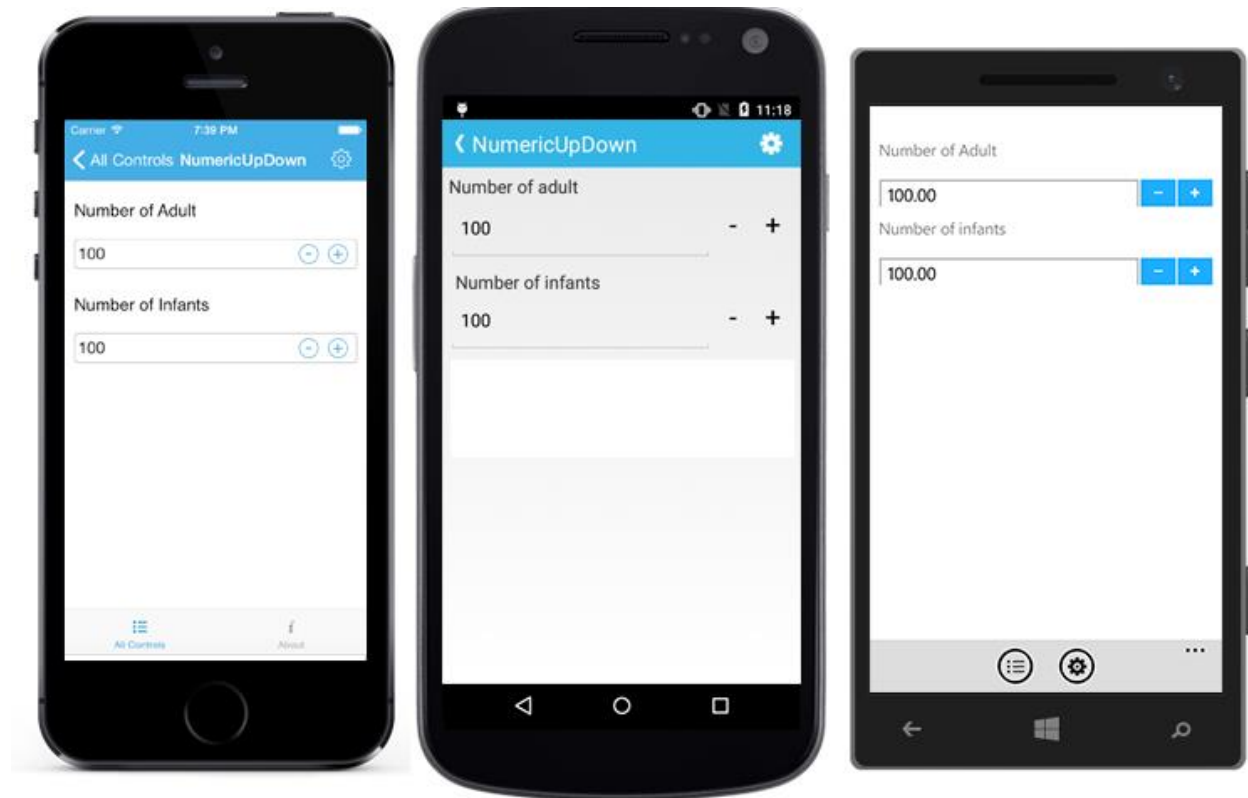
Spin Buttons will get aligned to the right side of the control.

#### C#

```
numericUpDown.SpinButtonAlignment = SpinButtonAlignment.Right;
```

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown"
    SpinButtonAlignment="Right"/>
```



### Left

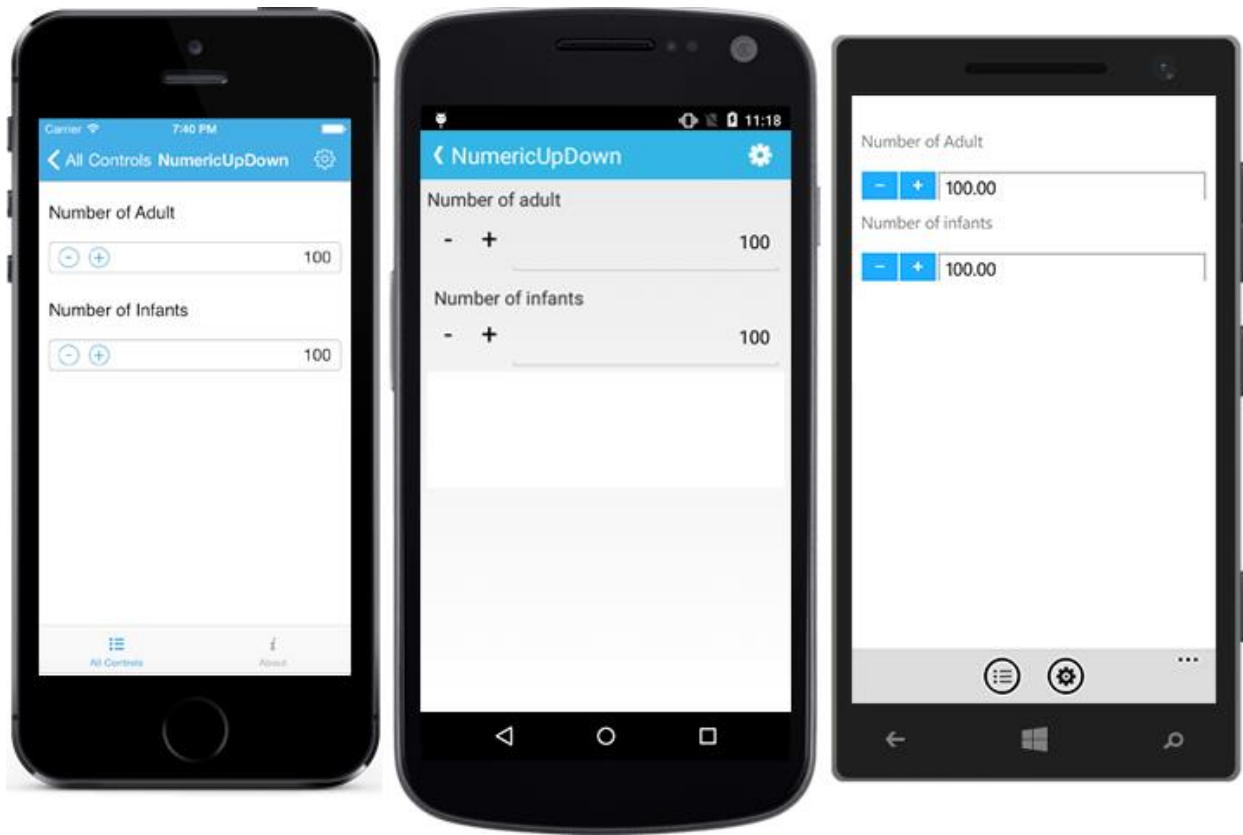
Spin Buttons will get aligned to the left side of the control.

#### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" SpinButtonAlignment="Left"/>
```

#### C#

```
numericUpDown.SpinButtonAlignment = SpinButtonAlignment.Left;
```



### *Both*

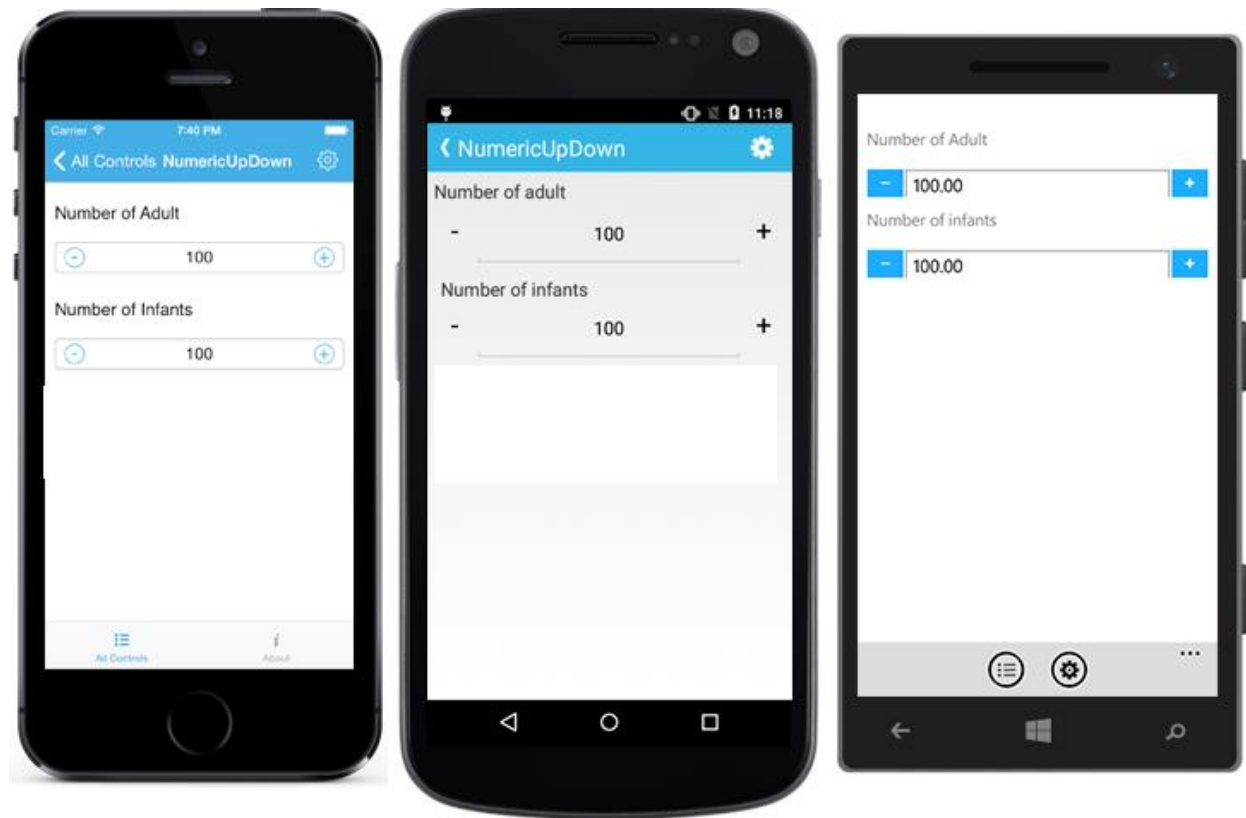
Spin Buttons will get aligned to the both side of the control.

### XML

```
<numeric:SfNumericUpDown x:Name="numericUpDown" SpinButtonAlignment="Both"/>
```

### C#

```
numericUpDown.SpinButtonAlignment = SpinButtonAlignment.Both;
```



**Note:** By default the property value is Right.

#### UpDownButtonSetting Customization

We can set the Up Down button of SfNumericUpDown control by using any of the below given ways.

1. View
2. Image
3. FontIconText

**Note:** For image and Font icon we need to add the the respective image and TTF file.

For android: Add image at Resource/Drawable/{Image} and .ttf file at Asserts/{.ttf}

For iOS: Add image and .ttf file to Resource/{Image/.ttf} and To use FontIcons, add respective FontFamily name in info.plist file under Fonts provided by application category.

For UWP: Add the image and .ttf file directly to the project.

*By using View*

#### XML

```
<updown:SfNumericUpDown x:Name="upDown" SpinButtonAlignment="Both"
    TextAlign="Center">
    <updown:SfNumericUpDown.IncrementButtonSettings>
    <updown:UpDownButtonSettings ButtonHeight="45" ButtonWidth="45">
    <updown:UpDownButtonSettings.ButtonView>
    <Grid HeightRequest="40" WidthRequest="40">
    <Image Source="up.png" Aspect="AspectFit" />
    </Grid>
```

```

</updown:UpDownButtonSettings.ButtonView>
</updown:UpDownButtonSettings>
</updown:SfNumericUpDown.IncrementButtonSettings>
<updown:SfNumericUpDown.DecrementButtonSettings>
<updown:UpDownButtonSettings ButtonHeight="45" ButtonWidth="45">
<updown:UpDownButtonSettings.ButtonView>
<Grid HeightRequest="40" WidthRequest="40">
<Image Source="down.png" Aspect="AspectFit" />
</Grid>
</updown:UpDownButtonSettings.ButtonView>
</updown:UpDownButtonSettings>
</updown:SfNumericUpDown.DecrementButtonSettings>
</updown:SfNumericUpDown>

```

**C#**

```

public partial class MainPage : ContentPage
{
    SfNumericUpDown upDown;
    Grid incrementGrid, decrementGrid;
    UpDownButtonSettings incSettings, decrementSettings;
    Image incrementImage, decrementImage;
    public MainPage()
    {
        InitializeComponent();
        incrementGrid = new Grid
        {
            HeightRequest = 40,
            WidthRequest = 40,
            BackgroundColor = Color.Blue
        };
        incSettings = new UpDownButtonSettings
        {
            ButtonView = incrementGrid,
            ButtonHeight = 45,
            ButtonWidth = 45
        };
        incrementImage = new Image
        {
            HorizontalOptions = LayoutOptions.Center,
            VerticalOptions = LayoutOptions.Center,
            Source = (FileImageSource)ImageSource.FromFile("up.png"),
            Aspect = Aspect.AspectFit
        };
        decrementSettings = new UpDownButtonSettings
        {
            ButtonView = decrementGrid,
            ButtonHeight = 45,
            ButtonWidth = 45
        };
        decrementGrid = new Grid
        {
            HeightRequest = 40,
            WidthRequest = 40,
            BackgroundColor = Color.Red
        };
    }
}

```



```

decrementImage = new Image
{
    Source = (FileImageSource) ImageSource.FromFile("down.png"),
    Aspect = Aspect.AspectFit,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center
};
upDown = new SfNumericUpDown
{
    SpinButtonAlignment = SpinButtonAlignment.Left,
    IncrementButtonSettings = incSettings,
    DecrementButtonSettings = decrementSettings
};
incrementGrid.Children.Add(incrementImage);
decrementGrid.Children.Add(decrementImage);
this.Content = upDown;
}
}

```

*By using Image with Button Height & Width*

#### **XML**

```

<updown:SfNumericUpDown x:Name="upDown" SpinButtonAlignment="Both"
    TextAlignment="Center">
    <updown:SfNumericUpDown.IncrementButtonSettings>
    <updown:UpDownButtonSettings ButtonImage="up" ButtonHeight="35"
        ButtonWidth="35" />
    </updown:SfNumericUpDown.IncrementButtonSettings>
    <updown:SfNumericUpDown.DecrementButtonSettings>
    <updown:UpDownButtonSettings ButtonImage="down" ButtonHeight="35"
        ButtonWidth="35" />
    </updown:SfNumericUpDown.DecrementButtonSettings>
</updown:SfNumericUpDown>

```

#### **C#**

```

public partial class MainPage : ContentPage
{
    SfNumericUpDown upDown;
    Grid incrementGrid, decrementGrid;
    UpDownButtonSettings incSettings, decrementSettings;
    Image incrementImage, decrementImage;
    public MainPage()
    {
        InitializeComponent();
        incSettings = new UpDownButtonSettings
        {
            ButtonImage = "up",
            ButtonHeight = 45,
            ButtonWidth = 45
        };
        decrementSettings = new UpDownButtonSettings
        {
            ButtonImage = "down",
            ButtonHeight = 45,

```

```

ButtonWidth = 45
};
upDown = new SfNumericUpDown
{
    SpinButtonAlignment = SpinButtonAlignment.Both,
    IncrementButtonSettings = incSettings,
    DecrementButtonSettings = decrementSettings
};
this.Content = upDown;
}
}

```



*By using FontIconText*

#### **XML**

```

<updown:SfNumericUpDown x:Name="upDown" SpinButtonAlignment="Both"
    TextAlignment="Center">
    <updown:SfNumericUpDown.IncrementButtonSettings>
    <updown:UpDownButtonSettings.ButtonFontFamily>
    <OnPlatform
    x:TypeArguments="x:String">
    <On
    Platform="Android"
    Value="numeric.ttf">
    </On>
    <On
    Platform="iOS"
    Value="numeric">
    </On>
    </OnPlatform>
    </updown:UpDownButtonSettings.ButtonFontFamily>
    <updown:UpDownButtonSettings ButtonFontIcon="#xe701;" ButtonHeight="35"
    ButtonWidth="35" />
    </updown:SfNumericUpDown.IncrementButtonSettings>
    <updown:SfNumericUpDown.DecrementButtonSettings>
    <updown:UpDownButtonSettings.ButtonFontFamily>
    <OnPlatform
    x:TypeArguments="x:String">
    <On
    Platform="Android"
    Value="numeric.ttf">
    </On>
    <On
    Platform="iOS"
    Value="numeric">
    </On>
    </OnPlatform>

```

```

</updown:UpDownButtonSettings.ButtonFontFamily>
<updown:UpDownButtonSettings ButtonFontIcon="&#xe700;" ButtonHeight="35"
ButtonWidth="35" />
</updown:SfNumericUpDown.DecrementButtonSettings>
</updown:SfNumericUpDown>

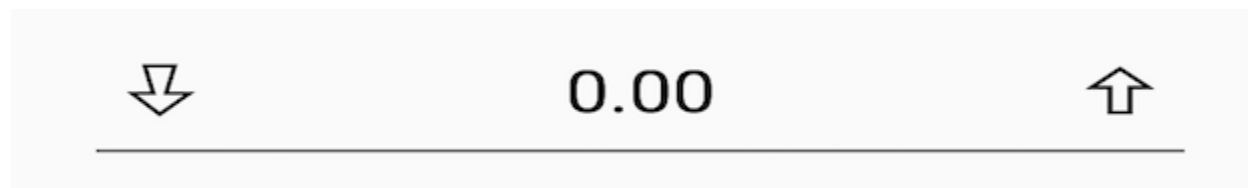
```

**C#**

```

SfNumericUpDown upDown = new SfNumericUpDown();
upDown.SpinButtonAlignment = SpinButtonAlignment.Both;
UpDownButtonSettings incSettings = new UpDownButtonSettings();
incSettings.ButtonFontIcon = "\xe701";
if (Device.OS == TargetPlatform.Android)
{
    incSettings.ButtonFontFamily = "numeric.ttf";
    decrementSettings.ButtonFontFamily = "numeric.ttf";
}
else
{
    incSettings.ButtonFontFamily = "numeric";
    decrementSettings.ButtonFontFamily = "numeric";
}
upDown.IncrementButtonSettings = incSettings;
UpDownButtonSettings decrementSettings = new UpDownButtonSettings();
decrementSettings.ButtonFontIcon = "\xe700";
decrementSettings.ButtonWidth = 45;
upDown.DecrementButtonSettings = decrementSettings;
this.Content = upDown;

```



Additional customization properties of UpDownButtonSettings

*BackgroundColor*

This property is used to change the background color of increment and decrement buttons.

**XML**

```

<updown:SfNumericUpDown >
<updown:SfNumericUpDown.IncrementButtonSettings>
<updown:UpDownButtonSettings BackgroundColor="Red"/>
</updown:SfNumericUpDown.IncrementButtonSettings>
<updown:SfNumericUpDown.DecrementButtonSettings>
<updown:UpDownButtonSettings BackgroundColor="Green"/>
</updown:SfNumericUpDown.DecrementButtonSettings>
</updown:SfNumericUpDown>

```

**C#**

```

SfNumericUpDown numericUpDown = new SfNumericUpDown();
UpDownButtonSettings incrementButtonSettings = new UpDownButtonSettings();

```

```
UpDownButtonSettings decrementButtonSettings = new UpDownButtonSettings();
numericUpDown.IncrementButtonSettings = incrementButtonSettings;
numericUpDown.DecrementButtonSettings = decrementButtonSettings;
incrementButtonSettings.BackgroundColor = Color.Red;
decrementButtonSettings.BackgroundColor = Color.Green;
```



#### *HighlightedBackgroundColor*

This property is used to change the background color of when press the increment or decrement button.

#### **XML**

```
<updown:SfNumericUpDown >
<updown:SfNumericUpDown.IncrementButtonSettings>
<updown:UpDownButtonSettings HighlightedBackgroundColor="Red"/>
</updown:SfNumericUpDown.IncrementButtonSettings>
<updown:SfNumericUpDown.DecrementButtonSettings>
<updown:UpDownButtonSettings HighlightedBackgroundColor="Green"/>
</updown:SfNumericUpDown.DecrementButtonSettings>
</updown:SfNumericUpDown>
```

#### **C#**

```
SfNumericUpDown numericUpDown = new SfNumericUpDown();
UpDownButtonSettings incrementButtonSettings = new UpDownButtonSettings();
UpDownButtonSettings decrementButtonSettings = new UpDownButtonSettings();
numericUpDown.IncrementButtonSettings = incrementButtonSettings;
numericUpDown.DecrementButtonSettings = decrementButtonSettings;
incrementButtonSettings.HighlightedBackgroundColor = Color.Red;
decrementButtonSettings.HighlightedBackgroundColor = Color.Green;
```

#### *ButtonFontColor*

This property is used to change the text color of increment and decrement buttons.

#### **XML**

```
<updown:SfNumericUpDown >
<updown:SfNumericUpDown.IncrementButtonSettings>
<updown:UpDownButtonSettings ButtonFontColor="Red"/>
</updown:SfNumericUpDown.IncrementButtonSettings>
<updown:SfNumericUpDown.DecrementButtonSettings>
<updown:UpDownButtonSettings ButtonFontColor="Green"/>
</updown:SfNumericUpDown.DecrementButtonSettings>
</updown:SfNumericUpDown>
```

#### **C#**

```
SfNumericUpDown numericUpDown = new SfNumericUpDown();
UpDownButtonSettings incrementButtonSettings = new UpDownButtonSettings();
UpDownButtonSettings decrementButtonSettings = new UpDownButtonSettings();
numericUpDown.IncrementButtonSettings = incrementButtonSettings;
numericUpDown.DecrementButtonSettings = decrementButtonSettings;
```

```
incrementButtonSettings.ButtonFontColor = Color.Red;
decrementButtonSettings.ButtonFontColor = Color.Green;
```

### HighlightedButtonFontColor

This property is used to change the text color of button when press the increment or decrement button.

### XML

```
<updown:SfNumericUpDown >
  <updown:SfNumericUpDown.IncrementButtonSettings>
    <updown:UpDownButtonSettings HighlightedButtonFontColor="Red"/>
  </updown:SfNumericUpDown.IncrementButtonSettings>
  <updown:SfNumericUpDown.DecrementButtonSettings>
    <updown:UpDownButtonSettings HighlightedButtonFontColor="Green"/>
  </updown:SfNumericUpDown.DecrementButtonSettings>
</updown:SfNumericUpDown>
```

### C#

```
SfNumericUpDown numericUpDown = new SfNumericUpDown();
UpDownButtonSettings incrementButtonSettings = new UpDownButtonSettings();
UpDownButtonSettings decrementButtonSettings = new UpDownButtonSettings();
numericUpDown.IncrementButtonSettings = incrementButtonSettings;
numericUpDown.DecrementButtonSettings = decrementButtonSettings;
incrementButtonSettings.HighlightedButtonFontColor = Color.Red;
decrementButtonSettings.HighlightedButtonFontColor = Color.Green;
```

### AutomationId

The NumericUpDown control has built-in AutomationId for inner elements. To keep unique id for inner elements, AutomationId of inner elements are updated based on NumericUpDown's AutomationId. For example, if NumericUpDown's AutomationId is set as sfNumericUpDown.AutomationId = "numericUpDown", then AutomationId of inner elements will be updated as follows.

Element	Element AutomationId	Qualified AutomationId
Increment Button	increment button	numericUpDown increment button
Decrement Button	decrement button	numericUpDown decrement button

## PDF

## SfParallaxView

### Overview

SfParallaxView for Xamarin.Forms provides a perfect way for scrolling any control that implements the IParallaxView interface with a background element that translates slower than the foreground element. The SfParallaxView control binds the scroll position of a foreground element (e.g., list) to a background element and moves the background element at different speeds.

## Key features

- Customizes the speed of the parallax scroll.
- Supports horizontal and vertical orientation.
- Provides built-in integration with SfListView and SfRotator.
- Bind any control which implements IParallaxView interface.



## Getting Started

This section explains the steps required to configure the [SfParallaxView](#) control and add basic elements to it using various APIs.

### Adding SfParallaxView reference

You can add SfParallaxView reference using one of the following methods:

#### Method 1: Adding SfParallaxView reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfParallaxView). To add SfParallaxView to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfParallaxView](https://www.nuget.org/packages/Syncfusion.Xamarin.SfParallaxView), and then install it.

![[Adding SfParallaxView reference from NuGet](ParallaxView\_Images/Adding SfParallaxView reference.png)]

---

**Note:** Install the same version of SfParallaxView NuGet in all the projects.

---

### Method 2: Adding SfParallaxView reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfParallaxView control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfParallaxView assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfParallaxView.XForms.dll Syncfusion.Core.XForms.dll
Android	Syncfusion.SfParallaxView.XForms.Android.dll Syncfusion.SfParallaxView.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll
iOS	Syncfusion.SfParallaxView.XForms.iOS.dll Syncfusion.SfParallaxView.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll
UWP	Syncfusion.SfParallaxView.XForms.UWP.dll Syncfusion.SfParallaxView.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

### Launch an application in iOS

To launch the parallax view in iOS, call the `SfParallaxViewRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    global::Xamarin.Forms.Forms.Init();
    Syncfusion.XForms.iOS.ParallaxView.SfParallaxViewRenderer.Init();
    LoadApplication(new App());
}
```

### Universal Windows Platform (UWP)

You need to initialize the parallax view assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with parallax view in Release mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    // Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.ParallaxView.SfParallax
ViewRenderer).GetTypeInfo().Assembly);
    // Replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Android

The Android platform does not require any additional configuration to render the parallax view.

#### Initialize parallax view

1. Import SfParallaxView control namespace as `xmlns:parallax="clr-namespace:Syncfusion.XForms.ParallaxView;assembly=Syncfusion.SfParallaxView.XForms"` in the XAML page.
2. Set the SfParallaxView control as content to the ContentPage.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ParallaxSample"
xmlns:parallax="clr-
namespace:Syncfusion.XForms.ParallaxView;assembly=Syncfusion.SfParallaxView.
XForms"
x:Class="ParallaxSample.MainPage">
<ContentPage.Content>
<parallax:SfParallaxView />
</ContentPage.Content>
</ContentPage>
```



**C#**

```
using Syncfusion.XForms.ParallaxView;
using Xamarin.Forms;
public class App : Application
{
    public App()
    {
        MainPage = new ParallaxSample.MainPage();
    }
}
public class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        SfParallaxView parallaxView = new SfParallaxView();
        this.Content = parallaxView;
    }
}
```

Add content to the parallax view

*Content*

The **Content** represents the background view of a parallax view. You can set any kind of view to the **Content** property such as Image and StackLayout.

The following code sample demonstrates how to set the content property to the parallax view.

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ParallaxSample"
xmlns:parallax="clr-namespace:Syncfusion.XForms.ParallaxView;assembly=Syncfusion.SfParallaxView.XForms"
x:Class="ParallaxSample.MainPage">
    <ContentPage.Content>
        <Grid>
            <parallax:SfParallaxView x:Name="parallaxview">
                <parallax:SfParallaxView.Content>
                    <Image Source="{Binding Image}" BackgroundColor="Transparent"
HorizontalOptions="Fill" VerticalOptions="Fill" Aspect="AspectFill" />
                </parallax:SfParallaxView.Content>
            </parallax:SfParallaxView>
        </Grid>
    </ContentPage.Content>
</ContentPage>
```

**C#**

```
using Syncfusion.XForms.ParallaxView;
using Xamarin.Forms;
public partial class MainPage : ContentPage
```

```

{
    public MainPage()
    {
        InitializeComponent();
        BindingContext = new ParallaxViewModel();
    }
}

public class ParallaxViewModel
{
    public ImageSource Image { get; set; }
    public ParallaxViewModel()
    {
        Image = ImageSource.FromResource("ParallaxSample.ParallaxGuitar1.png",
        typeof(MainPage).GetTypeInfo().Assembly);
    }
}

```

Bind source to the parallax view

#### Source

The **Source** represents the foreground view of the parallax view. The value of **Source** should be a scrollable content or the view which implements **IParallaxView** interface.

As of now, the SfParallaxView supports the following controls directly. You can simply bind the control to the **Source** property.

1. ScrollView
2. SfListView
3. SfRotator

The following code sample demonstrates how to bind the SfListView to the **Source** property.

#### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ParallaxSample"
xmlns:list="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:parallax="clr-namespace:Syncfusion.XForms.ParallaxView;assembly=Syncfusion.SfParallaxView.XForms"
x:Class="ParallaxSample.MainPage">
<ContentPage.Content>
<Grid>
<parallax:SfParallaxView Source="{x:Reference Name = listview}"
x:Name="parallaxview" >
<parallax:SfParallaxView.Content>
<Image BackgroundColor="Transparent" Source="{Binding Image}"
HorizontalOptions="Fill" VerticalOptions="Fill" Aspect="AspectFill" />
</parallax:SfParallaxView.Content>
</parallax:SfParallaxView>
<list:SfListView x:Name="listview" ItemsSource="{Binding Items}"
BackgroundColor="Transparent" ItemSize="100">
<list:SfListView.ItemTemplate>

```

```

<DataTemplate>
<ViewCell>
<Grid Padding="20,0,20,0" RowSpacing="50">
<StackLayout BackgroundColor="Transparent" Grid.Column="1" Padding="0,0,0,0"
VerticalOptions="CenterAndExpand" HorizontalOptions="StartAndExpand"
Orientation="Vertical">
<Label HorizontalOptions="Start" TextColor="White" Text="{Binding Name}"
Font="25">
</Label>
<Label HorizontalOptions="Start" Text="{Binding Author}" TextColor="White">
</Label>
</StackLayout>
</Grid>
</ViewCell>
</DataTemplate>
</list:SfListView.ItemTemplate>
</list:SfListView>
</Grid>
</ContentPage.Content>
</ContentPage>

```

## C#

```

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        BindingContext = new ParallaxViewModel();
    }
}

public class ParallaxViewModel
{
    public ImageSource Image { get; set; }
    public ObservableCollection<Contacts> Items { get; set; }
    public ParallaxViewModel()
    {
        Image = ImageSource.FromResource("ParallaxSample.ParallaxGuitar1.png",
        typeof(MainPage).GetTypeInfo().Assembly);
        Items = new ObservableCollection<Contacts>()
        {
            new Contacts() { Name = "Thriller", Author = "Michael Jackson" },
            new Contacts() { Name = "Like a Prayer", Author = "Madonna" },
            new Contacts() { Name = "When Doves Cry", Author = "Prince" },
            new Contacts() { Name = "I Wanna Dance", Author = "Whitney Houston" },
            new Contacts() { Name = "It's Gonna Be Me", Author = "N Sync" },
            new Contacts() { Name = "Everybody", Author = "Backstreet Boys" },
            new Contacts() { Name = "Rolling in the Deep", Author = "Adele" },
            new Contacts() { Name = "Don't Stop Believing", Author = "Journey" },
            new Contacts() { Name = "Billie Jean", Author = "Michael Jackson" },
            new Contacts() { Name = "Firework", Author = "Katy Perry" },
            new Contacts() { Name = "Thriller", Author = "Michael Jackson" },
            new Contacts() { Name = "Like a Prayer", Author = "Madonna" },
            new Contacts() { Name = "When Doves Cry", Author = "Prince" },
            new Contacts() { Name = "I Wanna Dance", Author = "Whitney Houston" },
            new Contacts() { Name = "It's Gonna Be Me", Author = "N Sync" },

```

```
new Contacts() { Name = "Everybody", Author = "Backstreet Boys" },  
new Contacts() { Name = "Rolling in the Deep", Author = "Adele" },  
new Contacts() { Name = "Don't Stop Believing", Author = "Journey"},  
};  
}  
}  
public class Contacts  
{  
    public string Name  
    {  
        get;  
        set;  
    }  
    public string Author  
    {  
        get;  
        set;  
    }  
}
```

**Note:** The size of the Content view will automatically be stretched to the size of the Source view.



You can find the complete getting started sample from this link: [Sample](#)

## Customization

### Speed Customization

The **Speed** value denotes the scrolling speed of the **Content** added as a background view. Based on the speed value, the background view will scroll along with the foreground view.

#### XML

```
<parallax:SfParallaxView Source="{x:Reference Name = listview}"
x:Name="parallaxview" Speed="0.5" >
<parallax:SfParallaxView.Content>
<Image BackgroundColor="Transparent" Source="{Binding Image}"
HorizontalOptions="Fill" VerticalOptions="Fill" Aspect="AspectFill" />
</parallax:SfParallaxView.Content>
</parallax:SfParallaxView>
```

#### C#

```
using Syncfusion.XForms.ParallaxView;
using Syncfusion.ListView.XForms;
namespace ParallaxView_GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            ParallaxViewModel view = new ParallaxViewModel();
            BindingContext = view;
            SfParallaxView parallax = new SfParallaxView();
            SfListView listview = new SfListView();
            Image image = new Image();
            Assembly assembly = typeof(MainPage).GetTypeInfo().Assembly;
            image.Source =
                ImageSource.FromResource("ParallaxView_GettingStarted.Images.ParallaxWallpaper.png", assembly);
            parallax.Content = image;
            parallax.Speed = 0.5;
            listview.ItemsSource = view.Items;
            parallax.Source = listview;
        }
    }
}
```

## Orientation

The orientation of the content scrolling can be customized to vertical or horizontal using the value of **Orientation** property.

#### XML

```
<Grid>
<parallax:SfParallaxView Source="{x:Reference Name = listview}"
x:Name="parallaxview" Orientation="Horizontal" >
<parallax:SfParallaxView.Content>
```

```

. . .
</parallax:SfParallaxView.Content>
</parallax:SfParallaxView>
<list:SfListView x:Name="listview" Orientation="Horizontal"
ItemsSource="{Binding Items}" BackgroundColor="Transparent" ItemSize="100">
. . .
</list:SfListView>
</Grid>

```

**C#**

```

using Syncfusion.XForms.ParallaxView;
using Syncfusion.ListView.XForms;
namespace ParallaxView_GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            ParallaxViewModel view = new ParallaxViewModel();
            BindingContext = view;
            SfParallaxView parallax = new SfParallaxView();
            SfListView listview = new SfListView();
            Image image = new Image();
            Assembly assembly = typeof(MainPage).GetTypeInfo().Assembly;
            image.Source =
            ImageSource.FromResource("ParallaxView_GettingStarted.Images.ParallaxWallpaper.png", assembly);
            parallax.Content = image;
            parallax.Speed = 0.5;
            parallax.Orientation =
            Syncfusion.XForms.ParallaxView.Orientation.Horizontal;
            listview.ItemsSource = view.Items;
            parallax.Source = listview;
        }
    }
}

```

## Scrolling support for custom controls

The Parallax view supports custom scrollable controls using the `IParallaxView` interface. This interface implements the `ScrollableContentSize` property and the `Scrolling` event.

**C#**

```

public class CustomListView : ListView, IParallaxView
{
    public Size ScrollableContentSize { get ; set ; }
    public event EventHandler<ParallaxScrollingEventArgs> Scrolling;
}

```

## Scrollable ContentSize

The `ScrollableContentSize` is the total content size of the scrollable custom control.

**C#**

```
public class CustomListView : ListView, IParallaxView
{
    public Size ScrollableContentSize { get; set; }
    public CustomListView()
    {
        this.ScrollableContentSize = ContentSize; // Total scrollable size of the
        custom control
    }
}
```

## Scrolling event

The **Scrolling** event occurs whenever the **ParallaxScrollingEventArgs** value is set through the scrollable custom control scrolled event.

The **ParallaxScrollingEventArgs** has the following three arguments:

- **ScrollX**: Denotes X position of the finished scroll.
- **ScrollY**: Denotes Y position of the finished scroll.
- **CanAnimate**: Defines whether to animate the scroll or not.

**XML**

```
<Grid>
<parallax:SfParallaxView Source="{x:Reference Name = listView}"
x:Name="parallaxview">
<parallax:SfParallaxView.Content>
<Image x:Name="image" BackgroundColor="Transparent" HorizontalOptions="Fill"
VerticalOptions="Fill" Aspect="AspectFill" />
</parallax:SfParallaxView.Content>
</parallax:SfParallaxView>
<local:CustomListView x:Name="listView" >
. . .
</local:CustomListView>
</Grid>
```

**C#**

```
public class CustomListView : ListView, IParallaxView
{
    public ListViewScrollingEventArgs scrollingEventArgs;
    internal event EventHandler<ListViewScrollingEventArgs>
CustomListViewScrolling;
    private Size scrollableContentSize = new Size();
    public Size ScrollableContentSize
    {
        get
        {
            return this.scrollableContentSize;
        }
        set
        {
            this.scrollableContentSize = value;
        }
    }
}
```

```

OnPropertyChanged("ScrollableContentSize");
}
}
public event EventHandler<ParallaxScrollingEventArgs> Scrolling;
protected virtual void OnScrollChanged(ParallaxScrollingEventArgs e)
{
    Scrolling?.Invoke(this, e);
}
public CustomListView()
{
    this.scrollingEventArgs = new ListViewScrollingEventArgs(); // Need to
    initiate event args
    CustomListViewScrolling += CustomListView_CustomListViewScrolling; // Need
    to invoke custom scroll event
}
private void CustomListView_CustomListViewScrolling(object sender,
    ListViewScrollingEventArgs e)
{
    OnScrollChanged(new ParallaxScrollingEventArgs(e.ScrollX, e.ScrollY,
        false));
}
}
public class ListViewScrollingEventArgs : EventArgs
{
    /// <summary>
    /// Initializes a new instance of the <see cref="ListViewScrollingEventArgs"
    /> class.
    /// </summary>
    public ListViewScrollingEventArgs()
    {
    }

    /// <summary>
    /// Gets or sets ScrollX value.
    /// </summary>
    public double ScrollX { get; set; }
    /// <summary>
    /// Gets or sets ScrollY value.
    /// </summary>
    public double ScrollY { get; set; }
}

```

By default, ParallaxView control supports `Xamarin.Forms ScrollView`. For custom controls you need to implement the `IParallaxView` interface.

You can achieve the parallax scroll support to the custom controls using the native renderers to calculate the total size of the scrollable content. Refer to this [KB article](#) for more details.

## SfPdfViewer

### PDF Viewer

PDF Viewer for Xamarin allows the user to view PDF documents within your Xamarin.Forms application.



**Key features:**

The following list shows the key features available in PDF Viewer control.

- View PDF documents
- Search a text in PDF document
- Select text and copy it to clipboard
- Scroll, pan, zoom in and out
- Page navigation

---

**Note:** PDF Viewer for Xamarin.Forms.Android will be supported from Android 5.0 (API Level 21) onwards.

**Note:** PDF Viewer for Xamarin.Forms.iOS will be supported from iOS version 9.0 onwards.

---

## Getting started

This section demonstrates how to create an application that displays a PDF file using SfPdfViewer control.

### Adding SfPdfViewer reference

You can add SfPdfViewer reference using one of the following methods:

**Method 1: Adding SfPdfViewer reference from nuget.org**

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfPdfViewer). To add SfPdfViewer to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfPdfViewer](https://www.nuget.org/packages/Syncfusion.Xamarin.SfPdfViewer), and then install it.

!{Adding SfPdfViewer reference from NuGet}(pdfviewer\_images/Adding SfPdfViewer reference.png)

**Note:** Install the same version of SfPdfViewer NuGet in all the projects.

**Method 2: Adding SfPdfViewer reference from toolbox**

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfPdfViewer control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

**Method 3: Adding SfPdfViewer assemblies manually from the installed location**

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Compression.Portable.dll Syncfusion.Pdf.Portable.dll Syncfusion.SfPdfViewer.XForms.dll Syncfusion.SfBusyIndicator.XForms.dll Syncfusion.SfRangeSlider.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Compression.Portable.dll Syncfusion.Pdf.Portable.dll Syncfusion.SfPdfViewer.XForms.dll Syncfusion.SfPdfViewer.XForms.Android.dll Syncfusion.SfBusyIndicator.XForms.dll Syncfusion.SfBusyIndicator.XForms.Android.dll Syncfusion.SfBusyIndicator.Android.dll Syncfusion.SfRangeSlider.XForms.dll Syncfusion.SfRangeSlider.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Compression.Portable.dll Syncfusion.Pdf.Portable.dll Syncfusion.SfPdfViewer.XForms.dll Syncfusion.SfPdfViewer.XForms.iOS.dll Syncfusion.SfBusyIndicator.XForms.dll Syncfusion.SfBusyIndicator.XForms.iOS.dll Syncfusion.SfBusyIndicator.iOS.dll Syncfusion.SfRangeSlider.XForms.dll Syncfusion.SfRangeSlider.XForms.iOS.dll

	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Compression.Portable.dll Syncfusion.Pdf.Portable.dll Syncfusion.SfPdfViewer.XForms.dll Syncfusion.SfPdfViewer.XForms.UWP.dll Syncfusion.SfRangeSlider.XForms.dll Syncfusion.SfRangeSlider.XForms.UWP.dll Syncfusion.SfShared.UWP.dll Syncfusion.SfInput.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Creating a simple PDF Viewer application

Create a new cross platform application for Xamarin.Forms.Portable in the Visual Studio with the project name "GettingStarted" and refer the above mentioned assemblies to the respective projects.

**Note:** If you are adding the references from toolbox, this step is not needed.

An additional step is required to render the SfPdfViewer control in iOS project. You need to call the Syncfusion.SfPdfViewer.XForms.iOS.SfPdfDocumentViewRenderer.Init() and Syncfusion.SfRangeSlider.XForms.iOS.SfRangeSliderRenderer.Init(); in the FinishedLaunching overridden method of the AppDelegate class after the Xamarin.Forms Framework has been initialized and before the LoadApplication is called, as demonstrated in the following code example.

### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    Syncfusion.SfPdfViewer.XForms.iOS.SfPdfDocumentViewRenderer.Init();
    Syncfusion.SfRangeSlider.XForms.iOS.SfRangeSliderRenderer.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

### Resolving issue when deploying an application in ReleaseMode in the UWP platform

There is a known Framework issue in the UWP platform when an application using custom control is deployed in Release Mode, it will not be rendered.

This can be resolved by initializing the SfPdfViewer related assemblies in **App.xaml.cs** in the UWP project. Refer to the following code snippet.

### C#

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    //Add the assemblies `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies that your app uses
    assembliesToInclude.Add(typeof(SfPdfDocumentViewRenderer).GetTypeInfo().Assembly);
    assembliesToInclude.Add(typeof(SfRangeSliderRenderer).GetTypeInfo().Assembly);
    //Replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Loading a PDF using MVVM binding

File handling with Xamarin.Forms can be done using embedded resources or writing against the native filesystem APIs. Please find more details in the below link:

<https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/files/>

Add a folder in the portable project with the name "Assets" and add the PDF document you need to display in the PDF viewer, here a PDF file named "GIS Succinctly.pdf" was used. In the properties of the PDF document set the build action property to be "Embedded Resource".

Add a folder in the portable project with the name "ViewModel" and include a class file with a name "PdfViewerViewModel" inside it. Add below code snippet in the "PdfViewerViewModel" class. Ensure to maintain the namespace as "GettingStarted".

### C#

```
using System.IO;
using System.Reflection;
using System.ComponentModel;
namespace GettingStarted
{
    class PdfViewerViewModel : INotifyPropertyChanged
    {
        private Stream m_pdfDocumentStream;
        /// <summary>
        /// An event to detect the change in the value of a property.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
        /// <summary>
        /// The PDF document stream that is loaded into the instance of the PDF
        /// viewer.
        /// </summary>
        public Stream PdfDocumentStream
        {
            get
```

```

{
    return m_pdfDocumentStream;
}
set
{
    m_pdfDocumentStream = value;
    NotifyPropertyChanged("PdfDocumentStream");
}
}
/// <summary>
/// Constructor of the view model class
/// </summary>
public PdfViewerViewModel()
{
    //Accessing the PDF document that is added as embedded resource as stream.
    m_pdfDocumentStream =
    typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("GettingStarted
    .Assets.GIS Succinctly.pdf");
}
private void NotifyPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
}
}
}

```

Add the following XAML code in the MainPage.xaml in the portable project.

- Includes the necessary namespace where the control class resides
- Set BindingContext of ContentPage to PdfViewerViewModel
- Includes PDF Viewer into the page
- Binds InputFileStream property of PDF Viewer control to PdfDocumentStream property of PdfViewerViewModel

#### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
ms"
>
<ContentPage.BindingContext>
<local:PdfViewerViewModel></local:PdfViewerViewModel>
</ContentPage.BindingContext>
<Grid x:Name="pdfViewGrid">
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding
PdfDocumentStream}"/>
</Grid>

```

---

</ContentPage>

---

The sample that illustrates loading a PDF in MVVM binding can be downloaded from the link below.

<http://www.syncfusion.com/downloads/support/directtrac/general/ze/GettingStarted-2072455774>

#### *Loading a PDF in code-behind*

In code-behind a PDF can be loaded to PdfViewer just by using the [LoadDocument](#) method without the need of the "PdfViewerViewModel" class described in the previous section. Use the following code snippet in the code-behind of the XAML page.

#### **C#**

```
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        Stream fileStream;
        public MainPage()
        {
            InitializeComponent();
        }
        protected override void OnAppearing()
        {
            base.OnAppearing();
            fileStream =
                typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("GettingStarted
                .Assets.GIS Succinctly.pdf");
            //Load the PDF
            pdfViewerControl.LoadDocument(fileStream);
        }
    }
}
```

In the XAML code described in the previous section, remove the binding of InputFileStream property of SfPdfViewer and the BindingContext of the page.

#### **XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:GettingStarted"
    x:Class="GettingStarted.MainPage"
    xmlns:syncfusion="clr-
    namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
    ms"
>
    <Grid x:Name="pdfViewGrid">
        <syncfusion:SfPdfViewer x:Name="pdfViewerControl" />
    </Grid>
</ContentPage>
```

The sample which illustrates loading a PDF in code-behind using LoadDocument method can be downloaded from the link below.

<http://www.syncfusion.com/downloads/support/directtrac/general/ze/GettingStarted-81946798>

### Unloading PDF document from the Viewer

The SfPdfViewer control allows you to unload the PDF document from the viewer, when the PDF document is not in use anymore. This releases the PDF document and all its associated resources of the application.

You need to call the Unload method of SfPdfViewer control as in the below code snippet to achieve the same.

#### C#

```
//Unloads the PDF document from the PDF viewer, freeing all the accessed resources.  
pdfViewerControl.Unload();
```

### How to get & set the current page number?

PDF viewer has a BindableProperty "PageNumber" using which the current page number can be retrieved and it can be set.

#### XML

```
<Entry Keyboard="Numeric" FontSize="18" x:Name="pageNumberEntry"  
HorizontalTextAlignment="Center" VerticalOptions="Center" Text="{Binding  
PageNumber, Source={x:Reference Name=pdfViewerControl}}"/>
```

On binding the Entry control to PageNumber property of the PDF viewer instance, the number of the current page being displayed in the PDF viewer will be displayed in the Entry control and the PDF viewer would navigate to the page number being entered in the same.

### How to get the total page number?

PDF viewer has a BindableProperty "PageCount" using which the total number of pages in the document can be retrieved.

#### XML

```
<Label x:Name="pageCountLabel" FontSize="18" VerticalTextAlignment="Center"  
HorizontalTextAlignment="Center" HorizontalOptions="FillAndExpand"  
VerticalOptions="Center" Text="{Binding PageCount, Source={x:Reference  
Name=pdfViewerControl}}"/>
```

On binding the Label to PageCount property of the PDF viewer instance, the total number of pages in the document can be displayed.

### How to navigate to next page and previous page?

PDF viewer has "GoToNextPageCommand" and "GoToPreviousPageCommand" which will navigate to the next page and previous page respectively.

#### XML

```
<Button x:Name="goToNextButton" Grid.Column="3"  
BackgroundColor="Transparent" Image="PageDown.png"  
HorizontalOptions="Center" VerticalOptions="Center" Command="{Binding  
GoToNextPageCommand, Source={x:Reference Name=pdfViewerControl}}"/>
```

```
<Button x:Name="goToPreviousButton" Grid.Column="4"
        BackgroundColor="Transparent" Image="PageUp.png" HorizontalOptions="Center"
        VerticalOptions="Center" Command="{Binding GoToPreviousPageCommand,
        Source={x:Reference Name=pdfViewerControl}}"/>
```

If the current page is already the last page of PDF document, then GoToNextPageCommand does not have any effect. Similarly, If the current page is already the first page of PDF document, then GoToPreviousPageCommand does not have any effect.

#### How to detect a page change?

PDF viewer has PageChanged event, which helps detecting changes in the page being displayed in the viewer.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" Grid.Row="1"
        PageChanged="pdfViewerControl_PageChanged"/>
```

#### C#

```
private void pdfViewerControl_PageChanged(object sender,
    Syncfusion.SfPdfViewer.XForms.PageChangedEventArgs args)
{
    int currentPageNumber = args.PageNumber;
}
```

The arguments of the PageChanged event contains details about to which page the document is navigated to.

#### How to get and set the zoom for SfPdfViewer?

PDF viewer has the BindableProperty ZoomPercentage that is used to retrieve and set the current zoom factor.

#### XML

```
<Entry Keyboard="Numeric" FontSize="18" x:Name="zoomPercentage"
        HorizontalTextAlignment="Center" VerticalOptions="Center" Text="{Binding
        ZoomPercentage, Source={x:Reference Name=pdfViewerControl}}"/>
```

On binding the entry control to ZoomPercentage property of the PDF viewer instance, the current zoom percentage being displayed in the PDF viewer is displayed in the entry control, and the PDF viewer would be zoomed based on the value entered.

#### How to get and set Horizontal and Vertical Offsets in PDF Viewer?

Navigate to the specified vertical and horizontal offset values in PDF Viewer using ScrollToOffset (HorizontalOffset and VerticalOffset) methods and you can also retrieve the current horizontal and vertical offset position by using HorizontalOffset and VerticalOffset properties respectively in [SfPdfViewer](#) class.

#### C#

```
//Retrieves the current horizontal offset of the PdfViewerControl
m_currentHorizontalOffset = pdfViewerControl.HorizontalOffset;
//Retrieves the current vertical offset of the PdfViewerControl
```



```
m_currentVerticalOffset = pdfViewerControl.VerticalOffset;  
//Scrolls the content to the specified vertical offset position in the PdfViewerControl  
pdfViewerControl.ScrollToOffset(m_currentHorizontalOffset+10,  
m_currentVerticalOffset+10);
```

## AutomationId

The SfPdfViewer control has built-in **AutomationId** for inner elements. The **AutomationId** API allows the automation framework to find and interact with the inner elements of the SfPdfViewer control. The following screenshots illustrate the AutomationIds of SfPdfViewer's inner elements.

### Top toolbar

![Top Toolbar](pdfviewerimages/automationid/Top Toolbar.png)

### Bottom toolbar

![Bottom Toolbar](pdfviewerimages/automationid/Bottom Toolbar.png)

### Search toolbar

![Search Toolbar](pdfviewerimages/automationid/Search Toolbar.png)

### Annotation toolbar

![Annotation Toolbar](pdfviewerimages/automationid/Annotation Toolbar.png)

### Text markup annotation toolbar

![Text markup annotation Toolbar](pdfviewerimages/automationid/Text Markups Toolbar.png)

### Highlight annotation toolbar

![Highlight annotation Toolbar](pdfviewerimages/automationid/Highlight Toolbar.png)

### Underline annotation toolbar

![Underline annotation Toolbar](pdfviewerimages/automationid/Underline Toolbar.png)

### Strikethrough annotation toolbar

![Strikethrough annotation Toolbar](pdfviewerimages/automationid/Strikethrough Toolbar.png)

### Shape annotation toolbar

![Shape annotation Toolbar](pdfviewerimages/automationid/Shapes Toolbar.png)

### Rectangle annotation toolbar

![Rectangle annotation Toolbar](pdfviewerimages/automationid/Rectangle Toolbar.png)

### Ellipse annotation toolbar

![Ellipse annotation Toolbar](pdfviewerimages/automationid/Ellipse Toolbar.png)

### Line annotation toolbar

![Line annotation Toolbar](pdfviewerimages/automationid/Line Toolbar.png)

### Arrow annotation toolbar

![Arrow annotation Toolbar](pdfviewerimages/automationid/Arrow Toolbar.png)

**Ink annotation toolbar**

![[Ink annotation Toolbar](pdfviewerimages/automationid/Ink Toolbar.png)]

**Free text annotation toolbar**

![[Free text annotation Toolbar](pdfviewerimages/automationid/Free Text Toolbar.png)]

**Thickness selection toolbar**

![[Thickness selection Toolbar](pdfviewerimages/automationid/Thickness Toolbar.png)]

**Color selection toolbar**

![[Color selection Toolbar](pdfviewerimages/automationid/Color Picker.png)]

**Ink Annotation - Undo, Redo toolbar**

![[Ink Annotation Undo Redo Toolbar](pdfviewerimages/automationid/Undo Redo Ink.png)]

**Signature pad**

![[Signature Pad](pdfviewerimages/automationid/Signature Pad.png)]

**View mode toolbar**

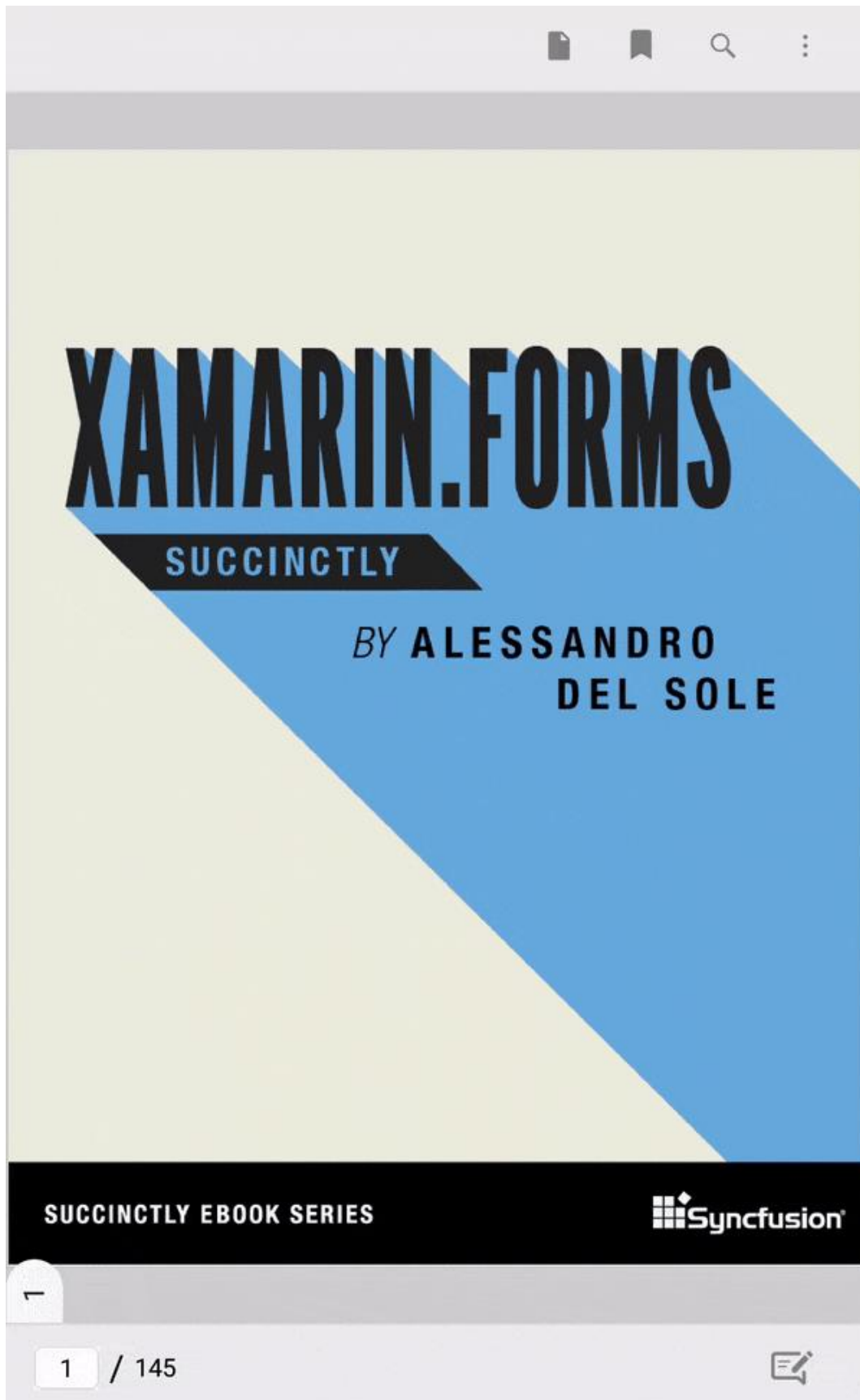
![[View mode Toolbar](pdfviewerimages/automationid/Flipview Mode.png)]

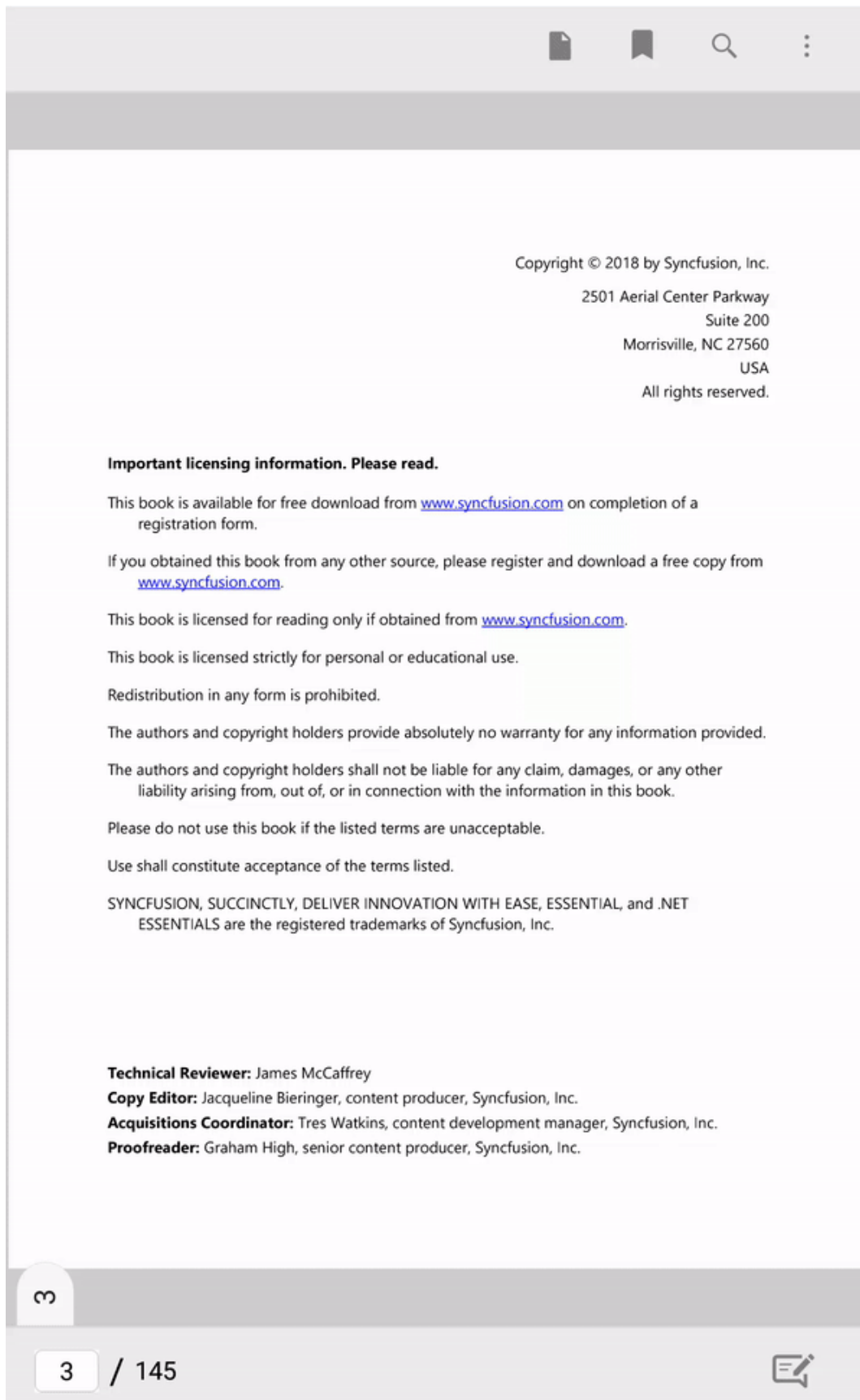
**More options toolbar**

![[More options Toolbar](pdfviewerimages/automationid/More Options Toolbar.png)]

**[Single page view mode](#)**

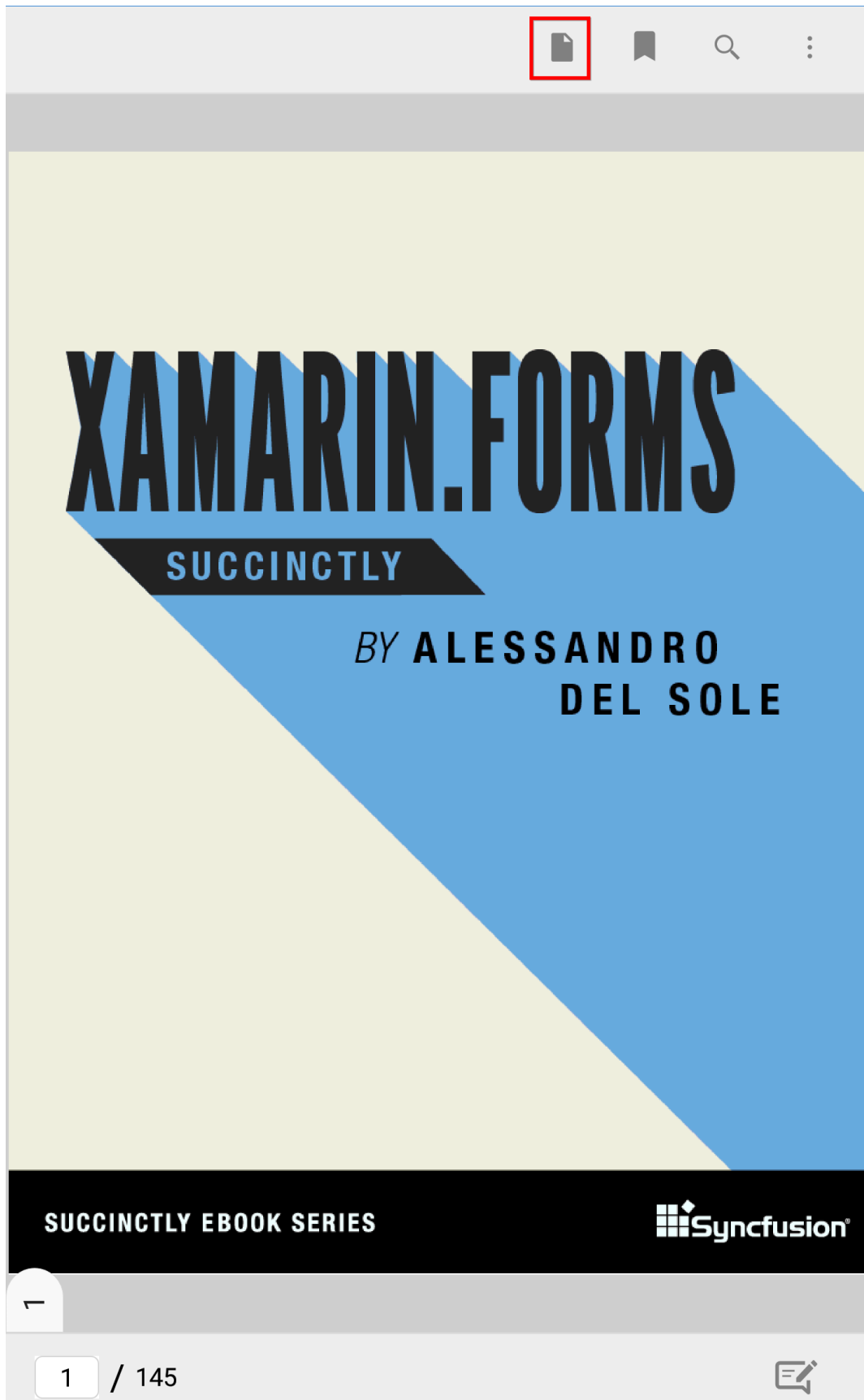
The default continuous view mode of PDF Viewer allows vertical scrolling. In addition to that, PDF Viewer also provides options to view PDFs page by page with horizontal navigation support.

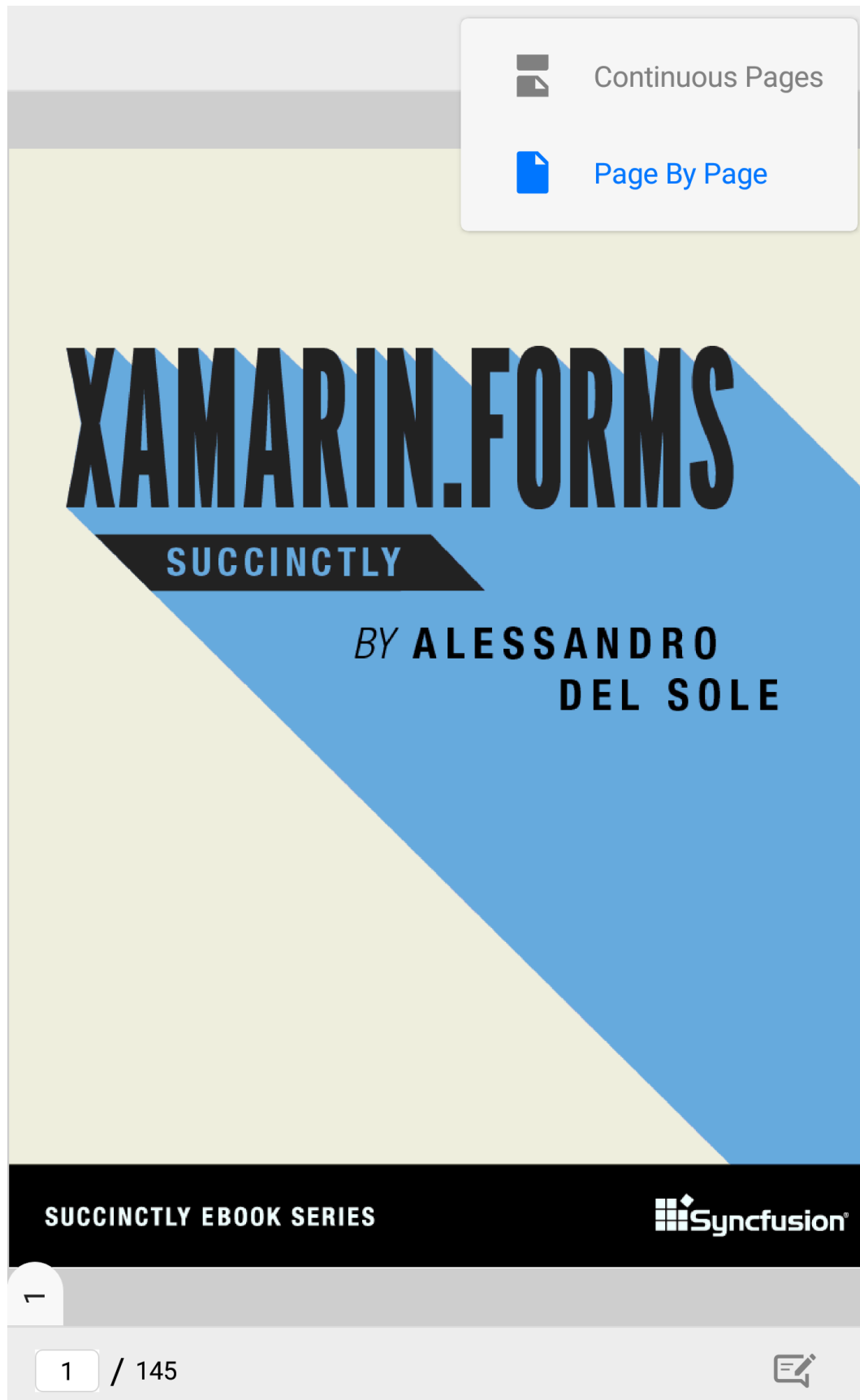




### Switching between view modes

The view mode can be switched by choosing the corresponding option from the menu bar that appears when the view mode button is clicked. The view mode button is available on the top toolbar.





The view mode can also be changed programmatically using the `PageViewMode` property of the PDF viewer. The enum `PageViewMode` has two constants - `Continuous`, `PageByPage`. The default value is `Continuous`. The below code snippet illustrates how to switch to the page by page view mode.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" PageViewMode="PageByPage" />
```

#### C#

```
pdfViewerControl.PageViewMode = PageViewMode.PageByPage;
```

#### Tracking the changes in the 'PageViewMode' property

When the `PageViewMode` property changes, the `PageViewModeChanged` event is raised. The second parameter of the event handler is of type `PageViewModeChangedEventArgs` which provides the details about the old and new view modes.

#### C#

```
pdfViewerControl.PageViewModeChanged +=
PdfViewerControl_PageViewModeChanged1;
private void PdfViewerControl_PageViewModeChanged1(object sender,
PageViewModeChangedEventArgs e)
{
    PageViewMode oldViewMode = e.PreviousPageViewMode;
    PageViewMode newViewMode = e.CurrentPageViewMode;
}
```

#### Designing a custom toolbar

The built-in toolbar of the PDF Viewer can be disabled and a custom toolbar can be used at the sample. The below code snippet demonstrates how to create a simple toolbar for PDF Viewer with the options such as current page number, total page count in the PDF document, go to previous page and go to next page.

In this project, two images have been used with the names - "PageDown.png" and "PageUp.png", place the images in the below mentioned location within the respective projects.

OS	Location
Android	Resources/drawable
iOS	Resources
UWP	Directly in the project location

The MainPage.xaml would be as mentioned below.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```



```

xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
ms"
>
<ContentPage.BindingContext>
<local:PdfViewerViewModel></local:PdfViewerViewModel>
</ContentPage.BindingContext>
<Grid x:Name="mainGrid">
<Grid.RowDefinitions>
<RowDefinition Height="50" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<AbsoluteLayout>
<Grid x:Name="toolbar" Grid.Row="0" BackgroundColor="#E9E9E9"
HorizontalOptions="Fill" VerticalOptions="Fill">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="1*" />
<ColumnDefinition Width="0.4*" />
<ColumnDefinition Width="1*" />
<ColumnDefinition Width="1*" />
<ColumnDefinition Width="1*" />
</Grid.ColumnDefinitions>
<Entry Keyboard="Numeric" FontSize="18" x:Name="pageNumberEntry"
HorizontalTextAlignment="Center" Grid.Column="0" VerticalOptions="Center"
Text="{Binding PageNumber, Source={x:Reference Name=pdfViewerControl}}"/>
<Label Text="/" Grid.Column="1" FontSize="18" x:Name="slashLabel"
VerticalTextAlignment="Center" HorizontalTextAlignment="Center"
HorizontalOptions="FillAndExpand" VerticalOptions="Center"/>
<Label x:Name="pageCountLabel" Grid.Column="2" FontSize="18"
VerticalTextAlignment="Center" HorizontalTextAlignment="Center"
HorizontalOptions="FillAndExpand" VerticalOptions="Center" Text="{Binding
PageCount, Source={x:Reference Name=pdfViewerControl}}"/>
<Button x:Name="goToNextButton" Grid.Column="3"
BackgroundColor="Transparent" Image="PageDown.png"
HorizontalOptions="Center" VerticalOptions="Center" Command="{Binding
GoToNextPageCommand, Source={x:Reference Name=pdfViewerControl}}"/>
<Button x:Name="goToPreviousButton" Grid.Column="4"
BackgroundColor="Transparent" Image="PageUp.png" HorizontalOptions="Center"
VerticalOptions="Center" Command="{Binding GoToPreviousPageCommand,
Source={x:Reference Name=pdfViewerControl}}"/>
</Grid>
</AbsoluteLayout>
<Grid x:Name="pdfViewGrid" Grid.Row="1">
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding
PdfDocumentStream}"/>
</Grid>
</Grid>
</ContentPage>

```

The final output will look like the below on iOS, Android and Windows (UWP) devices



This demo can be downloaded from the below link.

<http://www.syncfusion.com/downloads/support/directtrac/general/ze/GettingStarted-1533352385>

### Working with built-in toolbar

The SfPdfViewer has a built-in toolbar that has provisions to perform majority of the operations in the PDF Viewer and that can be disabled or enabled. You can disable the built-in toolbar and develop your own toolbar.

#### How to disable/enable built-in toolbar

By default, the built-in toolbar will be enabled. It can be disabled by setting the `Toolbar.Enabled` property of PDF Viewer to false.

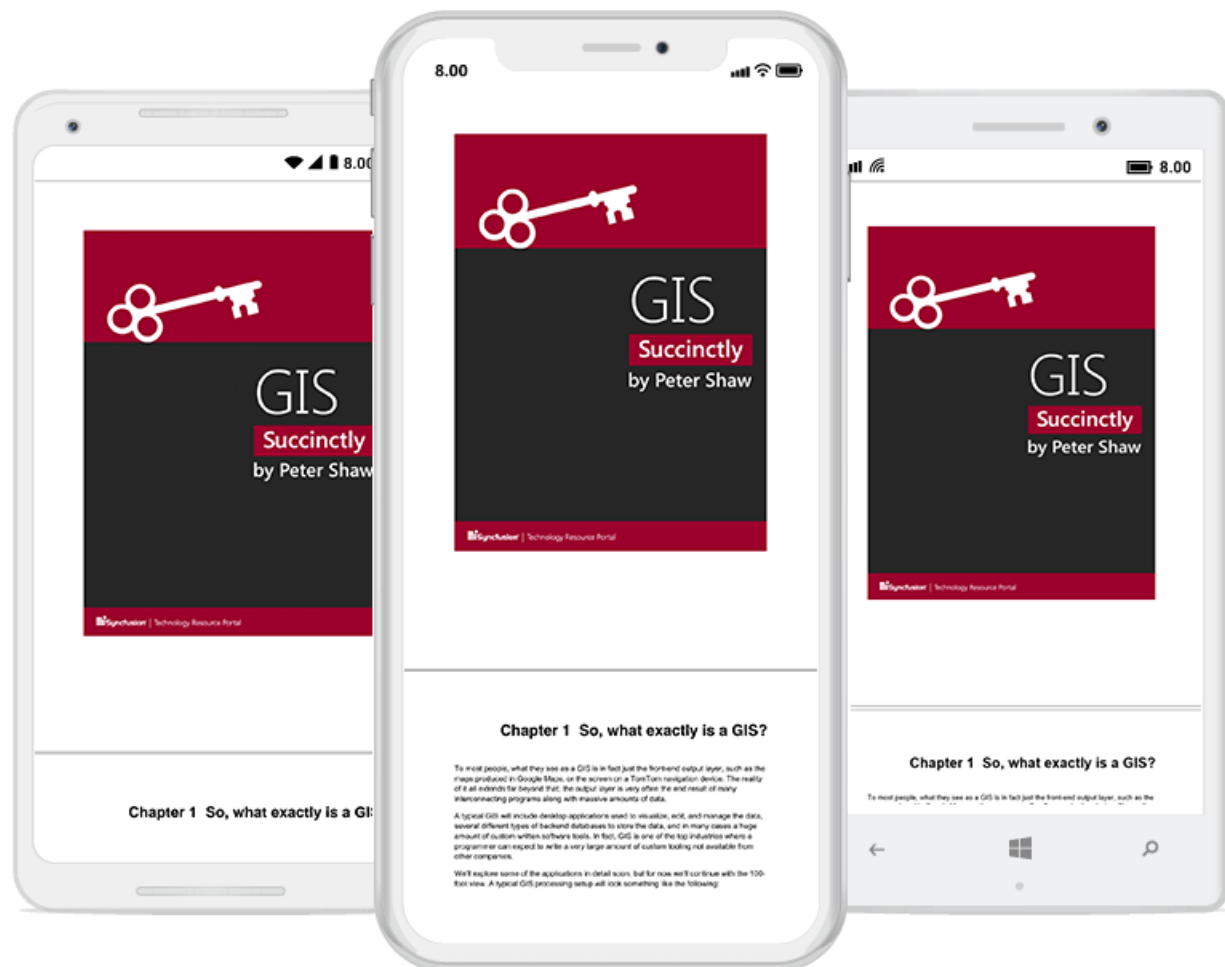
#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace: PdfViewerGettingStarted "
xmlns:syncfusion ="clr-
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
ms"
x:Class=" PdfViewerGettingStarted.MainPage">
<ContentPage.Resources>
```

```
<ResourceDictionary>
<syncfusion:Toolbar x:Key="ToolbarSettings">
<syncfusion:Toolbar.Enabled>>false
</syncfusion:Toolbar.Enabled>
</syncfusion:Toolbar>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" Toolbar = "{StaticResource
ToolbarSettings}" />
</ContentPage.Content>
</ContentPage>
```

## C#

```
pdfViewerControl.Toolbar.Enabled = false;
```



The toolbar can be enabled by setting the `Toolbar.Enabled` property to true.

## XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace: PdfViewerGettingStarted "
xmlns:syncfusion ="clr-
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
ms"
x:Class=" PdfViewerGettingStarted.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<syncfusion:Toolbar x:Key="ToolbarSettings">
<syncfusion:Toolbar.Enabled>true
</syncfusion:Toolbar.Enabled>
</syncfusion:Toolbar>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" Toolbar = "{StaticResource
ToolbarSettings}"/>
</ContentPage.Content>
</ContentPage>
```

### C#

```
pdfViewerControl.Toolbar.Enabled = true;
```




---

**Note:** By default, the built-in toolbar of SfPdfViewer is always visible.

---

#### How to disable/enable bookmark

By default, the bookmark button will be enabled. It can be disabled by setting the `BookmarkNavigationEnabled` property of PDF Viewer to false.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace: PdfViewerGettingStarted "
xmlns:syncfusion ="clr-
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
ms"
x:Class=" PdfViewerGettingStarted.MainPage">
<ContentPage.Content>
<syncfusion:SfPdfViewer x:Name="pdfViewerControl"
BookmarkNavigationEnabled="False"/>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
//Bookmark feature is disabled
pdfViewerControl.BookmarkNavigationEnabled = false;
```

### Search for a text instance

PDF Viewer provides support to find and highlight texts in the PDF document and it provides the following commands to perform text search operation.

Command/Binding property	Action
SearchTextCommand	Command that is executed to search for the specified text instance that is provided as the command parameter.
CancelSearchTextCommand	Command that is executed to cancel the text search operation when it is in progress and used to clear the highlighted text instances when the text search is completed.
SearchNextTextCommand	Command that is executed to navigate to the next text instance being searched.
SearchPreviousTextCommand	Command that is executed to navigate to the previous text instance of the text.

### How to initiate text search?

SearchTextCommand is used to initiate the text search, it takes the text to be searched as a parameter. The command searches for the text in the current page and highlights all the instances of the texts in complete document.

#### XML

```
<Entry x:Name="textSearchEntry" FontSize="18"
HorizontalTextAlignment="Center" HorizontalOptions="Fill"
VerticalOptions="Center"/>
<Button x:Name="searchTextButton" BackgroundColor="Transparent"
Image="SearchIcon.png" HorizontalOptions="Start" Command="{Binding
SearchTextCommand, Source={x:Reference Name=pdfViewerControl}}"
CommandParameter="{Binding Source={x:Reference textSearchEntry},
Path=Text}"/>
```

Here the text entered in the textSearchEntry control is provided as the parameter to the SearchTextCommand.

### How to identify if there is no instance of the text being searched?

SearchCompleted event of the PDF viewer can be used to identify if no instance of the searched text is found in the PDF document. This event has an event argument "NoMatchFound", when this it is true it means that the document does not have any match to the text searched and vice versa.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding
PdfDocumentStream}" SearchCompleted="pdfViewerControl_SearchCompleted" />
```

## C#

```
private void pdfViewerControl_SearchCompleted(object sender,
TextSearchCompletedEventArgs args)
{
    bool isNoMatchFound = args.NoMatchFound;
}
```

How to navigate to the next instance of the text?

SearchNextTextCommand is used to navigate to the next text instances in the PDF document. This command will highlight the current instance of the text with dark blue and all other instances in light blue. On executing the command continuously, the highlighted instance will navigate down the pages.

## XML

```
<Entry x:Name="textSearchEntry" FontSize="18"
HorizontalTextAlignment="Center" HorizontalOptions="Fill"
VerticalOptions="Center"/>
<Button x:Name="searchTextButton" BackgroundColor="Transparent"
Image="SearchIcon.png" HorizontalOptions="Start" Command="{Binding
SearchNextTextCommand, Source={x:Reference Name=pdfViewerControl}}"
CommandParameter="{Binding Source = {x:Reference textSearchEntry},
Path=Text}"/>
```

This command works both with and without a parameter. When a text parameter is provided to this command, it searches for the text provided as the parameter, when no parameter is sent to this command, it searches for the text that is provided as the parameter to the SearchTextCommand.

How to navigate to the previous instance of the text?

SearchPreviousTextCommand is used to navigate to the previous text instances in the PDF document. This command will highlight the current instance of the text with dark blue and all other instances in light blue. On executing the command continuously, the highlighted instance will navigate upwards in the pages.

## XML

```
<Entry x:Name="textSearchEntry" FontSize="18"
HorizontalTextAlignment="Center" HorizontalOptions="Fill"
VerticalOptions="Center"/>
<Button x:Name="searchTextButton" BackgroundColor="Transparent"
Image="SearchIcon.png" HorizontalOptions="Start" Command="{Binding
SearchPreviousTextCommand, Source={x:Reference Name=pdfViewerControl}}"
CommandParameter="{Binding Source = {x:Reference textSearchEntry},
Path=Text}"/>
```

This command works both with and without a parameter. When a text parameter is provided to this command, it searches for the text provided as the parameter, when no parameter is sent to this command, it searches for the text that is provided as the parameter to the SearchTextCommand.

How to identify if a complete cycle of text search is performed?

The first instance of the text in the page the text search is initiated is the first instance of the text, this first instance would be reset when we manually scroll the PDF document while navigating using the SearchNextTextCommand or SearchPreviousTextCommand.

When the search happens to hit the first instance again, the `NoMoreOccurrence` property of the `TextSearchEventArgs` of `SearchCompleted` event will be set to true.

For instance, if the text search is initiated at page 10 of a PDF document, the first instance in the page 10 will be considered as the first instance, on the occurrence of the same instance again in the cycle of text search `NoMoreOccurrence` property will be set to true.

### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding PdfDocumentStream}" SearchCompleted="pdfViewerControl_SearchCompleted" />
```

### C#

```
private void pdfViewerControl_SearchCompleted(object sender,
TextSearchEventArgs args)
{
    bool isNoMoreOccurrenceFound = args.NoMoreOccurrence;
}
```

### How to cancel text search?

`CancelSearchTextCommand` is used to cancel the text search progress. When the text search is in progress this command can be used to cancel the same, when text search is completed, this command can be used to clear all the highlighted texts in the PDF viewer.

### XML

```
<Button x:Name="cancelSearchButton" Grid.Column="3"
BackgroundColor="Transparent" Image="CancelSearch.png"
HorizontalOptions="Start" VerticalOptions="Center" Command="{Binding
CancelSearchTextCommand, Source={x:Reference Name=pdfViewerControl}}"/>
```

### How to track search progress?

Text search progress can be tracked using the `TextSearchInitiated` event and `TextMatchFound` event.

#### *SearchInitiated*

`TextSearchInitiated` event will be triggered soon after the call of the `SearchTextCommand`, the event argument of this event will contain details about the text being searched.

#### *TextMatchFound*

`TextMatchFound` event will be triggered once when the match of the text is found in the page of the PDF document. When text search is triggered using `SearchTextCommand` this event would be triggered for every page in which the match is found, when text search is triggered using `SearchPreviousCommand` or `SearchNextCommand` this event would be triggered for every match being found.

#### *SearchCompleted*

`TextSearchCompleted` event will be triggered once when all the pages are being search for the match of the input text at-least once. The event argument would contain properties `NoMatchFound` and `NoMoreOccurrenceFound`.

When `NoMatchFound` is set to true, it means that the PDF viewer could not find any match for the searched text in the PDF document.



When NoMoreOccurrence is set to true, it means that the PDF viewer had completed one full cycle of search and has hit the first instance again.

#### Implementing search bar with search features.

With the continuation of the getting started sample, you can extend the UI design to perform the text search in the PDF Viewer. Design the search toolbar in parallel to the main toolbar, here when the main toolbar is visible, search bar will be invisible and vice versa.

- Main toolbar – A new option to initiate text search toolbar will be added to the existing options.
- Search toolbar - A new and separate toolbar is created to search a text instance – it includes option to enter the text, perform search, cancel search and navigate back to the main toolbar.

In the PdfViewerViewModel class, create a new command (named – SearchAndToolbarToggleCommand) to toggle between the main toolbar and search toolbar. The view model class after the creation of the command and necessary boolean variables to toggle the visibility will be like

#### C#

```
using System.IO;
using System.Reflection;
using System.ComponentModel;
using System.Windows.Input;
using Xamarin.Forms;
namespace GettingStarted
{
    class PdfViewerViewModel : INotifyPropertyChanged
    {
        private Stream m_pdfDocumentStream;
        private bool m_isToolbarVisible = true;
        private bool m_isSearchBarVisible = false;
        /// <summary>
        /// An event to detect the change in the value of a property.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
        /// <summary>
        /// Command used to toggle between search bar and document toolbar.
        /// </summary>
        public ICommand SearchAndToolbarToggleCommand { get; set; }
        /// <summary>
        /// The PDF document stream that is loaded into the instance of the PDF
        viewer.
        /// </summary>
        public Stream PdfDocumentStream
        {
            get
            {
                return m_pdfDocumentStream;
            }
            set
            {
                m_pdfDocumentStream = value;
                NotifyPropertyChanged("PdfDocumentStream");
            }
        }
    }
}
```

```
/// <summary>
/// Gets or sets the bool value which will toggle the visibility of the
/// toolbar.
/// </summary>
/// <remarks>
/// The value true will make the toolbar visible and false would make the
/// toolbar invisible.
/// </remarks>
public bool IsToolbarVisible
{
    get { return m_isToolbarVisible; }
    set
    {
        if (m_isToolbarVisible == value)
            return;
        m_isToolbarVisible = value;
        NotifyPropertyChanged("IsToolbarVisible");
    }
}
/// <summary>
/// Gets or sets the bool value which will toggle the visibility of the
/// search bar.
/// </summary>
/// <remarks>
/// The value true will make the toolbar visible and false would make the
/// search bar invisible.
/// </remarks>
public bool IsSearchBarVisible
{
    get { return m_isSearchBarVisible; }
    set
    {
        if (m_isSearchBarVisible == value)
            return;
        m_isSearchBarVisible = value;
        NotifyPropertyChanged("IsSearchBarVisible");
    }
}
/// <summary>
/// Constructor of the view model class
/// </summary>
public PdfViewerViewModel()
{
    //Accessing the PDF document that is added as embedded resource as stream.
    m_pdfDocumentStream =
        typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("GettingStarted
        .Assets.GIS Succinctly.pdf");
    //Command used to toggle the visibility of the toolbar and search bar.
    SearchAndToolbarToggleCommand= new
        Command<object>(OnSearchAndToolbarToggleCommand, CanExecute);
}
private void NotifyPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

```

}
private bool CanExecute(object parameter)
{
    return true;
}
/// <summary>
/// Method used to toggle the visibility of the toolbar and search bar.
/// </summary>
private void OnSearchAndToolbarToggleCommand(object destinationPageParam)
{
    IsToolbarVisible = !IsToolbarVisible;
    IsSearchBarVisible = !IsToolbarVisible;
}
}
}

```

The complete XAML code after the design of the search bar will look like below

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
ms"
>
<ContentPage.BindingContext>
<local:PdfViewerViewModel></local:PdfViewerViewModel>
</ContentPage.BindingContext>
<Grid x:Name="mainGrid">
<Grid.RowDefinitions>
<RowDefinition Height="50" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<AbsoluteLayout>
<Grid x:Name="toolbar" Grid.Row="0" BackgroundColor="#E9E9E9"
HorizontalOptions="Fill" VerticalOptions="Fill" IsVisible="{Binding
IsGridVisible}">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="1*" />
<ColumnDefinition Width="0.4*" />
<ColumnDefinition Width="1*" />
<ColumnDefinition Width="1*" />
<ColumnDefinition Width="1*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="1*" />
</Grid.ColumnDefinitions>
<Entry Keyboard="Numeric" FontSize="18" x:Name="pageNumberEntry"
HorizontalTextAlignment="Center" Grid.Column="0" VerticalOptions="Center"
Text="{Binding PageNumber, Source={x:Reference Name=pdfViewerControl}}"/>
<Label Text="/" Grid.Column="1" FontSize="18" x:Name="slashLabel"
VerticalTextAlignment="Center" HorizontalTextAlignment="Center"
HorizontalOptions="FillAndExpand" VerticalOptions="Center"/>

```

```

<Label x:Name="pageCountLabel" Grid.Column="2" FontSize="18"
VerticalTextAlignment="Center" HorizontalTextAlignment="Center"
HorizontalOptions="FillAndExpand" VerticalOptions="Center" Text="{Binding
PageCount, Source={x:Reference Name=pdfViewerControl}}"/>
<Button x:Name="goToNextButton" Grid.Column="3"
BackgroundColor="Transparent" Image="PageDown.png"
HorizontalOptions="Center" VerticalOptions="Center" Command="{Binding
GoToNextPageCommand, Source={x:Reference Name=pdfViewerControl}}"/>
<Button x:Name="goToPreviousButton" Grid.Column="4"
BackgroundColor="Transparent" Image="PageUp.png" HorizontalOptions="Center"
VerticalOptions="Center" Command="{Binding GoToPreviousPageCommand,
Source={x:Reference Name=pdfViewerControl}}"/>
<Button x:Name="searchButton" Grid.Column="6" BackgroundColor="Transparent"
Image="SearchIcon.png" HorizontalOptions="Start" Command="{Binding
SearchAndToolbarToggleCommand}"/>
</Grid>
<Grid x:Name="searchBar" Grid.Row="0" BackgroundColor="#E9E9E9"
HorizontalOptions="Fill" VerticalOptions="Fill" IsVisible="{Binding
IsSearchBarVisible}">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="5*" />
<ColumnDefinition Width="1.5*" />
<ColumnDefinition Width="1.5*" />
<ColumnDefinition Width="1.5*" />
</Grid.ColumnDefinitions>
<Button x:Name="backIcon" Grid.Column="0" BackgroundColor="Transparent"
Image="BackIcon.png" HorizontalOptions="Start" VerticalOptions="Center"
Command="{Binding SearchAndToolbarToggleCommand}"/>
<Entry Grid.Column="1" x:Name="textSearchEntry" FontSize="18"
HorizontalTextAlignment="Center" HorizontalOptions="Fill"
VerticalOptions="Center"/>
<Button x:Name="searchTextButton" Grid.Column="2"
BackgroundColor="Transparent" Image="SearchIcon.png"
HorizontalOptions="Start" Command="{Binding SearchTextCommand,
Source={x:Reference Name=pdfViewerControl}}" CommandParameter="{Binding
Source={x:Reference textSearchEntry}, Path=Text}"/>
<Button x:Name="cancelSearchButton" Grid.Column="3"
BackgroundColor="Transparent" Image="CancelSearch.png"
HorizontalOptions="Start" VerticalOptions="Center" Command="{Binding
CancelSearchTextCommand, Source={x:Reference Name=pdfViewerControl}}"/>
</Grid>
</AbsoluteLayout>
<Grid x:Name="pdfViewGrid" Grid.Row="1">
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding
PdfDocumentStream}"/>
</Grid>
</Grid>
</ContentPage>

```

### Customizing search highlight color

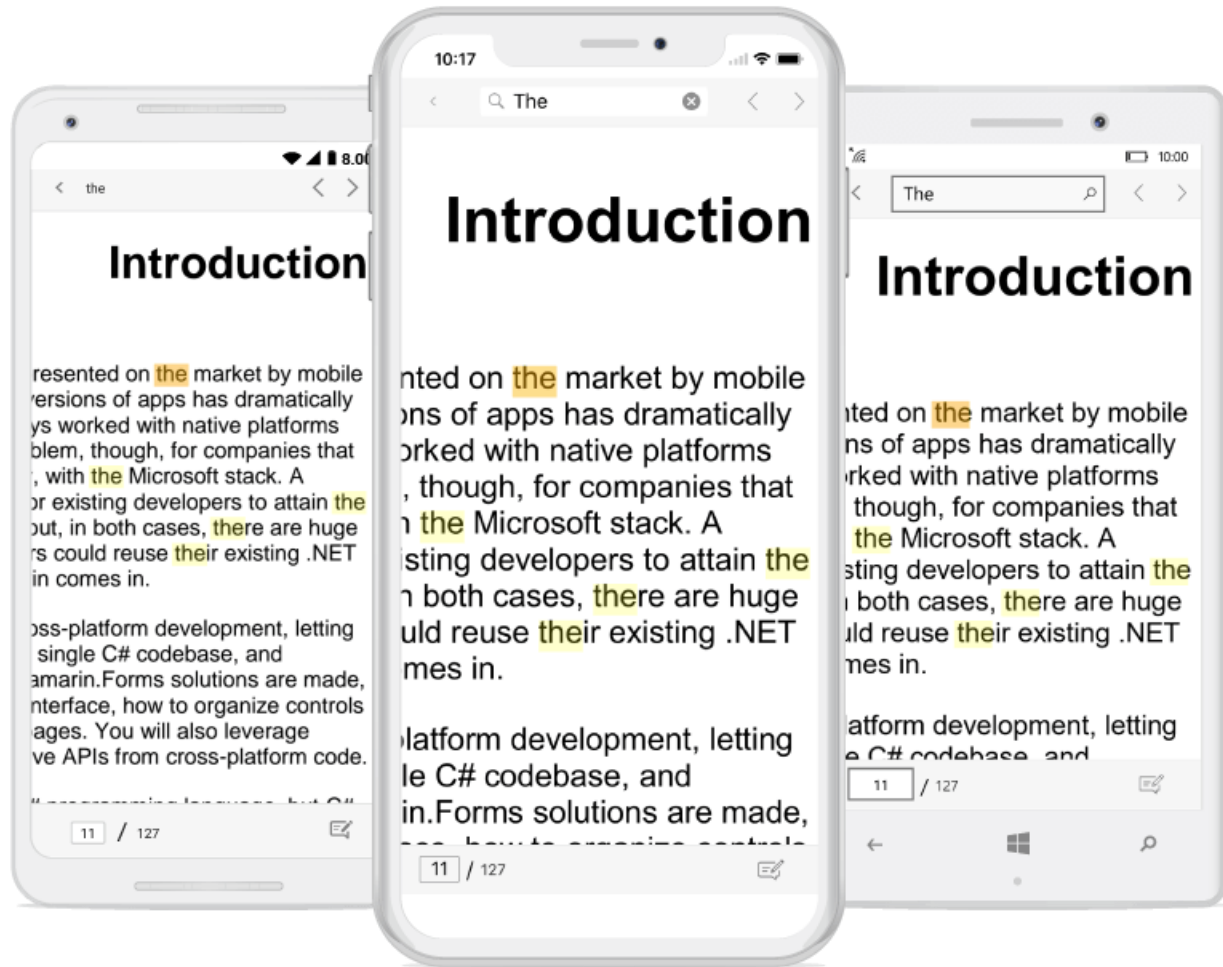
The colors in which the current instance and other instances are highlighted can be customized. The `TextSearchSettings` class in the PDF viewer instance can be used for this purpose.

### C#

```
pdfViewerControl.TextSearchSettings.OtherInstanceColor = Color.FromRgba(255, 255, 0, 50);  
pdfViewerControl.TextSearchSettings.CurrentInstanceColor =  
Color.FromRgba(255, 255, 0, 90);
```

## XML

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:local="clr-namespace:GettingStarted"  
x:Class="GettingStarted.MainPage"  
xmlns:syncfusion="clr-  
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor  
ms"  
>  
<ContentPage.Resources>  
<ResourceDictionary>  
<sfpdfviewer:TextSearchSettings x:Key="SearchSettings">  
<sfpdfviewer:TextSearchSettings.CurrentInstanceColor>  
<Color>#90FFFF00</Color>  
</sfpdfviewer:TextSearchSettings.CurrentInstanceColor>  
<sfpdfviewer:TextSearchSettings.OtherInstanceColor>  
<Color>#50FFFF00</Color>  
</sfpdfviewer:TextSearchSettings.OtherInstanceColor>  
</sfpdfviewer:TextSearchSettings>  
</ResourceDictionary>  
</ContentPage.Resources>  
<Grid x:Name="mainGrid">  
<syncfusion:SfPdfViewer x:Name="pdfViewerControl"  
TextSearchSettings="{StaticResource Key=SearchSettings}"/>  
</Grid>  
</ContentPage>
```



### Select and copy text

The PDF viewer supports text selection and copy feature, which allows user to select the text in the PDF document and copy it to the clipboard. This section illustrates about how to use this feature.

Property	Action
IsTextSelectionEnabled	Gets or sets the value that enables or disables the text selection feature in the PDF viewer. This property when set to true enables text selection and vice versa. By default, this property is set to true.

### XML

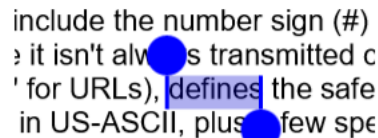
```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl"
IsTextSelectionEnabled="False"/>
```

### C#

```
//Text selection feature in the PDF viewer is disabled.
pdfViewerControl.IsTextSelectionEnabled = false;
```

### How to select text?

To select a text in the PDF page, press and hold any word of the text until selection bubbles appear at the top-left and bottom-right corners of its bounds. Then, use the left bubble to select the text at the left and top, and the right bubble to select the text at the right and bottom directions.



### How to enable or disable the context menu?

By default, PDF viewer comes with a context menu that will be displayed above the selected text in the PDF document, which has a button (option) to copy the selected text. The display of the context menu can be disabled by setting ShowContextMenu property of the TextSelectionSettings class to false. The below code illustrates the same. By default, context menu will be enabled in the PDF viewer.

#### C#

```
//The display of the default context menu for the text selection is disabled.
pdfViewerControl.TextSelectionSettings.ShowContextMenu = false;
```

### How to modify the selection and its handle color?

The color used for text selection and the color of the handle can be customized based on the developer's requirements. The properties TextSelectionColor and TextSelectionHandleColor of the TextSelectionSettings class can be used to customize them. The below code snippet illustrates the same.

#### C#

```
//Customizing the color being displayed while selecting the text from PDF document.
pdfViewerControl.TextSelectionSettings.TextSelectionColor =
Color.FromRgba(0, 0, 205, 80);
//Customizing the color of text selection handler displayed while selecting the text from PDF document.
pdfViewerControl.TextSelectionSettings.TextSelectionHandleColor =
Color.FromRgb(0, 0, 255);
```

### How to acquire selected text?

The completion of the text selection action would trigger TextSelectionCompleted event. The event argument of this event will contain a copy of the selected text in the String format.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding PdfDocumentStream}"
TextSelectionCompleted="PdfViewerControl_TextSelectionCompleted"/>
```

#### C#

```
private void PdfViewerControl_TextSelectionCompleted(object sender,
TextSelectionCompletedEventArgs args)
{
```

```
//The selected text is acquired and stored in the variable selectedText.  
string selectedText = args.SelectedText;  
}
```

How to acquire page number, page bounds and selected region?

The completion of the text selection action would trigger TextSelectionCompleted event. The event argument would contain details about the page number in which the selection operation is performed, bounds of the page and the selection region.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding  
PdfDocumentStream}"  
TextSelectionCompleted="PdfViewerControl_TextSelectionCompleted"/>
```

#### C#

```
private void PdfViewerControl_TextSelectionCompleted(object sender,  
TextSelectionCompletedEventArgs args)  
{  
//The number of the page in which the selection is performed is acquired.  
int pageNumber = args.PageNumber;  
//The bounds of the page in which the selection is performed is acquired.  
Rectangle pageBounds= args.PageBounds;  
//The region of the text being selected is acquired.  
Rectangle selectedRegion= args.SelectedRegion;  
}
```

### Working with Hyperlink navigation

PDF viewer supports hyperlink navigation that detects hyperlinks and tapping on the hyperlink will open the URL in the browser.

How to disable hyperlink navigation in PDF document using PDF viewer control?

You can disable the hyperlink navigation by setting the "AllowHyperlinkNavigation" property of PDF viewer to false. By default, hyperlink navigation is enabled in PDF viewer.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl"  
AllowHyperlinkNavigation="False" InputFileStream="{Binding  
PdfDocumentStream}" />
```

How to acquire the properties of clicked URI from PDF viewer?

You can acquire the details of the hyperlink that is tapped in the PDF viewer control from the HyperlinkClickedEventArgs and HyperlinkClicked event. Refer to the following code snippet for more details.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl"  
HyperlinkClicked="pdfViewerControl_HyperlinkClicked"  
InputFileStream="{Binding PdfDocumentStream}" />
```



## C#

```
public void pdfViewerControl_HyperlinkClicked(object sender, HyperlinkClickedEventArgs args)
{
    // Gets or sets a value indicating whether the hyperlink navigation was handled.
    args.Handled = false;
    // Gets the hyperlink being clicked.
    string uri = args.Uri;
    // Gets the current page index of the hyperlink.
    int pageIndex = args.PageIndex;
    //Gets the bounds of the hyperlink is being clicked.
    Rectangle hyperlinkBound = args.Bounds;
}
```

## Working with Document Link Annotation (Table of content)

The PDF viewer navigates to a specific destination within the PDF document.

How to disable document link navigation in PDF document using PDF viewer control?

Set the "EnableDocumentLinkAnnotation" property to false to disable the document link navigation in PDF viewer.

## XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl"
    EnableDocumentLinkAnnotation="false" InputFileStream="{Binding PdfDocumentStream}" />
```

---

**Note:** By default, the EnableDocumentLinkAnnotation property is set to true.

---

## Working with ink annotation

PDF viewer allows you to include ink annotation in the PDF document and provides options to edit or remove the existing ink annotation in the PDF document.

### Inclusion of ink annotation

This section describes how to include ink annotation in PDF viewer control Xamarin.Forms.

### Enabling ink annotation mode

Set the `AnnotationMode` property of the PDF viewer object to `ink` to enable the ink annotation. By setting this, zooming and scrolling will be disabled and touch interactions will be converted into inks on the PDF pages. Refer to the following code.

## C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();
pdfViewer.AnnotationMode = AnnotationMode.Ink;
```

## XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"/>
<Button x:Name="inkAnnotationButton" Command="{Binding AnnotationModeCommand, Source={x:Reference Name=pdfViewer}}"
    CommandParameter="Ink" />
```

Use the following code to reset the annotation mode to none.

### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();  
pdfViewer.AnnotationMode = AnnotationMode.None;
```

### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"/>  
<Button x:Name="resetAnnotationButton" Command="{Binding  
AnnotationModeCommand, Source={x:Reference Name=pdfViewer}}"  
CommandParameter="None" />
```

### *Adding ink annotation*

Steps involved in adding ink annotation to the PDF document:

1. Switch to ink annotation mode. 2. Add inks to the pages of the PDF document, this will be recorded as a session. 3. You can perform undo and redo operations for every stroke made in the session. 4. You can also choose the color and thickness of the ink, here color and thickness are restricted to the complete ink and not individual strokes. 5. After adding ink annotation, you can either accept or discard the changes.

### *How to accept and include the ink annotation on to the PDF viewer?*

The following code can be used to accept and include the drawn ink annotation to the PDF viewer control.

### C#

```
pdfViewer.EndInkSession(true);
```

### *How to ignore/discard drawn ink annotation from the PDF viewer before inclusion?*

The following code can be used to ignore or discard the drawn ink annotation to the PDF viewer control.

### C#

```
pdfViewer.EndInkSession(false);
```

### *How to identify whether the ink annotation is included?*

You can identify whether the drawn ink annotation has been added to the PDF viewer control or not by using the **InkAdded** event. This event will be raised once the annotation is added.

### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer" InkAdded="PdfViewer_InkAdded"/>
```

### C#

```
private void PdfViewer_InkAdded(object sender, InkAddedEventArgs args)  
{  
    //Retrieves the bounds of the deselected annotation.  
    Rectangle bounds = args.Bounds;
```

```
//Retrieves the page number where the deselected annotation resides.  
int pageNumber = args.PageNumber;  
//Retrieves the color of the deselected annotation.  
Color color = args.Color;  
//Retrieves the thickness of the deselected annotation.  
float thickness = args.Thickness;  
//Retrieves the opacity of the deselected annotation.  
float opacity = args.Opacity;  
}
```

How to perform undo and redo operation during inking session?

You can perform undo and redo operations on the drawn ink annotations during the pre-conformance session.

The following code can be used to perform undo operation on the ink annotations drawn over the PDF viewer control.

#### **C#**

```
pdfViewer.UndoInk();
```

You can identify the undo operation in this session using the following event.

#### **XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer" CanUndoInkModified =  
"PdfViewer_CanUndoInkModified" />
```

#### **C#**

```
private void PdfViewer_CanUndoInkModified(object sender,  
CanUndoInkModifiedEventArgs args)  
{  
    bool canUndoInk = args.CanUndoInk;  
}
```

The following code can be used to perform redo operation on the ink annotations drawn over the PDF viewer control.

#### **C#**

```
pdfViewer.RedoInk();
```

You can identify the redo operation in this session using the following event.

#### **XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer" CanRedoInkModified =  
"PdfViewer_CanRedoInkModified" />
```

#### **C#**

```
private void PdfViewer_CanRedoInkModified(object sender,  
CanRedoInkModifiedEventArgs args)
```

```
{  
    bool CanRedoInk = args.CanRedoInk;  
}
```

### Deleting ink annotation

The PDF viewer can remove a selected annotation or all the annotations in the PDF document.

#### *Removing a selected annotation.*

The following code snippet illustrates removing a selected annotation from the PDF document.

#### **XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"  
    InkSelected="PdfViewer_InkSelected"/>
```

#### **C#**

```
private void PdfViewer_InkSelected(object sender, InkSelectedEventArgs args)  
{  
    //Casts the sender object as Ink annotation.  
    InkAnnotation selectedInkAnnotation = sender as InkAnnotation;  
    //Removes the selected annotation from the PDF viewer.  
    pdfViewer.RemoveAnnotation(selectedInkAnnotation);  
}
```

#### *How to identify the removal of ink annotation?*

You can identify removal of ink annotation using the **InkRemoved** event, which is available in the PDF viewer control. This event will be raised when you remove the ink annotation from the PDF viewer control.

#### **XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"  
    InkRemoved="PdfViewer_InkRemoved"/>
```

The following C# code shows how to acquire bounds, page number, color, thickness, and opacity of the removed ink annotation.

#### **C#**

```
private void PdfViewer_InkRemoved(object sender, InkRemovedEventArgs args)  
{  
    //Retrieves the bounds of the deselected annotation.  
    Rectangle bounds = args.Bounds;  
    //Retrieves the page number where the deselected annotation resides.  
    int pageNumber = args.PageNumber;  
    //Retrieves the color of the deselected annotation.  
    Color color = args.Color;  
    //Retrieves the thickness of the deselected annotation.  
    float thickness = args.Thickness;  
    //Retrieves the opacity of the deselected annotation.  
    float opacity = args.Opacity;  
}
```

## Working with ink annotation settings

You can customize the color, opacity, and thickness of the ink annotation.

### Changing the color

You can set the color of the ink annotation by using the `AnnotationSettings.Ink.Color` property. Refer to the following code sample.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace: PdfViewerGettingStarted "
xmlns:syncfusion ="clr-
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
ms"
x:Class=" PdfViewerGettingStarted.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<syncfusion:AnnotationSettings x:Key="InkAnnotationSettings">
<syncfusion:AnnotationSettings.Ink>
<Color>#FFFF0000</Color>
</syncfusion:AnnotationSettings.Ink>
</syncfusion:AnnotationSettings>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<syncfusion:SfPdfViewer x:Name="pdfViewer"
AnnotationSettings="{StaticResource InkAnnotationSettings}"/>
</ContentPage.Content>
</ContentPage>
```

#### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();
pdfViewer.AnnotationMode = AnnotationMode.Ink;
pdfViewer.AnnotationSettings.Ink.Color = Color.Red;
```

### Changing the opacity

You can set the opacity of the ink annotation by using the `AnnotationSettings.Ink.Opacity` property. Opacity value ranges from 0 to 1. Refer to the following code example.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace: PdfViewerGettingStarted "
xmlns:syncfusion ="clr-
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
ms"
x:Class=" PdfViewerGettingStarted.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<syncfusion:AnnotationSettings x:Key="InkAnnotationSettings">
<syncfusion:AnnotationSettings.Ink>
<Opacity>0.5</Opacity>
```

```

</syncfusion:AnnotationSettings.Ink>
</syncfusion:AnnotationSettings>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<syncfusion:SfPdfViewer x:Name="pdfViewer"
AnnotationSettings="{StaticResource InkAnnotationSettings}"/>
</ContentPage.Content>
</ContentPage>

```

## C#

```

SfPdfViewer pdfViewer = new SfPdfViewer();
pdfViewer.AnnotationMode = AnnotationMode.Ink;
pdfViewer.AnnotationSettings.Ink.Opacity = 0.5f;

```

### Changing the thickness

You can set the thickness of the ink annotation by using the `AnnotationSettings.Ink.Thickness` property. Refer to the following code example.

## XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace: PdfViewerGettingStarted "
xmlns:syncfusion ="clr-
namespace:Syncfusion.SfPdfViewer.XForms;assembly=Syncfusion.SfPdfViewer.XFor
ms"
x:Class=" PdfViewerGettingStarted.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<syncfusion:AnnotationSettings x:Key="InkAnnotationSettings">
<syncfusion:AnnotationSettings.Ink>
<Thickness>5</Thickness>
</syncfusion:AnnotationSettings.Ink>
</syncfusion:AnnotationSettings>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<syncfusion:SfPdfViewer x:Name="pdfViewer"
AnnotationSettings="{StaticResource InkAnnotationSettings}"/>
</ContentPage.Content>
</ContentPage>

```

## C#

```

SfPdfViewer pdfViewer = new SfPdfViewer();
pdfViewer.AnnotationMode = AnnotationMode.Ink;
pdfViewer.AnnotationSettings.Ink.Thickness = 5;

```

### How to select the ink annotation?

You can select the ink annotation by tapping the annotation.

### How to change the properties of ink annotation after selection?

You can select and change the properties of the ink annotation using the `InkSelected` event. This event will be raised when you select the ink annotation. This is illustrated in the following code sample.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
InkSelected="PdfViewer_InkSelected" />
```

You can change the properties of the selected annotation using the `InkSelectedEventArgs` object. The following code shows how to change the properties of the selected annotation in the `InkSelectedEventHandler`.

#### C#

```
private void pdfViewer_InkSelected(object sender, InkSelectedEventArgs args)
{
    //Casts the sender object as Ink annotation.
    InkAnnotation selectedInkAnnotation = sender as InkAnnotation;
    //Sets the color of the selected annotation using ink annotation settings.
    selectedInkAnnotation.Settings.Color = Color.Blue;
    //Sets the opacity of the selected annotation using ink annotation settings.
    selectedInkAnnotation.Settings.Opacity = 0.3f;
    //Sets the thickness of the selected annotation using ink annotation
    settings.
    selectedInkAnnotation.Settings.Thickness = 3;
}
```

You can identify whether the ink annotation is tapped or not using the `InkTapped` event. This event will be triggered if you tap the ink annotation whether to select or deselect the annotation.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer" InkTapped="PdfViewer_InkTapped"/>
```

#### C#

```
private void PdfViewer_InkTapped(object sender, InkTappedEventArgs args)
{
    //Retrieves the bounds of the deselected annotation.
    Rectangle bounds = args.Bounds;
    //Retrieves the page number where the deselected annotation resides.
    int pageNumber = args.PageNumber;
    //Retrieves the color of the deselected annotation.
    Color color = args.Color;
    //Retrieves the thickness of the deselected annotation.
    float thickness = args.Thickness;
    //Retrieves the opacity of the deselected annotation.
    float opacity = args.Opacity;
}
```

### How to deselect the ink annotation?

You can deselect the ink annotation by tapping on the selected annotation or on the PDF page. Deselection can be identified using the `InkDeselected` event.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
InkDeselected="PdfViewer_InkDeselected"/>
```

The following code shows how to identify the deselected annotation from the raised event.

#### C#

```
private void PdfViewer_InkDeselected(object sender, InkDeselectedEventArgs
args)
{
    //Retrieves the bounds of the deselected annotation.
    Rectangle bounds = args.Bounds;
    //Retrieves the page number where the deselected annotation resides.
    int page = args.PageNumber;
    //Retrieves the color of the deselected annotation.
    Color color = args.Color;
    //Retrieves the thickness of the deselected annotation.
    float thickness = args.Thickness;
    //Retrieves the opacity of the deselected annotation.
    float opacity = args.Opacity;
}
```

### How to move or resize the selected ink annotation?

Select and drag the annotation to move or resize the ink annotation.

#### How to identify whether the ink annotation is moved or resized?

The event `AnnotationMovedOrResized` will be raised when you move or resize the selected annotation.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
AnnotationMovedOrResized="PdfViewer_AnnotationMovedOrResized"/>
```

#### C#

```
private void PdfViewer_AnnotationMovedOrResized(object sender,
AnnotationMovedOrResizedEventArgs args)
{
    //Retrieves the old bounds of the annotation
    Rectangle oldBounds = args.OldBounds;
    //Retrieves the new bounds of the annotation
    Rectangle newBounds = args.NewBounds;
}
```

## Working with handwritten signatures

PDF viewer allows you to include handwritten signatures in PDF documents and provides options to modify or remove the existing ones.



## Adding handwritten signatures

### Enabling handwritten signature mode

Set the `AnnotationMode` property of the PDF viewer instance to `HandwrittenSignature`. After setting the signature mode the signature pad would appear on which the signature can be drawn. After drawing the signature, the `Done` button should be tapped to add the drawn signature to the PDF, you can use clear button to redraw the signature or close button to cancel signing.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"/>
<Button x:Name="signatureButton" Command="{Binding AnnotationModeCommand,
Source={x:Reference Name=pdfViewer}}"
CommandParameter="HandwrittenSignature" />
```

#### C#

```
pdfViewer.AnnotationMode = AnnotationMode.HandwrittenSignature;
```

### Disabling handwritten signature mode

Setting the `AnnotationMode` to `None` will disable the signature mode and would hide the signature pad.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"/>
<Button x:Name="resetAnnotationButton" Command="{Binding
AnnotationModeCommand, Source={x:Reference Name=pdfViewer}}"
CommandParameter="None" />
```

#### C#

```
pdfViewer.AnnotationMode = AnnotationMode.None;
```

## Customizing the appearance of handwritten signatures

You can customize the default values such as stroke color, opacity, and thickness of all signatures to be added. This will not affect the already added signatures.

### Setting the default stroke color

You can set the default stroke color of handwritten signatures by using the `SfPdfViewer.AnnotationSettings.HandwrittenSignature.Color` property. Refer to the following code.

#### C#

```
pdfViewer.AnnotationSettings.HandwrittenSignature.Color = Color.Red;
```

### Setting the default opacity

You can set the default opacity of handwritten signatures by using the `SfPdfViewer.AnnotationSettings.HandwrittenSignature.Opacity` property. Opacity value ranges from 0 to 1. Refer to the following code example.

#### C#

```
pdfViewer.AnnotationSettings.HandwrittenSignature.Opacity = 0.5f;
```

### Setting the default thickness

You can set the thickness of handwritten signatures by using the `SfPdfViewer.AnnotationSettings.HandwrittenSignature.Thickness` property. Refer to the following code example.

#### C#

```
pdfViewer.AnnotationSettings.HandwrittenSignature.Thickness = 5;
```

### Similarity with Ink annotations

The handwritten signatures are preserved as ink annotations in PdfViewer. However, on saving you can choose whether to preserve the signature as ink annotation or flatten it. This can be achieved using the `pdfViewer.AnnotationSettings.HandwrittenSignature.FlattenSignatureOnSave` property.

#### C#

```
pdfViewer.AnnotationSettings.HandwrittenSignature.FlattenSignatureOnSave = true;
```

**Note:** The default value of the property is `true`.

### Events

Since the handwritten signature is preserved as ink annotation, the events for both handwritten signatures and ink annotations are same. The events supported by ink annotations are described in detail [here](#).

When the common events occur, ink annotation and handwritten signature can be distinguished using the event argument's `IsChildSignature` property. Also, the `sender` parameter will be of type `InkAnnotation` and `HandwrittenSignature` for the respective annotations. For brevity, only the `InkSelected` event is illustrated below.

#### C#

```
pdfViewer.InkSelected += PdfViewer_InkSelected;
private void PdfViewer_InkSelected(object sender,
Syncfusion.SfPdfViewer.XForms.InkSelectedEventArgs args)
{
    if (args.IsSignature)
    {
        //The event occurred was raised by handwritten signature
        //The "sender" parameter is of type "HandwrittenSignature"
    }
    else
    {
        //The event occurred was raised by ink annotation
        //The "sender" parameter is of type "InkAnnotation"
    }
}
```

## Working with shape annotations

PDF viewer allows you to include shape annotations in a PDF document and provides options to modify or remove the existing shape annotations. The supported shape annotations are:

1. Rectangle
2. Circle
3. Line
4. Arrow

In all the code snippets, Rectangle shape annotation is used for illustration purpose.

### Adding shape annotations

#### Enabling shape annotation mode

Set the `AnnotationMode` property of the PDF viewer to `Rectangle`. After setting the annotation mode to any of the shapes, the zooming, panning, and scrolling options will be disabled. The shapes can be drawn only on the currently visible page area. Refer to the following code.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"/>
<Button x:Name="shapeAnnotationButton" Command="{Binding
AnnotationModeCommand, Source={x:Reference Name=pdfViewer}}"
CommandParameter="Rectangle" />
```

#### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();
pdfViewer.AnnotationMode = AnnotationMode.Rectangle;
```

#### Disabling shape annotation mode

Setting the `AnnotationMode` to `None` disables the shape annotation mode. When the annotation mode is reset, the zooming, panning, and scrolling will be enabled again.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"/>
<Button x:Name="resetAnnotationButton" Command="{Binding
AnnotationModeCommand, Source={x:Reference Name=pdfViewer}}"
CommandParameter="None" />
```

#### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();
pdfViewer.AnnotationMode = AnnotationMode.None;
```

#### Detecting the inclusion shape annotations

The event `ShapeAnnotationAdded` will be raised when shape annotations are added to the PDF.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
ShapeAnnotationAdded="PdfViewer_ShapeAnnotationAdded"/>
```

**C#**

```
SfPdfViewer pdfViewer = new SfPdfViewer();  
pdfViewer.ShapeAnnotationAdded += PdfViewer_ShapeAnnotationAdded;
```

## Detecting tap on shape annotations

Tapping a shape annotation selects it or deselects it if it is already selected. The event `ShapeAnnotationTapped` is raised when a shape is tapped.

**XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"  
ShapeAnnotationTapped="PdfViewer_ShapeAnnotationTapped"/>
```

**C#**

```
SfPdfViewer pdfViewer = new SfPdfViewer();  
pdfViewer.ShapeAnnotationTapped += PdfViewer_ShapeAnnotationTapped;
```

## Selecting shape annotations

You can select a shape annotation by tapping on it, this action is followed by the appearance of the selector around the selected annotation. When a shape is selected, the `ShapeAnnotationSelected` event will be raised.

The properties of the selected shape annotation can be obtained from the `args` parameter of the event handler.

**C#**

```
private void PdfViewer_ShapeAnnotationSelected(object sender,  
ShapeAnnotationSelectedEventArgs args)  
{  
    //Get the type of the annotation. Here it is rectangle  
    AnnotationMode annotationMode = args.AnnotationType;  
    //Get the bounds of the rectangle  
    Rectangle bounds = args.Bounds;  
    //Get the page number in which the annotations are present  
    int pageNumber = args.PageNumber;  
}
```

## Deselecting shape annotations

You can deselect a selected shape annotation by tapping on it or somewhere else on the PDF page. Deselection can be detected using the `ShapeAnnotationDeselected` event.

**XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"  
ShapeAnnotationDeselected="PdfViewer_ShapeAnnotationDeselected"/>
```

**C#**

```
SfPdfViewer pdfViewer = new SfPdfViewer();  
pdfViewer.ShapeAnnotationDeselected += PdfViewer.ShapeAnnotationDeselected;
```

### Customizing the appearance of shape annotations

You can customize the default values of stroke color, opacity, and thickness of all shape annotations to be added. This will not affect the already added shape annotations. The appearance of a selected free text can also be modified.

#### Setting the default stroke color

You can set the default stroke color of the shape annotations by using the `SfPdfViewer.AnnotationSettings.Rectangle.Settings.StrokeColor` property. Refer to the following code.

#### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();  
pdfViewer.AnnotationSettings.Rectangle.Settings.StrokeColor = Color.Red;
```

#### Setting the default opacity

You can set the default opacity of the shape annotations by using the `SfPdfViewer.AnnotationSettings.Rectangle.Settings.Opacity` property. Opacity value ranges from 0 to 1. Refer to the following code example.

#### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();  
pdfViewer.AnnotationSettings.Rectangle.Settings.Opacity = 0.5f;
```

#### Setting the default thickness

You can set the thickness of the shape annotations by using the `SfPdfViewer.AnnotationSettings.Rectangle.Settings.Thickness` property. Refer to the following code example.

#### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();  
pdfViewer.AnnotationSettings.Rectangle.Settings.Thickness = 5;
```

### Changing the properties of a selected shape

You can change the properties of the selected annotation by casting the `sender` parameter of the `ShapeAnnotationSelected` event handler to `ShapeAnnotation` and modifying its properties. The following code shows how to change the properties.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"  
ShapeAnnotationSelected="PdfViewer.ShapeAnnotationSelected" />
```

#### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();  
pdfViewer.ShapeAnnotationSelected += PdfViewer.ShapeAnnotationSelected;
```

**C#**

```
private void PdfViewer_ShapeAnnotationSelected(object sender,
Syncfusion.SfPdfViewer.XForms.ShapeAnnotationSelectedEventArgs args)
{
    //Get the type of the annotation. Here it is Rectangle
    AnnotationMode annotationMode = args.AnnotationType;
    //Cast the sender object as shape annotation.
    ShapeAnnotation selectedShapeAnnotation = sender as ShapeAnnotation;
    //Set the stroke color of the selected annotation using shape annotation
    settings.
    selectedShapeAnnotation.Settings.StrokeColor = Color.Blue;
    //Set the opacity of the selected annotation using shape annotation
    settings.
    selectedShapeAnnotation.Settings.Opacity = 0.3f;
    //Set the thickness of the selected annotation using shape annotation
    settings.
    selectedShapeAnnotation.Settings.Thickness = 3;
}
```

**Moving or resizing shape annotations**

Select the shape annotation to move or resize it. After the selector appears around the annotation, dragging the annotation by tapping anywhere inside the selector will move the annotation. Similarly, tapping and dragging on the corner bubbles will resize the selected annotation.

**Detecting the move or resize of a shape**

The event **AnnotationMovedOrResized** will be raised when you move or resize the selected annotation.

**XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
AnnotationMovedOrResized="PdfViewer_AnnotationMovedOrResized"/>
```

**C#**

```
SfPdfViewer pdfViewer = new SfPdfViewer();
pdfViewer.AnnotationMovedOrResized += PdfViewer_AnnotationMovedOrResized;
```

**C#**

```
private void PdfViewer_AnnotationMovedOrResized(object sender,
AnnotationMovedOrResizedEventArgs args)
{
    //Determine whether the moved or resized annotation is a shape or some other
    annotation.
    if(sender as ShapeAnnotation == null)
    {
        //The annotation is not a shape
    }
    else
    {
        //The annotation is a shape
    }
}
```

```
}  
//Retrieve the old bounds of the annotation  
Rectangle oldBounds = args.OldBounds;  
//Retrieve the new bounds of the annotation  
Rectangle newBounds = args.NewBounds;  
}
```

### Deleting shape annotations

The PDF viewer supports removing a single annotation and all the annotations in the PDF document.

#### Removing a selected annotation

The following code snippet illustrates removing a selected annotation from the PDF document.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"  
ShapeAnnotationSelected="PdfViewer_ShapeAnnotationSelected"/>  
<Button x:Name="deleteShapeAnnotationButton" Grid.Row="1"  
Clicked="deleteShapeAnnotationButton_Clicked" />
```

#### C#

```
ShapeAnnotation selectedShapeAnnotation;  
private void PdfViewer_ShapeAnnotationSelected(object sender,  
ShapeAnnotationSelectedEventArgs args)  
{  
    //Cast the sender object as Shape annotation.  
    selectedShapeAnnotation = sender as ShapeAnnotation;  
}  
private void deleteShapeAnnotationButton_Clicked(object sender, EventArgs e)  
{  
    //Delete the selected shape annotation  
    pdfViewer.RemoveAnnotation(selectedShapeAnnotation);  
}
```

#### Removing all annotations

The following code snippet illustrates removing all annotations from the PDF.

#### C#

```
pdfViewer.ClearAllAnnotations();
```

#### Detecting the removal of a shape

The event **ShapeAnnotationRemoved** will be raised when a shape annotation is removed from the PDF.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"  
ShapeAnnotationRemoved="PdfViewer_ShapeAnnotationRemoved"/>
```

#### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();
```

```
pdfViewer.ShapeAnnotationRemoved += PdfViewer.ShapeAnnotationRemoved;
```

The properties of the removed shape annotation can be obtained from the `args` parameter of the event handler.

### C#

```
private void PdfViewer_ShapeAnnotationRemoved(object sender,
ShapeAnnotationRemovedEventArgs args)
{
    //Get the type of the annotation. Here it is rectangle
    AnnotationMode annotationMode = args.AnnotationType;
    //Get the bounds of the rectangle
    Rectangle bounds = args.Bounds;
    //Get the data points of the shape
    List<Point> dataPoints = args.DataPoints;
    //Get the opacity value
    float opacity = args.Opacity;
    //Get the page number in which the annotations are present
    int pageNumber = args.PageNumber;
    //Get the position of the shape within the page
    Point position = args.Position;
    //Get the stroke color of the shape
    Color strokeColor = args.StrokeColor;
    //Get the thickness of the shape
    float thickness = args.Thickness;
}
```

## Working with free text annotations

PDF viewer allows you to include free text annotations in a PDF document and provides options to modify or remove the existing ones.

### Adding free text annotations

#### Enabling free text annotation mode

Set the `AnnotationMode` property of the PDF viewer to `FreeText`. After setting the annotation mode, the zooming, panning, and scrolling will be disabled. Tap anywhere on the displayed PDF page, a popup will appear. Type the text in the popup and click `&#34;Ok&#34;` to add text to the page. Once the free text annotation is added, the zooming, panning, and scrolling will be enabled again.

### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"/>
<Button x:Name="freeTextAnnotationButton" Command="{Binding
AnnotationModeCommand, Source={x:Reference Name=pdfViewer}}">
    CommandParameter="FreeText" />
```

### C#

```
SfPdfViewer pdfViewer = new SfPdfViewer();
pdfViewer.AnnotationMode = AnnotationMode.FreeText;
```

#### Disabling free text annotation mode

Setting the annotation mode to `None` disables the free text mode.



### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"/>
<Button x:Name="resetAnnotationButton" Command="{Binding
AnnotationModeCommand, Source={x:Reference Name=pdfViewer}}"
CommandParameter="None" />
```

### C#

```
pdfViewer.AnnotationMode = AnnotationMode.None;
```

#### Detecting the inclusion of free text annotations

The event `FreeTextAnnotationAdded` will be raised when a free text annotation is added to the PDF.

### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
FreeTextAnnotationAdded="PdfViewer_FreeTextAnnotationAdded"/>
```

### C#

```
pdfViewer.FreeTextAnnotationAdded += PdfViewer_FreeTextAnnotationAdded;
```

#### Detecting tap on free text annotations

Tapping a free text annotation selects it or deselects if it is already selected. The event `FreeTextAnnotationTapped` is raised when a free text is tapped.

### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
FreeTextAnnotationTapped="PdfViewer_FreeTextAnnotationTapped"/>
```

### C#

```
pdfViewer.FreeTextAnnotationTapped += PdfViewer_FreeTextAnnotationTapped;
```

#### Selecting free text annotations

You can select a free text annotation by tapping on it. When a free text is selected, the `FreeTextAnnotationSelected` event will be raised. The properties of the selected free text can be retrieved using the `args` parameter of the event handler.

### C#

```
private void PdfViewer_FreeTextAnnotationSelected(object sender,
FreeTextAnnotationSelectedEventArgs args)
{
    //Get the bounds
    Rectangle bounds = args.Bounds;
    //Get the page number on which the deselected free text is
    int pageNumber = args.PageNumber;
    //Get the text of the free text annotation
    string text = args.Text;
    //Get the text color
}
```

```
Color textColor = args.TextColor;
//Get the text size
float textSize = args.TextSize;
}
```

### Deselecting free text annotations

You can deselect a selected free text annotation by tapping on it or somewhere on the PDF page. Deselection can be detected using the `FreeTextAnnotationDeselected` event.

#### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
FreeTextAnnotationDeselected="PdfViewer_FreeTextAnnotationDeselected"/>
```

#### C#

```
pdfViewer.FreeTextAnnotationDeselected +=
PdfViewer_FreeTextAnnotationDeselected;
```

### Customizing the appearance of free text annotations

You can customize the default text color and text size of free text annotations to be added. This will not affect the already added free text annotations. The appearance of a selected free text can also be modified.

#### Setting the default text color

You can set the default text color of the free text annotations by using the `SfPdfViewer.AnnotationSettings.FreeTextAnnotationSettings.TextColor` property. Refer to the following code.

#### C#

```
pdfViewer.AnnotationSettings.FreeTextAnnotationSettings.TextColor =
Color.Red;
```

#### Setting the default text size

You can set the default text size of the free text annotations by using the `SfPdfViewer.AnnotationSettings.FreeTextAnnotationSettings.TextSize` property. Refer to the following code example.

#### C#

```
pdfViewer.AnnotationSettings.FreeTextAnnotationSettings.TextSize = 4;
```

### Changing the properties of a selected free text

You can change the properties of the selected annotation by casting the `sender` parameter of the `FreeTextAnnotationSelected` event handler to `FreeTextAnnotation` and modify its properties. The following code shows how to change the properties.

#### C#

```
Button changeFreeTextPropertiesButton = new Button();
```

```
changeFreeTextPropertiesButton.Clicked +=  
changeFreeTextPropertiesButton_Clicked;  
FreeTextAnnotation selectedFreeTextAnnotation;  
private void PdfViewer_FreeTextAnnotationSelected(object sender,  
FreeTextAnnotationSelectedEventArgs args)  
{  
    //Cast the sender object to FreeTextAnnotation  
    selectedFreeTextAnnotation = sender as FreeTextAnnotation;  
}  
private void changeFreeTextPropertiesButton_Clicked(object sender, EventArgs  
e)  
{  
    //Change the color of the text  
    selectedFreeTextAnnotation.Settings.TextColor = Color.Blue;  
    //Change the size of the text  
    selectedFreeTextAnnotation.Settings.TextSize = 7;  
}
```

### Moving or resizing free text annotations

To move or resize the annotation, it should be selected. After the selector appears, tapping and dragging anywhere inside the selector will move the annotation. Tapping on the bubbles around the selector and dragging will resize the annotation.

#### *Detecting the move or resize of a free text annotation*

The event `FreeTextAnnotationMovedOrResized` will be raised when you move or resize the free text annotation.

### **XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"  
FreeTextAnnotationMovedOrResized="PdfViewer_FreeTextAnnotationMovedOrResized"  
/>
```

### **C#**

```
pdfViewer.FreeTextAnnotationMovedOrResized +=  
PdfViewer_FreeTextAnnotationMovedOrResized;
```

The properties of the moved or resized free text can be obtained from the `args` parameter of the event handler.

### **C#**

```
private void PdfViewer_FreeTextAnnotationMovedOrResized(object sender,  
FreeTextAnnotationMovedOrResizedEventArgs args)  
{  
    //Retrieve the old bounds of the annotation  
    Rectangle oldBounds = args.OldBounds;  
    //Retrieve the new bounds of the annotation  
    Rectangle newBounds = args.NewBounds;  
    //Retrieve the page number in which the free text is  
    int pageNumber = args.PageNumber;  
}
```

## Deleting free text annotations

PDF viewer can remove a selected annotation or all annotations in the PDF document.

### *Removing a selected free text annotation*

The following code snippet illustrates removing a selected free text from the PDF document.

#### **XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
FreeTextAnnotationSelected="PdfViewer_FreeTextAnnotationSelected"/>
<Button x:Name="deleteFreeTextAnnotationButton" Grid.Row="1"
Clicked="deleteFreeTextAnnotationButton_Clicked" />
```

#### **C#**

```
FreeTextAnnotation selectedFreeTextAnnotation;
private void PdfViewer_FreeTextAnnotationSelected(object sender,
FreeTextAnnotationSelectedEventArgs args)
{
    //Cast the sender object to free text annotation.
    selectedFreeTextAnnotation = sender as FreeTextAnnotation;
}
private void deleteFreeTextAnnotationButton_Clicked(object sender, EventArgs
e)
{
    //Delete the selected free text annotation
    pdfViewer.RemoveAnnotation(selectedFreeTextAnnotation);
}
```

### *Removing all annotations*

The `ClearAllAnnotations` method can be used to delete all annotations irrespective of their types from the PDF.

#### **XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer" />
<Button x:Name="deleteAllAnnotationsButton" Command="{Binding
ClearAllAnnotationsCommand, Source={x:Reference Name=pdfViewer}}" />
```

#### **C#**

```
private void DeleteAllAnnotationsButton_Clicked(object sender, EventArgs e)
{
    pdfViewer.ClearAllAnnotations();
}
```

### *Detecting the removal of a free text*

The event `FreeTextAnnotationRemoved` will be raised when a free text annotation is removed from the PDF.

#### **XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewer"
FreeTextAnnotationRemoved="PdfViewer_FreeTextAnnotationRemoved"/>
```

**C#**

```
pdfViewer.FreeTextAnnotationRemoved += PdfViewer_FreeTextAnnotationRemoved;
```

The properties of the removed free text can be obtained from the `args` parameter of the event handler.

**C#**

```
private void PdfViewer_FreeTextAnnotationRemoved(object sender,
FreeTextAnnotationRemovedEventArgs args)
{
    //Get the bounds
    Rectangle bounds = args.Bounds;
    //Get the page number on which the deselected free text is
    int pageNumber = args.PageNumber;
    //Get the text of the free text annotation
    string text = args.Text;
    //Get the text color
    Color textColor = args.TextColor;
    //Get the text size
    float textSize = args.TextSize;
}
```

### Working with text markup annotation

The PDF viewer supports to add, edit, and delete text markup annotations (highlight, underline, and strikethrough) in the PDF document.

#### Highlight a text

The two ways to highlight a text in the PDF document are:

1. By selecting the text and highlight option
  - o Select the text in the PDF document.
  - o Select “Highlight” option in the context menu that appears.
2. Enabling the highlight mode and selecting the text

By default, the PDF viewer will be in the scrolling and text selection mode. Once highlight annotation mode is activated, scrolling of the document will be frozen and highlight annotations will be included when you swipe over the texts in the pages of the PDF document.

XAML code to switch to highlight annotation mode

**XML**

```
<Button x:Name="highlightAnnotationButton" BackgroundColor="Transparent"
Image="TextHighlightIcon.png" HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand" Command="{Binding AnnotationModeCommand,
Source={x:Reference Name=pdfViewerControl}}" CommandParameter="Highlight" />
```

C# code to switch to highlight annotation mode

**C#**

```
pdfViewerControl.AnnotationMode = AnnotationMode.Highlight;
```

XAML code to switch to normal mode from annotation mode.

#### XML

```
<Button x:Name="resetAnnotationButton" BackgroundColor="Transparent"
Image="resetIcon.png" HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand" Command="{Binding AnnotationModeCommand,
Source={x:Reference Name=pdfViewerControl}}" CommandParameter="None" />
```

C# code to switch to normal mode from annotation mode.

#### C#

```
pdfViewerControl.AnnotationMode = AnnotationMode.None;
```

### Underline a text

The two ways to underline a text in the PDF document are:

1. By selecting the text and underline option
  - o Select the text in the PDF document.
  - o Select "Underline" option in the context menu that appears.
2. Enabling the underline mode and selecting the text

By default, the PDF viewer will be in the scrolling and text selection mode. Once underline annotation mode is activated, scrolling of the document will be frozen and underline annotations will be included when you swipe over the texts in the pages of the PDF document.

XAML code to switch to underline annotation mode

#### XML

```
<Button x:Name="underlineAnnotationButton" BackgroundColor="Transparent"
Image="TextUnderlineIcon.png" HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand" Command="{Binding AnnotationModeCommand,
Source={x:Reference Name=pdfViewerControl}}" CommandParameter="Underline" />
```

C# code to switch to underline annotation mode

#### C#

```
pdfViewerControl.AnnotationMode = AnnotationMode.Underline;
```

XAML code to switch to normal mode from annotation mode.

#### XML

```
<Button x:Name="resetAnnotationButton" BackgroundColor="Transparent"
Image="resetIcon.png" HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand" Command="{Binding AnnotationModeCommand,
Source={x:Reference Name=pdfViewerControl}}" CommandParameter="None" />
```

C# code to switch to normal mode from annotation mode.

### **C#**

```
pdfViewerControl.AnnotationMode = AnnotationMode.None;
```

### Strikethrough a text

The two ways to strikethrough a text in the PDF document are:

1. By selecting the text and strikethrough option
  - o Select the text in the PDF document.
  - o Select “Strikethrough” option in the context menu that appears.
2. Enabling the strikethrough mode and selecting the text

By default, the PDF viewer will be in the scrolling and text selection mode. Once strikethrough annotation mode is activated, scrolling of the document will be frozen and strikethrough annotations will be included when you swipe over the texts in the pages of the PDF document.

XAML code to switch to strikethrough annotation mode

### **XML**

```
<Button x:Name="strikethroughAnnotationButton" BackgroundColor="Transparent"
Image="TextStrikethroughIcon.png" HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand" Command="{Binding AnnotationModeCommand,
Source={x:Reference Name=pdfViewerControl}}"
CommandParameter="Strikethrough" />
```

C# code to switch to strikethrough annotation mode

### **C#**

```
pdfViewerControl.AnnotationMode = AnnotationMode.Strikethrough;
```

XAML code to switch to normal mode from annotation mode.

### **XML**

```
<Button x:Name="resetAnnotationButton" BackgroundColor="Transparent"
Image="resetIcon.png" HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand" Command="{Binding AnnotationModeCommand,
Source={x:Reference Name=pdfViewerControl}}" CommandParameter="None" />
```

C# code to switch to normal mode from annotation mode.

### **C#**

```
pdfViewerControl.AnnotationMode = AnnotationMode.None;
```

### Deleting a text markup annotation

The PDF viewer removes a single annotation in the PDF document, removes all the annotations in a page, and removes all the annotations in the document.

*Removing a single annotation.*

The following code snippet illustrates removing a single annotation from the PDF document.

**XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl"
TextMarkupSelected="PdfViewerControl_TextMarkupSelected" />
```

C# code to remove selected annotation from the PDF document

**C#**

```
private void PdfViewerControl_TextMarkupSelected(object sender,
TextMarkupSelectedEventArgs args)
{
TextMarkupAnnotation selectedAnnotation = sender as TextMarkupAnnotation;
pdfViewerControl.RemoveAnnotation(selectedAnnotation);
}
```

*Removing all the annotations in a page*

The ClearAllAnnotations(int pageNumber) API can be used to remove all the annotations in a page of the PDF document.

**XML**

```
<Button x:Name="clearAllAnnotationsButton"
Clicked="clearAllAnnotationsButton_Clicked" />
```

C# code to clear all the annotations in a given page number.

**C#**

```
private void clearAllAnnotationsButton_Clicked(object sender, EventArgs e)
{
pdfViewerControl.ClearAllAnnotations(pageNumber);
}
```

*Removing all the annotations in the PDF document*

The ClearAllAnnotations API can be used to remove all the annotations in the PDF document.

**XML**

```
<Button x:Name="clearAllAnnotationsButton"
Clicked="clearAllAnnotationsButton_Clicked" />
```

C# code to clear all the annotations in a PDF document.

**C#**

```
private void clearAllAnnotationsButton_Clicked(object sender, EventArgs e)
{
pdfViewerControl.ClearAllAnnotations();
}
```



### Editing the color of the text markup annotation

You can edit the color of the text markup annotation before including the annotation and after including the annotation.

#### *Before inclusion of the annotation or Default color*

You can alter the default color of the annotation to change the color while including annotation. Refer to the following code sample.

#### **C#**

```
pdfViewerControl.AnnotationSettings.TextMarkup.Highlight.Color = Color.Red;
pdfViewerControl.AnnotationSettings.TextMarkup.Underline.Color =
Color.Magenta;
pdfViewerControl.AnnotationSettings.TextMarkup.Strikethrough.Color =
Color.Yellow;
```

---

**Note:** Setting this property will not edit the color of annotations that are included in prior.

---

#### *After inclusion of the annotation*

- Select the annotation.
- TextMarkupSelected event will be triggered and you will get a copy of selected annotation.
- Edit the copy of annotation, so that you can edit the color of the text markup annotation.

The following code snippet illustrates the same.

#### **XML**

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl"
TextMarkupSelected="PdfViewerControl_TextMarkupSelected" />
```

#### **C#**

```
private void PdfViewerControl_TextMarkupSelected(object sender,
TextMarkupSelectedEventArgs args)
{
    TextMarkupAnnotation selectedAnnotation = sender as TextMarkupAnnotation;
    selectedAnnotation.Settings.Color = Color.Yellow;
}
```

### Performing undo and redo

The PDF viewer performs undo and redo for the changes made in annotations in the PDF document. In text markup annotation undo and redo actions are provided for:

- Inclusion of new text markup annotation.
- Deletion of text markup annotation.
- Changing the color of the text markup annotation.

Refer to the following code sample to perform undo and redo operations:

#### **XML**

```
<Button x:Name="undoButton" BackgroundColor="Transparent"
Image="UndoIcon.png" HorizontalOptions="Center" VerticalOptions="Center"
Command="{Binding PerformUndoCommand ,Source={x:Reference
Name=pdfViewerControl}}"/>
<Button x:Name="redoButton" BackgroundColor="Transparent"
Image="RedoIcon.png" HorizontalOptions="Center" VerticalOptions="Center"
Command="{Binding PerformRedoCommand ,Source={x:Reference
Name=pdfViewerControl}}"/>
```

**C#**

```
pdfViewerControl.PerformUndo();
pdfViewerControl.PerformRedo();
```

**Saving the PDF document**

After performing all the changes over the PDF document in the PDF viewer, the resultant document can be saved using the `SaveDocument()` API.

**XML**

```
<Button x:Name="saveButton" Clicked="saveButton_Clicked" />
```

**C#**

```
private void saveButton_Clicked(object sender, EventArgs e)
{
    Stream stream = pdfViewerControl.SaveDocument();
}
```

**Note:** The `CanUndo` property is used to identify whether a document loaded in the PDF viewer is edited or not. When this property is set to true, means that the document in the PDF viewer is edited.

**Working with Bookmark Navigation**

PDF viewer allows users to view the bookmarks present in the loaded PDF document and navigate to the destination saved in the bookmarks.

**Enabling and disabling bookmark feature**

You can enable and disable the bookmark button from the built-in toolbar using the `BookmarkNavigationEnabled` property available in PDF viewer.

Property	Action
<code>BookmarkNavigationEnabled</code>	Gets or sets the value that enable and disable the bookmark feature in PDF viewer. By default, this property is set to true.

This property removes the bookmark button from the built-in toolbar and disable the bookmark feature, when it is set to false and vice versa.

**XML**

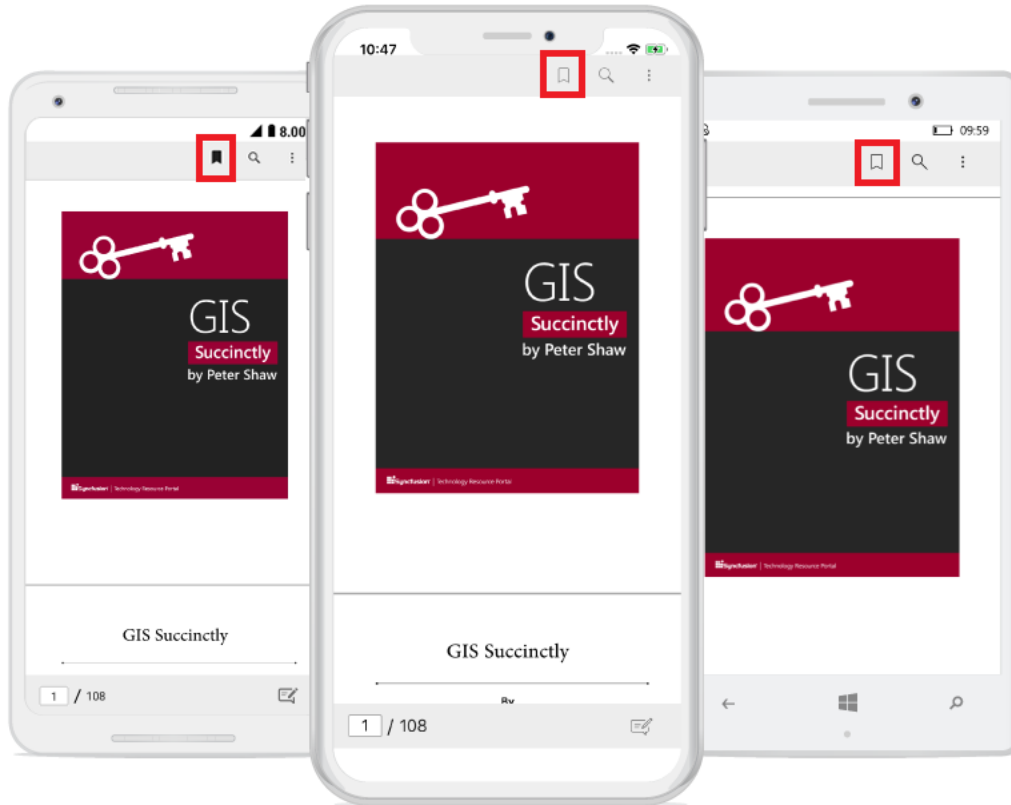
```
<pdfviewer:SfPdfViewer x:Name="pdfViewerControl"
BookmarkNavigationEnabled="False"/>
```

**C#**

```
//Bookmark feature is disabled
pdfViewerControl. BookmarkNavigationEnabled = false;
```

**Expand and collapse the bookmark pane from built-in toolbar**

The bookmark pane is expanded and collapsed by clicking the bookmark button from the built-in toolbar.

**Expand and collapse the bookmark pane programmatically**

The bookmark pane is expanded and collapsed programmatically by using the **BookmarkPaneVisible** bindable property available in PDF viewer.

Property	Action
BookmarkPaneVisible	Gets or sets the value to expand and collapse the bookmark pane in PDF viewer. Setting this property to true expands the pane and vice versa. By default, this property is set to true.

**XML**

```
<pdfviewer:SfPdfViewer x:Name="pdfViewerControl"
BookmarkPaneVisible="False"/>
```

**C#**

```
//Bookmark pane will be collapsed, if it is expanded  
pdfViewerControl.BookmarkPaneVisible = true;
```

## Programmatically navigate to a bookmark destination

You can navigate to a desired bookmark destination using the `GotoBookmark(PdfBookmark)` method. The target/destination bookmark should be provided as the argument to this method. Refer to the following code sample.

**C#**

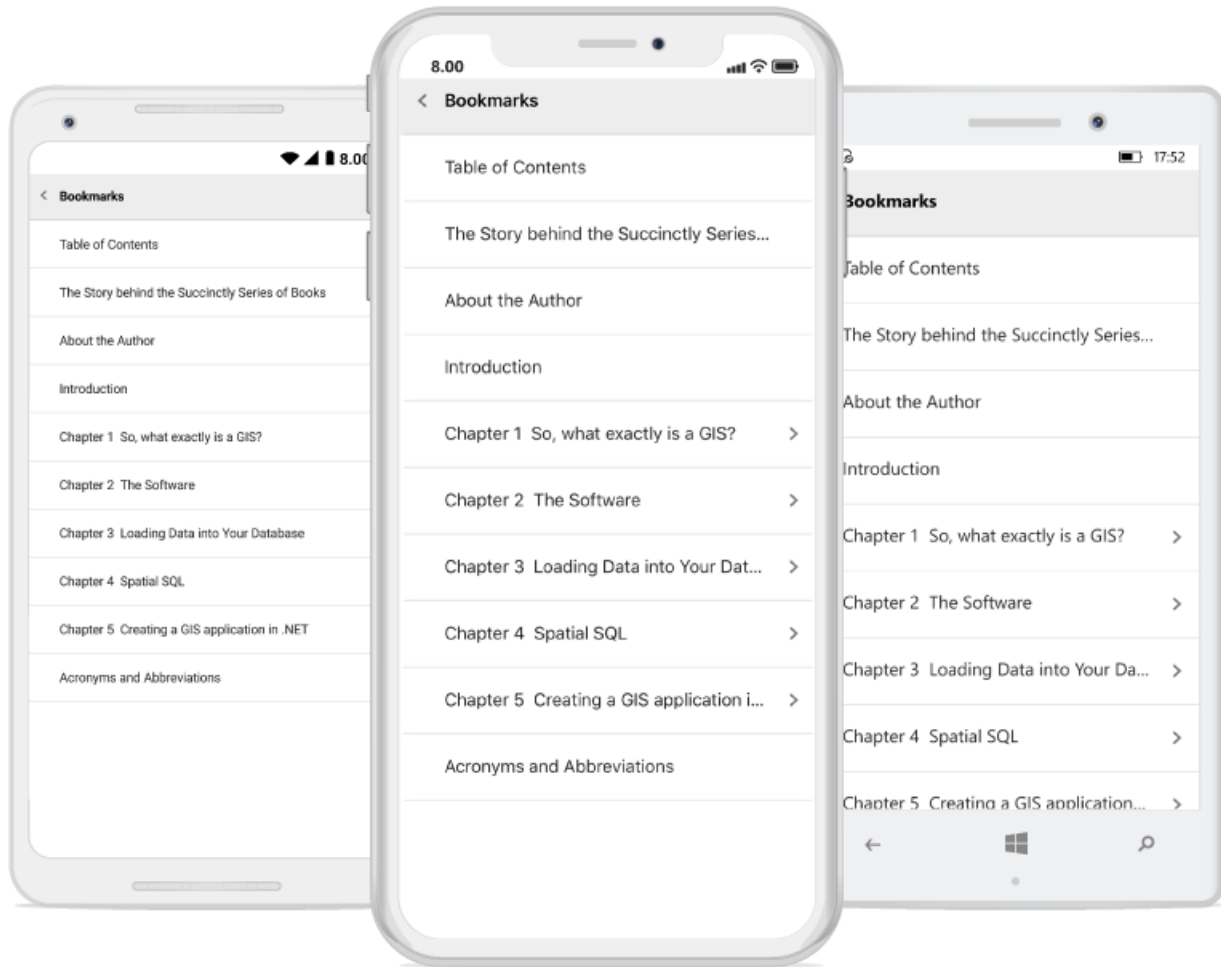
```
//Loads the PDF document in PdfLoadedDocument  
PdfLoadedDocument loadedDocument = new PdfLoadedDocument(documentStream);  
//Retrieves the bookmark collection from the loaded PDF document  
PdfBookmarkBase bookmark = loadedDocument.Bookmarks;  
//Navigate to the specified bookmark destination offset  
pdfViewerControl.GoToBookmark(bookmark[0]);
```

## Track the occurrence of the bookmark navigation

You can track the bookmark navigation operation using the `BookmarkNavigationOccurred` event. The argument of this event will provide the instance of the bookmark, destination page number, and vertical offset. Refer to the following code example.

**C#**

```
pdfViewerControl.BookmarkNavigationOccurred +=  
PdfViewerControl_BookmarkNavigationOccurred;  
private void PdfViewerControl_BookmarkNavigationOccurred(object sender,  
Syncfusion.SfPdfViewer.XForms.BookmarkNavigationOccurredEventArgs e)  
{  
//Retrieves the current bookmark  
PdfBookmark bookmark = e.Bookmark;  
//Gets the vertical offset of the bookmark  
double destinationOffset = e.Offset;  
//Gets the current page number in which the bookmark destination resides  
int pageNumber = e.PageNumber;  
}
```



## Touch interaction events

### Tapped

The [Tapped](#) event will be triggered when the user taps on the document display area of the PDF Viewer control. The event argument of this event contains the following information:

- [PageNumber](#) : The page number in which the tap event occurred.
- [Position](#) : The position of the tap event location with respect to the start of the page.

### DoubleTapped

The [DoubleTapped](#) event will be triggered when the user double taps on the document display area of the PDF Viewer control. The event argument of this event contains the following information:

- [PageNumber](#) : The page number in which the double tap event occurred.
- [Position](#) : The position of the double tap event location with respect to the start of the page.

### LongPressed

The [LongPressed](#) event triggered when the user long presses on the document display area of the PDF Viewer control. The event argument of this event contains the following information:

- [PageNumber](#) : The page number in which the double tap event occurred.
- [Position](#) : The position of the double tap event location with respect to the start of the page.

### XML

```
<Grid x:Name="pdfViewGrid">
<syncfusion:SfPdfViewer x:Name="pdfViewerControl"
Tapped="PdfViewerControl_Tapped"
DoubleTapped="PdfViewerControl_DoubleTapped"
LongPressed="PdfViewerControl_LongPressed"
InputFileStream="{Binding PdfDocumentStream}"/>
</Grid>
```

### C#

```
public MainPage()
{
    InitializeComponent();
}

private void PdfViewerControl_LongPressed(object sender,
Syncfusion.SfPdfViewer.XForms.TouchInteractionEventArgs e)
{
    //Gets the page number of the PDF Viewer where you tapped.
    int pageNumber = e.PageNumber;
    //Gets the position of the PDF Viewer where you tapped.
    Point position = e.Position;
}

private void PdfViewerControl_DoubleTapped(object sender,
Syncfusion.SfPdfViewer.XForms.TouchInteractionEventArgs e)
{
    //Gets the page number of the PDF Viewer where you double tapped.
    int pageNumber = e.PageNumber;
    //Gets the position of the PDF Viewer where you double tapped.
    Point position = e.Position;
}

private void PdfViewerControl_Tapped(object sender,
Syncfusion.SfPdfViewer.XForms.TouchInteractionEventArgs e)
{
    //Gets the page number of the PDF Viewer where you long pressed.
    int pageNumber = e.PageNumber;
    //Gets the position of the PDF Viewer where you long pressed.
    Point position = e.Position;
}
```

## Working with PDF AcroForms

PDF Viewer supports adding/modifying the existing forms fields content present in the PDF document. By default, it detects the form fields present in the PDF document and enables modifying or filling the values.

### How to restrict editing the form field values in the existing PDF document

By setting the `EnableFormFilling` property of the PdfViewerControl instance to false, you can avoid modifying the values of the form field elements present in the loaded PDF document.

### C#

```
//Does not load the form fields in PDF viewer
pdfViewerControl.EnableFormFilling = false;
```

**Note:** By default, the EnableFormFilling property is set to true.

#### How to flatten and save the form fields data

Flattening PDF form is a process of removing the form fields in the PDF document, thereby rendering the form fields appearance and its content in the page graphics. This will avoid the PDF form being edited in any device. PDF Viewer supports flattening the PDF form when saving. You can perform this action by passing the parameter `true` to the Save method of PDF Viewer.

#### C#

```
//Flatten and save the form fields during saving process
pdfViewerControl.SaveDocument(true);
```

### Working with ScrollHead

The scroll head in the PDF Viewer allows users to easily scroll through and navigate to destination page in the PDF document. ScrollHead always appears on the right side of PDF Viewer.

#### Enabling and disabling ScrollHead

By setting the `EnableScrollHead` property of the PdfViewerControl instance to false, you can disable the scroll head.

Property	Action
EnableScrollHead	Gets or sets the value that enable and disable the ScrollHead feature in PDF Viewer. By default, this property is set to true.

#### XML

```
<pdfviewer:SfPdfViewer x:Name="pdfViewerControl" EnableScrollHead="False"/>
```

#### C#

```
//Disables the scroll head
pdfViewerControl.EnableScrollHead = false;
```

#### Navigate to a desired page using ScrollHead

PDF Viewer allows users to navigate to a desired destination page using ScrollHead. On tapping the ScrollHead of PDF Viewer, a page navigation pop-up will appear and users can navigate to the destination page by entering a valid page number.

#### Localization

Localization is the process of configuring the application to a specific language. PdfViewerControl supports to localize its static text. SfPdfViewer uses the following static text that can be localized in application level:

- SfPdfViewerCancel
- SfPdfViewerCopy

- SfPdfViewerHighlight
- SfPdfViewerHyperlinkCancel
- SfPdfViewerHyperlinkMessage
- SfPdfViewerHyperlinkMessageTitle
- SfPdfViewerHyperlinkOpen
- SfPdfViewerInvalidPageMessage
- SfPdfViewerInvalidPageMessageTitle
- SfPdfViewerNoMatches
- SfPdfViewerNoOccurrences
- SfPdfViewerOkMessage
- SfPdfViewerPageEntryCancel
- SfPdfViewerPageEntryOkay
- SfPdfViewerPageNumberEntryMessage
- SfPdfViewerPageNumberEntryMessageTitle
- SfPdfViewerPlaceholderText
- SfPdfViewerRedo
- SfPdfViewerSave
- SfPdfViewerSearchResult
- SfPdfViewerStrikethrough
- SfPdfViewerUnderline
- SfPdfViewerUndo

To localize the SfPdfViewer, follow the steps in application level:

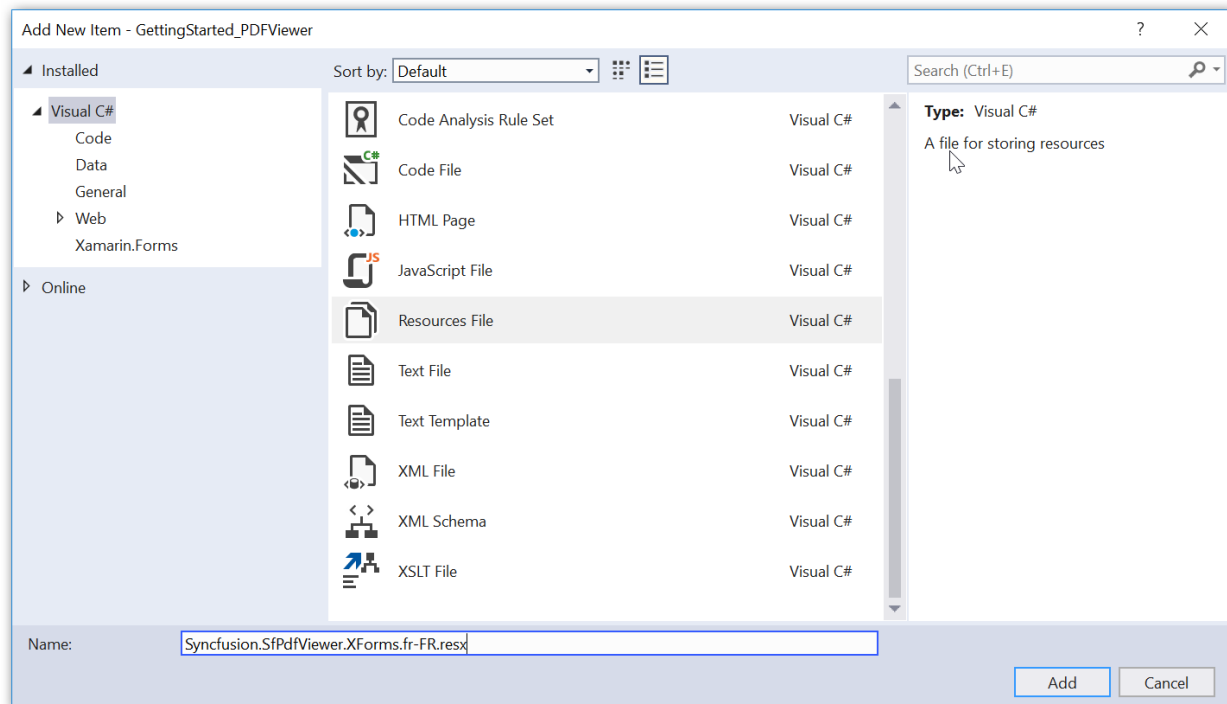
1. Add a .resx file.
2. Convert the platform specific language format to .NET format.
3. Apply the converted format.

#### Add a .resx file

In the portable project of your application, add a .resx file inside the resources folder with **Build Action - > EmbeddedResource**. File name should be **Syncfusion control's Namespace + language code** format.

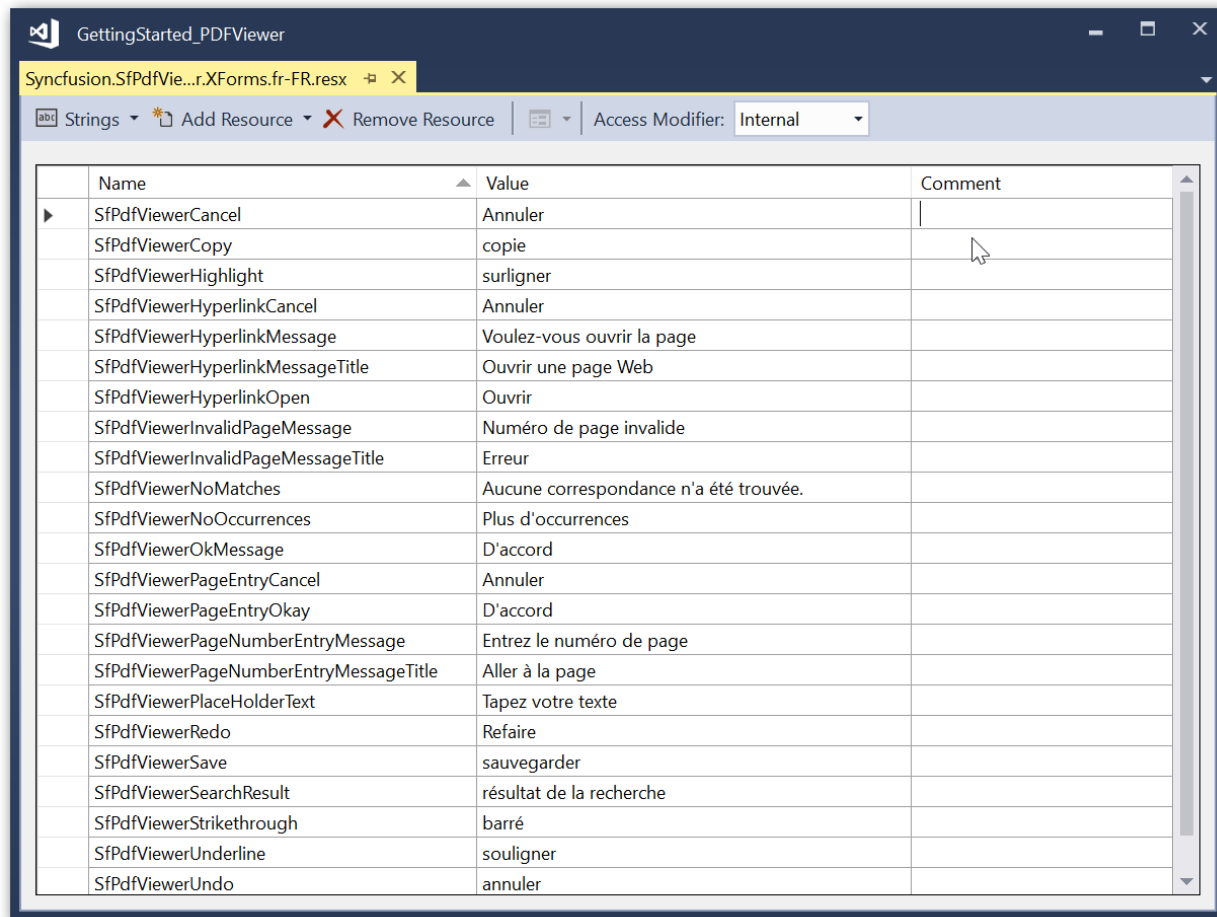
For example, to set the culture as French, the file should be named as **Syncfusion.SfPdfViewer.XForms.fr-FR.resx**.





Based on the language, set the appropriate equivalent text to the static text in the .resx file.

**Note:** You should create and add separate .resx files for individual languages.



### Convert the platform specific language format to .NET format

To get the localized text from the added `.resx` file, declare an interface named `ILocalize` in your PCL project and implement the interface in each platform renderer. This will query the language set in the device using platform specific code and convert to .NET format.

Refer to the following code snippet to declare the interface in PCL project.

#### C#

```
public interface ILocalize
{
    CultureInfo GetCurrentCultureInfo();
    void SetLocale(CultureInfo cultureInfo);
}

public class PlatformCulture
{
    public PlatformCulture(string platformCultureString)
    {
        if (!String.IsNullOrEmpty(platformCultureString))
        {
            PlatformString = platformCultureString.Replace("_", "-"); // .NET expects
            // dash, not underscore
            var dashIndex = PlatformString.IndexOf("-", StringComparison.Ordinal);
            if (dashIndex > 0)
            {

```

```
var parts = PlatformString.Split('-');
LanguageCode = parts[0];
LocaleCode = parts[1];
}
else
{
    LanguageCode = PlatformString;
    LocaleCode = "";
}
}
}
public string PlatformString
{
    get; private set;
}
public string LanguageCode
{
    get; private set;
}
public string LocaleCode
{
    get; private set;
}
public override string ToString()
{
    return PlatformString; ;
}
}
```

Refer to the following code to implement the interface in Android renderer project.

### **C#**

```
public class Localize : ILocalize
{
    public void SetLocale(CultureInfo cultureInfo)
    {
        Thread.CurrentThread.CurrentCulture = cultureInfo;
        Thread.CurrentThread.CurrentUICulture = cultureInfo;
    }
    public CultureInfo GetCurrentCultureInfo()
    {
        var netLanguage = "en";
        var androidLocale = Java.Util.Locale.Default;
        netLanguage = AndroidToDotnetLanguage(androidLocale.ToString().Replace("_", "-"));
        // this gets called a lot - try/catch can be expensive so consider caching or something
        CultureInfo cultureInfo = null;
        try
        {
            cultureInfo = new CultureInfo(netLanguage);
        }
        catch
        {
            // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
        }
    }
}
```

```

// fallback to first characters, in this case "en"
try
{
    var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
    cultureInfo = new CultureInfo(fallback);
}
catch
{
    // iOS language not valid .NET culture, falling back to English
    cultureInfo = new CultureInfo("en");
}
}
return cultureInfo;
}
private string AndroidToDotnetLanguage(string androidLanguage)
{
    var netLanguage = androidLanguage;
    //certain languages need to be converted to CultureInfo equivalent
    switch (androidLanguage)
    {
        case "ms-BN": // "Malaysian (Brunei)" not supported .NET culture
        case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture
        case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture
            netLanguage = "ms"; // closest supported
            break;
        case "in-ID": // "Indonesian (Indonesia)" has different code in .NET
            netLanguage = "id-ID"; // correct code for .NET
            break;
        case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET
            culture
            netLanguage = "de-CH"; // closest supported
            break;
        // add more application-specific cases here (if required)
        // ONLY use cultures that have been tested and known to work
    }
    return netLanguage;
}
private string ToDotnetFallbackLanguage(PlatformCulture platformCulture)
{
    var netLanguage = platformCulture.LanguageCode; // use the first part of the
    identifier (two chars, usually);
    switch (platformCulture.LanguageCode)
    {
        case "gsw":
            netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
            break;
        // add more application-specific cases here (if required)
        // ONLY use cultures that have been tested and known to work
    }
    return netLanguage;
}
}

```

Refer to the following code to implement the interface in iOS renderer project.

### **C#**

```
public class Localize : ILocalize
{
    public void SetLocale(CultureInfo cultureInfo)
    {
        Thread.CurrentThread.CurrentCulture = cultureInfo;
        Thread.CurrentThread.CurrentUICulture = cultureInfo;
    }
    public CultureInfo GetCurrentCultureInfo()
    {
        var netLanguage = "en";
        if (NSLocale.PreferredLanguages.Length > 0)
        {
            var pref = NSLocale.PreferredLanguages[0];
            netLanguage = iOSToDotnetLanguage(pref);
        }
        // this gets called a lot - try/catch can be expensive so consider caching
        // or something
        CultureInfo cultureInfo = null;
        try
        {
            cultureInfo = new CultureInfo(netLanguage);
        }
        catch
        {
            // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
            // fallback to first characters, in this case "en"
            try
            {
                var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
                cultureInfo = new CultureInfo(fallback);
            }
            catch
            {
                // iOS language not valid .NET culture, falling back to English
                cultureInfo = new CultureInfo("en");
            }
        }
        return cultureInfo;
    }
    private string iOSToDotnetLanguage(string iOSLanguage)
    {
        var netLanguage = iOSLanguage;
        //certain languages need to be converted to CultureInfo equivalent
        switch (iOSLanguage)
        {
            {
                case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture
                case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture
                    netLanguage = "ms"; // closest supported
                    break;
                case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET
                    culture
                    netLanguage = "de-CH"; // closest supported
                    break;
                // add more application-specific cases here (if required)
                // ONLY use cultures that have been tested and known to work
            }
        }
        return netLanguage;
    }
}
```

```

}
private string ToDotnetFallbackLanguage(PlatformCulture platCulture)
{
    var netLanguage = platCulture.LanguageCode; // use the first part of the
identifier (two chars, usually);
    switch (platCulture.LanguageCode)
    {
        //
        case "pt":
            netLanguage = "pt-PT"; // fallback to Portuguese (Portugal)
            break;
        case "gsw":
            netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
            break;
        // add more application-specific cases here (if required)
        // ONLY use cultures that have been tested and known to work
    }
    return netLanguage;
}
}

```

Implementation of the interface is not required for UWP project, since the resources automatically recognizes the selected language.

#### Apply the converted format

After setting the root/main page of the application in your MainPage.Xaml.cs file of the PCL project, initialize a new instance of the `ResourceManager` class and set it to the `PdfViewerResourceManager.Manager` property to look up into the resources with specified root name in the given assembly. Using `DependencyService`, call `SetLocale()` of the implemented interface with necessary language code as parameter.

#### C#

```

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        //The PDF is in the Assets folder of this project. Read it into a stream
        Stream stream =
            typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("GettingStarted
            .Assets.Xamarin_Forms_Succinctly.pdf");
        //Load the stream to PdfViewer
        pdfViewerControl.LoadDocument(stream);
        //Assign the localized string to the specific culture
        PdfViewerResourceManager.Manager=new
        ResourceManager("GettingStarted.Resources.Syncfusion.SfPdfViewer.XForms",
        GetType().GetTypeInfo().Assembly);
    }
}

```

For Android and iOS, it is mandatory to implement the previous steps. However, to set the specific language to the application irrespective of the selected language in the device, use `CultureInfo.CurrentUICulture` in a specific project of UWP platform.

Refer to the following code example to localize the text in UWP platform.

MainPage.Xaml.cs

**C#**

```
public MainPage ()
{
    this.InitializeComponent();
    SfPdfViewerRenderer.Init();
    // Applying localization for UWP
    CultureInfo.CurrentUICulture = new CultureInfo("fr");
    LoadApplication(new GettingStarted.App());
}
```



## Loading password protected PDFs

Password protected PDFs can be loaded using the `LoadDocument(Stream pdfStream, string password)` method.

**C#**

```
string password = "PASSWORD";
pdfViewerControl.LoadDocument(pdfStream, password);
```

In the above code snippet, `pdfStream` is the Stream instance read from the encrypted PDF and `password` is the key with which the PDF is encrypted.

### Handling invalid passwords

If the password provided with the `LoadDocument(Stream pdfStream, string password)` is invalid, then the `PasswordErrorOccurred` event is raised. The `Title` property of the `PasswordErrorOccurredEventArgs` parameter helps identify whether the event raised due to invalid password. In that case the `Title` property will read "Error loading encrypted PDF document".

#### C#

```
pdfViewerControl.PasswordErrorOccurred +=  
PdfViewerControl_PasswordErrorOccurred;  
private void PdfViewerControl_PasswordErrorOccurred(object sender,  
PasswordErrorOccurredEventArgs args)  
{  
    //Get the details regarding the password error occurred.  
    string title = args.Title;  
    string description = args.Description;  
}
```

**Note:** The event will also be raised when an encrypted PDF is loaded without providing a password using the `LoadDocument(Stream pdfStream)` overload.

### Importing and exporting form data

PDF viewer provides options to import and export data to and from form fields in the PDF document. The form data files with the following extensions are supported.

1. fdf
2. xfdf
3. json
4. xml

The required file type can be chosen from the `DataFormat` enumeration. In the following sections only the fdf file type is illustrated for brevity.

#### Exporting form data

The `ExportFormData` method exports the current data filled in the form fields into a stream in the specified file format. The name of the PDF, with extension, from which the data are exported should be given as the second argument to the method.

#### C#

```
Stream fdfStreamToSave = pdfViewerControl.ExportFormData(DataFormat.Fdf,  
"PdfFileName.Pdf");
```

**Note:** While saving the stream returned by the `ExportFormData` method, the file name must have the same extension as the file type that was given as the first argument. e.g. In the above case the file type provided is fdf, so the saved file extension should be .fdf.

#### Importing form data

The `ImportFormData` method imports the data from a file of specified type and fills the new data into the form fields.



## C#

```
Stream fdfStreamToImport =  
typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("SampleDefaultN  
amespace.Assets.FileName.fdf");  
pdfViewerControl.ImportFormData(fdfStreamToImport, DataFormat.Fdf);
```

## Exporting the pages of PDF document as images

The PdfViewerControl allows users to export the pages of a PDF document as image streams using the **ExportAsImage** method. The resultant image streams can be saved as image files in the local storage.

### Exporting a single PDF page as image

To export a single PDF page as image, you should use the **ExportAsImage** method that accepts page index as its parameter. The following code example describes exporting page at index 0 as image stream.

## C#

```
//Accessing the PDF document that is added as embedded resource as stream  
var fileStream =  
typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("GettingStarted  
.Assets.Xamarin-Forms-Succinctly.pdf");  
//Initialize the SfPdfViewer  
SfPdfViewer pdfViewerControl = new SfPdfViewer();  
//Load the PDF document as stream  
pdfViewerControl.LoadDocument(fileStream);  
//Export the page of PDF document to image stream with the given index  
Stream stream = pdfViewerControl.ExportAsImage(0);
```

### Exporting a single PDF page as image with custom scale factor

To export a single PDF page as image with custom scale factor, you should use the **ExportAsImage** method that accepts page index and scale factor as its parameters. The following code example describes exporting page at index 0 with the scale factor 2.0 as image stream.

## C#

```
//Accessing the PDF document that is added as embedded resource as stream  
var fileStream =  
typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("GettingStarted  
.Assets.Xamarin-Forms-Succinctly.pdf");  
//Initialize the SfPdfViewer  
SfPdfViewer pdfViewerControl = new SfPdfViewer();  
//Load the PDF document as stream  
pdfViewerControl.LoadDocument(fileStream);  
//Export the 0th index of PDF page to image stream with size two times  
greater than the original size  
Stream stream = pdfViewerControl.ExportAsImage(0, 2.0f);
```

### Exporting a range of PDF pages as images

To export a range of PDF pages as images, you should use the **ExportAsImage** method that accepts start page index and end page index as its parameter. The following code example describes exporting pages ranging between index 0 and 3 as image stream array.

## C#

```
var fileStream =  
typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("GettingStarted  
.Assets.Xamarin-Forms-Succinctly.pdf");  
//Initialize the SfPdfViewer  
SfPdfViewer pdfViewerControl = new SfPdfViewer();  
//Load the PDF document as stream  
pdfViewerControl.LoadDocument(fileStream);  
//Export the pages of PDF document from the index 0 to 3 to the array of  
image stream  
Stream[] stream = pdfViewerControl.ExportAsImage(0, 3);
```

### Exporting a range of PDF pages as images with custom scale factor

To export a range of PDF pages as images, you should use the `ExportAsImage` method that accepts start page index, end page index, and scale factor as its parameter. The following code example describes exporting pages ranging between index 0 and 3 with the scale factor 2 as image stream array.

## C#

```
var fileStream =  
typeof(App).GetTypeInfo().Assembly.GetManifestResourceStream("GettingStarted  
.Assets.Xamarin-Forms-Succinctly.pdf");  
//Initialize the SfPdfViewer  
SfPdfViewer pdfViewerControl = new SfPdfViewer();  
//Load the PDF document as stream  
pdfViewerControl.LoadDocument(fileStream);  
//Export the pages of PDF document from the index 0 to 3 to the array of  
image stream  
//All the exported images are scaled 2 times larger than its original size  
Stream[] stream = pdfViewerControl.ExportAsImage(0, 3, 2.0f);
```

## Working with magnification

Magnification of the PDF document can be done in multiple ways and the different ways are explained below. By default, the PdfViewer supports panning and Ctrl+ mouse scroll (in XForms.UWP) to manipulate the magnification of the document.

### Set custom zoom percentage

The users can set custom magnification factor between 100 and 300 to magnify the document displayed. The following code explains this,

## XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding  
PdfDocumentStream}" ZoomPercentage="150" />
```

## C#

```
pdfViewerControl.ZoomPercentage = 150;
```

### Get custom zoom factor

The following code explains accessing zoom factor in which the document is displayed in the viewer.

### XML

```
<Entry Keyboard="Numeric" FontSize="18" x:Name="zoomPercentage"
HorizontalTextAlignment="Center" VerticalOptions="Center" Text="{Binding
ZoomPercentage, Source={x:Reference Name=pdfViewerControl}}"/>
```

### C#

```
int zoom = pdfViewerControl.ZoomPercentage;
```

On binding entry control to the ZoomPercentage property of the PDF Viewer instance, the current zoom percentage being displayed in PDF Viewer is displayed in the entry control, and the PDF Viewer will be zoomed based on the value entered.

#### Set maximum zoom percentage

The PDF Viewer control allows you to set the maximum zoom percentage value for the PDF document being displayed. The following code example will set the maximum zoom percentage of PDF Viewer instance to 200.

### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding
PdfDocumentStream}" MaximumZoomPercentage="200" />
```

### C#

```
pdfViewerControl.MaximumZoomPercentage = 200;
```

#### Set minimum zoom percentage

The PDF Viewer control allows you to set the minimum zoom percentage value for the PDF document being displayed. The following code example will set the minimum zoom percentage of PDF Viewer instance to 10.

### XML

```
<syncfusion:SfPdfViewer x:Name="pdfViewerControl" InputFileStream="{Binding
PdfDocumentStream}" MinimumZoomPercentage="10" />
```

### C#

```
pdfViewerControl.MinimumZoomPercentage = 10;
```

## SfPicker

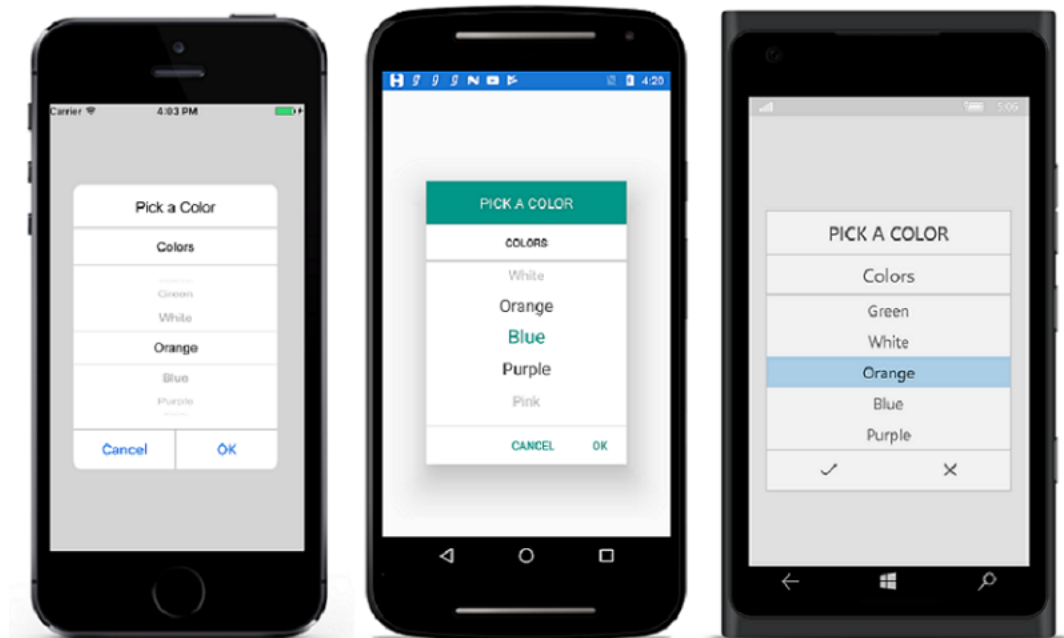
### Overview

The picker control allows you to pick an item among a list of items that can be customized with custom view. This control can be opened as dialog. Its rich feature set includes functionalities such as data binding, multi column, header/footer, custom view on header/footer, and default validation buttons.

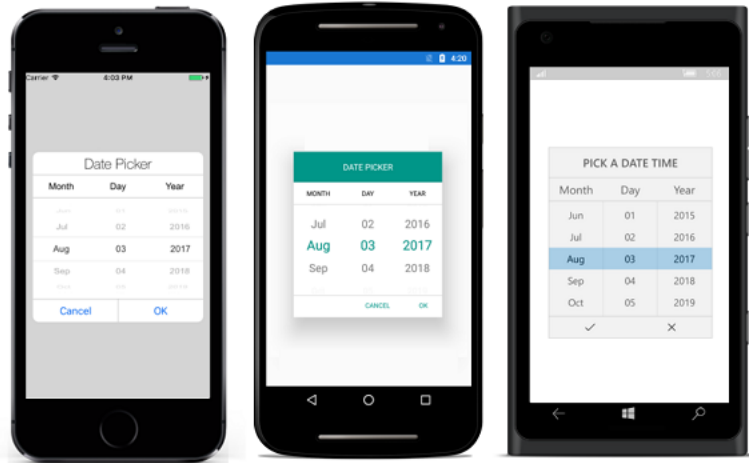
## Feature comparisons across iOS, Android, and UWP

Features	Xamarin.Forms (Android, iOS and UWP)	Xamarin.Android	Xamarin.iOS
Show Picker in Dialog			
Selection			
Header Customization			
Footer Customization			
Item Customization			
Multi Column Support			
Customize each column width			
Column Header support			
Column Header Customization			

The following screenshot illustrates picker's color selection



The following screenshot illustrates picker's multicolumn



## Getting Started

This section explains the steps required to configure a picker control in a real-time scenario, and provides a walk-through on some of the customization features available in picker control.

### Adding SfPicker reference

You can add SfPicker reference using one of the following methods:

#### Method 1: Adding SfPicker reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.syncfusion.com/nuget-packages). To add SfPicker to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfPicker](https://www.syncfusion.com/nuget-packages), and then install it.

!{Adding SfPicker reference from NuGet}(images/Adding SfPicker reference.png)

---

**Note:** Install the same version of SfPicker NuGet in all the projects.

---

#### Method 2: Adding SfPicker reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfPicker control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfPicker assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfPicker.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfPicker.Android.dll Syncfusion.SfPicker.XForms.Android.dll Syncfusion.SfPicker.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll

iOS	Syncfusion.SfPicker.iOS.dll Syncfusion.SfPicker.XForms.iOS.dll Syncfusion.SfPicker.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfInput.UWP.dll Syncfusion.SfShared.UWP.dll Syncfusion.SfPicker.XForms.UWP.dll Syncfusion.SfPicker.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Initialize picker on each platform

To use picker in Xamarin application, each platform project must initialize the picker renderer. These initializing steps vary from platform to platform, and it is discussed in the following sections.

#### Android

The Android launches the picker without any initialization, and it is enough to only initialize the Xamarin.Forms Framework to launch the application.

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch the picker in iOS, call the `SfPickerRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework has been initialized and before the `LoadApplication` is called as demonstrated in the following code example.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    SfPickerRenderer.Init ();
    LoadApplication (new App ());
    ...
}
```

### Universal Windows Platform (UWP)

To launch the picker in UWP, call the `SfPickerRenderer.Init()` in the `MainPage` constructor before the `LoadApplication` is called as demonstrated in the following code example.

#### C#

```
public MainPage()
{
    ...
    SfPickerRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

### ReleaseMode issue in UWP platform

There is a known Framework issue in UWP platform. The custom controls will not render when deployed application is in Release Mode.

The above issues can be resolved by initializing the picker assemblies in `App.xaml.cs` in UWP project as shown in the following code snippet.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfPickerRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Create a simple picker

This section explains how to create a simple picker control and configure it. picker can be configured by using XAML or C# code.

### Create a Xamarin.Forms project

Create a new blank project (Xamarin.Forms portable) by using Visual Studio or Xamarin Studio for Xamarin.Forms.

### Adding picker in Xamarin.Forms project

1. Add the required assembly reference in PCL, and other renderer projects as discussed in **Adding picker reference** section.
2. Add picker control's two way XAML or C#.
  - o XAML Page

- Set SfPicker control namespace as `xmlns:syncfusion="clr-namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms"` in XAML Content page.
- Set the SfPicker control in content property of contentPage.
- C# Page
- Import SfPicker control namespace as `using Syncfusion.SfPicker.XForms;` in C# ContentPage.
- Create a new SfPicker instance in ContentPage constructor, and assign SfPicker instance to ContentPage content property.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:syncfusion="clr-namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="GettingStarted.PickerSample">
<ContentPage.Content>
<syncfusion:SfPicker x:Name="picker" />
</ContentPage.Content>
</ContentPage>
```

### C#

```
using Syncfusion.SfPicker.XForms;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace GettingStarted
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class PickerSample : ContentPage
    {
        SfPicker picker;
        public PickerSample()
        {
            InitializeComponent();
            picker = new SfPicker();
            this.Content = picker;
        }
    }
}
```

#### *Set header to the picker*

The picker control allows you to define header text by setting the `SfPicker.HeaderText`, and enable SfPicker header by setting `SfPicker.ShowHeader` property to true. Default value of `SfPicker.ShowHeader` is true.

### XML

```
<syncfusion:SfPicker x:Name="picker" HeaderText="Select a Color" />
```



## C#

```
picker.HeaderText = "Select a Color";
```

### *Adding picker items*

picker control is a data bounded control. Hence, you must create collection of data's and bind it to picker control.

- Create a simple Observable Collection with string type of data for the picker

## C#

```
public class ColorInfo
{
    private ObservableCollection<string> _color;
    public ObservableCollection<string> Colors
    {
        get { return _color; }
        set { _color = value; }
    }
    public ColorInfo()
    {
        Colors = new ObservableCollection<string>();
        Colors.Add("Red");
        Colors.Add("Green");
        Colors.Add("Yellow");
        Colors.Add("Blue");
        Colors.Add("SkyBlue");
        Colors.Add("Orange");
        Colors.Add("Gray");
        Colors.Add("Pink");
    }
}
```

- Bind the Collection to picker

picker control allows you to bind collection of data by setting the `SfPicker.ItemsSource` property. You can bind the collection of data in both XAML or C#.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    x:Class="GettingStarted.PickerSample"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:GettingStarted"
    xmlns:syncfusion="clr-namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms">
    <ContentPage.BindingContext>
        <local:ColorInfo />
    </ContentPage.BindingContext>
    <ContentPage.Content>
        <syncfusion:SfPicker
```

```
x:Name="picker"
HeaderText="Select a Color"
ItemsSource="{Binding Colors}" />
</ContentPage.Content>
</ContentPage>
```

### C#

```
ColorInfo info = new ColorInfo();
picker.ItemsSource = info.Colors;
```

#### Set title to the items

picker control allows you to define title to the picker items by setting `SfPicker.ColumnHeaderText` and enable title of the picker items by setting `SfPicker.ShowColumnHeader` property to true. Default value of `SfPicker.ShowColumnHeader` is false.

### XML

```
<syncfusion:SfPicker
x:Name="picker"
ColumnHeaderText="Color"
ShowColumnHeader="True" />
```

### C#

```
picker.ColumnHeaderText = "Color";
picker.ShowColumnHeader = true;
```

#### Enable validation button in footer

In picker control, validation buttons (OK and Cancel) can be enabled by setting `SfPicker.ShowFooter` property to true. Default value of `SfPicker.ShowFooter` property is false

### XML

```
<syncfusion:SfPicker
x:Name="picker"
ShowFooter="True" />
```

### C#

```
picker.ShowFooter = true;
```

#### Open as dialog

picker can be rendered as a dialog by setting the `SfPicker.PickerMode` property to Dialog. Default value of `SfPicker.PickerMode` property is Default.

### XML

```
<syncfusion:SfPicker x:Name="picker" PickerMode="Dialog" />
```

### C#

```
picker.PickerMode = PickerMode.Dialog;
```

The picker can be opened programmatically by setting `SfPicker.IsOpen` property to true. Default value of `SfPicker.IsOpen` is false.

Note: This property is automatically changed to false when you close the dialog by click outside of dialog.

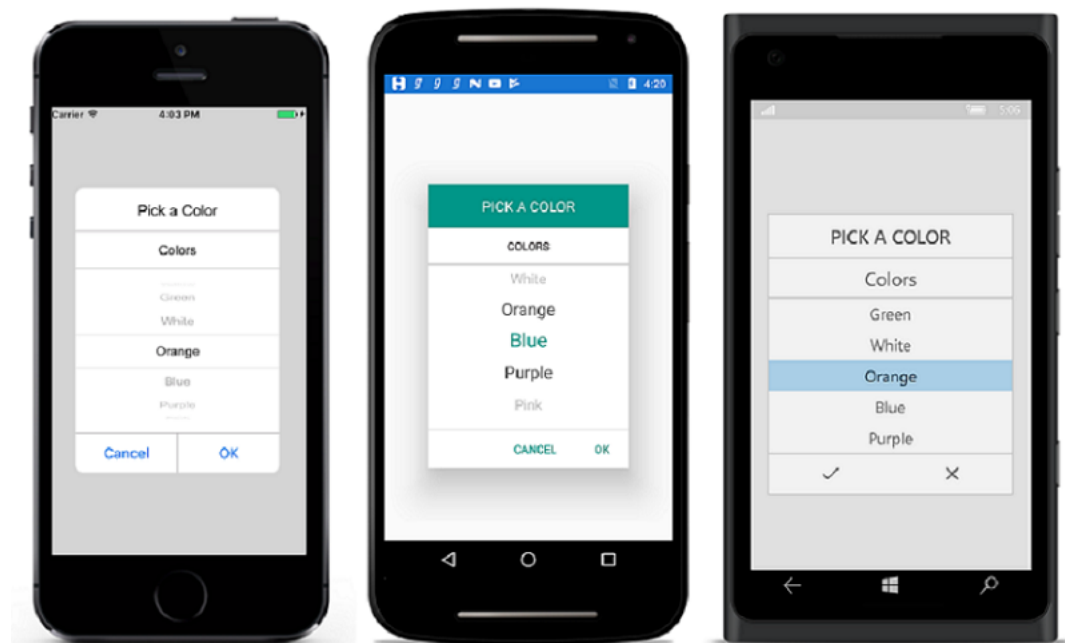
### XML

```
<syncfusion:SfPicker  
x:Name="picker"  
IsOpen="True"  
PickerMode="Dialog" />
```

### C#

```
picker.IsOpen = true;
```

The following screenshot illustrates the output of above code.



We have attached sample for reference. You can download the sample from the following link.

Sample link: [GettingStarted](#)

### Populating Items

This section explains the ways of populating items for picker control.

#### Binding data source

picker control is bound to the external data source to display the data. It supports any collections that implements the `IEnumerable` interface.

To bind the data source in picker, set the `SfPicker.ItemsSource` property as shown in the following code.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
  x:Class="GettingStarted.PickerSample"
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:GettingStarted"
  xmlns:syncfusion="clr-namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms">
  <ContentPage.BindingContext>
  <local:ColorInfo />
  </ContentPage.BindingContext>
  <ContentPage.Content>
  <syncfusion:SfPicker
    x:Name="picker"
    HeaderText="Select a Color"
    ItemsSource="{Binding Colors}" />
  </ContentPage.Content>
</ContentPage>
```

### C#

```
ColorInfo info = new ColorInfo();
picker.ItemsSource = info.Colors;
```

### Multi-column items

The picker automatically populates the items as Multi-column based on the data source.

Collection of items can be created and assigned to a Collection, and each item Collection is a column of picker.

The following code example illustrates how to populate Month, Day, and Year values in each column of picker.

### C#

```
public class DateTimePicker : SfPicker
{
  #region Public Properties
  // Months api is used to modify the Day collection as per change in Month
  internal Dictionary<string, string> Months { get; set; }
  /// <summary>
  /// Date is the actual DataSource for SfPicker control which will holds the
  collection of Day ,Month and Year
  /// </summary>
  /// <value>The date.</value>
  public ObservableCollection<object> Dates { get; set; }
  //Day is the collection of day numbers
  internal ObservableCollection<object> Day { get; set; }
  //Month is the collection of Month Names
  internal ObservableCollection<object> Month { get; set; }
  //Year is the collection of Years from 1990 to 2042
```

```

internal ObservableCollection<object> Year { get; set; }
/// <summary>
/// Headers api is holds the column name for every column in date picker
/// </summary>
/// <value>The Headers.</value>
public ObservableCollection<string> Headers { get; set; }
#endregion
public DateTimePicker()
{
    Months = new Dictionary<string, string>();
    Dates = new ObservableCollection<object>();
    Day = new ObservableCollection<object>();
    Month = new ObservableCollection<object>();
    Year = new ObservableCollection<object>();
    Headers = new ObservableCollection<string>();
    //First column of picker
    Headers.Add("Month");
    //Second column of picker
    Headers.Add("Day");
    //Third column of picker
    Headers.Add("Year");
    HeaderText = "Date Time Picker";
    PopulateDateCollection();
    this.ItemsSource = Dates;
    this.ColumnHeaderText = Headers;
    ShowFooter = true;
    ShowHeader = true;
    ShowColumnHeader = true;
}
private void PopulateDateCollection()
{
    //populate months
    for (int i = 1; i < 13; i++)
    {
        if
        (!Months.ContainsKey(CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(
            i).Substring(0, 3)))
        Months.Add(CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i).Substri
            ng(0, 3), CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i));
        Month.Add(CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i).Substrin
            g(0, 3));
    }
    //populate year
    for (int i = 1990; i < 2050; i++)
    {
        Year.Add(i.ToString());
    }
    //populate Days
    for (int i = 1; i <= DateTime.DaysInMonth(DateTime.Now.Year,
        DateTime.Now.Month); i++)
    {
        if (i < 10)
        {
            Day.Add("0" + i);
        }
        else
        Day.Add(i.ToString());
    }
}

```

```

}
Dates.Add(Month);
Dates.Add(Day);
Dates.Add(Year);
}
}

```

## XML

```

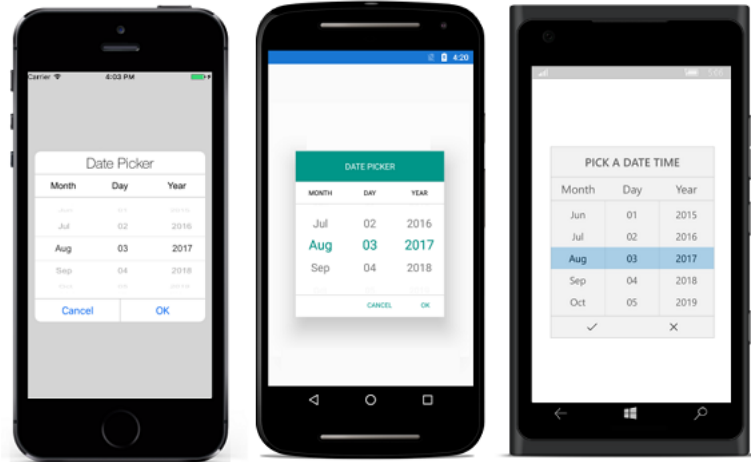
<ContentPage.BindingContext>
<local:DateTimeViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<local:DateTimePicker x:Name="picker"
PickerHeight="400"
PickerWidth="300"
HeaderText="Date Picker"
SelectedItem="{Binding StartDate}"
ColumnHeaderHeight="50">
</local:DateTimePicker>
</ContentPage.Content>

```

You can download the multi column sample from the following link.

Sample link: [MultiColumn](#)

The following screenshot illustrates the output of the above code.



Set items colors and font attributes customization

picker control, both items text color and font can be selected and unselected by customizing as shown in the following code.

*Selected item customization*

Text Color

Selected item's text color can be customized by setting `SfPicker.SelectedItemTextColor` property.

## XML

```

<syncfusion:SfPicker
x:Name="picker"

```

```
ItemsSource="{Binding Colors}"
SelectedItemTextColor="Red" />
```

### **C#**

```
picker.SelectedItemTextColor = Color.Red;
```

### *Font*

This section explains about the customization of selected item's font.

#### *FontFamily*

Selected item's text FontFamily can be customized by setting SfPicker.SelectedItemFontFamily property.

### **XML**

```
<syncfusion:SfPicker
x:Name="picker"
ItemsSource="{Binding Colors}"
SelectedItemFontFamily="Arial" />
```

### **C#**

```
picker.SelectedItemFontFamily = "Arial";
```

### *FontSize*

Selected item's text FontSize can be customized by setting SfPicker.SelectedItemFontSize property.

### **XML**

```
<syncfusion:SfPicker
x:Name="picker"
ItemsSource="{Binding Colors}"
SelectedItemFontSize="12" />
```

### **C#**

```
picker.SelectedItemFontSize = 12;
```

### *FontAttribute*

Selected item's text FontAttribute can be customized by setting SfPicker.SelectedItemFontAttribute property.

### **XML**

```
<syncfusion:SfPicker
x:Name="picker"
ItemsSource="{Binding Colors}"
SelectedItemFontAttribute="Bold" />
```

### **C#**

```
picker.SelectedItemFontAttribute = FontAttributes.Bold;
```

### *Unselected item customization*

#### Text Color

Unselected item's text color can be customized by setting `SfPicker.UnSelectedItemTextColor` property.

#### **XML**

```
<syncfusion:SfPicker
  x:Name="picker"
  ItemsSource="{Binding Colors}"
  UnSelectedItemTextColor="Gray" />
```

#### **C#**

```
picker.UnSelectedItemTextColor = Color.Gray;
```

#### Font

This section explains about the customization of unselected item's font.

#### *FontFamily*

Unselected item's text FontFamily can be customized by setting `SfPicker.UnSelectedItemFontFamily` property.

#### **XML**

```
<syncfusion:SfPicker
  x:Name="picker"
  ItemsSource="{Binding Colors}"
  UnSelectedItemFontFamily="Calibri" />
```

#### **C#**

```
picker.UnSelectedItemFontFamily = "Calibri";
```

#### *FontSize*

Unselected item's text FontSize can be customized by setting `SfPicker.UnSelectedItemFontSize` property.

#### **XML**

```
<syncfusion:SfPicker
  x:Name="picker"
  ItemsSource="{Binding Colors}"
  UnSelectedItemFontSize="11" />
```

#### **C#**

```
picker.UnSelectedItemFontSize = 11;
```



### FontAttribute

Unselected item's text FontAttribute can be customized by setting SfPicker.UnSelectedItemFontAttribute property.

### XML

```
<syncfusion:SfPicker
x:Name="picker"
ItemsSource="{Binding Colors}"
UnSelectedItemFontAttribute="Italic" />
```

### C#

```
picker.UnSelectedItemFontAttribute = FontAttributes.Italic;
```

### Adding custom view to items

In picker control, the items can be customized with custom view of each item by hooking SfPicker.OnPickerItemLoaded event, and assign custom view in PickerViewEventArgs.View property to add the all the item with custom view.

### XML

```
<Grid x:Name="main">
<Button
x:Name="button"
Text="Open Picker"
Clicked="Handle_Clicked"
HeightRequest="100"
WidthRequest="200"
VerticalOptions="Center"
HorizontalOptions="Center" />
<picker:SfPicker
x:Name="picker"
HeaderHeight="40"
ShowHeader="true"
HeaderText="SfPicker Sample"
ShowColumnHeader="True"
ColumnHeaderHeight="46"
PickerMode="Dialog"
ItemsHeight="40"
PickerHeight="350"
PickerWidth="350"
ShowFooter="True"
FooterHeight="46"/>
</Grid>
```

### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        ObservableCollection<object> mainCollection = new
        ObservableCollection<object>();
    }
}
```

```

ObservableCollection<object> column0 = new ObservableCollection<object>();
ObservableCollection<object> column1 = new ObservableCollection<object>();
ObservableCollection<string> columnHeader = new
ObservableCollection<string>();
columnHeader.Add("Custom View");
columnHeader.Add("Labels");
picker.ColumnHeaderText = columnHeader;
picker.OnPickerItemLoaded+=HandlePickerViewEvent;
column0.Add("India.png");
column0.Add("UAE.png");
column0.Add("USA.png");
column0.Add("UK.png");
column0.Add("Germany.png");
column1.Add("TestingLabel 1");
column1.Add("TestingLabel 2");
column1.Add("TestingLabel 3");
column1.Add("TestingLabel 4");
column1.Add("TestingLabel 5");
mainCollection.Add(column0);
mainCollection.Add(column1);
picker.ItemsSource = mainCollection;
picker.Parent = main;
}
void HandlePickerViewEvent(object sender,
Syncfusion.SfPicker.XForms.PickerViewEventArgs e)
{
    if (e.Column == 0)
    {
        Country country = new Country() { Name = e.Item.ToString() };
        e.View = new ItemView(country);
    }
}
void Handle_Clicked(object sender, System.EventArgs e)
{
    picker.IsOpen = true;
}
}

```

### CustomView Xaml

#### XML

```

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="40" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Image
Source="{Binding Name}"
Margin="8,0,0,0"
VerticalOptions="Center" />
<Label
Text="{Binding Text}"
Grid.Column="1" />
</Grid>

```

**C#**

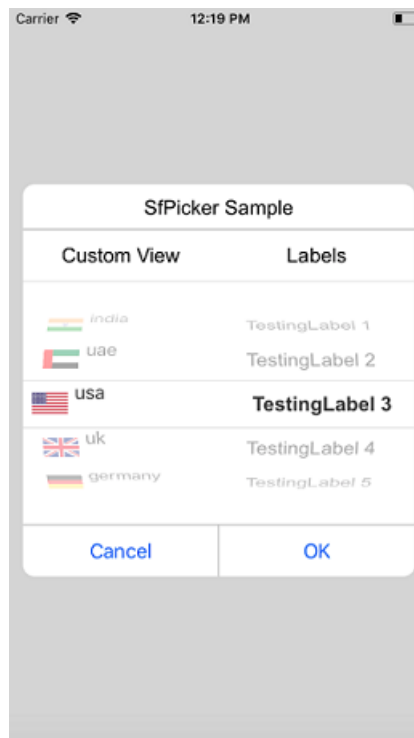
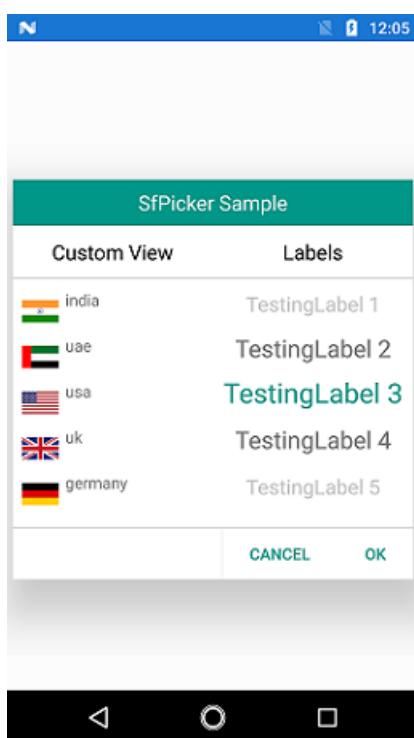
```

public partial class ItemView : ContentView
{
    Country countryName;
    int i;
    public ItemView(Country country)
    {
        i = 0;
        countryName = country;
        InitializeComponent();
        foreach(char count in country.Name)
        {
            i++;
        }
        countryName.Text = country.Name.Remove(i - 4);
        this.BindingContext = countryName;
    }
}

public class Country
{
    public string Name { get; set; }
    public string Text { get; set; }
}

```

Screen shot for the above code.

[DataTemplateSelector](#)

SfPicker supports DataTemplateSelector which you can choose a DataTemplate based on the data object.

**XML**

```

<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="DefaultTemplate">
<Grid >
<Label Text="{Binding LanguageName}" FontSize="Medium"
HorizontalOptions="Center" VerticalOptions="Center"/>
</Grid>
</DataTemplate>
<DataTemplate x:Key="SepcificTemplate">
<Grid BackgroundColor="Green">
<Label HorizontalOptions="CenterAndExpand" FontSize="Medium"
VerticalOptions="CenterAndExpand" Text="{Binding LanguageName}"/>
</Grid>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<Grid>
<syncfusion:SfPicker
ItemsSource="{Binding LanguageCollection }" >
<syncfusion:SfPicker.ItemTemplate>
<local:DataTemplateSelectorViewModel DefaultTemplate="{StaticResource
DefaultTemplate}" SpecificTemplate="{StaticResource SepcificTemplate}"/>
</syncfusion:SfPicker.ItemTemplate>
</syncfusion:SfPicker>
</Grid>
</Grid>
</Grid>
</ContentPage.Content>

```

**C#**

```

public partial class MainPage : ContentPage
{
    SfPicker picker = new SfPicker();
    DataTemplate defaultTemplate;
    DataTemplate specifictempalte;
    public MainPage()
    {
        InitializeComponent();
        this.BindingContext = new ViewModel();
        ViewModel view = new ViewModel();
        defaultTemplate = new DataTemplate(() =>
        {
            Grid grid = new Grid();
            Label label = new Label();
            label.SetBinding(Label.TextProperty, "LanguageName");
            label.HorizontalOptions = LayoutOptions.Center;
            label.VerticalOptions = LayoutOptions.Center;
            label.FontSize = 20;
            grid.Children.Add(label);
            return grid;
        });
        specifictempalte = new DataTemplate(() =>

```

```

{
    Grid maingrid = new Grid();
    Grid labelgrid = new Grid();
    Grid imagegrid = new Grid();
    Label label = new Label();
    Image image = new Image();
    label.SetBinding(Label.TextProperty, "LanguageName");
    label.FontSize = 20;
    label.HorizontalOptions = LayoutOptions.Center;
    label.VerticalOptions = LayoutOptions.Center;
    image.Source = "Crown.png";
    image.HeightRequest = 15;
    image.WidthRequest = 15;
    image.HorizontalOptions = LayoutOptions.Center;
    image.VerticalOptions = LayoutOptions.Center;
    labelgrid.Children.Add(label);
    imagegrid.Children.Add(image);
    imagegrid.Padding = new Thickness(-80, 0, 0, 0);
    maingrid.Children.Add(labelgrid);
    maingrid.Children.Add(imagegrid);
    return maingrid;
});
Grid mainGrid = new Grid();
RowDefinition firstrow = new RowDefinition();
RowDefinition secondrow = new RowDefinition();
Button button = new Button();
button.Clicked += Button_Clicked;
button.HorizontalOptions = LayoutOptions.Center;
button.VerticalOptions = LayoutOptions.Center;
button.Text = "Pick a Language";
Grid.SetRow(button, 0);
picker.ShowHeader = true;
picker.ShowFooter = true;
picker.HeaderText = "Select a Language";
picker.PickerMode = PickerMode.Dialog;
picker.ItemsSource = view.LanguageCollection;
picker.ItemTemplate = new DataTemplateSelectorViewModel { DefaultTemplate =
defaultTemplate, SpecificTemplate = specifictempalte };
Grid.SetRow(picker, 1);
mainGrid.RowDefinitions.Add(firstrow);
mainGrid.RowDefinitions.Add(secondrow);
mainGrid.Children.Add(button);
mainGrid.Children.Add(picker);
firstrow.Height = new GridLength(80);
secondrow.Height = new GridLength(500);
this.Content = mainGrid;
}

```

### *OnSelectTemplate*

The `OnSelectTemplate` is a overridden method to return a particular `DataTemplate`, which shown in the following code:

### **C#**

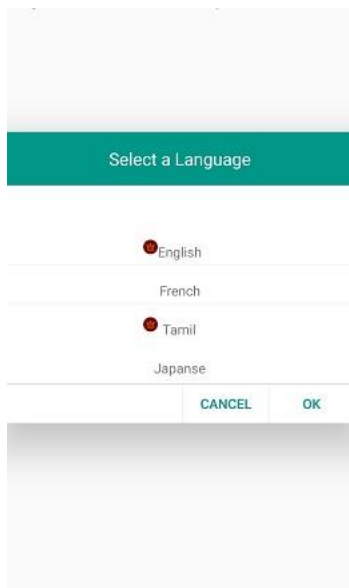
```

public class DataTemplateSelectorViewModel : DataTemplateSelector
{

```

```
private DataTemplate defaulttemplate;
public DataTemplate DefaultTemplate
{
    get { return defaulttemplate; }
    set { defaulttemplate = value; }
}
private DataTemplate specifictemplate;
public DataTemplate SpecificTemplate
{
    get { return specifictemplate; }
    set { specifictemplate = value; }
}
protected override DataTemplate OnSelectTemplate(object item, BindableObject
container)
{
    var message = item as Model;
    if (message == null)
        return null;
    return message.IsPremium ? SpecificTemplate : DefaultTemplate;
}
```

The following screenshot illustrates the output of above code.



We have attached sample for reference. You can download the sample from the following link.

Sample link: [DataTemplateSelectorSample](#)

## Dealing with Header and Footer

This section explains about the header and footer customization of picker control.

### Enable or disable header

SfPicker allows enabling or disabling the header section by setting `SfPicker.ShowHeader` property to True or False. Default value of `SfPicker.ShowHeader` property is True.

### XML

```
<syncfusion:SfPicker
x:Name="picker" ShowColumnHeader="False"
HeaderText="Select a Date" />
```

### C#

```
picker.ShowHeader = false;
```

### Set custom header

SfPicker allows providing custom text to its header by setting `SfPicker.HeaderText` property. Default value of `SfPicker.HeaderText` property is Null.

### XML

```
<syncfusion:SfPicker
x:Name="picker">
<syncfusion:SfPicker.HeaderView>
<Grid>
<Button Text="Select a Color" TextColor="Red" />
</Grid>
</syncfusion:SfPicker.HeaderView>
</syncfusion:SfPicker>
```

### C#

```
picker.HeaderText = "Select a Date";
```

### Header customization

SfPicker allows customizing background, text color, and fonts.

#### Background

Header's background color can be customized by setting `SfPicker.HeaderBackgroundColor` property.

### XML

```
<syncfusion:SfPicker
x:Name="picker"
HeaderBackgroundColor="SkyBlue"
HeaderText="Select a Date" />
```

### C#

```
picker.HeaderBackgroundColor = Color.SkyBlue;
```

#### Text-Color

Header text's color can be customized by setting `SfPicker.HeaderTextColor` property.

### XML

```
<syncfusion:SfPicker
x:Name="picker"
HeaderText="Select a Date" />
```

**C#**

```
picker.HeaderTextColor = Color.Red;
```

*Font*

This section explains about the customization of header text's of Font.

*FontFamily*

Header text's FontFamily can be customized by setting `SfPicker.HeaderFontFamily` property.

**XML**

```
<syncfusion:SfPicker
x:Name="picker"
HeaderFontFamily="Arial"
HeaderText="Select a Date" />
```

**C#**

```
picker.HeaderFontFamily = "Arial";
```

*FontSize*

Header text's FontSize can be customized by setting `SfPicker.HeaderFontSize` property.

**XML**

```
<syncfusion:SfPicker
x:Name="picker"
HeaderFontSize="18"
HeaderText="Select a Date" />
```

**C#**

```
picker.HeaderFontSize = 18;
```

*FontAttribute*

Header text's FontAttribute can be customized by setting `SfPicker.HeaderFontAttribute` property.

**XML**

```
<syncfusion:SfPicker
x:Name="picker"
HeaderFontAttribute="Italic"
HeaderText="Select a Date" />
```

**C#**

```
picker.HeaderFontAttribute = FontAttributes.Italic;
```



### Enable or disable footer

Picker allows enabling or disabling the footer section by setting `SfPicker.ShowFooter` property to True or False. Default value of `SfPicker.ShowFooter` property is False.

#### XML

```
<syncfusion:SfPicker  
x:Name="picker" ShowFooter="True"  
>
```

#### C#

```
picker.ShowFooter = true;
```

### Set custom footer

picker allows providing custom view to its footer by setting `SfPicker.FooterView` property. Default value of `SfPicker.FooterView` property is Null.

#### XML

```
<syncfusion:SfPicker x:Name="picker" ShowFooter="True">  
  <syncfusion:SfPicker.FooterView>  
    <Grid>  
      <Label Text="Cancel" TextColor="Red" />  
    </Grid>  
  </syncfusion:SfPicker.FooterView>  
</syncfusion:SfPicker>
```

#### C#

```
picker.ShowFooter = true;  
Grid layout = new Grid();  
layout.Children.Add(new Button() { Text = "Ok", TextColor = Color.Red });  
picker.FooterView = layout;
```

### Perform validation with default validation button

Picker allows performing validation based on OK or Cancel button by hooking `SfPicker.OkButtonClicked` and `SfPicker.CancelButtonClicked`. In this event, from the `SelectionChangedEventArgs` argument, current selected items can be obtained.

#### XML

```
<syncfusion:SfPicker  
x:Name="picker"  
CancelButtonClicked="picker_CancelButtonClicked"  
OkButtonClicked="picker_OkButtonClicked"  
ShowFooter="True" />
```

#### C#

```
picker.OkButtonClicked += picker_OkButtonClicked;  
picker.CancelButtonClicked += picker_CancelButtonClicked;
```

## Dealing with Columns

This section explains about the customization of picker columns

### Adjust column width

Picker allows user to adjust the column width by hooking `SfPicker.OnColumnLoaded` event, and check the column by using the `ColumnLoadedEventArgs.Column` property, and then adjust width of column by setting the `ColumnLoadedEventArgs.ColumnWidth` property.

The following code snippets demonstrate the DateTimePicker sample by using `OnColumnLoaded` event.

#### MainPage

Column width of the each column can be adjusted by using the `OnColumnLoaded` event in picker. That is implemented in the following code.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:ColumnDateTime"
x:Class="ColumnDateTime.MainPage"
xmlns:picker="clr-
namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms">
<!--Assign the TimePickerViewModel to BindingContext of Page-->
<ContentPage.BindingContext>
<local:DateTimePickerViewModel />
</ContentPage.BindingContext>
<Grid>
<Button
Clicked="Button_Clicked"
HeightRequest="50"
HorizontalOptions="Center"
Text="Show DateTimePicker"
VerticalOptions="Center"
WidthRequest="200" />
<!--Initialize the CustomDateTimePicker-->
<local:CustomDateTimePicker
x:Name="date"
ColumnHeaderHeight="40"
HorizontalOptions="Center"
OnColumnLoaded="date_OnColumnLoaded"
VerticalOptions="Center"
PickerHeight="300"
PickerMode="Dialog"
PickerWidth="310" SelectedItem="{Binding SelectedTime,Mode=TwoWay}" />
</Grid>
</ContentPage>
```

#### C#

```
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class MainPage : ContentPage
{
    public MainPage ()
    {
```

```
InitializeComponent();
}
private void Button_Clicked(object sender, EventArgs e)
{
    //open picker dialog
    date.IsOpen = !date.IsOpen;
}
private void date_OnColumnLoaded(object sender, ColumnLoadedEventArgs e)
{
    //Column width adjusted based on platform
    if (Device.OS == TargetPlatform.Android)
    {
        if (e.Column == 0)
            e.ColumnWidth = 400;
        if (e.Column == 1)
            e.ColumnWidth = 150;
        if (e.Column == 2)
            e.ColumnWidth = 150;
        if (e.Column == 3)
            e.ColumnWidth = 200;
    }
    if (Device.OS == TargetPlatform.iOS)
    {
        if (e.Column == 0)
            e.ColumnWidth = 130;
        if (e.Column == 1)
            e.ColumnWidth = 50;
        if (e.Column == 2)
            e.ColumnWidth = 50;
        if (e.Column == 3)
            e.ColumnWidth = 70;
    }
    else
    {
        if (e.Column == 0)
            e.ColumnWidth = 130;
        }
    }
}
```

You can download the sample for reference from the following link.

Sample link: [DateTimePicker](#)

#### Add caption

picker allows users to add header for each column by setting the `SfPicker.ColumnHeaderText` property and enabling the `SfPicker.ShowColumnHeader` property to True.

The `ColumnHeaderText` property is object type and user can assign string or collection.

If a string type is assigned in `SfPicker.ColumnHeaderText`, that string will be updated in all the column of picker.

To assign the collection in `SfPicker.ColumnHeaderText`, picker column header should be updated based on index with value or collection.

The following code illustrates assigning the ColumnHeaderText for picker

### C#

```
public class DatesInfo
{
    public ObservableCollection<object> Dates { get; set; }
    //Day is the collection of day numbers
    private ObservableCollection<string> Day { get; set; }
    //Month is the collection of Month Names
    private ObservableCollection<string> Month { get; set; }
    //Year is the collection of Years from 1990 to 2050
    private ObservableCollection<string> Year { get; set; }
    //ColumnHeader is the collection of Header Day,Month and Year
    public ObservableCollection<string> ColumnHeader { get; set; }
    public DatesInfo()
    {
        Dates = new ObservableCollection<object>();
        //Populate Day, Month and Year values of each collection
        PopulateDates();
        //first column of SfPicker
        Dates.Add(Day);
        //Second column of SfPicker
        Dates.Add(Month);
        //Third column of SfPicker
        Dates.Add(Year);
        ColumnHeader = new ObservableCollection<string>();
        ColumnHeader.Add("Day");
        ColumnHeader.Add("Month");
        ColumnHeader.Add("Year");
    }
    private void PopulateDates()
    {
        Day = new ObservableCollection<string>();
        Month = new ObservableCollection<string>();
        Year = new ObservableCollection<string>();
        for (int i = 1; i <= 31; i++)
            Day.Add(i.ToString());
        for (int i = 1; i <= 12; i++)
            Month.Add(System.Globalization.CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i));
        for (int i = 1990; i <= 2050; i++)
            Year.Add(i.ToString());
    }
}
```

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    x:Class="GettingStarted.PickerSample"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:GettingStarted"
    xmlns:syncfusion="clr-namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms">
    <ContentPage.BindingContext>
```

```
<local:DatesInfo />
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfPicker
x:Name="picker"
ColumnHeaderText="{Binding ColumnHeader}"
HeaderText="Select a Date"
ItemsSource="{Binding Dates}"
ShowColumnHeader="True" />
</ContentPage.Content>
</ContentPage>
```

### Caption customization

This section explains about the column header's Background, TextColor, and Fonts customization of picker.

#### Background

Column header's background color can be customized by setting the `SfPicker.ColumnHeaderBackgroundColor` property.

#### XML

```
<syncfusion:SfPicker
x:Name="picker"
ColumnHeaderBackgroundColor="SkyBlue"
ColumnHeaderText="Color"
ShowColumnHeader="True" />
```

#### C#

```
picker.ColumnHeaderBackgroundColor = Color.SkyBlue;
```

#### Text-Color

Column header's text color can be customized by setting the `SfPicker.ColumnHeaderTextColor` property.

#### XML

```
<syncfusion:SfPicker
x:Name="picker"
ColumnHeaderTextColor="Red"
ColumnHeaderText="Color"
ShowColumnHeader="True" />
```

#### C#

```
picker.ColumnHeaderTextColor = Color.Red;
```

#### Font

This section explains about the customization of column header's text or Font.

### FontFamily

Column header's text FontFamily can be customized by setting the SfPicker.ColumnHeaderFontFamily property.

#### XML

```
<syncfusion:SfPicker
x:Name="picker"
ColumnHeaderFontFamily="Courier New"
ColumnHeaderText="Color"
ShowColumnHeader="True" />
```

#### C#

```
picker.ColumnHeaderFontFamily = "Courier New";
```

### FontSize

Column header's text FontSize can be customized by setting the SfPicker.ColumnHeaderFontSize property.

#### XML

```
<syncfusion:SfPicker
x:Name="picker"
ColumnHeaderFontSize="16"
ColumnHeaderText="Color"
ShowColumnHeader="True" />
```

#### C#

```
picker.ColumnHeaderFontSize = 16;
```

### FontAttribute

Column header's text FontAttribute can be customized by setting the SfPicker.ColumnHeaderFontAttribute property.

#### XML

```
<syncfusion:SfPicker
x:Name="picker"
ColumnHeaderFontAttribute="Bold"
ColumnHeaderText="Color"
ShowColumnHeader="True" />
```

#### C#

```
picker.ColumnHeaderFontAttribute = FontAttributes.Bold;
```

### Cascading

Picker allows users to get selection change intimation by setting the SfPicker.SelectionChanged event.

Refer the following code example to prepare cascading sample for change the picker's background color when change the selection of picker.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
  x:Class="ColorCascading.MainPage"
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:ColorCascading"
  xmlns:syncfusion="clr-namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms">
  <ContentPage.BindingContext>
    <local:ColorInfo />
  </ContentPage.BindingContext>
  <Grid HorizontalOptions="Center" VerticalOptions="Center">
    <syncfusion:SfPicker
      x:Name="picker"
      HeaderText="Select a Color"
      ItemsSource="{Binding Colors}"
      PickerHeight="260"
      PickerWidth="300"
      SelectionChanged="picker_SelectionChanged"
    />
  </Grid>
</ContentPage>
```

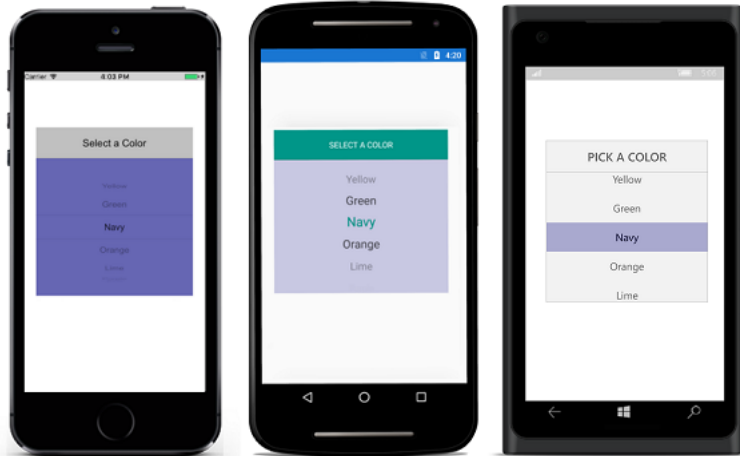
## C#

```
*MainPage*
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
    private void Button_Clicked(object sender, EventArgs e)
    {
        picker.IsOpen = true;
    }
    private void picker_SelectionChanged(object sender,
        Syncfusion.SfPicker.XForms.SelectionChangedEventArgs e)
    {
        if (e.NewValue != null)
        {
            var color = PickerHelper.GetColor(e.NewValue.ToString());
            if ((Device.OS==TargetPlatform.Android) || (Device.OS==TargetPlatform.iOS))
            {
                picker.BackgroundColor = Color.FromRgba(color.R, color.G, color.B, 0.2);
            }
            else
            {
                picker.SelectionBackgroundColor = color;
            }
        }
    }
    ColorInfo:
    public class ColorInfo
    {
```

```
private ObservableCollection<string> _color;
public ObservableCollection<string> Colors
{
    get { return _color; }
    set { _color = value; }
}
public ColorInfo()
{
    Colors = new ObservableCollection<string>();
    Colors.Add("Red");
    Colors.Add("Green");
    Colors.Add("Yellow");
    Colors.Add("Purple");
    Colors.Add("SkyBlue");
    Colors.Add("Orange");
    Colors.Add("Gray");
    Colors.Add("Pink");
    SelectedColor = "Yellow";
}
}
PickerHelper:
public static class PickerHelper
{
    static Dictionary<string, Color> colors = new Dictionary<string, Color>();
    public static Color GetColor(string color)
    {
        colors.Clear();
        colors.Add("Yellow", Color.Yellow);
        colors.Add("Green", Color.Green);
        colors.Add("Orange", Color.Orange);
        colors.Add("Lime", Color.Lime);
        colors.Add("Purple", Color.Purple);
        colors.Add("Pink", Color.Pink);
        colors.Add("Black", Color.Black);
        colors.Add("SkyBlue", Color.SkyBlue);
        colors.Add("Navy", Color.Navy);
        colors.Add("Red", Color.Red);
        colors.Add("Gray", Color.Gray);
        return colors[color.ToString()];
    }
}
```

The following screenshot illustrates the output of above code example.





You can download the sample for reference from the following link.

Sample link:[Cascading](#)

## Looping

The Looping support is used to automatically navigate the first item to repeat the list of items after reached the last item. Each forward iteration is followed by a backward iteration in the picker control. This can be achieved by `EnableLooping` property.

## EnableLooping

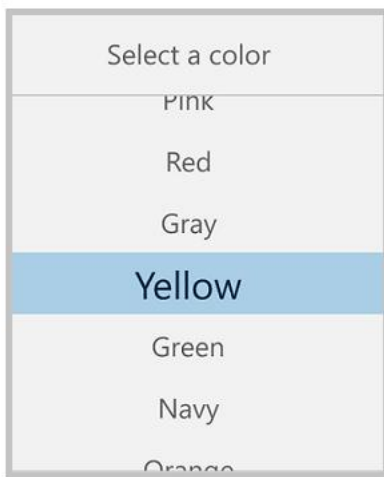
The looping support is achieved by setting the `EnableLooping` property to true.

## XML

```
<ContentPage.BindingContext>
<local:ColorInfo />
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfPicker
x:Name="picker"
HeaderText="Select a Color"
EnableLooping="True"
ItemsSource="{Binding Colors}" />
</ContentPage.Content>
</ContentPage>
```

## C#

```
SfPicker picker = new SfPicker();
ColorInfo info = new ColorInfo();
picker.ItemsSource = info.Colors;
// Enable Looping in carousel control
picker.EnableLooping = true;
```



You can find the complete Looping sample from this [link](#).

How to restrict Looping in a particular column of the picker

The looping support can be restricted in a particular column of the picker by setting the `EnableLooping` of `ColumnLoaded` event argument to false.

#### XML

```
<syncfusion:SfPicker
x:Name="picker"
OnColumnLoaded="Picker_OnColumnLoaded"/>
```

#### C#

```
private void Picker_OnColumnLoaded(object sender,
Syncfusion.SfPicker.XForms.ColumnLoadedEventArgs e)
{
    //restrict an Looping in Column 1
    if(e.Column == 1)
    {
        e.EnableLooping = false;
    }
}
```

TIME PICKER		
Hour	Minute	Format
10		
11		
12		AM
1	00	PM
2	01	
3	02	
4	03	
OK		Cancel

You can find the sample from this [link](#).

### Cascading

This section explains the steps required to create custom cascading sample by using picker control.

The cascading sample has been created for updating the state collection based on selected item picker.

Please refer the following steps to create the cascading sample

**Step 1 :** Create three ObservableCollection with object type in PickerCascading class.

#### Collection details :

Area collection, Country collection, and State collection.

Country collection -Add country names.

State collection -Add state names.

Area collection -Add above 2 collections.

Area collection is the main collection, and this collection has been assigned to ItemsSource of picker Control.

The following code demonstrates Area collection creation.

#### C#

```
public class PickerCascading:INotifyPropertyChanged
{
    #region Public Properties
    /// <summary>
    /// Area is the actual DataSource for SfPicker control which will holds the
    /// collection of Country and State
    /// </summary>
    /// <value>The area.</value>
    public ObservableCollection<object> Area { get; set; }
    //Country is the collection of country names
    private ObservableCollection<object> Country { get; set; }
```

```

//State is the collection of state names
private ObservableCollection<object> State { get; set; }
/// <summary>
/// Headers API is holds the column name for every column in cascading
picker
/// </summary>
/// <value>The Headers.</value>
public ObservableCollection<string> Header { get; set; }
private object _selectedArea;
public event PropertyChangedEventHandler PropertyChanged;
#endregion
//Identify the selected area using property changed method
public object SelectedArea
{
    get { return _selectedArea; }
    set { _selectedArea = value; RaisePropertyChanged("SelectedArea"); }
}
public PickerCascading()
{
    Area = new ObservableCollection<object>();
    Header = new ObservableCollection<string>();
    Country = new ObservableCollection<object>();
    State = new ObservableCollection<object>();
    //populate Countries
    Country.Add("UK");
    Country.Add("USA");
    Country.Add("India");
    Country.Add("UAE");
    Country.Add("Germany");
    //populate states
    State.Add("London");
    State.Add("Manchester");
    State.Add("Cambridge");
    State.Add("Edinburgh");
    State.Add("Glasgow");
    State.Add("Birmingham");
    Area.Add(Country);
    Area.Add(State);
    SelectedArea = new ObservableCollection<object>() { "UK", "London" };
}
//Hooked when changes occurred
public void RaisePropertyChanged(string name)
{
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(name));
}
}

```

**Step 2 :** Update the state collection based on selected item of country name by using the Selection changed event of picker control.

### C#

```

private void picker_SelectionChanged(object sender,
Syncfusion.SfPicker.XForms.SelectionChangedEventArgs e)
{

```

```

if (picker.ItemsSource != null && e.NewValue is IList && (picker.ItemsSource
as IList).Count > 1 && CurrentItem != (e.NewValue as IList)[0].ToString())
{
    //Updated the second column collection based on first column selected value.
    (picker.ItemsSource as ObservableCollection<object>).RemoveAt(1);
    (picker.ItemsSource as
    ObservableCollection<object>).Add(GetCountry((e.NewValue as
    IList)[0].ToString()));
}
}

```

**Step 3 :** Define column headers as “Country” and “State” by using ColumnHeaderText property of picker control. The following code demonstrates how to define header for each column of picker control.

### C#

```

public class PickerCascading:INotifyPropertyChanged
{
    /// <summary>
    /// Headers API is holds the column name for every column in cascading
    picker
    /// </summary>
    /// <value>The Headers.</value>
    public ObservableCollection<string> Header { get; set; }
    public PickerCascading()
    {
        Header = new ObservableCollection<string>();
        Header.Add("Country");
        Header.Add("State");
    }
}

```

**Step 4 :** Add the cascading picker control in main page of XAML. Please refer the following code snippets.

### XML

```

<ContentPage
x:Class="CascadingPicker.MainPage"
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:CascadingPicker"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms">
    <ContentPage.BindingContext>
    <local:PickerCascading />
    </ContentPage.BindingContext>
    <ContentPage.Content>
    <Grid HorizontalOptions="Center" VerticalOptions="Center">
    <StackLayout>
    <Button
Clicked="Button_Clicked"
HeightRequest="40"
Text="Open Picker"
WidthRequest="200" />

```

```

</StackLayout>
<syncfusion:SfPicker
x:Name="picker"
ColumnHeaderText="{Binding Header}"
HeaderText="Select your Area"
HeightRequest="350"
ItemsSource="{Binding Area}"
PickerHeight="250"
PickerMode="Dialog"
PickerWidth="280"
SelectedItem="{Binding SelectedArea}"
SelectionChanged="picker_SelectionChanged"
ShowColumnHeader="True"
WidthRequest="300" />
</Grid>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

Grid mainGrid = new Grid();
mainGrid.HorizontalOptions = LayoutOptions.Center;
mainGrid.VerticalOptions = LayoutOptions.Center;
StackLayout mainStack = new StackLayout();
mainStack.VerticalOptions = LayoutOptions.Center;
mainStack.HorizontalOptions = LayoutOptions.Center;
Button button = new Button();
button.HeightRequest = 40;
button.Text = "Open Picker";
button.WidthRequest = 200;
mainStack.Children.Add(button);
SfPicker picker = new SfPicker();
picker.HeaderText = "Select your Area";
picker.HeightRequest = 350;
picker.PickerHeight = 250;
picker.PickerMode = PickerMode.Dialog;
picker.PickerWidth = 280;
picker.SelectionChanged += Picker_SelectionChanged;
picker.ShowColumnHeader = true;
picker.SetBinding(Picker.SelectedItemProperty, "SelectedArea");
picker.SetBinding(Picker.ItemsSourceProperty, "Area");
picker.SetBinding(Picker.ColumnHeaderText, "Header");
picker.ShowFooter = true;
picker.WidthRequest = 300;
mainGrid.Children.Add(mainStack);
mainGrid.Children.Add(picker);
this.Content = mainGrid;

```

The code in the code behind is as follows.

**C#**

```

public partial class MainPage : ContentPage
{
    string CurrentItem;
    public MainPage()

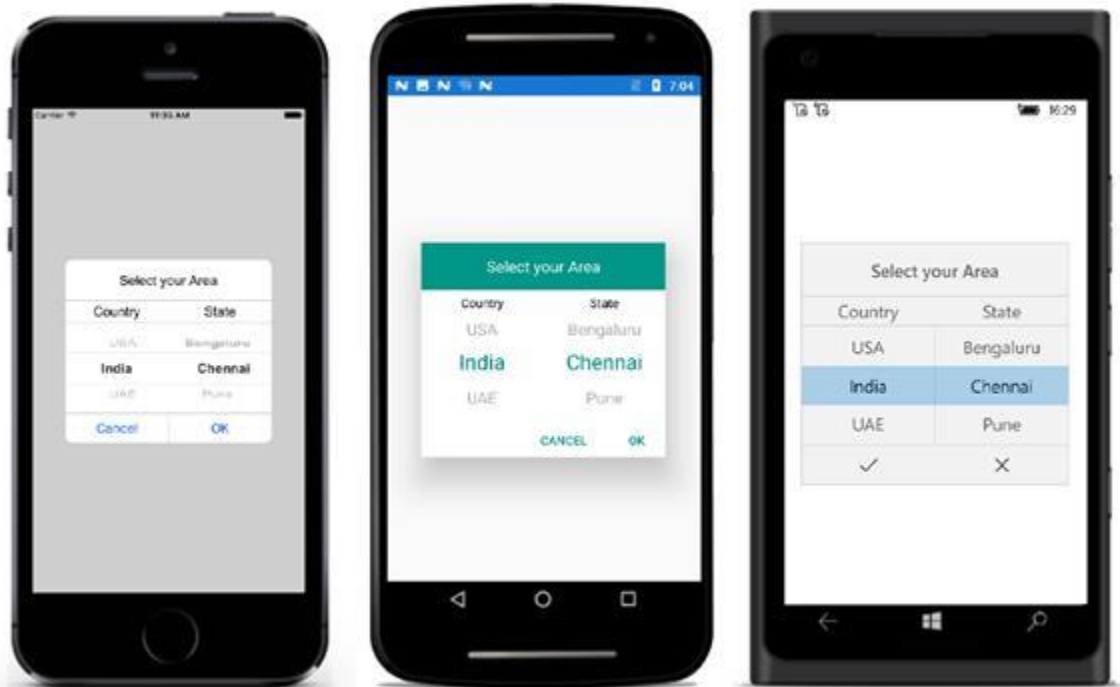
```

```

{
    InitializeComponent();
}
private void Button_Clicked(object sender, EventArgs e)
{
    picker.IsOpen = true;
}
}

```

The following screenshot illustrates the output of above code snippets.



You can download the sample for reference from the following link.

Sample link: [CascadingSample](#)

## Time Picker

This section explains the steps required to create custom TimePicker by using picker control.

**Step 1 :** Create a custom class, and name it as “CustomTimePicker”. This class should be inherited from the picker control.

### C#

```

public class CustomTimePicker: SfPicker
{
}

```

**Step 2 :** Create four ObservableCollection with object type in CustomTimePicker class.

### Collection details :

Time collection, Minute collection, Hour collection, and Format collection.

Time Collection -Add all the three collections.

Minute collection -Added minutes from 0 to 59.

Hour collection -Added hours from 1 to 12.

Format Collection -Add two formats, namely AM and PM.

The following code demonstrates time collection creation.

### **C#**

```
public class CustomTimePicker: SfPicker
{
    // Time api is used to modify the Hour collection as per change in Time
    /// <summary>
    /// Time is the actual DataSource for SfPicker control which will holds the
    collection of Hour ,Minute and Format
    /// </summary>
    public ObservableCollection<object> Time { get; set; }
    //Minute is the collection of minute numbers
    public ObservableCollection<object> Minute;
    //Hour is the collection of hour numbers
    public ObservableCollection<object> Hour;
    //Format is the collection of AM and PM
    public ObservableCollection<object> Format;
    /// <summary>
    /// Header api is holds the column name for every column in time picker
    /// </summary>
    public ObservableCollection<string> Headers { get; set; }
    public CustomTimePicker()
    {
        Time = new ObservableCollection<object>();
        Hour = new ObservableCollection<object>();
        Minute = new ObservableCollection<object>();
        Format = new ObservableCollection<object>();
        PopulateTimeCollection();
        this.ItemsSource = Time;
    }
    private void PopulateTimeCollection()
    {
        //Populate Hour
        for (int i = 1; i <= 12; i++)
        {
            Hour.Add(i.ToString());
        }
        //Populate Minute
        for (int j = 0; j < 60; j++)
        {
            if (j < 10)
            {
                Minute.Add("0" + j);
            }
            else
            {
                Minute.Add(j.ToString());
            }
        }
        //Populate Format
        Format.Add("AM");
    }
}
```



```
Format.Add("PM");  
Time.Add(Hour);  
Time.Add(Minute);  
Time.Add(Format);  
}  
}
```

**Step 3 :** Define each column headers “Hour”, “Minute”, and “Format” by using the ColumnHeaderText property of picker control. The following code demonstrates how to define header for each column of picker control.

### C#

```
public class CustomTimePicker: SfPicker  
{  
    /// <summary>  
    /// Header API is holds the column name for every column in time picker  
    /// </summary>  
    public ObservableCollection<string> Headers { get; set; }  
    public CustomTimePicker()  
    {  
        Headers = new ObservableCollection<string>();  
        if (Device.RuntimePlatform == Device.Android)  
        {  
            Headers.Add("HOUR");  
            Headers.Add("MINUTE");  
            Headers.Add("FORMAT");  
        }  
        else  
        {  
            Headers.Add("Hour");  
            Headers.Add("Minute");  
            Headers.Add("Format");  
        }  
        //SfPicker header text  
        HeaderText = "TIME PICKER";  
        // Column header text collection  
        this.ColumnHeaderText = Headers;  
    }  
}
```

**Step 4 :** Finally, enable the picker header, column header, and footer by using the ShowHeader, ShowFooter, and ShowColumnHeader properties.

### C#

```
public CustomTimePicker()  
{  
    //Enable Footer of SfPicker  
    ShowFooter = true;  
    //Enable Header of SfPicker  
    ShowHeader = true;  
    //Enable Column Header of SfPicker  
    ShowColumnHeader = true;  
}
```

**Step 5 :** Add the CustomTimePicker control in main XAML page. Please refer the following code snippets.

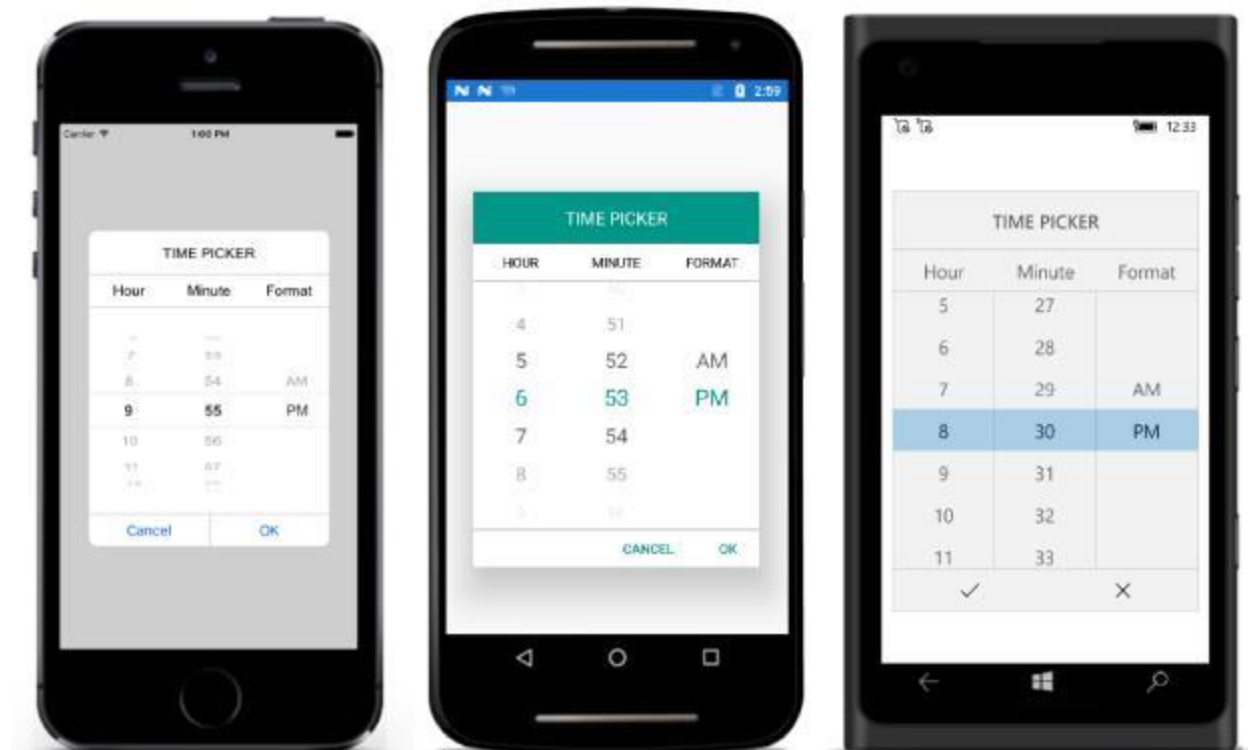
#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:TimePicker"
x:Class="TimePicker.MainPage"
xmlns:picker="clr-
namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms">
<!--Assign the TimePickerViewModel to BindingContext of Page-->
<ContentPage.BindingContext>
<local:TimePickerViewModel />
</ContentPage.BindingContext>
<Grid>
<Button
Clicked="Button_Clicked"
HeightRequest="50"
VerticalOptions="Center"
HorizontalOptions="Center"
Text="Show TimePicker"
WidthRequest="200" />
<!--Initialize the CustomTimePicker-->
<local:CustomTimePicker
x:Name="date"
ColumnHeaderHeight="40"
HorizontalOptions="Center"
VerticalOptions="Center"
PickerHeight="400"
PickerMode="Dialog"
PickerWidth="300"
SelectedItem="{Binding SelectedTime,Mode=TwoWay}" />
</Grid>
</ContentPage>
```

#### C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
    private void Button_Clicked(object sender, EventArgs e)
    {
        //open picker dialog
        date.IsOpen = !date.IsOpen;
    }
}
```

The following screenshot illustrates the output of the above code snippets.



You can download the TimePicker sample for reference from the following link.

Sample link: [TimePicker](#)

### Date Time Picker

In our Xamarin.Forms, picker control has multi column support. By using this, you we can populate day, month, year, hour, and minute values of collection in picker control.

This section explains create custom DateTimePicker by using picker control.

**Step 1 :** Create a custom class, and name it as “DateTimePicker”. This class should be inherited from picker control.

#### C#

```
public class DateTimePicker : SfPicker
{
}
```

**Step 2 :** Create six ObservableCollection with object type in DateTimePicker class.

#### Collection details :

Day collection, Month collection, Year collection, Hour collection, and Minute collection.

Day collection Add current month's days by using DateTime.DaysInMonth.

Month collection -Add months from January to December.

Year collection -Add years from 1990 to 2050.

Hour collection -Add hours from 0 to 24.

Minute collection -Add minutes from 00 to 59.

Date collection -Add all the five collections.

Date collection is the main collection, and this collection has been assigned to ItemsSource of picker control.

The following code demonstrates date collection creation.

### C#

```
public class DateTimePicker : SfPicker
{
    #region Public Properties
    // Months API is used to modify the Day collection as per change in Month
    internal Dictionary<string, string> Months { get; set; }
    /// <summary>
    /// Date is the actual DataSource for SfPicker control which will holds the
    collection of Day ,Month and Year
    /// </summary>
    /// <value>The date.</value>
    public ObservableCollection<object> Date { get; set; }
    //Day is the collection of day numbers
    internal ObservableCollection<object> Day { get; set; }
    //Month is the collection of Month Names
    internal ObservableCollection<object> Month{ get; set; }
    //Year is the collection of Years from 1990 to 2042
    internal ObservableCollection<object> Year{ get; set; }
    //Hour is the collection of Hours in Railway time format
    internal ObservableCollection<object> Hour{ get; set; }
    //Minute is the collection of Minutes from 00 to 59
    internal ObservableCollection<object> Minute{ get; set; }
    #endregion
    public DateTimePicker()
    {
        Months = new Dictionary<string, string>();
        Date = new ObservableCollection<object>();
        Day = new ObservableCollection<object>();
        Month = new ObservableCollection<object>();
        Year = new ObservableCollection<object>();
        Hour = new ObservableCollection<object>();
        Minute = new ObservableCollection<object>();
        PopulateDateCollection();
        this.ItemsSource = Date;
    }
    private void PopulateDateCollection()
    {
        //populate months
        for (int i = 1; i < 13; i++)
        {
            if
            (!Months.ContainsKey(CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(
            i).Substring(0, 3)))
            Months.Add(CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i).Substri
            ng(0, 3), CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i));
            Month.Add(CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i).Substrin
            g(0, 3));
        }
    }
}
```

```
//populate year
for (int i = 1990; i < 2050; i++)
{
    Year.Add(i.ToString());
}
//populate Days
for (int i = 1; i <= DateTime.DaysInMonth(DateTime.Now.Year,
    DateTime.Now.Month); i++)
{
    if (i < 10)
    {
        Day.Add("0" + i);
    }
    else
    {
        Day.Add(i.ToString());
    }
}
//populate Hours
for (int i = 1; i <= 24; i++)
{
    if (i < 10)
    {
        Hour.Add("0" + i.ToString());
    }
    else
    {
        Hour.Add(i.ToString());
    }
}
//populate Minutes
for (int j = 0; j < 60; j++)
{
    if (j < 10)
    {
        Minute.Add("0" + j);
    }
    else
    {
        Minute.Add(j.ToString());
    }
}
Date.Add(Year);
Date.Add(Month);
Date.Add(Day);
Date.Add(Hour);
Date.Add(Minute);
}
```

**Step 3 :** Update the day value based on month and year values by using Selection changed event of picker control. Since the days of each month differs, you should handle this collection.

### C#

```
public DateTimePicker()
{
    //hook selection changed event
    this.SelectionChanged += CustomDatePicker_SelectionChanged;
}
private void CustomDatePicker_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
```

```

{
    UpdateDays(Date, e);
}
//Update days method is used to alter the Date collection as per selection change in Month column(if Feb is Selected day collection has value from 1 to 28)
public void UpdateDays(ObservableCollection<object> Date,
    SelectionChangedEventArgs e)
{
    Device.BeginInvokeOnMainThread(() =>
    {
        if (Date.Count == 5)
        {
            bool flag = false;
            if (e.OldValue != null && e.NewValue != null && (e.OldValue is
                ObservableCollection<object>) && (e.OldValue as
                ObservableCollection<object>).Count > 0)
            {
                if (!object.Equals((e.OldValue as IList)[1], (e.NewValue as IList)[1]))
                {
                    flag = true;
                }
                if (!object.Equals((e.OldValue as IList)[0], (e.NewValue as IList)[0]))
                {
                    flag = true;
                }
            }
            if (flag)
            {
                ObservableCollection<object> days = new ObservableCollection<object>();
                int month = DateTime.ParseExact(Months[(e.NewValue as IList)[1].ToString()],
                    "MMMM", CultureInfo.InvariantCulture).Month;
                int year = int.Parse((e.NewValue as IList)[0].ToString());
                for (int j = 1; j <= DateTime.DaysInMonth(year, month); j++)
                {
                    if (j < 10)
                    {
                        days.Add("0" + j);
                    }
                    else
                    {
                        days.Add(j.ToString());
                    }
                }
                ObservableCollection<object> PreviousValue = new
                ObservableCollection<object>();
                foreach (var item in e.NewValue as IList)
                {
                    PreviousValue.Add(item);
                }
                if (days.Count > 0)
                {
                    Date.RemoveAt(2);
                    Date.Insert(2, days);
                }
                if ((Date[2] as IList).Contains(PreviousValue[2]))
                {
                    this.SelectedItem = PreviousValue;
                }
            }
        }
    }
}

```

```

else
{
    PreviousValue[2] = (Date[2] as IList)[(Date[2] as IList).Count - 1];
    this.SelectedItem = PreviousValue;
}
}
}
});
}

```

**Step 4 :** Define each column headers “Day”, “Month”, “Year”, “Hour”, and “Minute” by using the ColumnHeaderText property. The following code demonstrates how to define header for each column of picker control.

#### C#

```

public class DateTimePicker : SfPicker
{
    /// <summary>
    /// Headers API is holds the column name for every column in date picker
    /// </summary>
    /// <value>The Headers.</value>
    public ObservableCollection<string> Headers { get; set; }
    public DateTimePicker()
    {
        Headers = new ObservableCollection<string>();
        Headers.Add("Year");
        Headers.Add("Month");
        Headers.Add("Day");
        Headers.Add("Hour");
        Headers.Add("Minute");
        //SfPicker header text
        HeaderText = "Date Picker";
        // Column header text collection
        this.ColumnHeaderText = Headers;
    }
}

```

**Step 5 :** Finally enable the picker header, Column header and footer by using the ShowHeader, ShowFooter and ShowColumnHeader properties.

#### C#

```

public DateTimePicker()
{
    //Enable Footer
    ShowFooter = true;
    //Enable SfPicker Header
    ShowHeader = true;
    //Enable Column Header of SfPicker
    ShowColumnHeader = true;
}

```

**Step 6 :** Add the DateTimePicker control in main XAML page. Please refer the following code snippets.

## XML

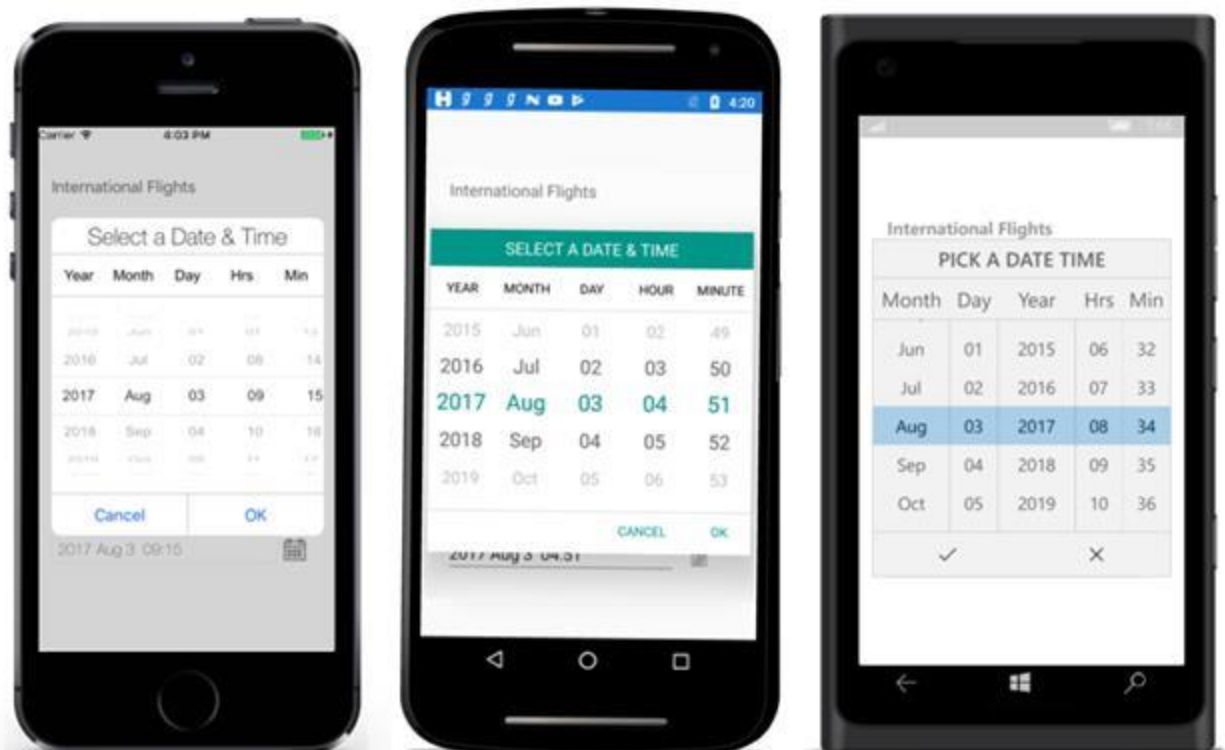
```
<ContentPage
x:Class="SfPDatetimeSample.MainPage"
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SfPDatetimeSample"
xmlns:picker="clr-
namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms">
  <ContentPage.BindingContext>
  <local:DateTimeViewModel />
  </ContentPage.BindingContext>
  <Grid>
    <Button
      Clicked="Button_Clicked"
      HeightRequest="50"
      HorizontalOptions="Center"
      Text="Show Picker"
      VerticalOptions="Center"
      WidthRequest="200" />
    <local:DateTimePicker
      x:Name="date"
      ColumnHeaderHeight="40"
      HorizontalOptions="Start"
      PickerHeight="400"
      PickerMode="Dialog"
      PickerWidth="300"
      SelectedItem="{Binding StartDate}"
      VerticalOptions="Center" />
  </Grid>
</ContentPage>
```

## C#

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
    private void Button_Clicked(object sender, EventArgs e)
    {
        date.IsOpen = !date.IsOpen;
    }
}
```

The following screenshot illustrates the output of the above codes.





You can download the DateTimePicker sample for reference from the following link.

Sample link: [DateTimePicker](#)

## Date Picker

This section explains the steps required to create custom Date Picker by using picker control.

**Step 1 :** Create a custom class, and named it as “CustomDatePicker”. This class should be inherited from picker control.

### C#

```
public class CustomDatePicker : SfPicker
{
}
```

**Step 2 :** Create four ObservableCollection with object type in CustomDatePicker class.

### Collection details :

Date collection, Day collection, Month collection, and Year collection.

Day collection -Add current month's days by using DateTime.DaysInMonth.

Month collection -Add months from January to December.

Year collection -Add years from 1990 to 2050.

Date collection -Add all the three collections.

Date Collection is the main collection, and this collection has been assigned to ItemsSource of picker control.

The following code demonstrates Date collection creation.

### C#

```
public class CustomDatePicker : SfPicker
{
    #region Public Properties
    // Months API is used to modify the Day collection as per change in Month
    internal Dictionary<string, string> Months { get; set; }
    /// <summary>
    /// Date is the actual DataSource for SfPicker control which will holds the
    collection of Day ,Month and Year
    /// </summary>
    /// <value>The date.</value>
    public ObservableCollection<object> Date { get; set; }
    //Day is the collection of day numbers
    internal ObservableCollection<object> Day { get; set; }
    //Month is the collection of Month Names
    internal ObservableCollection<object> Month{ get; set; }
    //Year is the collection of Years from 1990 to 2042
    internal ObservableCollection<object> Year{ get; set; }
    #endregion
    public CustomDatePicker()
    {
        Months = new Dictionary<string, string>();
        Date = new ObservableCollection<object>();
        Day = new ObservableCollection<object>();
        Month = new ObservableCollection<object>();
        Year = new ObservableCollection<object>();
        PopulateDateCollection();
        this.ItemsSource = Date;
    }
    private void PopulateDateCollection()
    {
        //populate months
        for (int i = 1; i < 13; i++)
        {
            if
            (!Months.ContainsKey(CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(
            i).Substring(0, 3)))
            Months.Add(CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i).Substri
            ng(0, 3), CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i));
            Month.Add(CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i).Substrin
            g(0, 3));
        }
        //populate year
        for (int i = 1990; i < 2050; i++)
        {
            Year.Add(i.ToString());
        }
        //populate Days
        for (int i = 1; i <= DateTime.DaysInMonth(DateTime.Now.Year,
            DateTime.Now.Month); i++)
        {
            if (i < 10)
            {
                Day.Add("0" + i);
            }
        }
    }
}
```

```

}
else
Day.Add(i.ToString());
}
Date.Add(Month);
Date.Add(Day);
Date.Add(Year);
}
}

```

**Step 3 :** Update the day value based on month and year values by using Selection changed event of picker control. Since the days of each month differs, you should handle this collection.

### C#

```

public CustomDatePicker()
{
    //hook selection changed event
    this.SelectionChanged += CustomDatePicker_SelectionChanged;
}

private void CustomDatePicker_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    UpdateDays(Date, e);
}

//Update days method is used to alter the Date collection as per selection
change in Month column(if Feb is Selected day collection has value from 1 to
28)
public void UpdateDays(ObservableCollection<object> Date,
SelectionChangedEventArgs e)
{
    Device.BeginInvokeOnMainThread(() =>
    {
        if (Date.Count == 3)
        {
            bool flag = false;
            if (e.OldValue != null && e.NewValue != null && (e.OldValue as
ObservableCollection<object>).Count == 3 && (e.NewValue as
ObservableCollection<object>).Count== 3 )
            {
                if (!object.Equals((e.OldValue as IList)[0], (e.NewValue as IList)[0]))
                {
                    flag = true;
                }
                if (!object.Equals((e.OldValue as IList)[2], (e.NewValue as IList)[2]))
                {
                    flag = true;
                }
            }
            if (flag)
            {
                ObservableCollection<object> days = new ObservableCollection<object>();
                int month = DateTime.ParseExact(Months[(e.NewValue as IList)[0].ToString()],
"MMMM", CultureInfo.InvariantCulture).Month;
                int year = int.Parse((e.NewValue as IList)[2].ToString());
                for (int j = 1; j <= DateTime.DaysInMonth(year, month); j++)

```

```

{
    if (j < 10)
    {
        days.Add("0" + j);
    }
    else
    {
        days.Add(j.ToString());
    }
    ObservableCollection<object> PreviousValue = new
    ObservableCollection<object>();
    foreach (var item in e.NewValue as IList)
    {
        PreviousValue.Add(item);
    }
    if (days.Count > 0)
    {
        Date.RemoveAt(1);
        Date.Insert(1, days);
    }
    if ((Date[1] as IList).Contains(PreviousValue[1]))
    {
        this.SelectedItem = PreviousValue;
    }
    else
    {
        PreviousValue[1] = (Date[1] as IList)[(Date[1] as IList).Count - 1];
        this.SelectedItem = PreviousValue;
    }
}
});
}

```

**Step 4 :** Define each column headers “Day”, “Month”, and “Year” by using ColumnHeaderText property of picker control. The following code demonstrates how to define header for each column of picker control.

#### C#

```

public class CustomDatePicker : SfPicker
{
    /// <summary>
    /// Headers API is holds the column name for every column in date picker
    /// </summary>
    /// <value>The Headers.</value>
    public ObservableCollection<string> Headers { get; set; }
    public CustomDatePicker()
    {
        Headers = new ObservableCollection<string>();
        Headers.Add("Month");
        Headers.Add("Day");
        Headers.Add("Year");
        //SfPicker header text
        HeaderText = "Date Picker";
        // Column header text collection
        this.ColumnHeaderText = Headers;
    }
}

```

```
}
}
```

**Step 5 :** Finally, enable the picker header, Column header, and footer by using ShowHeader, ShowFooter and ShowColumnHeader properties.

### C#

```
public CustomDatePicker()
{
    //Enable Footer
    ShowFooter = true;
    //Enable SfPicker Header
    ShowHeader = true;
    //Enable Column Header of SfPicker
    ShowColumnHeader = true;
}
```

**Step 6 :** Add the CustomDatePicker control in main XAML page. Please refer the following code snippets.

### XML

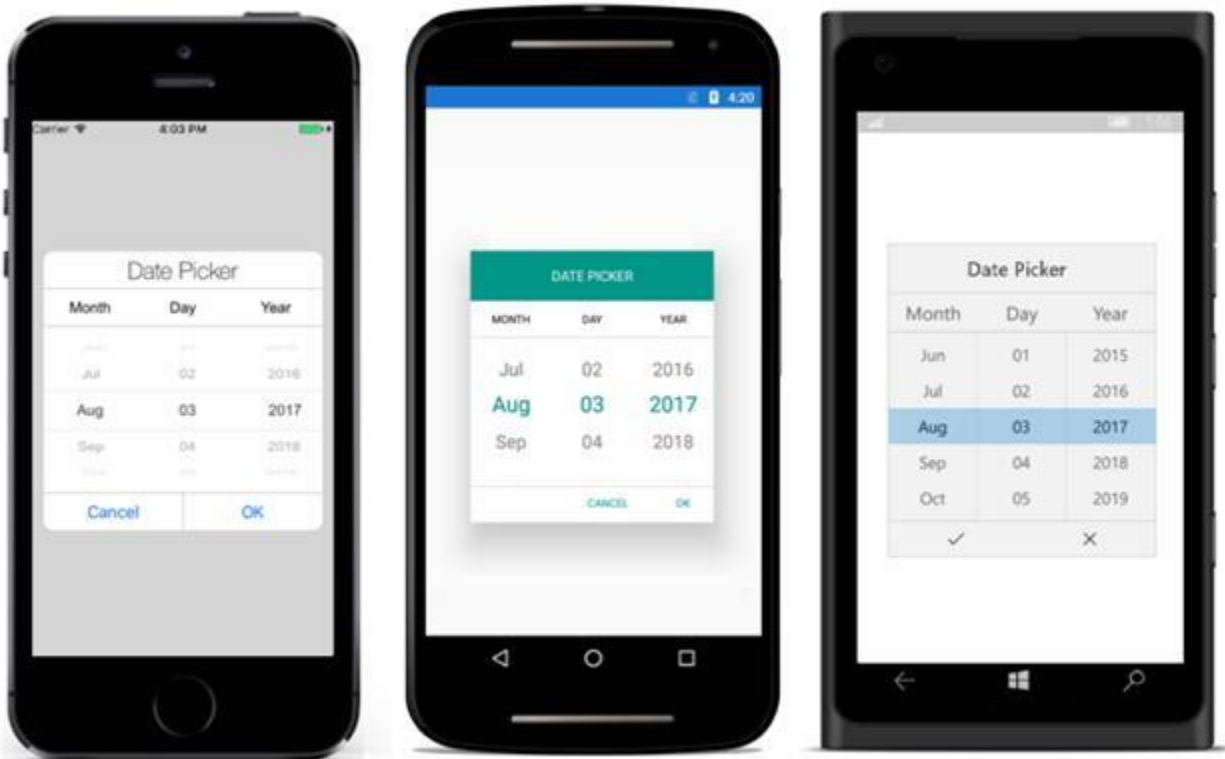
```
<ContentPage
x:Class="DatePicker.DatePickerPage"
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:DatePicker"
xmlns:picker="clr-
namespace:Syncfusion.SfPicker.XForms;assembly=Syncfusion.SfPicker.XForms">
<ContentPage.BindingContext>
<local:DatePickerViewModel />
</ContentPage.BindingContext>
<Grid>
<Button
Clicked="Button_Clicked"
HeightRequest="30"
HorizontalOptions="Center"
Text="Show Picker"
VerticalOptions="Center"
WidthRequest="200" />
<local:CustomDatePicker
x:Name="date"
ColumnHeaderHeight="40"
HorizontalOptions="Center"
PickerHeight="400"
PickerMode="Dialog"
PickerWidth="300"
SelectedItem="{Binding StartDate}"
VerticalOptions="Center" />
</Grid>
</ContentPage>
```

### C#

```
public partial class DatePickerPage : ContentPage
```

```
{  
public DatePickerPage()  
{  
InitializeComponent();  
}  
private void Button_Clicked(object sender, EventArgs e)  
{  
date.IsOpen = !date.IsOpen;  
}  
}
```

The following screenshot illustrates the output of above code snippets.



You can download the DatePicker sample for reference from the following link.

Sample link: [DatePicker](#)

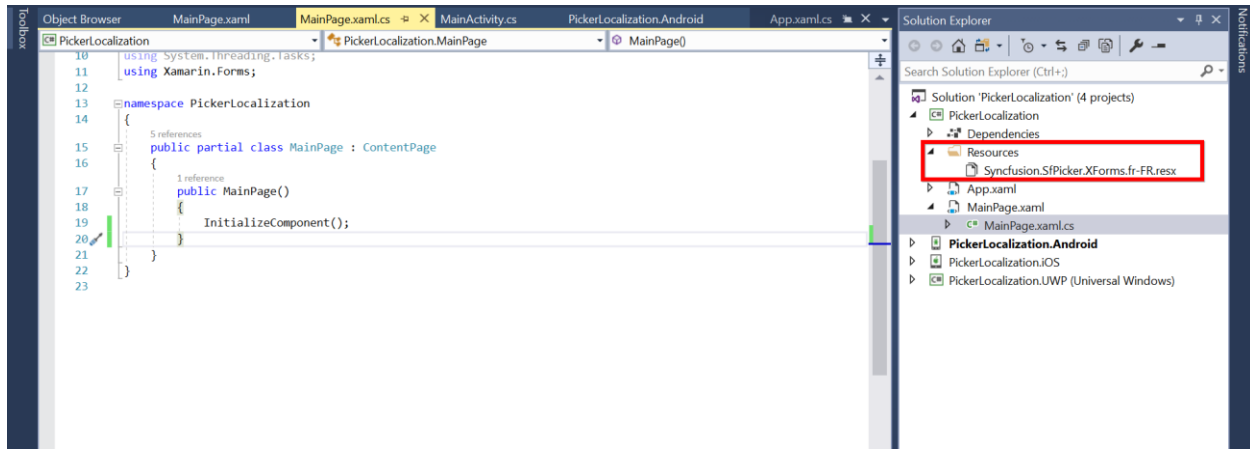
### Localization

You can localize Picker in all the platforms by adding a.resx file in a .NET Standard project alone. The following steps describe how to localize SfPicker in a project and you can download the complete sample from this [link](#).

**NOTE:** Here, the resources have been already created for some cultures and shared them on [Syncfusion GitHub](#) for your convenience.

1. Add a new folder in the .NET Standard project named Resources.
2. Add resource files for the languages you wish to support, and set their Build Action to EmbeddedResource. The name of the resource file should be \$name of the Syncfusion component\$+\$language code\$.resx. For example, if you add a resource file for the French

culture, add the Syncfusion.SfPicker.XForms.fr-FR.resx file to Resources folder as illustrated in the following screenshot.



3. Provide the French values for each key in the respective .resx files. Here, "Cancel" and "Ok" are the keys, and "Annuler" and "D'accord" are their respective French values.

#### XML

```
<data name="Cancel" xml:space="preserve">
<value>Annuler</value>
</data>
<data name="Ok" xml:space="preserve">
<value>D'accord</value>
</data>
```

4. Set the resource manager to `PickerResourceManager.Manager` as demonstrated in the following code to get the resource manager from the users. For more details, refer [Localization](#).

#### C#

```
PickerResourceManager.Manager = new
ResourceManager("PickerLocalization.Resources.Syncfusion.SfPicker.XForms",
Application.Current.GetType().Assembly);
```

#### Localize at application level

You can also localize the text at application-level regardless of the language selected on the device. The following platform-specific codes are needed to localize the text at application-level.

#### C#

```
//For Android and iOS,
Thread.CurrentThread.CurrentUICulture = new CultureInfo("fr-FR");
//For UWP,
CultureInfo.CurrentUICulture = new CultureInfo("fr-FR");
```

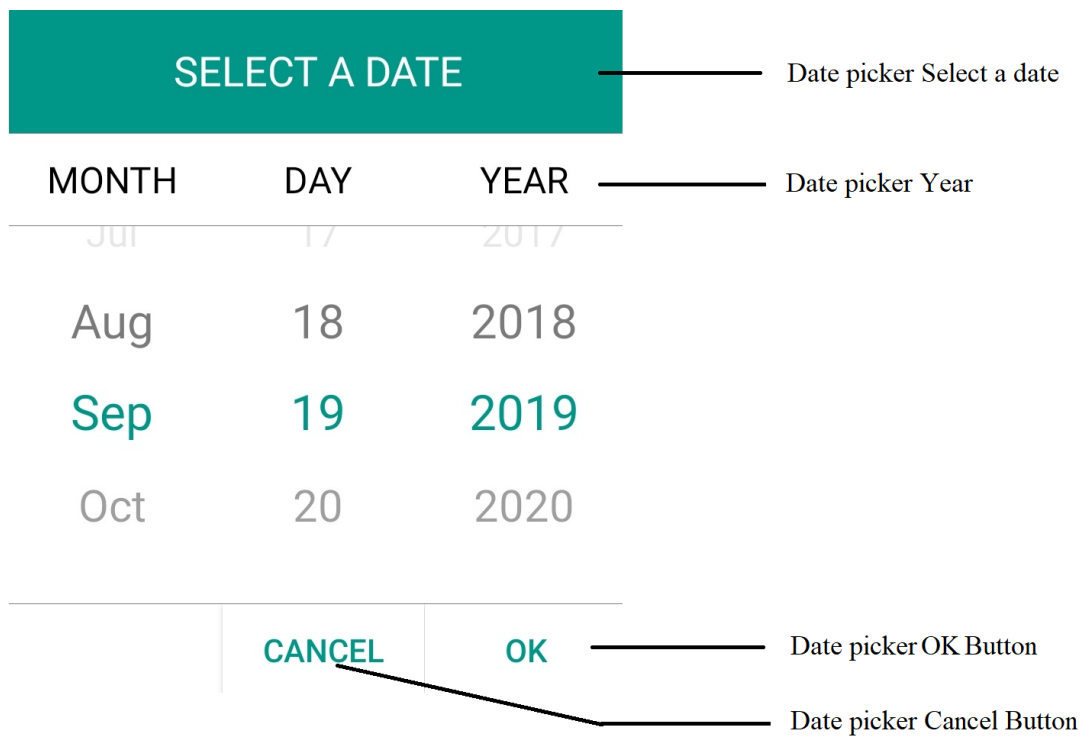
![Localization support in Xamarin.Forms SfPicker](images/Xamarin.Forms SfPicker.jpg)

### AutomationId

The SfPicker control has built-in **AutomationId** for inner elements. The **AutomationId** API allows the automation framework to find and interact with the inner elements of the **SfPicker** control. To keep unique **AutomationId**, these inner elements' **AutomationIds** are updated based on the control's **AutomationId**.

For example, if you set SfPicker's **AutomationId** as "Date picker", then the automation framework will interact with the OK button as "Date picker OK Button". The following screenshot illustrates the **AutomationIds** of inner elements.

You can only interact with the header, column header, OK, and Cancel buttons.



## SfPopupLayout

### Overview

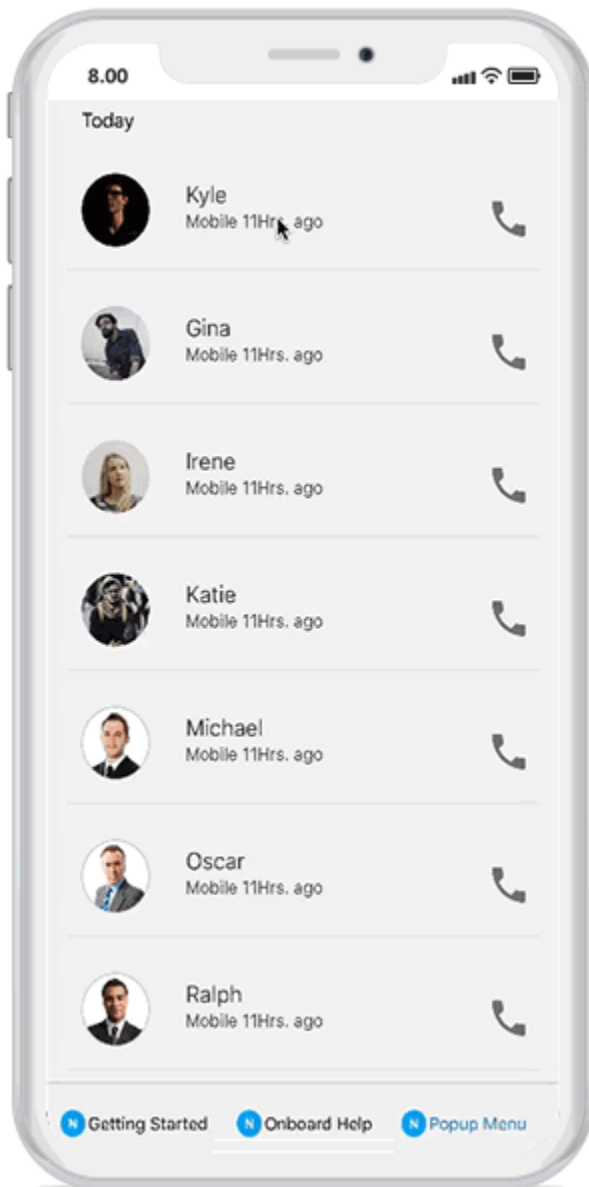
Popup layout provides a layout that allows users to display an alert message with customizable buttons or load any desired view inside a popup. It provides options to fully customize the popup with custom header, body, and footer capabilities, allowing you to display important information to use precisely how user want to.

### Key features

- Load complex views and layouts as content.



- Display popups with minimalistic code by calling only a single method.
- Choose from default layouts, such as one-button layout and two-button layout, to display simple and minimalistic information.
- Display popups in positions relative to a view on screen.
- Display popups at the touch interaction positions.
- Display popups at specific x and y coordinates with desired height and width or with default dimensions in the center.



## Getting Started

This section provides a quick overview for working with the SfPopupLayout for Xamarin.Forms.

### Assembly deployment

After installing Essential Studio for Xamarin, you can get all the required assemblies in the {Syncfusion Essential Studio Installed location}\Essential Studio\{{ site.releaseversion }}\Xamarin\lib installation folder.

E.g. C:\Program Files (x86)\Syncfusion\Essential Studio\{{ site.releaseversion }}\Xamarin\lib

---

**Note:** Assemblies can be found in an unzipped package location in Mac.

---

### NuGet configuration

To install the required NuGet for the SfPopupLayout control in the application, configure the NuGet packages of the Syncfusion components.

Refer to the following KB to configure the NuGet package of the Syncfusion components:

[How to configure package source and install Syncfusion NuGet packages in an existing project?](#)

The following NuGet package should be installed to use the SfPopupLayout control in the application.

Project	Required package
Xamarin.Forms	Syncfusion.Xamarin.SfPopupLayout

### Adding SfPopupLayout reference

You can add SfPopupLayout reference using one of the following methods:

#### Method 1: Adding SfPopupLayout reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfPopupLayout). To add SfPopupLayout to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfPopupLayout](#), and then install it.

![Adding SfPopupLayout reference from NuGet](GettingStarted\_images/Adding SfPopupLayout reference.png)

---

**Note:** Install the same version of SfPopupLayout NuGet in all the projects.

---

#### Method 2: Adding SfPopupLayout reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfPopupLayout control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfPopupLayout assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfPopupLayout.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfPopupLayout.XForms.dll Syncfusion.SfPopupLayout.XForms.Android.dll

	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfPopupLayout.XForms.dll Syncfusion.SfPopupLayout.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfPopupLayout.XForms.dll Syncfusion.SfPopupLayout.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the SfPopupLayout on each platform

To use the SfPopupLayout inside an application, each platform application must initialize the SfPopupLayout renderer. This initialization step varies from platform to platform and is discussed in the following sections.

#### Android

If you are using SfPopupLayout by [Displaying popup when the SfPopupLayout is set as root view](#), the Android launches the SfPopupLayout without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

If you are using SfPopupLayout by [Displaying popup on the go](#), call the `SfPopupLayoutRenderer.Init()` in the `OnCreate` overridden method of the MainActivity.cs class after the Xamarin.Forms Framework initialization and before the LoadApplication is called as demonstrated in the following code example.

#### C#

```
protected override void OnCreate(Bundle bundle)
{
    ...
    global::Xamarin.Forms.Forms.Init(this, bundle);
    Syncfusion.XForms.Android.PopupLayout.SfPopupLayoutRenderer.Init();
    LoadApplication(new App());
}
```

**Note:** If you are adding the references from toolbox, this step is not needed.

*iOS*

To launch the SfPopupLayout in iOS, call the `SfPopupLayoutRenderer.Init()` in the `FinishedLaunching` overridden method of the AppDelegate class after the Xamarin.Forms Framework initialization and before the `LoadApplication` is called as demonstrated in the following code example.

**C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.XForms.iOS.PopupLayout.SfPopupLayoutRenderer.Init ();
    LoadApplication (new App ());
    ...
}
```

*Universal Windows Platform (UWP)*

To launch the SfPopupLayout in UWP, call the `SfPopupLayoutRenderer.Init()` in the `MainPage` constructor before the `LoadApplication` is called as demonstrated in the following code example.

**C#**

```
public MainPage()
{
    ...
    Syncfusion.XForms.UWP.PopupLayout.SfPopupLayoutRenderer.Init ();
    LoadApplication (new App ());
    ...
}
```

*ReleaseMode issue in UWP platform*

The known Framework issue in UWP platform is the custom controls will not render when deployed the application in **Release Mode**. It can be resolved by initializing the SfPopupLayout assemblies in `App.xaml.cs` in UWP project as in the following code snippet.

**C#**

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfPopupLayoutRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Create a simple popup

This section explains how to create a SfPopupLayout and configure it.

Create a new BlankApp (Xamarin.Forms.Portable) application in Xamarin Studio or Visual Studio for Xamarin.Forms.

### Adding SfPopupLayout in Xamarin.Forms

1. Add the required assembly references to the pcl and renderer projects as discussed in the [Assembly deployment](#) section.
2. Import the control namespace as xmlns:syncfusion="clr-namespace:Syncfusion.XForms.SfPopupLayout;assembly=Syncfusion.SfPopupLayout.XForms" in XAML Page.
3. The SfPopupLayout can be displayed by the following cases.

#### Displaying popup when the SfPopupLayout is set as root view

The SfPopupLayout can be displayed by making it as base view or content view of the main page. For the first case, set the view over which the SfPopupLayout should be displayed as the content of the SfPopupLayout by using the [SfPopupLayout.Content](#) property.

Refer to the following code example for displaying popup.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="mainLayout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked"/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

#### C#

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

```
private void ClickToShowPopup_Clicked(object sender, EventArgs e)
{
    popupLayout.Show();
}
}
```

### Displaying popup on the go

You can continue to keep your view as the content view of the main page and still display popup over your view by simply calling the [SfPopupLayout.Show](#) method.

Refer to the following code example for displaying popup.

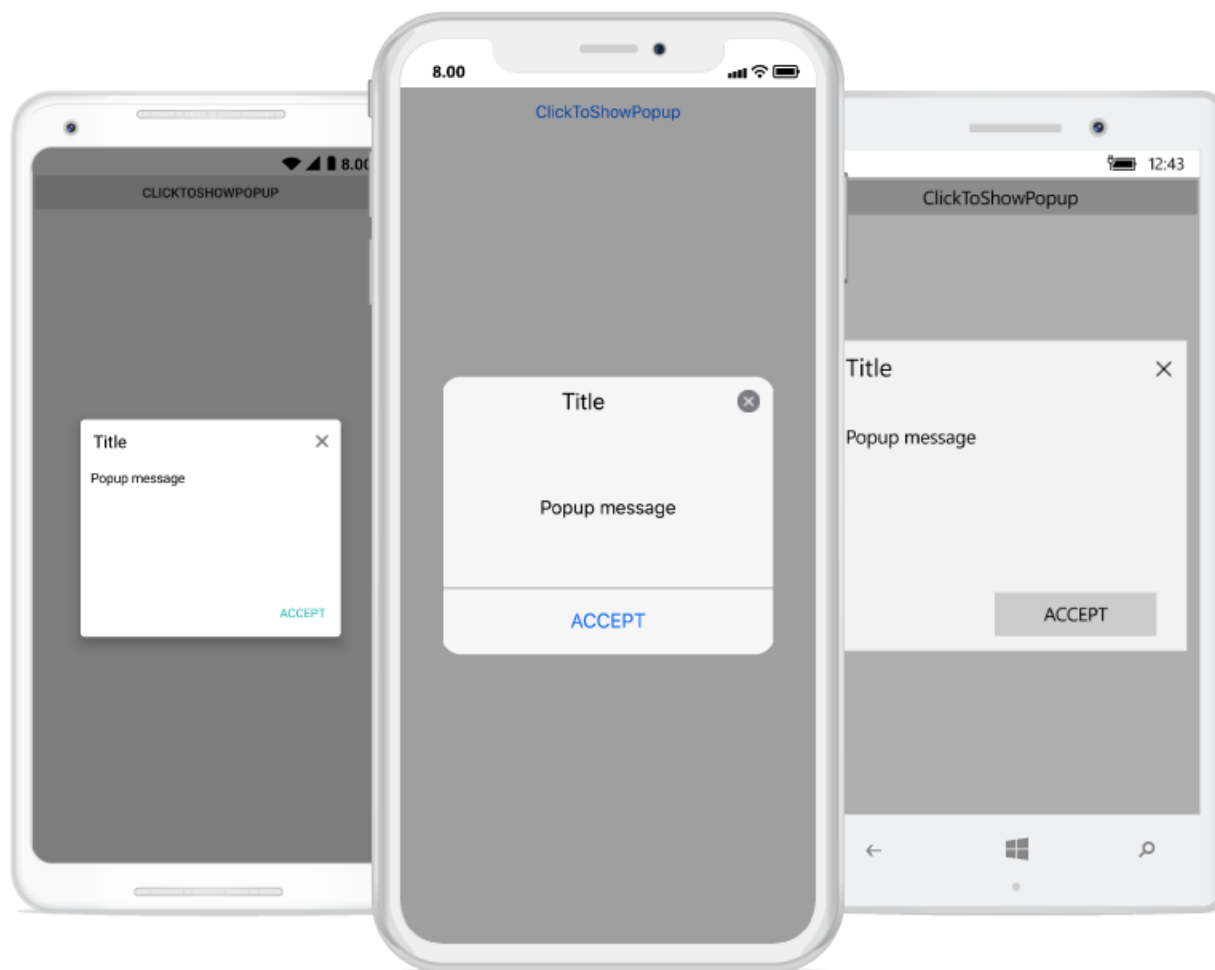
### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0">
<StackLayout x:Name="mainLayout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked" />
</StackLayout>
</ContentPage>
```

### C#

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        SfPopupLayout popupLayout;
        public MainPage()
        {
            InitializeComponent();
            popupLayout = new SfPopupLayout();
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



You can download the source code of this sample [here](#).

---

**Note:** Two Popups cannot be displayed at the same time in a page.

---

### Customize positioning

The SfPopupLayout allows showing the popup content at various positions.

The following list of options are available to position the SfPopupLayout in the desired position:

- **Center Positioning:** Use the [SfPopupLayout.IsOpen](#) property or [SfPopupLayout.Show](#) method to display the SfPopupLayout at the center.
- **Absolute Positioning:** Use the [SfPopupLayout.Show\(x-position, y-position\)](#) to display the SfPopupLayout at the specified X and y position.
- **OnTouch:** Use the [SfPopupLayout.ShowAtTouchPoint\(\)](#) to display the SfPopupLayout at the touch point.
- **Relative Positioning:** Use the [SfPopupLayout.ShowRelativeToView\(View, RelativePosition\)](#) to display the SfPopupLayout at any of the 8 positions relative to the specified view.
- **Absolute relative positioning:** Use the [SfPopupLayout.ShowRelativeToView\(View, RelativePosition, x position, y position\)](#) to display the SfPopupLayout at an absolute x,y coordinate from the relative position of the specified view.

More information for popup positioning is in this [link](#).

### Customizing layouts

By default, you can choose a layout from the following available layouts in the SfPopupLayout by using the property [SfPopupLayout.AppearanceMode](#).

- [OneButton](#): Shows the SfPopupLayout with one button in the footer view. This is the default value.
- [TwoButton](#): Shows the SfPopupLayout with two buttons in the footer view.

You can also customize the entire view of the popup by loading the templates or custom views for the header, body, and footer.

More information for popup positioning is in this [link](#).

### *Load your template view in the popup body*

Any view can be added as popup content by using the [SfPopupLayout.PopupView.ContentTemplate](#) property to refresh it. Refer to the following code example in which a label is added as a popup content and displaying the popup when the SfPopupLayout is set as root view.

### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView>
<sfPopup:PopupView.ContentTemplate>
<DataTemplate>
<Label Text="This is SfPopupLayout" BackgroundColor="SkyBlue"
HorizontalTextAlignment="Center"/>
</DataTemplate>
</sfPopup:PopupView.ContentTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start"
HorizontalOptions="FillAndExpand" Clicked="ClickToShowPopup_Clicked" />
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

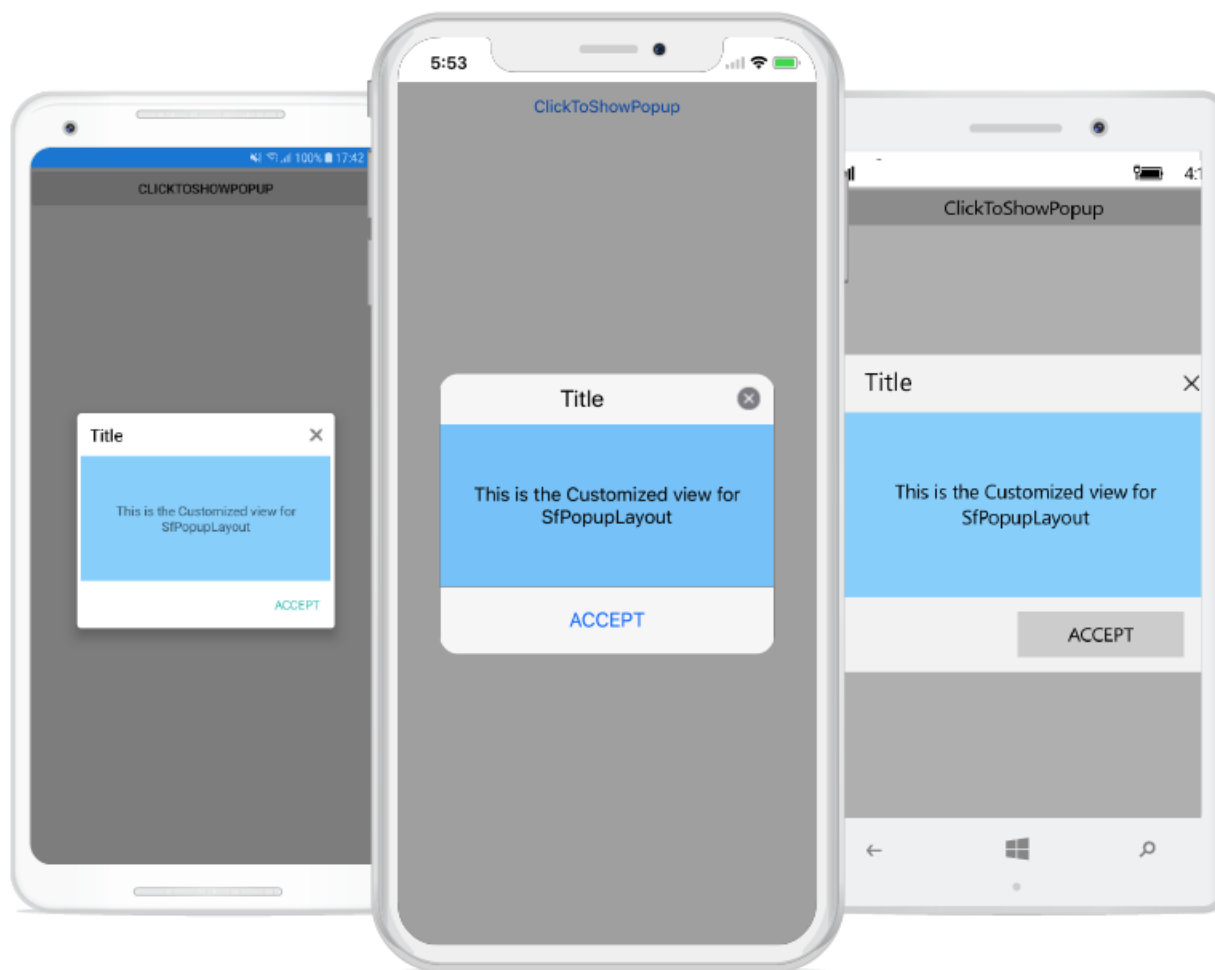
### **C#**

```
using Syncfusion.XForms.PopupLayout;
```



```
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        DataTemplate templateView;
        Label popupContent;
        public MainPage()
        {
            InitializeComponent();
            templateView = new DataTemplate(() =>
            {
                popupContent = new Label();
                popupContent.Text = "This is the SfPopupLayout";
                popupContent.BackgroundColor = Color.LightSkyBlue;
                popupContent.HorizontalTextAlignment = TextAlignment.Center;
                return popupContent;
            });
            // Adding ContentTemplate of the SfPopupLayout
            popupLayout.PopupView.ContentTemplate = templateView;
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



**Note:** Setting the content view is same for both cases i.e. displaying the popup when the SfPopupLayout is set as root view and vice versa.

### Customizing animations

Available built-in animations will be applied to the SfPopupLayout when the PopupView opens and closes in the screen. By default, you can choose from the following animations available in the SfPopupLayout by using the property [SfPopupLayout.AnimationMode](#):

- [Zoom](#): Zoom-out animation will be applied when the PopupView opens and zoom-in animation will be applied when the PopupView closes. This is the default AnimationMode.
- [Fade](#): Fade-out animation will be applied when the PopupView opens and fade-in animation will be applied when the PopupView closes.
- [SlideOnLeft](#): PopupView will be animated from left-to-right when it opens and from right-to-left when the PopupView closes.
- [SlideOnRight](#): PopupView will be animated from right-to-left when it opens and from left-to-right when the PopupView closes.
- [SlideOnTop](#): PopupView will be animated from top-to-bottom when it opens and from bottom-to-top when the PopupView closes.

- [SlideOnBottom](#): PopupView will be animated from bottom-to-top when it opens and from top-to-bottom when the PopupView closes
- [None](#): Animations will not be applied.

More information for popup animations is in this [link](#).

#### XML

```
<sfPopup:SfPopupLayout x:Name="popupLayout">
  <sfPopup:SfPopupLayout.PopupView>
    <sfPopup:PopupView AnimationMode="Fade" />
  </sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

#### C#

```
//MainPage.cs
public MainPage()
{
    InitializeComponent();
    popupLayout.PopupView.AnimationMode = AnimationMode.Fade;
}
```

### Popup Positioning

The SfPopupLayout allows showing the popup content at various available positions.

Following are the list of options available to show SfPopupLayout at various positions.

Methods / Properties	Description
<a href="#">SfPopupLayout.IsOpen</a>	Shows the SfPopupLayout at center.
<a href="#">SfPopupLayout.Show</a>	Similar as SfPopupLayout.IsOpen property.
<a href="#">SfPopupLayout.Show(x-position, y-position)</a>	Shows the SfPopupLayout at the specified X and y positions.
<a href="#">SfPopupLayout.ShowAtTouchPoint()</a>	Shows the SfPopupLayout at the touch point.
<a href="#">SfPopupLayout.ShowRelativeToView(View, RelativePosition)</a>	Shows the SfPopupLayout at the position relative to the specified view.
<a href="#">SfPopupLayout.ShowRelativeToView(View, RelativePosition,x-position,y-position)</a>	Shows the SfPopupLayout at an absolute x, y coordinate from the relative position of the specified view.
<a href="#">SfPopupLayout.PopupView.IsFullScreen</a>	Shows the SfPopupLayout in full width and height of the screen.
<a href="#">SfPopupLayout.Show(bool)</a>	Similar as SfPopupLayout.PopupView.IsFullScreen property.

### Center positioning

SfPopupLayout can be shown at the center by using the following options.

- [IsOpen property](#)
- [SfPopupLayout.Show](#)

To open the SfPopupLayout, use the `SfPopupLayout.IsOpen` property as in the following code sample.

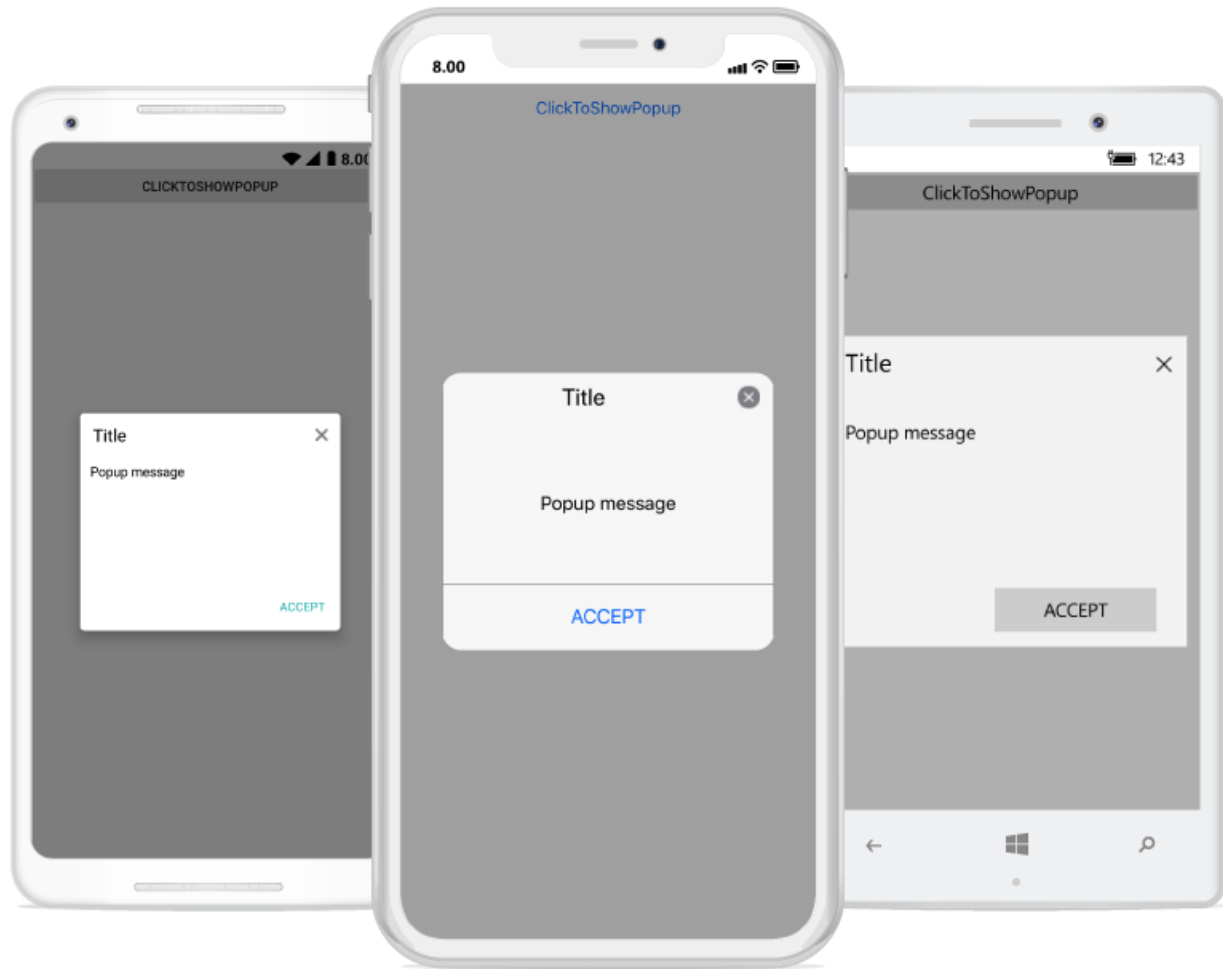
#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked" />
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

#### C#

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            // Shows SfPopupLayout at the center of the view.
            popupLayout.IsOpen = true;
        }
    }
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



To open the SfPopupLayout, use the [SfPopupLayout.Show](#) method as in the following code sample.

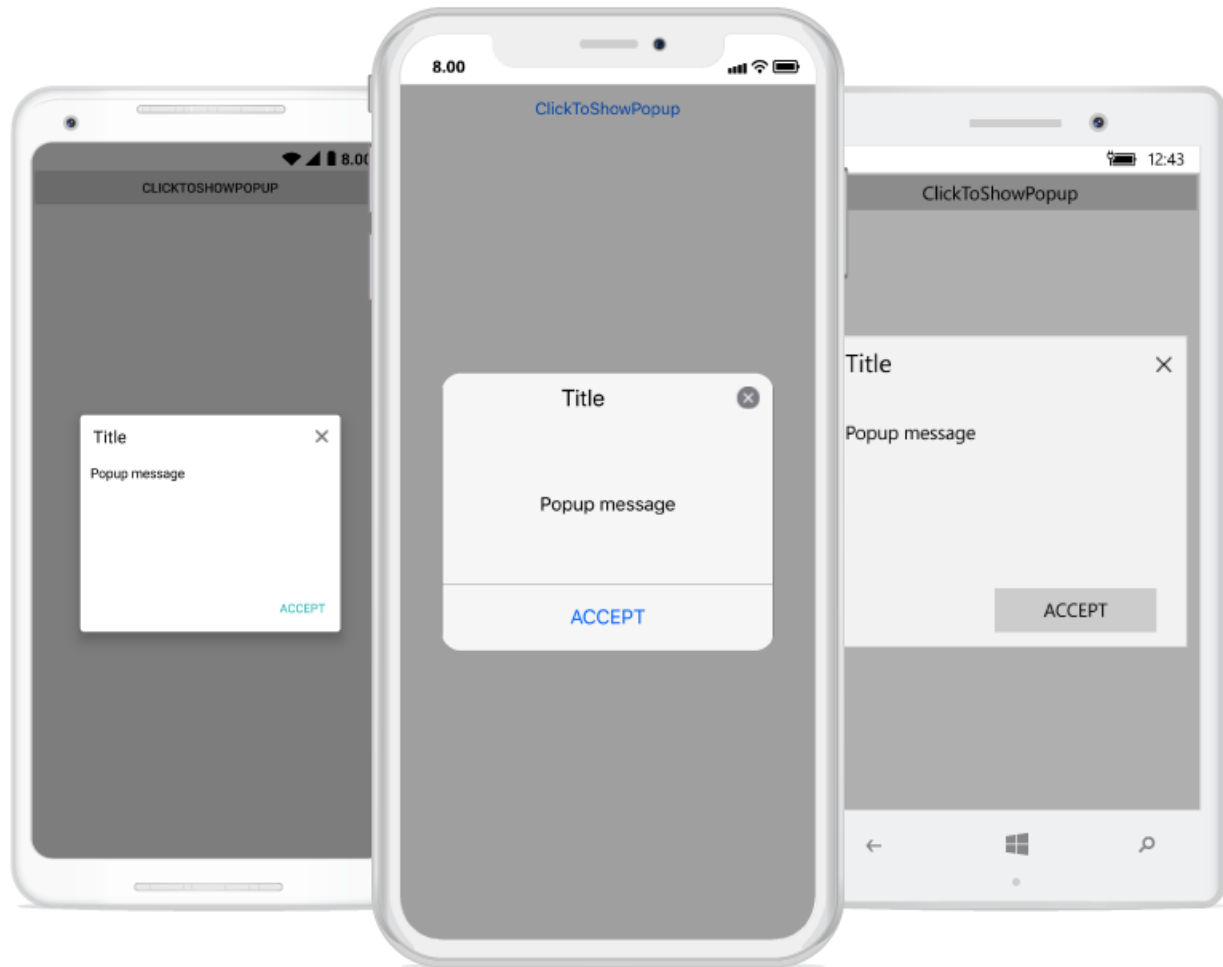
#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked" />
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

**C#**

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            // Shows SfPopupLayout at the center of the view.
            popupLayout.Show();
        }
    }
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



### Absolute positioning

To open the SfPopupLayout in specific X,Y coordinates, use the `SfPopupLayout.Show(x-position, y-position)` property as in the following code sample.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked" />
</StackLayout>
</sfPopup:SfPopupLayout>
```

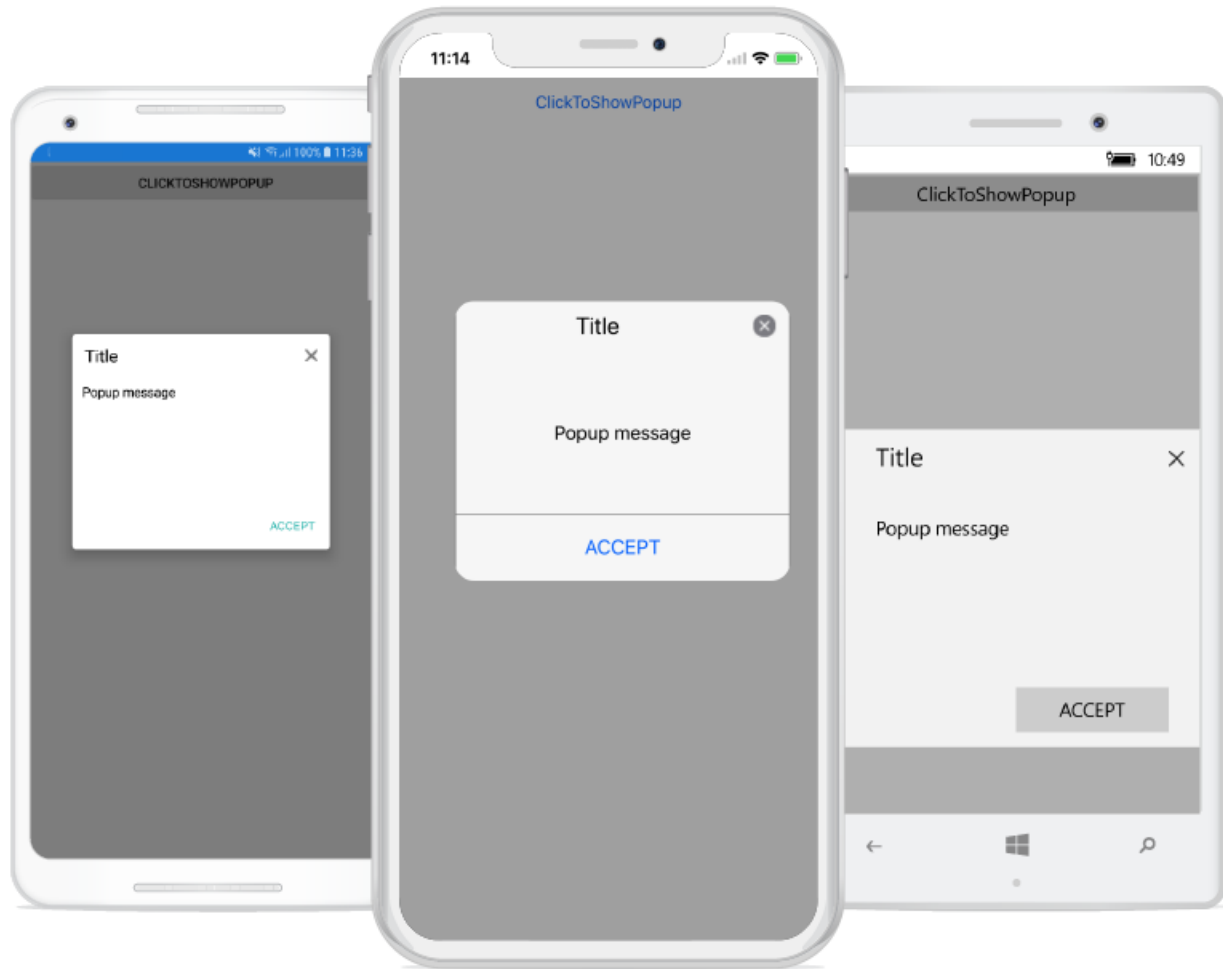
```
</sfPopup:SfPopupLayout.Content>  
</sfPopup:SfPopupLayout>  
</ContentPage>
```

## C#

```
using Syncfusion.XForms.PopupLayout;  
namespace GettingStarted  
{  
    public partial class MainPage : ContentPage  
    {  
        public MainPage()  
        {  
            InitializeComponent();  
        }  
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)  
        {  
            // Shows SfPopupLayout at x-position 100 and y position 100.  
            popupLayout.Show(100, 700);  
        }  
    }  
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.





Position popup at touch point

To open the SfPopupLayout from the touch point in the screen, use the [SfPopupLayout.ShowAtTouchPoint](#) method as in the following code sample.

#### XML

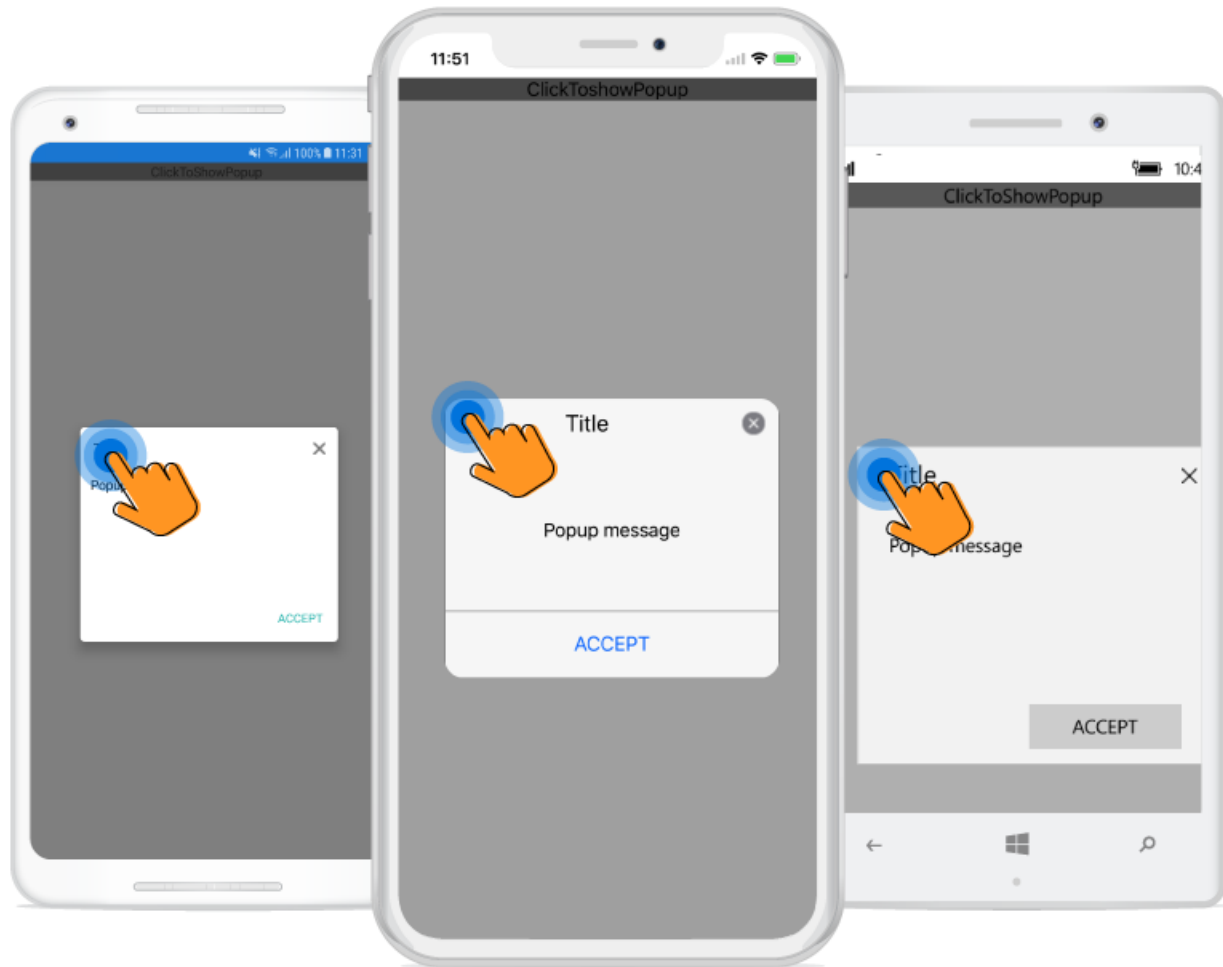
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked" />
</StackLayout>
```

```
</sfPopup:SfPopupLayout.Content>  
</sfPopup:SfPopupLayout>  
</ContentPage>
```

### **C#**

```
using Syncfusion.XForms.PopupLayout;  
namespace GettingStarted  
{  
    public partial class MainPage : ContentPage  
    {  
        public MainPage()  
        {  
            InitializeComponent();  
        }  
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)  
        {  
            // Shows SfPopupLayout at the touch point.  
            popupLayout.ShowAtTouchPoint();  
        }  
    }  
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



**Note:** Positioning popup at touch point is not supported when using the approach of **Displaying popup on the fly**.

### Relative positioning

SfPopupLayout can be shown at the relative position by using the following method.

#### *Display popup relative to a view*

To open the SfPopupLayout relative to a view, use the **SfPopupLayout.ShowRelativeToView(View, RelativePosition,x-position,y-position)** method.

### XML

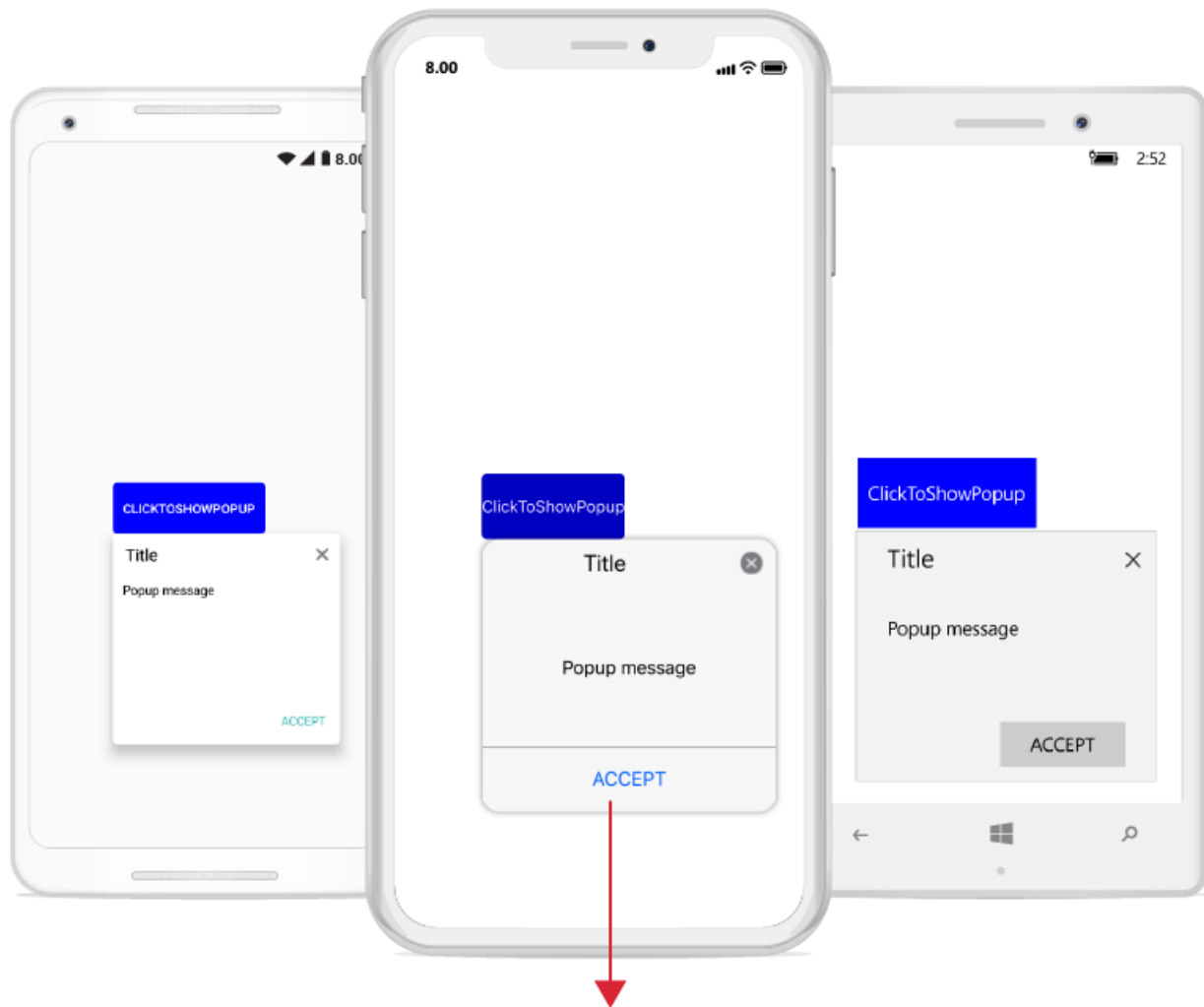
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
```

```
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="mainLayout" VerticalOptions="Center"
HorizontalOptions="Center">
<StackLayout VerticalOptions="CenterAndExpand"
HorizontalOptions="StartAndExpand">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup" TextColor="White"
HeightRequest="60" VerticalOptions="Center" Margin="100,0,0,0"
HorizontalOptions="Center" BackgroundColor="Blue"
Clicked="ClickToShowPopup_Clicked"/>
</StackLayout>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

### C#

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            // Shows SfPopupLayout at the bottom of the label.
            popupLayout.ShowRelativeToView(label, RelativePosition.AlignBottom, 0, 0);
        }
    }
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



Popup is shown at the bottom of the button

#### *Display popup relatively to a view with absolute coordinates*

The SfPopupLayout can be displayed at an absolute x, y coordinate from the relative position of the specified view by using the following method.

To open the SfPopupLayout in the specific x, y coordinate relative to a view, use the `SfPopupLayout.ShowRelativeToView(View, RelativePosition,x-position,y-position)` method.

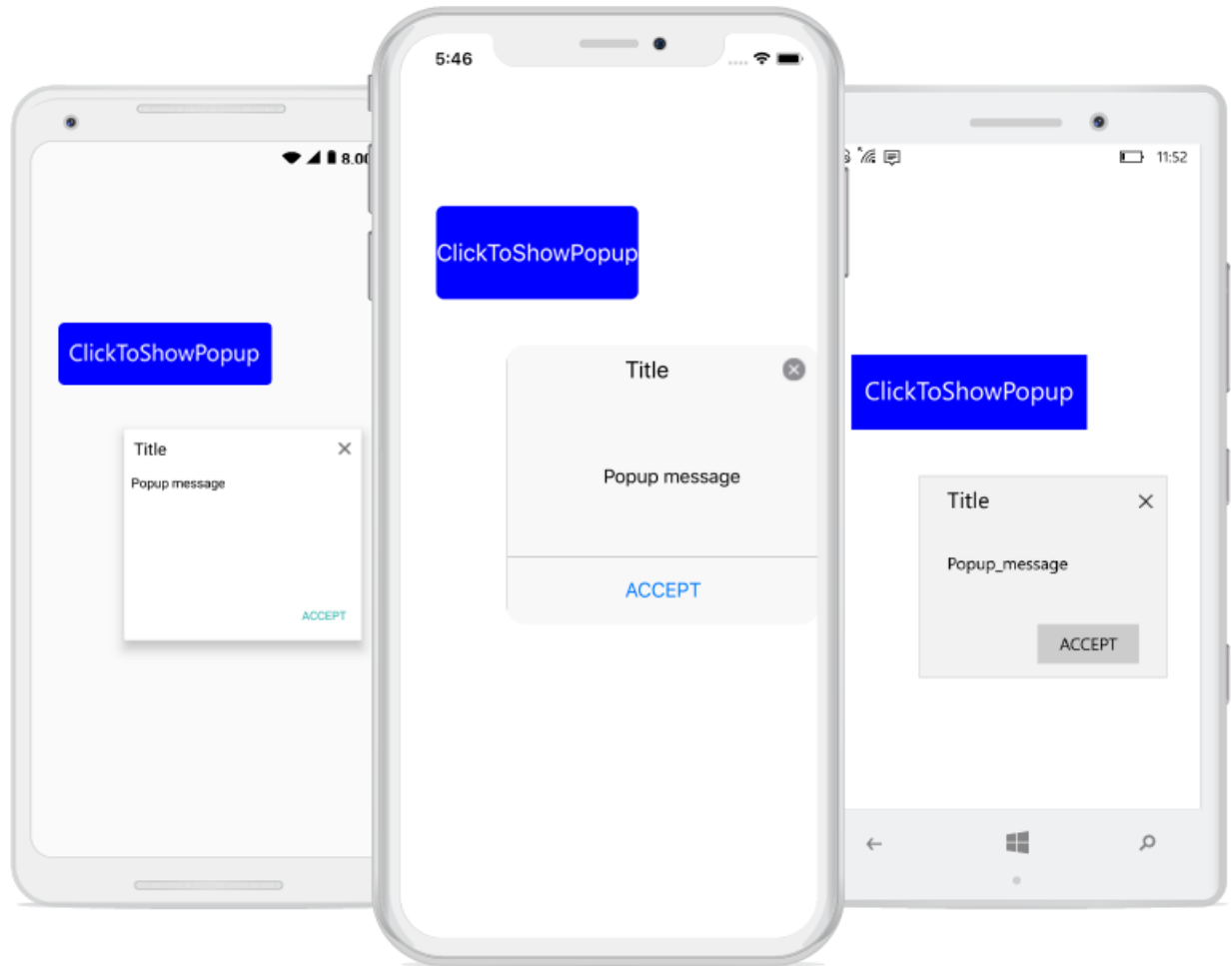
#### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,200,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
```

```
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="mainLayout" VerticalOptions="StartAndExpand"
HorizontalOptions="Start">
<StackLayout VerticalOptions="StartAndExpand"
HorizontalOptions="StartAndExpand">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup" TextColor="White"
HeightRequest="60" VerticalOptions="Start" Margin="50,0,0,0"
HorizontalOptions="StartAndExpand" BackgroundColor="Blue"
Clicked="ClickToShowPopup_Clicked"/>
</StackLayout>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

## C#

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            // Shows SfPopupLayout at the bottom of the label and with absolute relative
            position.
            popupLayout.ShowRelativeToView(label, RelativePosition.AlignBottom, 50, 50);
        }
    }
}
```



You can pass both negative and positive values as parameters to the `SfPopupLayout.ShowRelativeToView(View, RelativePosition, x-position, y-position)`. The popup will be positioned by considering the relative position as (0, 0) the center point. For example, if you have set the `RelativePosition` as `RelativePosition.BottomRight` and `RelativeLayout` as a button, bottom right corner of the button will be considered as the 0, 0 point and a negative x-position value will place the popup to the left of that point and a positive x-position value will place the popup to the right of that point. The same applies for y-position also.

---

**Note:** To open the `SfPopupLayout` relative to a view without absolute position, pass the x-position and y-position parameters as 0 in `SfPopupLayout.ShowRelativeToView(View, RelativePosition, x-position, y-position)`.

---

#### Show relative to view in MVVM

To open the `SfPopupLayout` relative to a view in MVVM assign values to the `SfPopupLayout.RelativeLayout` and `SfPopup.RelativePosition` properties and use the `SfPopupLayout.IsOpen` property to open or close the popup using binding.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
```

```

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<ContentPage.Content>
<sfPopup:SfPopupLayout x:Name="popupLayout" IsOpen="{Binding DisplayPopup}"
RelativePosition="AlignBottom" AbsoluteX="0" AbsoluteY="0">
<sfPopup:SfPopupLayout.Content>
<StackLayout >
<Label x:Name="relativeView" Text="Showing Popup at relative position in
MVVM" VerticalOptions="StartAndExpand" HorizontalOptions="FillAndExpand"
HorizontalTextAlignment="Center" HeightRequest="60" LineBreakMode="WordWrap"
FontSize="Medium" BackgroundColor="Blue" TextColor="White"/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
<sfPopup:SfPopupLayout.RelativeView>
<x:Reference Name="relativeView"/>
</sfPopup:SfPopupLayout.RelativeView>
</sfPopup:SfPopupLayout>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        protected override void OnAppearing()
        {
            base.OnAppearing();
            viewModel.DisplayPopup = true;
        }
    }
}

```

```
// ViewModel
```

```

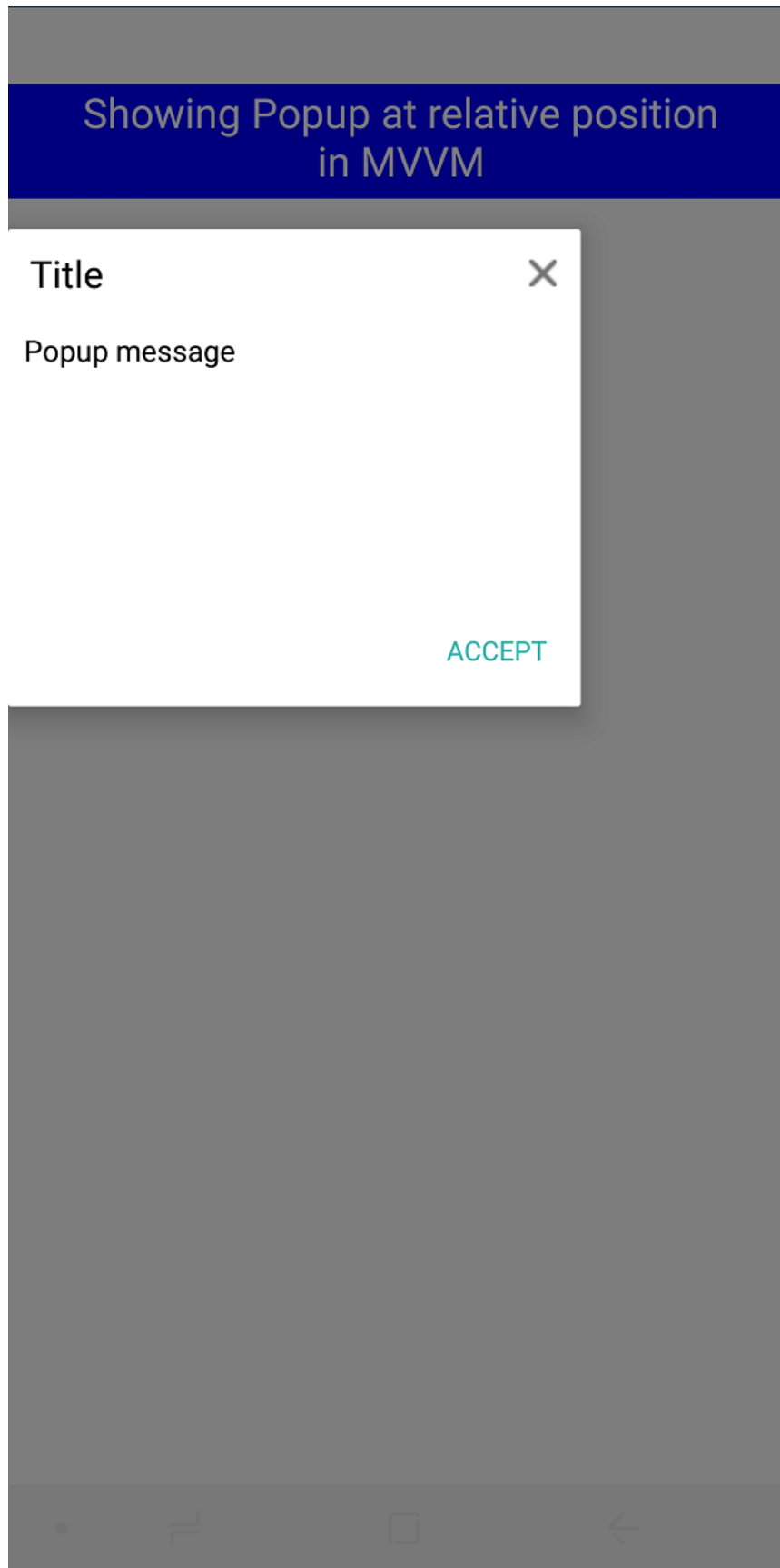
public class ViewModel: INotifyPropertyChanged
{
    private bool displayPopup;
    public bool DisplayPopup

```



```
{
get { return displayPopup; }
set { displayPopup = value; RaisePropertyChanged("DisplayPopup"); }
}
public ViewModel()
{
this.displayPopup = false;
}
public void RaisePropertyChanged(string propName)
{
if (this.PropertyChanged != null)
this.PropertyChanged(this, new PropertyChangedEventArgs(propName));
}
public event PropertyChangedEventHandler PropertyChanged;
}
```

Executing the above codes renders the following output in Android device respectively.



## Popup Size

SfPopupLayout allows displaying the Popup at any desired width and height by setting the SfPopupLayout.PopupView.WidthRequest and SfPopupLayout.PopupView.HeightRequest. The Popup size can also be changed by setting width request and height request to the views loaded inside the templates of the Popup.

In the following code snippet the Popup is loaded in 350x350 pixels, where width request is set for the ListView that is loaded as template content and height request is set for the PopupView.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:PopupDemo"
xmlns:sfPopup="clr-
namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XF
orms"
xmlns:sfListView="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
x:Class="PopupDemo.MainPage">
<ContentPage.BindingContext>
<local:ContactsViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView HeaderTitle="ListView" HeightRequest ="350">
<sfPopup:PopupView.ContentTemplate>
<DataTemplate>
<sfListView:SfListView x:Name="listView" ItemSpacing="5"
WidthRequest="350"
ItemsSource="{Binding Items}"
>
<sfListView:SfListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid x:Name="grid" RowSpacing="1">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="50" />
<ColumnDefinition Width="200" />
<ColumnDefinition Width="50" />
</Grid.ColumnDefinitions>
<Image Source="{Binding ContactImage}"
VerticalOptions="Center"
HorizontalOptions="Center"
HeightRequest="50"/>
<Label Grid.Column="1"
HorizontalTextAlignment="Center"
LineBreakMode="NoWrap"
Text="{Binding ContactName}"
FontSize="Medium" />
<Image Grid.Column="2"
```

```

Source="{Binding ContactType}"
VerticalOptions="End"
HorizontalOptions="End"
HeightRequest="50"/>
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</sfListView:SfListView.ItemTemplate>
</sfListView:SfListView>
</DataTemplate>
</sfPopup:PopupView.ContentTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout>
<Button Text="Click to show popup" Clicked="isOpenButton_Clicked"/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>

```

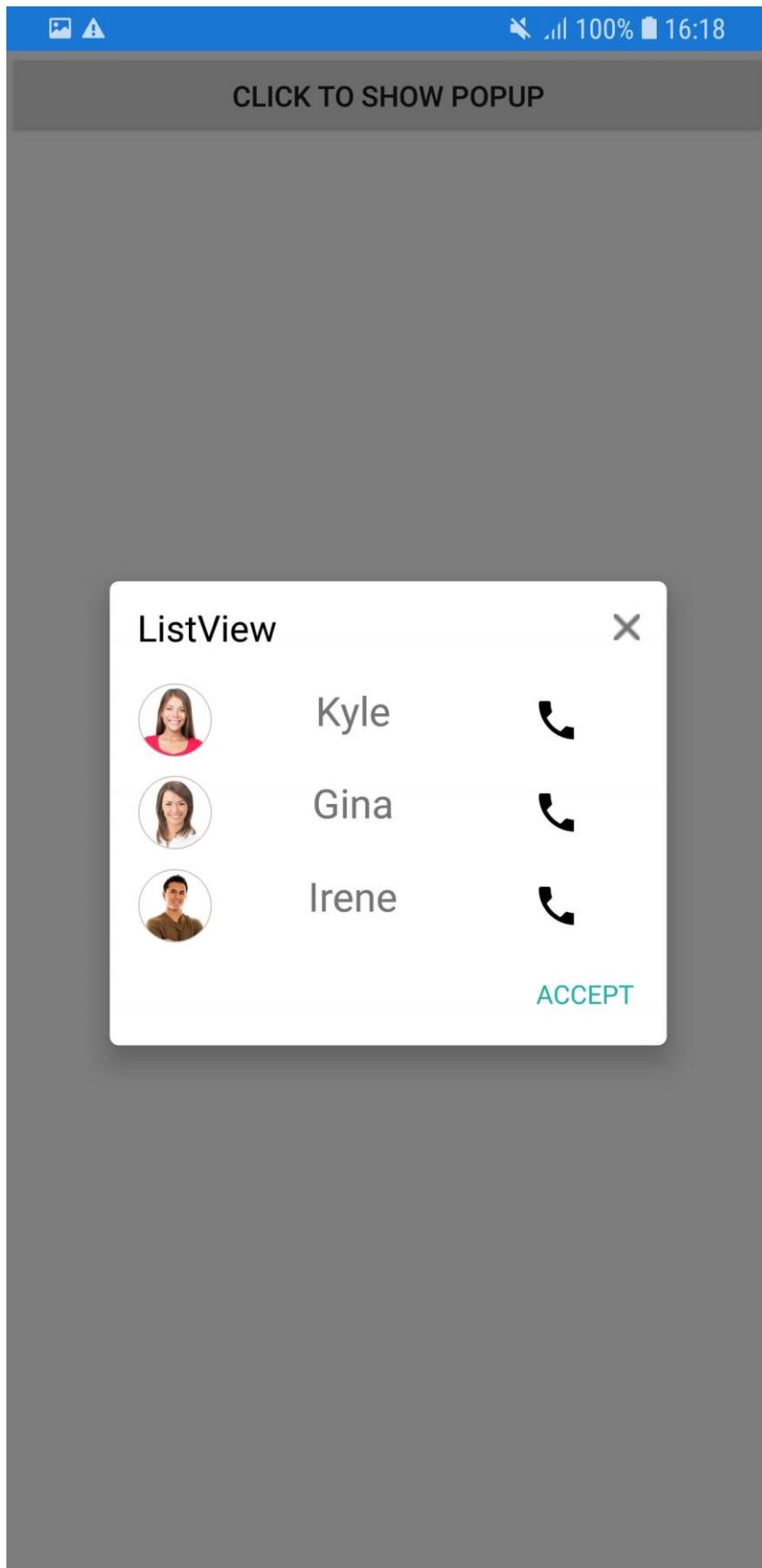
**C#**

```

using Syncfusion.ListView.XForms;
using Syncfusion.XForms.PopupLayout;
namespace PopupDemo
{
    public partial class MainPage : ContentPage
    {
        SfListView listView;
        ContactsViewModel viewModel;
        SfPopupLayout popupLayout;
        public MainPage()
        {
            InitializeComponent();
            listView = new SfListView() { ItemSpacing = 5 };
            listView.WidthRequest = 350;
            listView.ItemTemplate = new DataTemplate(() =>
            {
                ViewCell viewCell = new ViewCell();
                var grid = new Grid() { RowSpacing = 1 };
                grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = 50 });
                grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = 200 });
                grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = 50 });
                var contactImage = new Image()
                {
                    VerticalOptions = LayoutOptions.Center,
                    HorizontalOptions = LayoutOptions.Center,
                    HeightRequest = 50
                };
                contactImage.SetBinding(Image.SourceProperty, new Binding("ContactImage"));
                var contactName = new Label()
                {
                    HorizontalTextAlignment = TextAlignment.Center,
                    LineBreakMode = LineBreakMode.NoWrap,

```

```
FontSize = Font.SystemFontOfSize(NamedSize.Medium).FontSize,
};
contactName.SetBinding(Label.TextProperty, new Binding("ContactName"));
var contactType = new Image()
{
    VerticalOptions = LayoutOptions.End,
    HorizontalOptions = LayoutOptions.End,
    HeightRequest = 50,
};
contactType.SetBinding(Image.SourceProperty, new Binding("ContactType"));
grid.Children.Add(contactImage, 0, 0);
grid.Children.Add(contactName, 1, 0);
grid.Children.Add(contactType, 2, 0);
viewCell.View = grid;
return viewCell;
});
viewModel = new ContactsViewModel();
listView.ItemsSource = viewModel.Items;
popupLayout = new SfPopupLayout();
popupLayout.PopupView.HeaderTitle = "ListView";
popupLayout.PopupView.HeightRequest = 350;
popupLayout.PopupView.ContentTemplate = new DataTemplate(() =>
{
    return listView;
});
StackLayout stackLayout = new StackLayout();
Button isOpenButton = new Button();
isOpenButton.Clicked += isOpenButton_Clicked;
isOpenButton.Text = "Click to show popup";
stackLayout.Children.Add(isOpenButton);
popupLayout.Content = stackLayout;
this.Content = popupLayout;
}
private void isOpenButton_Clicked(object sender, EventArgs e)
{
    popupLayout.Show();
}
}
```



## Full Screen

SfPopupLayout can be shown in full width and height of the screen using,

- [SfPopupLayout.PopupView.IsFullScreen](#)
- [SfPopupLayout.Show\(bool\)](#)

Refer the below code example to open the popup in full screen.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:textinput="clr-
namespace:Syncfusion.XForms.TextInputLayout;assembly=Syncfusion.Core.XForms"
xmlns:sfPopup="clr-
namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XF
orms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<popuplayout:SfPopupLayout.PopupView>
<popuplayout:PopupView AppearanceMode="TwoButton"
AcceptButtonText="SAVE"
DeclineButtonText="CANCEL">
<popuplayout:PopupView.HeaderTemplate>
<DataTemplate>
<Label Text="ADD EVENT" VerticalTextAlignment="Center"
HorizontalTextAlignment="Start" FontAttributes="Bold"/>
</DataTemplate>
</popuplayout:PopupView.HeaderTemplate>
<popuplayout:PopupView.ContentTemplate>
<DataTemplate>
<Grid BackgroundColor="White" Padding="15,20,15,0">
<Grid.RowDefinitions>
<RowDefinition Height="100"/>
<RowDefinition Height="100"/>
<RowDefinition Height="30"/>
<RowDefinition Height="50"/>
<RowDefinition Height="30"/>
<RowDefinition Height="50"/>
<RowDefinition>
<RowDefinition.Height>
<OnPlatform x:TypeArguments="GridLength" Android="55" iOS="55">
<On Platform="UWP" Value="75"/>
</OnPlatform>
</RowDefinition.Height>
</RowDefinition>
</Grid.RowDefinitions>
<Grid Grid.Row="0" BackgroundColor="#F3F3F9" Margin="0,15,0,0">
<textinput:SfTextInputLayout Hint="Event name" ContainerType="Outlined"
BackgroundColor="Transparent">
<Entry HeightRequest="75" BackgroundColor="Transparent"/>
</textinput:SfTextInputLayout>
</Grid>
```

```

<Grid Grid.Row="1" BackgroundColor="#F3F3F9" Margin="0,15,0,0">
<textinput:SfTextInputLayout Hint="Location" ContainerType="Outlined"
BackgroundColor="Transparent">
<Entry HeightRequest="75" BackgroundColor="Transparent"/>
</textInput:SfTextInputLayout>
</Grid>
<Label Grid.Row="2" Text="From" Margin="0,10,0,0"/>
<Grid Grid.Row="3">
<Grid.ColumnDefinitions>
<ColumnDefinition/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<DatePicker Grid.Column="0" FontSize="Small"/>
<TimePicker Grid.Column="1" FontSize="Small"/>
</Grid>
<Label Grid.Row="4" Text="To" Margin="0,10,0,0"/>
<Grid Grid.Row="5">
<Grid.ColumnDefinitions>
<ColumnDefinition/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<DatePicker Grid.Column="0" FontSize="Small"/>
<TimePicker Grid.Column="1" FontSize="Small"/>
</Grid>
<Grid Grid.Row="6" Margin="0,35,0,0">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<Switch Grid.Column="0"/>
<Label Grid.Column="1" Text="All-day">
<Label.Margin>
<OnPlatform x:TypeArguments="Thickness">
<On Platform="UWP" Value="0,10,0,0"/>
</OnPlatform>
</Label.Margin>
</Label>
</Grid>
</Grid>
</DataTemplate>
</popuplayout:PopupView.ContentTemplate>
</popuplayout:PopupView>
</popuplayout:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked" />
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>

```

**C#**

```
using Syncfusion.XForms.PopupLayout;
```



```
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            // Shows SfPopupLayout in full width and height of the screen using the
            // SfPopupLayout.PopupView.IsFullScreen property.
            popupLayout.PopupView.IsFullScreen = true;
            popupLayout.IsOpen = true;
            // Shows SfPopupLayout in full width and height of the screen using the
            // SfPopupLayout.Show(bool) method.
            // popupLayout.Show(true);
        }
    }
}
```

Executing the above codes renders the following output in an Android device.

ADD EVENT

Event name

Location

From

21-03-2019 00:00

To

21-03-2019 00:00

☐ All-day

CANCEL SAVE

### Auto-size Popup

The SfPopupLayout can auto-size the popup view based on the contents loaded inside its SfPopupLayout.PopupView.ContentTemplate property using the SfPopupLayout.PopupView.AutoSizeMode property. The default value is AutoSizeMode.None. You can choose to auto-size the Popup in height, width, or in both height and width of its contents. By default, the HeightRequest and WidthRequest set to the SfPopupLayout.PopupView or the views loaded inside the template is given higher priority than the AutoSizeMode.

In the following code snippet the Popup is auto-sized in height based on the content loaded inside the SfPopupLayout.PopupView.ContentTemplate property.

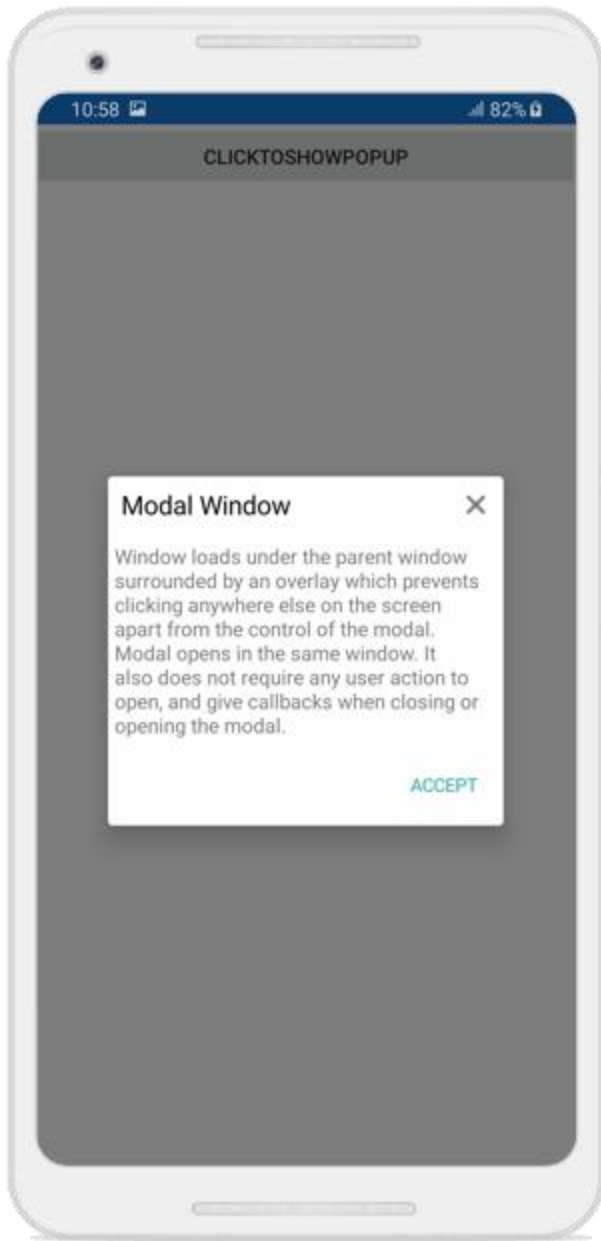
#### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">
  <sfPopup:SfPopupLayout.PopupView>
    <sfPopup:PopupView AutoSizeMode="Height">
      <sfPopup:PopupView.ContentTemplate>
        <DataTemplate>
          <StackLayout Padding="5,5,5,0">
            <Label Text="Window loads under the parent window surrounded by an overlay
which prevents clicking anywhere else on the screen apart from the control
of the modal. Modal opens in the same window. It also does not require any
user action to open, and give callbacks when closing or opening the modal."
BackgroundColor="White"
LineBreakMode="WordWrap"
/>
          </StackLayout>
        </DataTemplate>
      </sfPopup:PopupView.ContentTemplate>
    </sfPopup:PopupView>
  </sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

#### C#

```
popUpLayout.PopupView.AutoSizeMode = AutoSizeMode.Height;
```

Executing the above codes renders the following output in Android devices.



You can download the above sample code by clicking the following link: [Sample](#).

### Layout Customizations

The SfPopupLayout supports two types of [SfPopupLayout.PopupView.AppearanceMode](#). By default, the `AppearanceMode.OneButton` is set. You can change the appearance by using the `SfPopupLayout.PopupView.AppearanceMode` property.

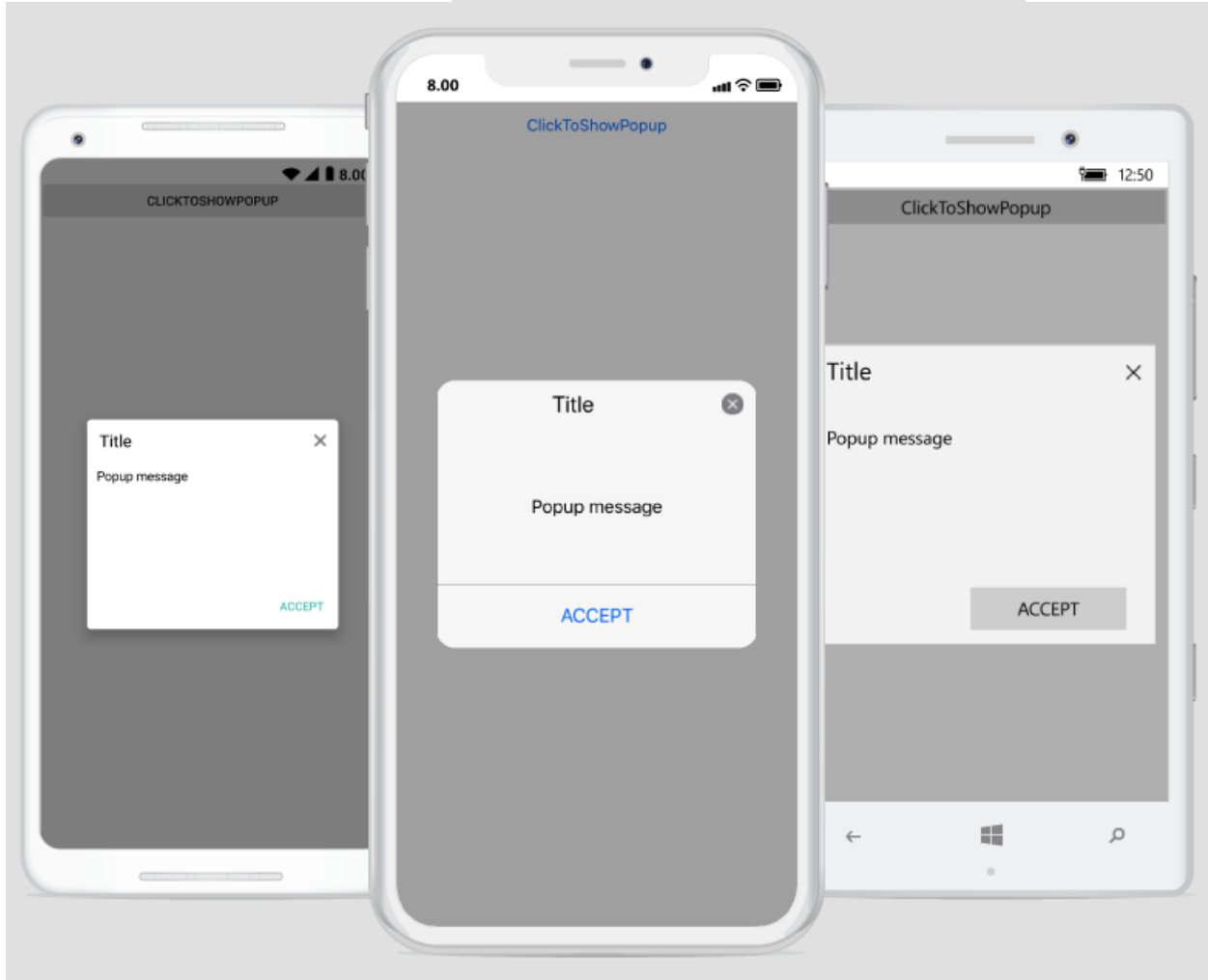
The two different appearance modes in the SfPopupLayout are as follows:

Modes	Description
<a href="#">OneButton</a>	Shows the SfPopupLayout with one button in the footer view. This is the default value.

<a href="#">TwoButton</a>	Shows the SfPopupLayout with two buttons in the footer view.
---------------------------	--

### Popup with one button in the footer

In the following code example, set the `SfPopupLayout.PopupView.AppearanceMode` property as



`OneButton` which displays only the Accept button in the footer view.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView AppearanceMode="OneButton" />
</sfPopup:SfPopupLayout>
</ContentPage>
```

```
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand" Clicked +=
ClickToShowPopup_Clicked/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

## C#

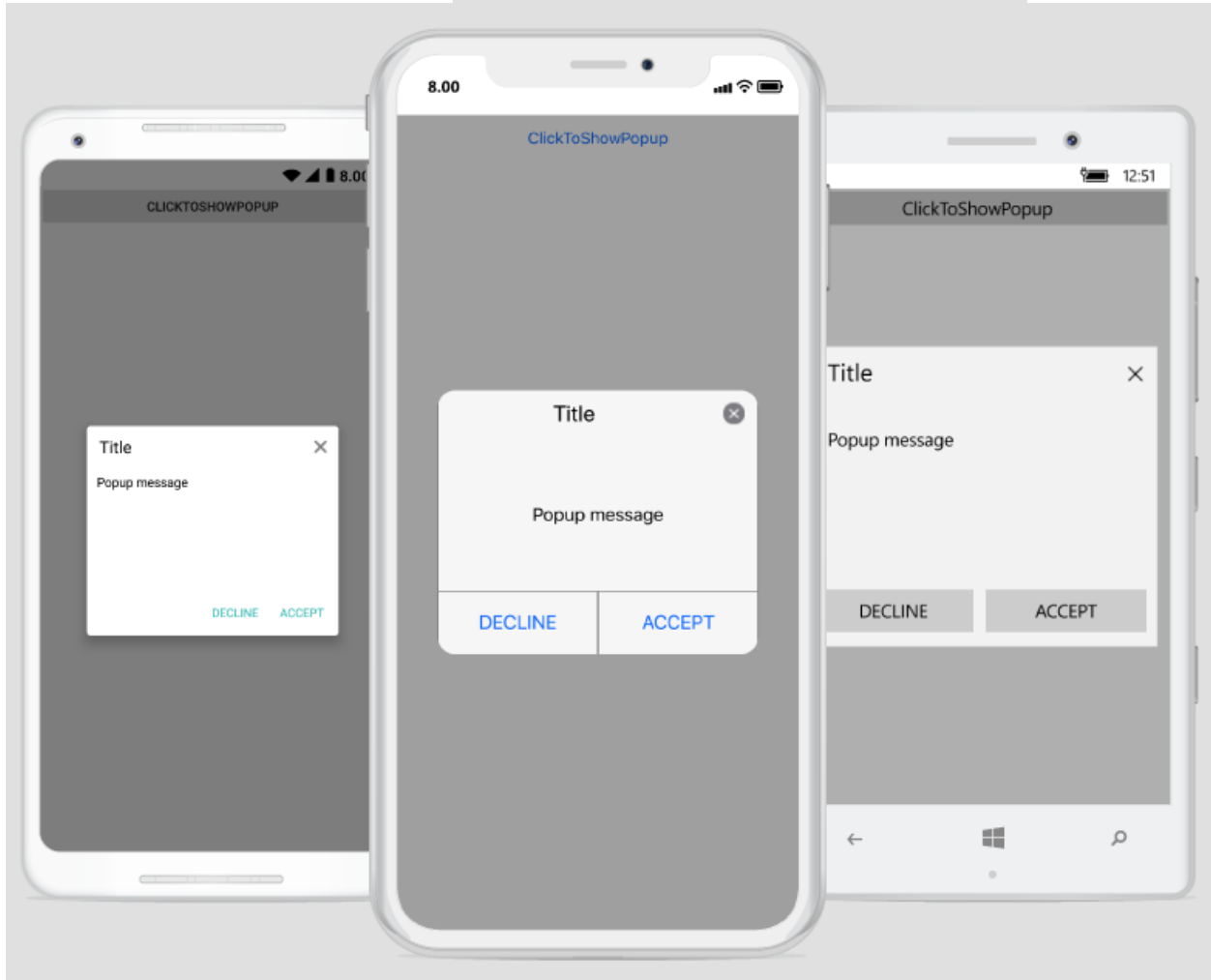
```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            // Setting the AppearanceMode as OneButton
            popupLayout.PopupView.AppearanceMode = AppearanceMode.OneButton;
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.

![Popup with one button](GettingStartedimages/AppearanceModeOneButton.png)

### Popup with two buttons in the footer

In the following code example, set the `SfPopupLayout.PopupView.AppearanceMode` property as



`TwoButton` which displays both Accept button and Decline button in the footer view.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:GettingStarted"
  x:Class="GettingStarted.MainPage"
  Padding="0,40,0,0"
  xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
  <sfPopup:SfPopupLayout x:Name="popupLayout">
    <sfPopup:SfPopupLayout.PopupView>
      <sfPopup:PopupView AppearanceMode="TwoButton" />
    </sfPopup:SfPopupLayout.PopupView>
    <sfPopup:SfPopupLayout.Content>
      <StackLayout x:Name="layout">
```

```
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked" />
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

**C#**

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            // Setting the AppearanceMode as TwoButton
            popupLayout.PopupView.AppearanceMode = AppearanceMode.TwoButton;
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.

![Popup with two button](GettingStartedimages/AppearanceModeTwoButton.png)

Popup without header

You can display the Popup without header by using the property, [SfPopupLayout.PopupView.ShowHeader](#), find the code example of the same below.

**XML**

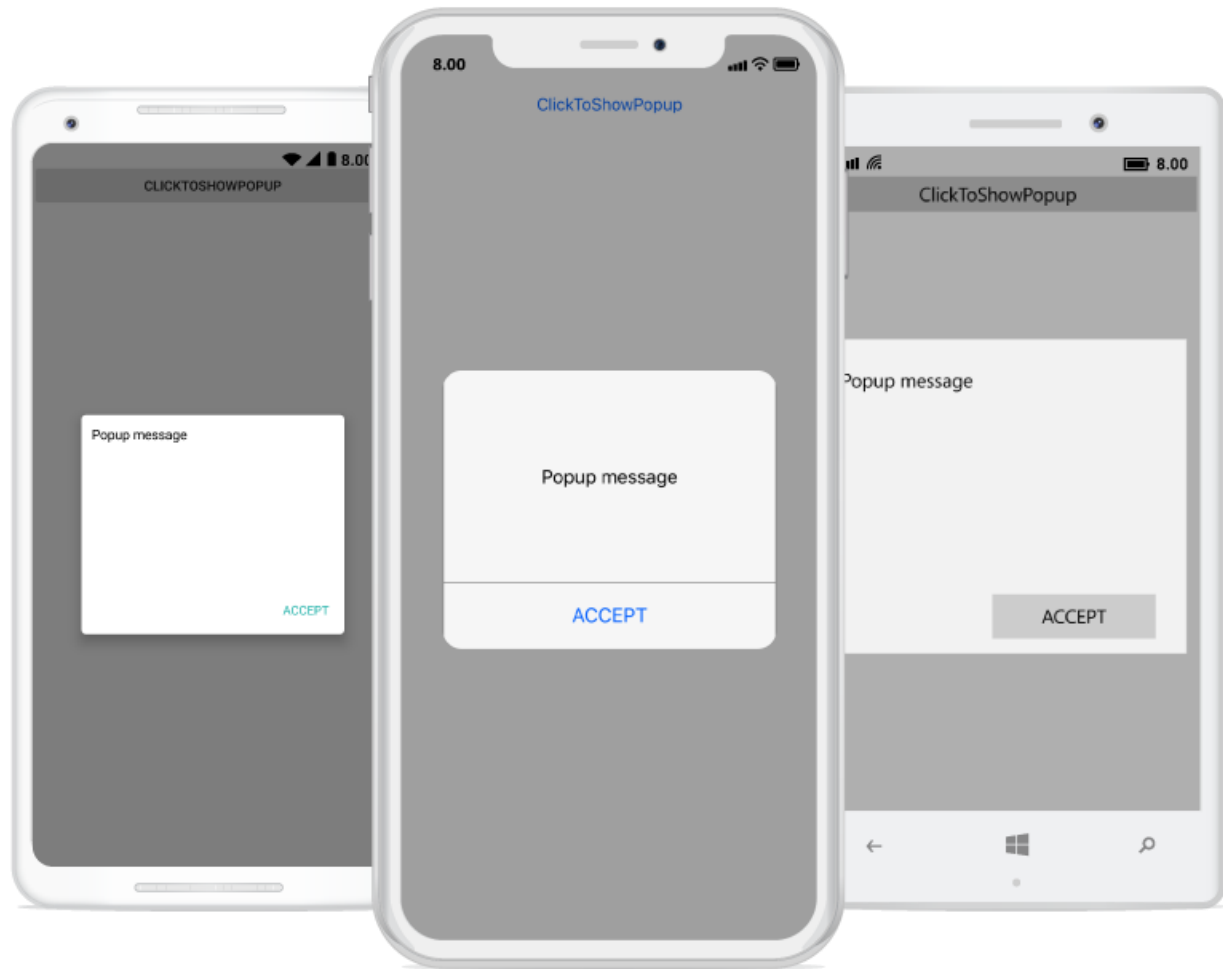
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
    <sfPopup:SfPopupLayout x:Name="popupLayout">
        <sfPopup:SfPopupLayout.PopupView>
            <sfPopup:PopupView ShowHeader="False"/>
        </sfPopup:SfPopupLayout.PopupView>
        <sfPopup:SfPopupLayout.Content>
            <StackLayout x:Name="layout">
```



```
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand" Clicked +=
ClickToShowPopup_Clicked/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

## C#

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            popupLayout.PopupView.ShowHeader = false;
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}
```



### Popup without Footer

You can display the Popup without footer by using the property, [SfPopupLayout.PopupView.ShowFooter](#), find the code example of the same below.

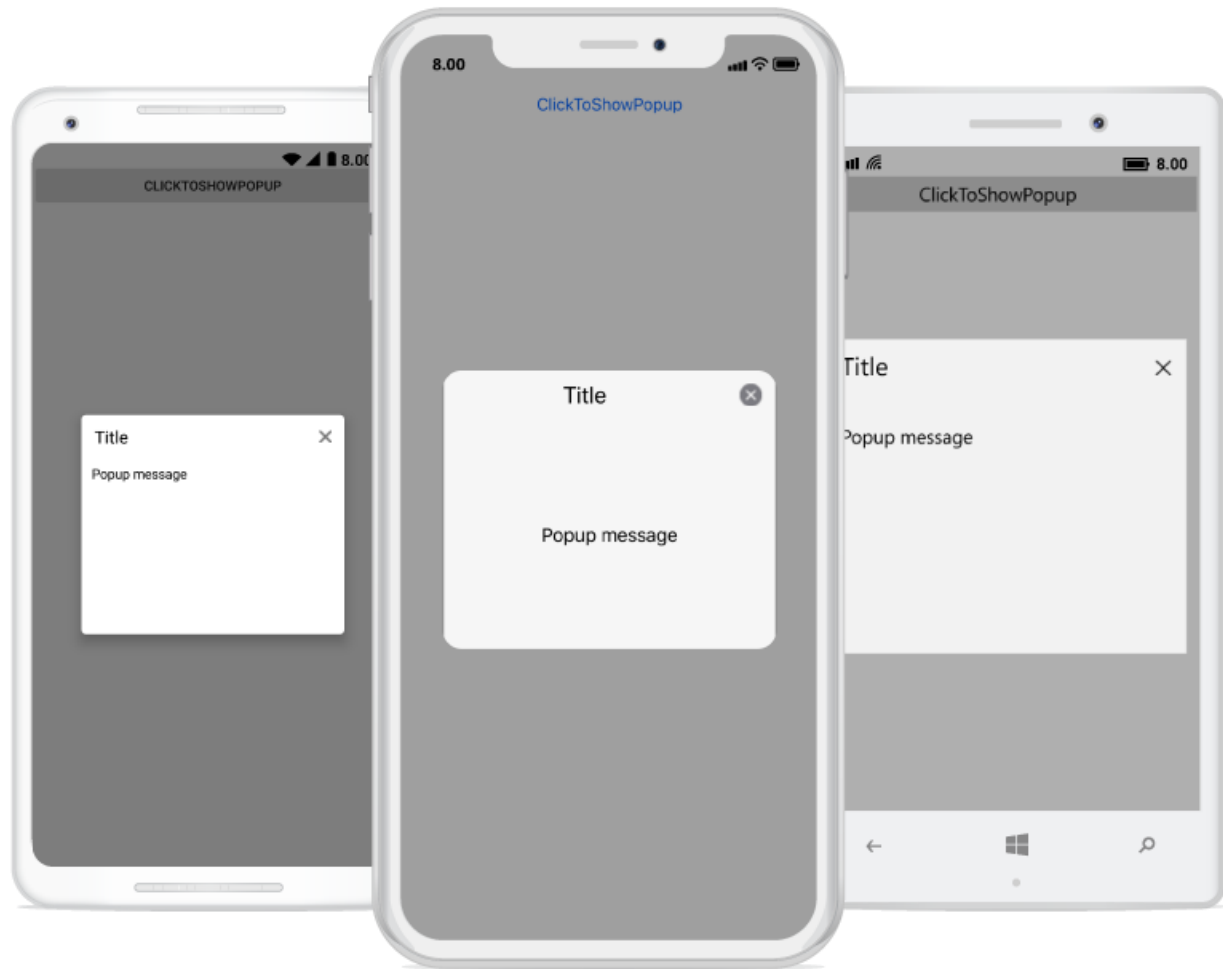
#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
  <sfPopup:SfPopupLayout x:Name="popupLayout">
    <sfPopup:SfPopupLayout.PopupView>
      <sfPopup:PopupView ShowFooter="False"/>
    </sfPopup:SfPopupLayout.PopupView>
    <sfPopup:SfPopupLayout.Content>
      <StackLayout x:Name="layout">
```

```
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand" Clicked +=
ClickToShowPopup_Clicked/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

## C#

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            popupLayout.PopupView.ShowFooter = false;
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}
```



### Popup without header and footer

You can display only the content of the SfPopupLayout, by removing the header and footer by using [SfPopupLayout.PopupView.ShowHeader](#) and [SfPopupLayout.PopupView.ShowFooter](#), code example of the same below.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView ShowFooter="False" ShowHeader="False"/>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
```

```
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand" Clicked +=
ClickToShowPopup_Clicked/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

**C#**

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            popupLayout.PopupView.ShowFooter = false;
            popupLayout.PopupView.ShowHeader = false;
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}
```

## Customizing popup header

Any view can be added as the header content using the [SfPopupLayout.PopupView.HeaderTemplate](#) property. Refer to the following code example in which a label is added as a header content.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
    <sfPopup:SfPopupLayout x:Name="popupLayout">
        <sfPopup:SfPopupLayout.PopupView>
            <sfPopup:PopupView ShowCloseButton="False">
                <sfPopup:PopupView.HeaderTemplate>
                    <DataTemplate>
                        <Label Text="Customized Header"
FontAttributes="Bold"
BackgroundColor="Blue"
FontSize="16"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"
/>
                    </DataTemplate>
                </sfPopup:PopupView>
            </sfPopup:SfPopupLayout>
        </sfPopup:SfPopupLayout>
    </ContentPage>
```

```

</DataTemplate>
</sfPopup:PopupView.HeaderTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand" />
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>

```

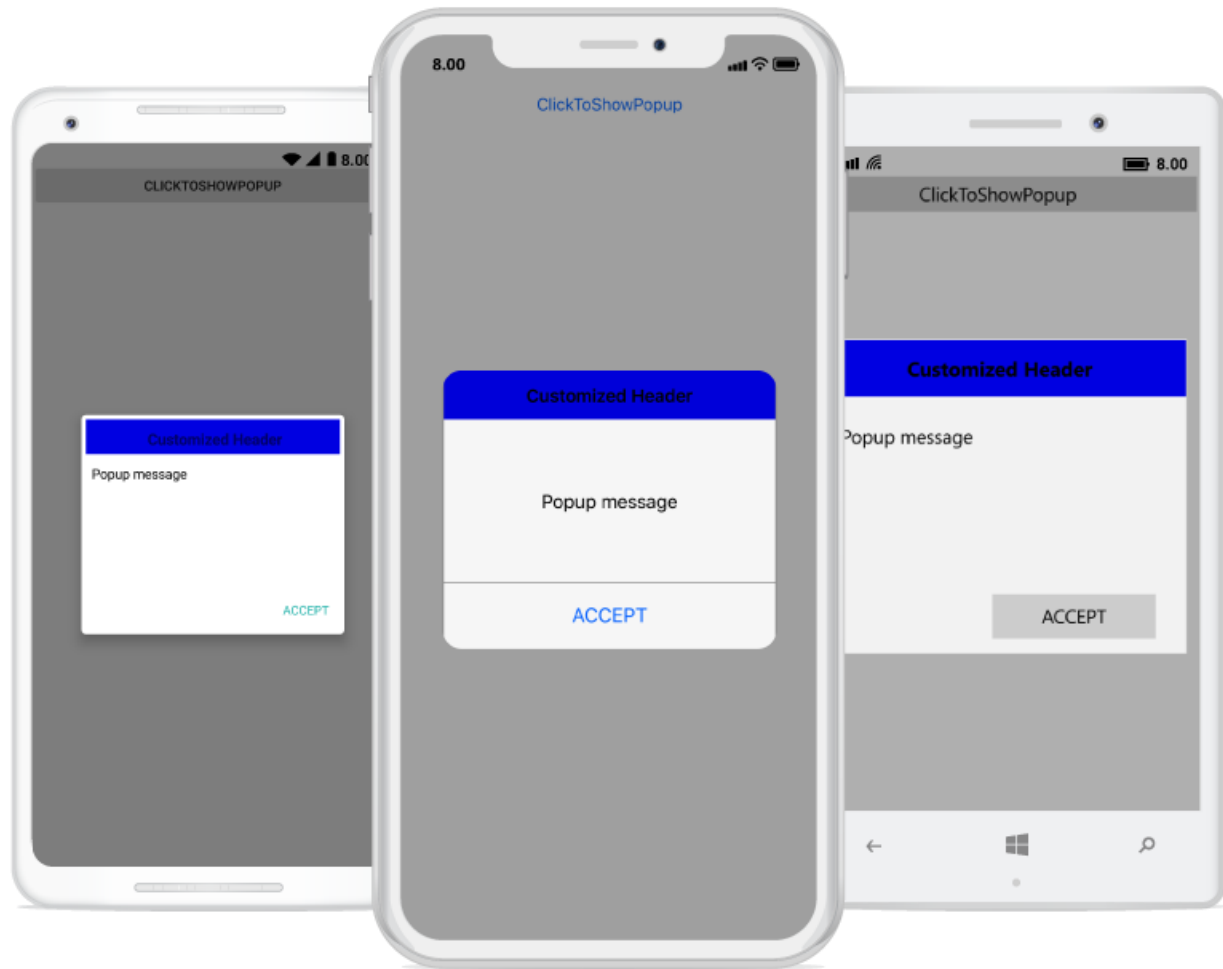
## C#

```

using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        DataTemplate headerTemplateView;
        Label headerContent;
        public MainPage()
        {
            InitializeComponent();
            clickToShowPopup.Clicked += ClickToShowPopup_Clicked;
            headerTemplateView = new DataTemplate(() =>
            {
                headerContent = new Label();
                headerContent.Text = "Customized Header";
                headerContent.FontAttributes = FontAttributes.Bold;
                headerContent.BackgroundColor = Color.FromRgb(0, 0, 225);
                headerContent.FontSize = 16;
                headerContent.HorizontalTextAlignment = TextAlignment.Center;
                headerContent.VerticalTextAlignment = TextAlignment.Center;
                return headerContent;
            });
            popupLayout.PopupView.ShowCloseButton = false;
            // Adding HeaderTemplate of the SfPopupLayout
            popupLayout.PopupView.HeaderTemplate = headerTemplateView;
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}

```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



### Customizing popup footer

Any view can be added as the footer content using the [SfPopupLayout.PopupView.FooterTemplate](#) property. Refer to the following code example in which a label is added as a footer content.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
  <sfPopup:SfPopupLayout x:Name="popupLayout">
    <sfPopup:SfPopupLayout.PopupView>
      <sfPopup:PopupView>
        <sfPopup:PopupView.FooterTemplate>
          <DataTemplate>
            <Label Text="Customized Footer"
FontAttributes="Bold"
BackgroundColor="Blue"/>
          </DataTemplate>
        </sfPopup:PopupView.FooterTemplate>
      </sfPopup:PopupView>
    </sfPopup:SfPopupLayout.PopupView>
  </sfPopup:SfPopupLayout>
</ContentPage>
```

```

FontSize="16"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"
/>
</DataTemplate>
</sfPopup:PopupView.FooterTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand" />
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>

```

**C#**

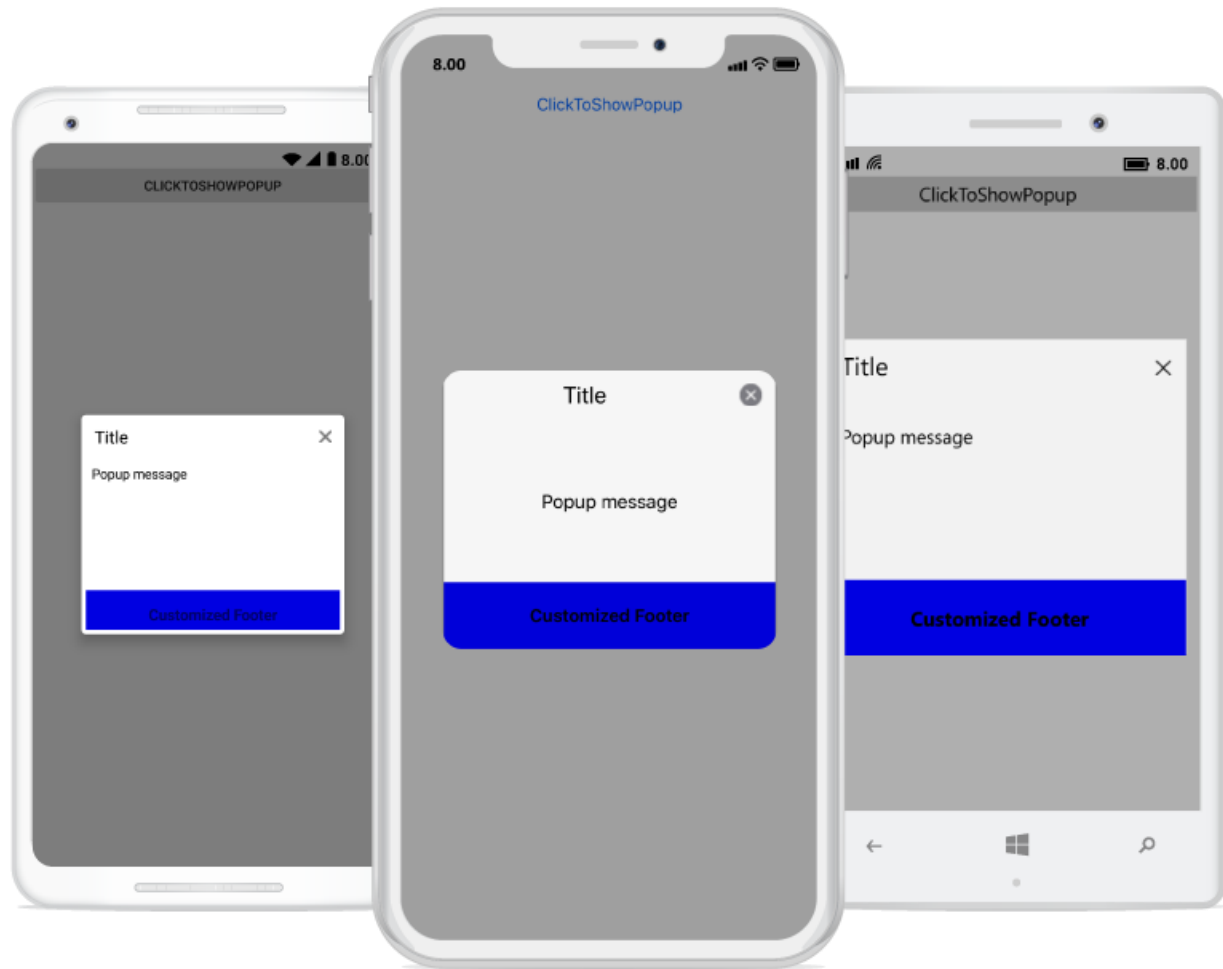
```

using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
public partial class MainPage : ContentPage
{
DataTemplate footerTemplateView;
Label footerContent;
public MainPage()
{
InitializeComponent();
clickToShowPopup.Clicked += ClickToShowPopup_Clicked;
footerTemplateView = new DataTemplate(() =>
{
footerContent = new Label();
footerContent.Text = "Customized Footer";
footerContent.FontAttributes = FontAttributes.Bold;
footerContent.BackgroundColor = Color.FromRgb(0, 0, 225);
footerContent.FontSize = 16;
footerContent.HorizontalTextAlignment = TextAlignment.Center;
footerContent.VerticalTextAlignment = TextAlignment.Center;
return footerContent;
});
// Adding FooterTemplate of the SfPopupLayout
popupLayout.PopupView.FooterTemplate = footerTemplateView;
}
private void ClickToShowPopup_Clicked(object sender, EventArgs e)
{
popupLayout.Show();
}
}
}

```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.





### Customizing popup content

Any view can be added as popup content by using the [SfPopupLayout.PopupView.ContentTemplate](#) property. Refer to the following code example in which a label is added as a popup content.

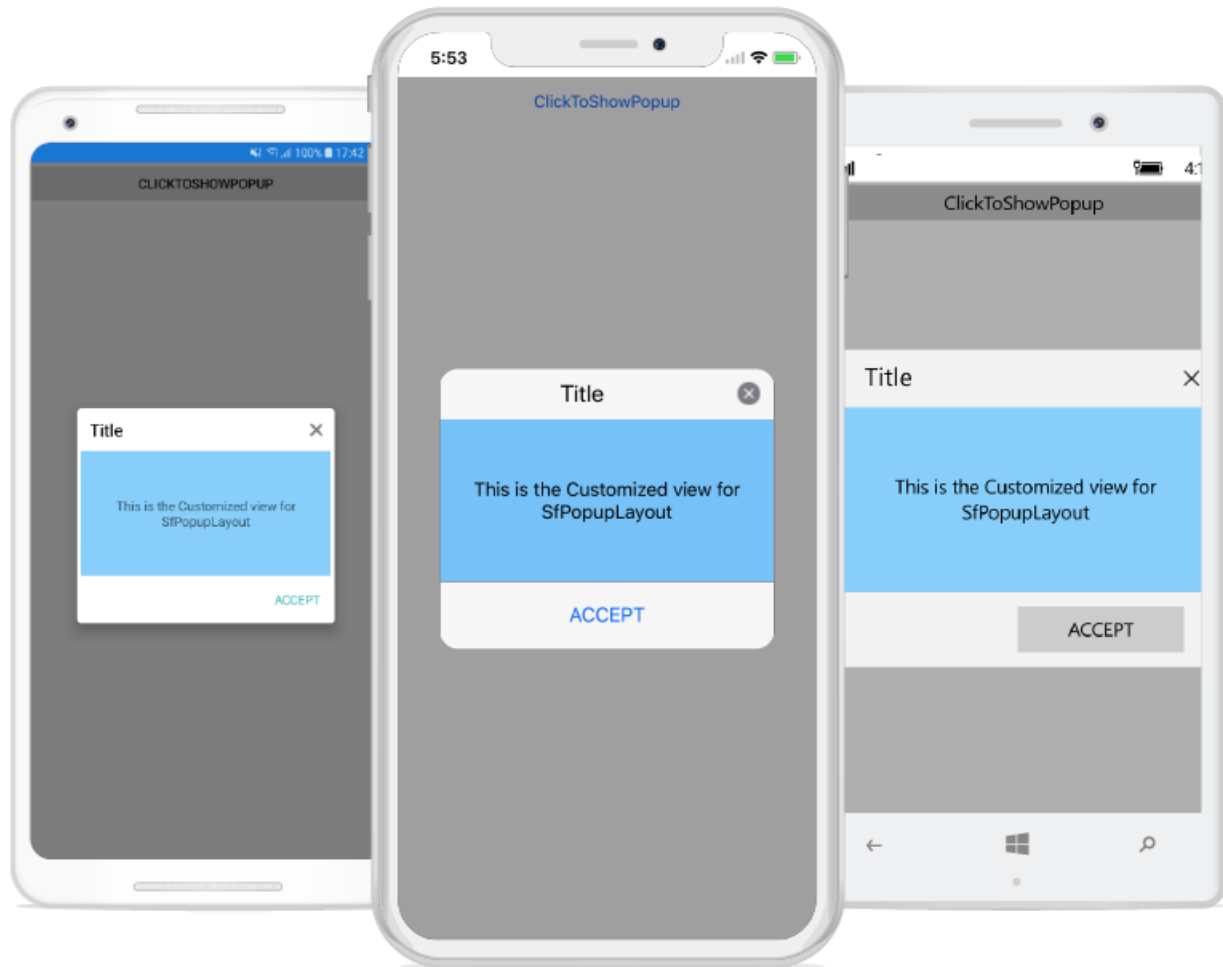
#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
  <sfPopup:SfPopupLayout x:Name="popupLayout">
    <sfPopup:SfPopupLayout.PopupView>
      <sfPopup:PopupView>
        <sfPopup:PopupView.ContentTemplate>
          <DataTemplate>
            <Label Text="This is SfPopupLayout" BackgroundColor="SkyBlue"
HorizontalTextAlignment="Center"/>
          </DataTemplate>
        </sfPopup:PopupView>
      </sfPopup:SfPopupLayout.PopupView>
    </sfPopup:SfPopupLayout>
  </ContentPage>
```

```
</sfPopup:PopupView.ContentTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="layout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start"
HorizontalOptions="FillAndExpand" Clicked="ClickToShowPopup_Clicked" />
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

## C#

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        DataTemplate templateView;
        Label popupContent;
        public MainPage()
        {
            InitializeComponent();
            templateView = new DataTemplate(() =>
            {
                popupContent = new Label();
                popupContent.Text = "This is the SfPopupLayout";
                popupContent.BackgroundColor = Color.LightSkyBlue;
                popupContent.HorizontalTextAlignment = TextAlignment.Center;
                return popupContent;
            });
            // Adding ContentTemplate of the SfPopupLayout
            popupLayout.PopupView.ContentTemplate = templateView;
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}
```



To display ListView as content of the Popup, refer to this [documentation](#).

### Styles

The SfPopupLayout applies style to all of its elements by using the [SfPopupLayout.PopupView.PopupStyle](#) property.

#### Styling popup header

The SfPopupLayout allows customizing the header elements with various header customizations as follows:

Property	Description
<a href="#">SfPopupLayout.PopupView.PopupStyle.HeaderBackgroundColor</a>	Gets or sets the background color for the header.
<a href="#">SfPopupLayout.PopupView.PopupStyle.HeaderFontAttribute</a>	Gets or sets the font attribute for the header title.
<a href="#">SfPopupLayout.PopupView.PopupStyle.HeaderFontFamily</a>	Gets or sets the font style for the header title.

<a href="#">SfPopupLayout.PopupView.PopupStyle.HeaderFontSize</a>	Gets or sets the font size for the header title.
<a href="#">SfPopupLayout.PopupView.PopupStyle.HeaderTextAlignment</a>	Gets or sets the text alignment for the header.
<a href="#">SfPopupLayout.PopupView.PopupStyle.HeaderTextColor</a>	Gets or sets the text color to be applied for the header title.

Refer to the following code example for customizing the header elements.

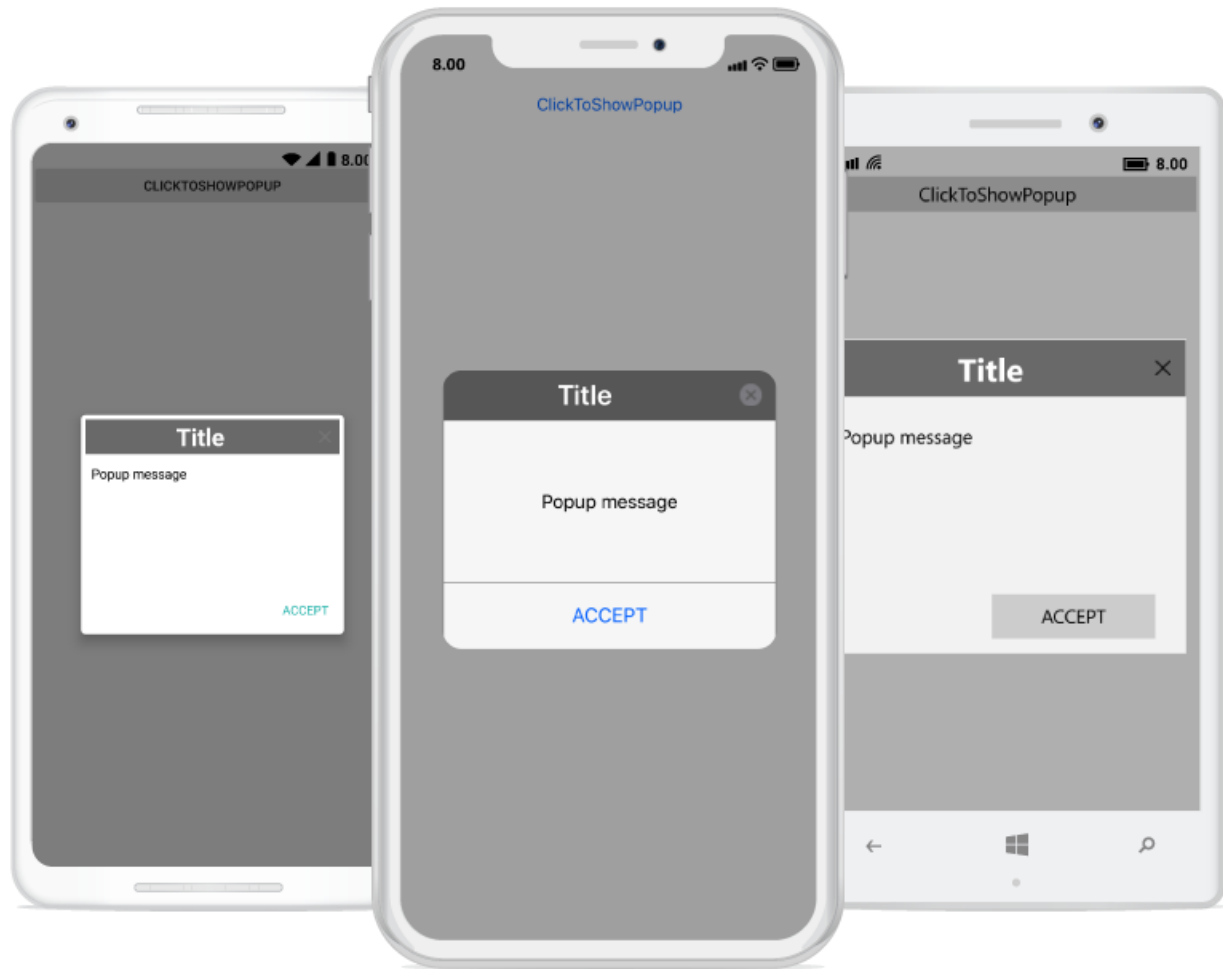
#### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">
  <sfPopup:SfPopupLayout.PopupView>
    <sfPopup:PopupView>
      <sfPopup:PopupView.PopupStyle>
        <sfPopup:PopupStyle HeaderBackgroundColor="DimGray"
HeaderFontAttribute="Bold"
HeaderFontFamily="Helvetica-Bold"
HeaderFontSize="25"
HeaderTextAlignment="Center"
HeaderTextColor="White"/>
      </sfPopup:PopupView.PopupStyle>
    </sfPopup:PopupView>
  </sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

#### C#

```
//MainPage.cs
public MainPage()
{
    ....
    InitializeComponent();
    popupLayout.PopupView.PopupStyle.HeaderBackgroundColor =
    Color.FromRgb(105, 105, 105);
    popupLayout.PopupView.PopupStyle.HeaderFontAttribute = FontAttributes.Bold;
    popupLayout.PopupView.PopupStyle.HeaderFontFamily = "Helvetica-Bold";
    popupLayout.PopupView.PopupStyle.HeaderFontSize = 25;
    popupLayout.PopupView.PopupStyle.HeaderTextAlignment = TextAlignment.Center;
    popupLayout.PopupView.PopupStyle.HeaderTextColor = Color.White;
    ....
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



### Styling popup footer

The SfPopupLayout allows customizing the footer elements with various footer customizations as follows:

Property	Description
<a href="#">SfPopupLayout.PopupView.PopupStyle.FooterBackgroundColor</a>	Gets or sets the background color for the footer.
<a href="#">SfPopupLayout.PopupView.PopupStyle.AcceptButtonBackgroundColor</a>	Gets or sets the background color for the Accept button in the footer.
<a href="#">SfPopupLayout.PopupView.PopupStyle.AcceptButtonTextColor</a>	Gets or sets the foreground color for the Accept button in the footer.
<a href="#">SfPopupLayout.PopupView.PopupStyle.DeclineButtonBackgroundColor</a>	Gets or sets the background color for the Decline button in the footer.

<a href="#">SfPopupLayout.PopupView.PopupStyle.DeclineButtonTextColor</a>	Gets or sets the foreground color for the Decline button in the footer.
---	---

Refer to the following code example for customizing the footer elements.

#### **XML**

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">
  <sfPopup:SfPopupLayout.PopupView>
    <sfPopup:PopupView AppearanceMode="TwoButton">
      <sfPopup:PopupView.PopupStyle>
        <sfPopup:PopupStyle FooterBackgroundColor="LightGray"
          AcceptButtonBackgroundColor="DimGray"
          AcceptButtonTextColor="White"
          DeclineButtonBackgroundColor="DimGray"
          DeclineButtonTextColor="White"
        />
      </sfPopup:PopupView.PopupStyle>
    </sfPopup:PopupView>
  </sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

#### **C#**

```
//MainPage.cs
public MainPage()
{
    ....
    InitializeComponent();
    // Setting the AppearanceMode as TwoButton
    popupLayout.PopupView.AppearanceMode =
    Syncfusion.XForms.PopupLayout.AppearanceMode.TwoButton;
    // Footer customization
    popupLayout.PopupView.PopupStyle.FooterBackgroundColor = Color.LightGray;
    popupLayout.PopupView.PopupStyle.AcceptButtonBackgroundColor =
    Color.FromRgb(105, 105, 105);
    popupLayout.PopupView.PopupStyle.AcceptButtonTextColor = Color.White;
    popupLayout.PopupView.PopupStyle.DeclineButtonBackgroundColor =
    Color.FromRgb(105, 105, 105);
    popupLayout.PopupView.PopupStyle.DeclineButtonTextColor = Color.White;
    ....
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



### Border customization

The SfPopupLayout allows customizing the border appearance with various border customizations as follows:

Property	Description
<a href="#">SfPopupLayout.PopupView.PopupStyle.BorderColor</a>	Gets or sets the border color for the PopupView.
<a href="#">SfPopupLayout.PopupView.PopupStyle.BorderThickness</a>	Gets or sets the border thickness for the PopupView.
<a href="#">SfPopupLayout.PopupView.PopupStyle.CornerRadius</a>	Gets or sets the corner radius for the PopupView.

Refer to the following code example for customizing the border elements.

### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">
  <sfPopup:SfPopupLayout.PopupView>
```

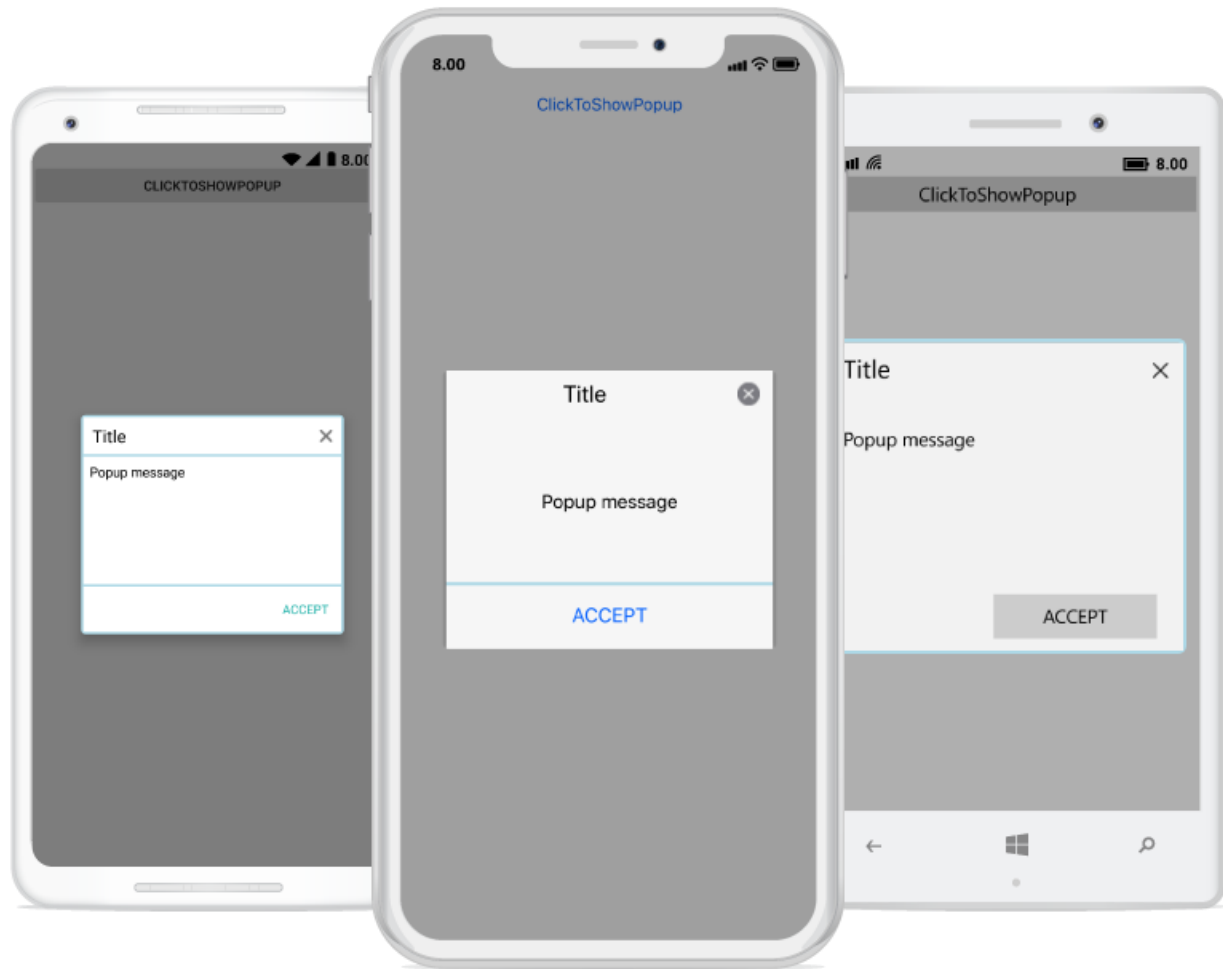
```
<sfPopup:PopupView>
<sfPopup:PopupView.PopupStyle>
<sfPopup:PopupStyle BorderColor="LightBlue"
BorderThickness="3"
CornerRadius="5"
/>
</sfPopup:PopupView.PopupStyle>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

## C#

```
//MainPage.cs
public MainPage()
{
    ....
    InitializeComponent();
    popupLayout.PopupView.PopupStyle.BorderColor = Color.LightBlue;
    popupLayout.PopupView.PopupStyle.BorderThickness = 3;
    popupLayout.PopupView.PopupStyle.CornerRadius = 5;
    ....
}
```

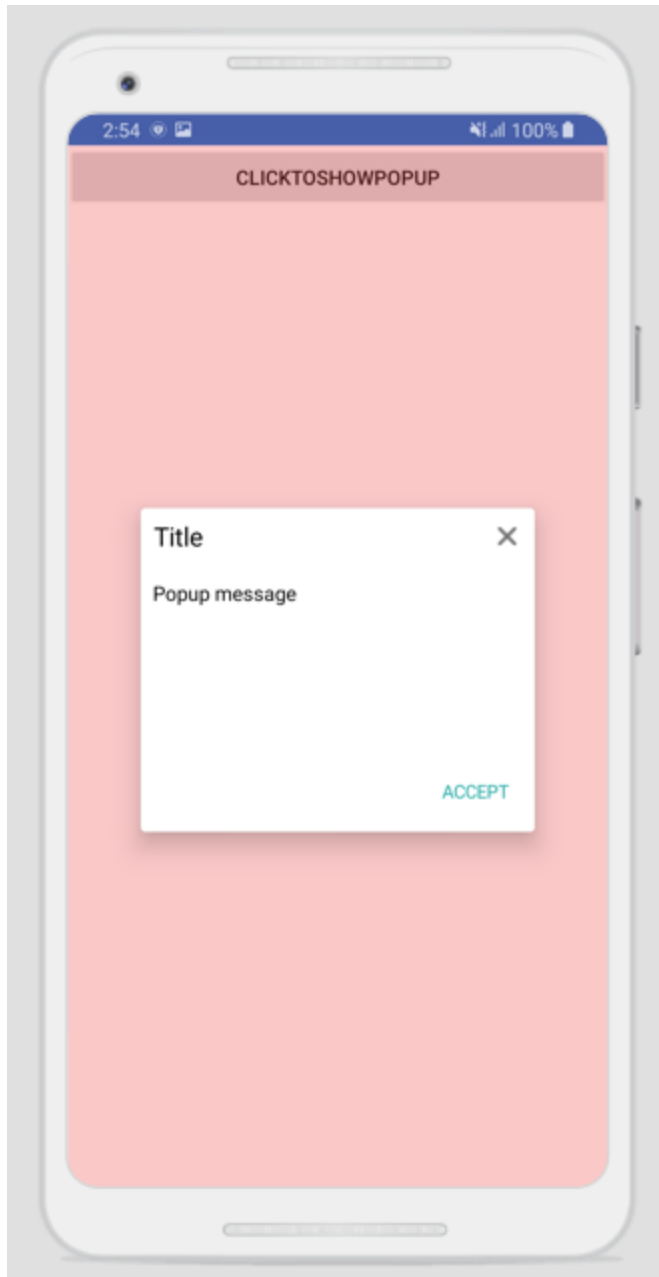
Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.





### Styling overlay background

The SfPopupLayout allows to customize the opacity and the background color of overlay using the SfPopupLayout.PopupView.PopupStyle.OverlayOpacity and



`SfPopupLayout.PopupView.PopupStyle.OverlayColor` properties, respectively.

Refer to the following code example for customizing the overlay opacity and the background color.

#### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">
  <sfPopup:SfPopupLayout.PopupView>
    <sfPopup:PopupView>
      <sfPopup:PopupView.PopupStyle>
        <sfPopup:PopupStyle OverlayColor="Red" OverlayOpacity="0.2">
        </sfPopup:PopupStyle>
      </sfPopup:PopupView.PopupStyle>
    </sfPopup:PopupView>
  </sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

```
</sfPopup:SfPopupLayout>
```

### C#

```
popUpLayout.PopupView.PopupStyle.OverlayColor = Color.Red;
popUpLayout.PopupView.PopupStyle.OverlayOpacity = 0.2;
```

Executing the above codes renders the following output in Android devices.

![Popup with overlay customization](PopupLayoutimages/PopupViewOverlay.png)

### Popup Animations

Built-in animations are available in SfPopupLayout, which is applied when the PopupView opens and closes in the screen.

The SfPopupLayout has different animation modes as listed below:

- [Zoom](#)
- [Fade](#)
- [SlideOnLeft](#)
- [SlideOnRight](#)
- [SlideOnTop](#)
- [SlideOnBottom](#)
- [None](#)

**Note:** Setting of AnimationMode is same for both **Displaying popup when the SfPopupLayout is set as root view** and **Displaying popup on the go**.

### Zoom

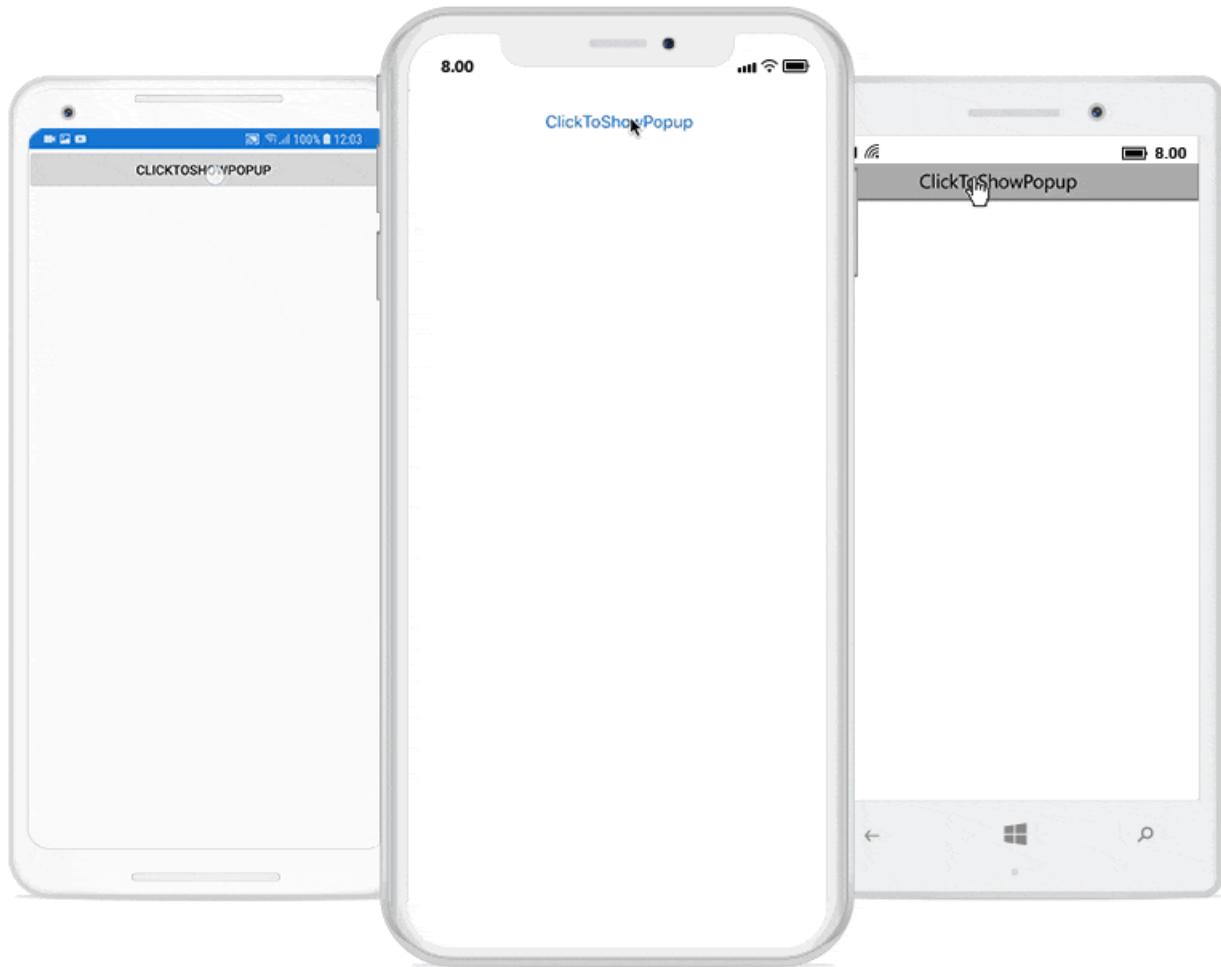
Zoom-out animation will be applied when the PopupView opens and Zoom-in animation will be applied when the PopupView closes.

### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">
  <sfPopup:SfPopupLayout.PopupView>
    <sfPopup:PopupView AnimationMode="Zoom" />
  </sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

### C#

```
//MainPage.cs
public MainPage ()
{
    ....
    InitializeComponent();
    popUpLayout.PopupView.AnimationMode = AnimationMode.Zoom;
    ....
}
```



## Fade

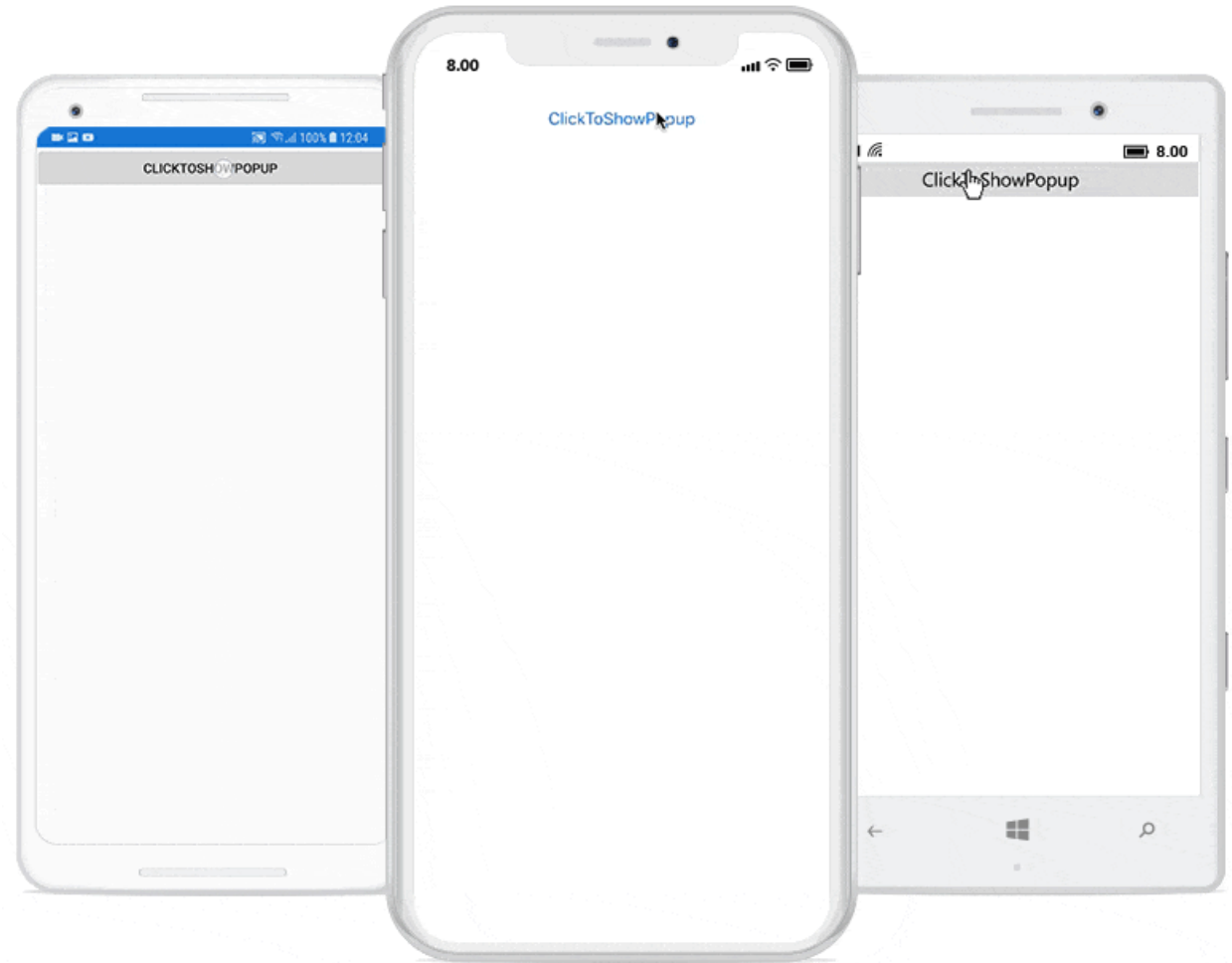
Fade-out animation will be applied when the PopupView opens and Fade-in animation will be applied when the PopupView closes.

### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">
  <sfPopup:SfPopupLayout.PopupView>
    <sfPopup:PopupView AnimationMode="Fade" />
  </sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

### C#

```
//MainPage.cs
public MainPage()
{
    ....
    InitializeComponent();
    popupLayout.PopupView.AnimationMode = AnimationMode.Fade;
    ....
}
```



### SlideOnLeft

PopupView will be animated from left-to-right when it opens and from right-to-left when it closes.

#### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">
  <sfPopup:SfPopupLayout.PopupView>
    <sfPopup:PopupView AnimationMode="SlideOnLeft" />
  </sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

#### C#

```
//MainPage.cs
public MainPage()
{
    ....
    InitializeComponent();
    popupLayout.PopupView.AnimationMode = AnimationMode.SlideOnLeft;
    ....
}
```

```
}
```



### SlideOnRight

PopupView will be animated from right-to-left when it opens and from left-to-right when it closes.

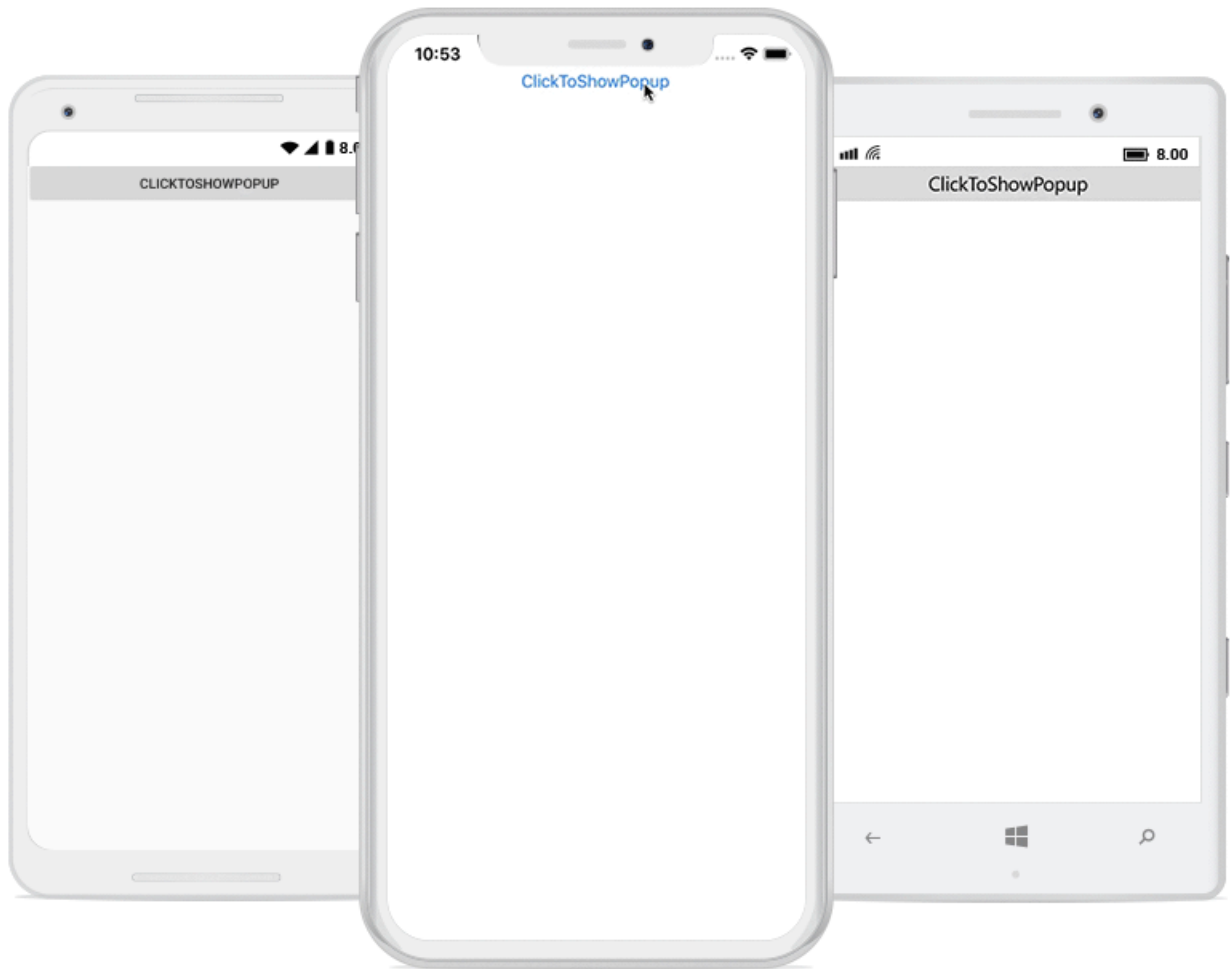
#### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">
  <sfPopup:SfPopupLayout.PopupView>
    <sfPopup:PopupView AnimationMode="SlideOnRight" />
  </sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

#### C#

```
//MainPage.cs
public MainPage()
{
    ....
    InitializeComponent();
    popupLayout.PopupView.AnimationMode = AnimationMode.SlideOnRight;
}
```

```
.....  
}
```



### SlideOnTop

PopupView will be animated from top-to-bottom when it opens and from bottom-to-top when it closes.

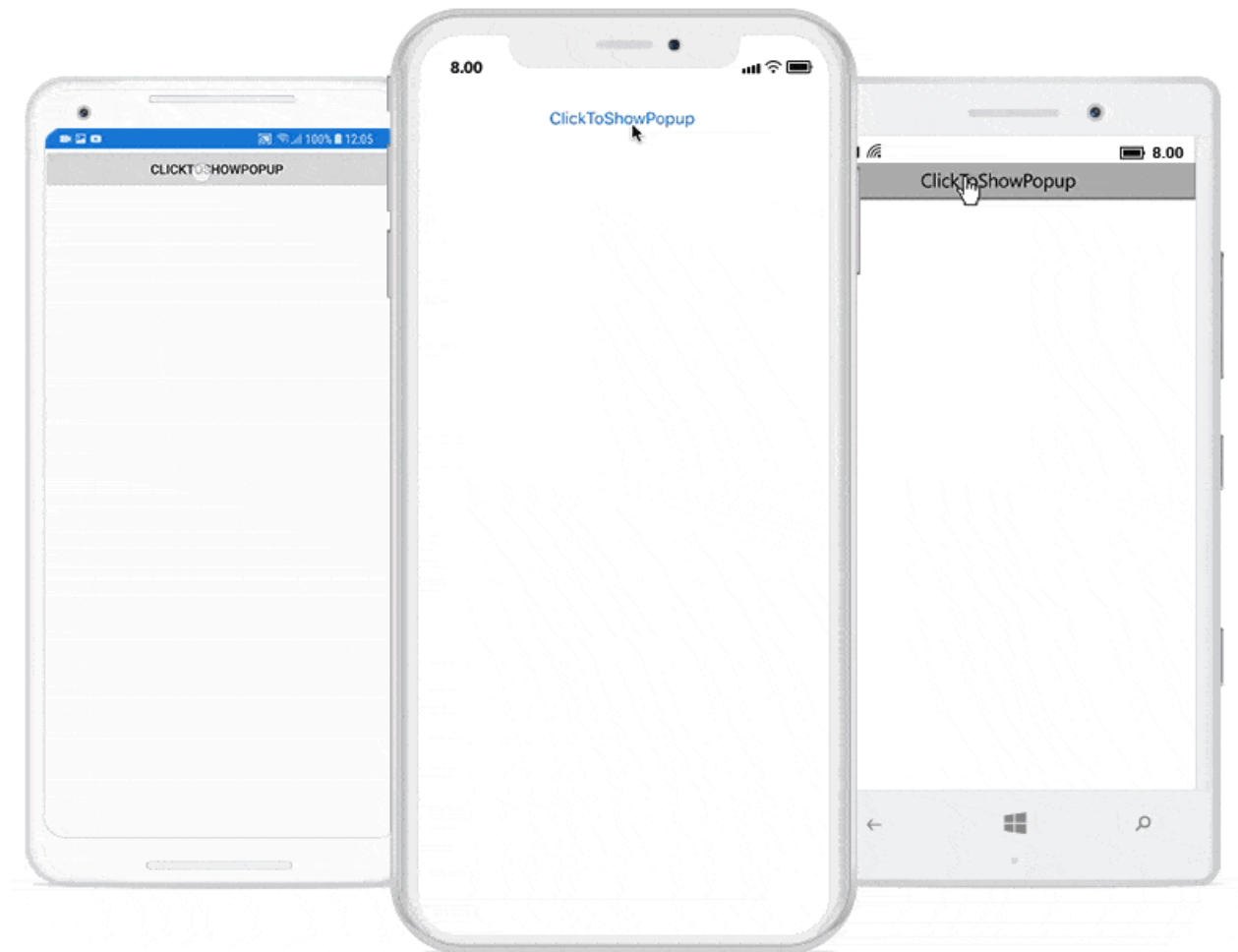
#### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">  
  <sfPopup:SfPopupLayout.PopupView>  
    <sfPopup:PopupView AnimationMode="SlideOnTop" />  
  </sfPopup:SfPopupLayout.PopupView>  
</sfPopup:SfPopupLayout>
```

#### C#

```
//MainPage.cs  
public MainPage()  
{  
  .....  
  InitializeComponent();  
}
```

```
popupLayout.PopupView.AnimationMode = AnimationMode.SlideOnTop;  
....  
}
```



### SlideOnBottom

PopupView will be animated from bottom-to-top when it opens and from top-to-bottom when it closes.

#### XML

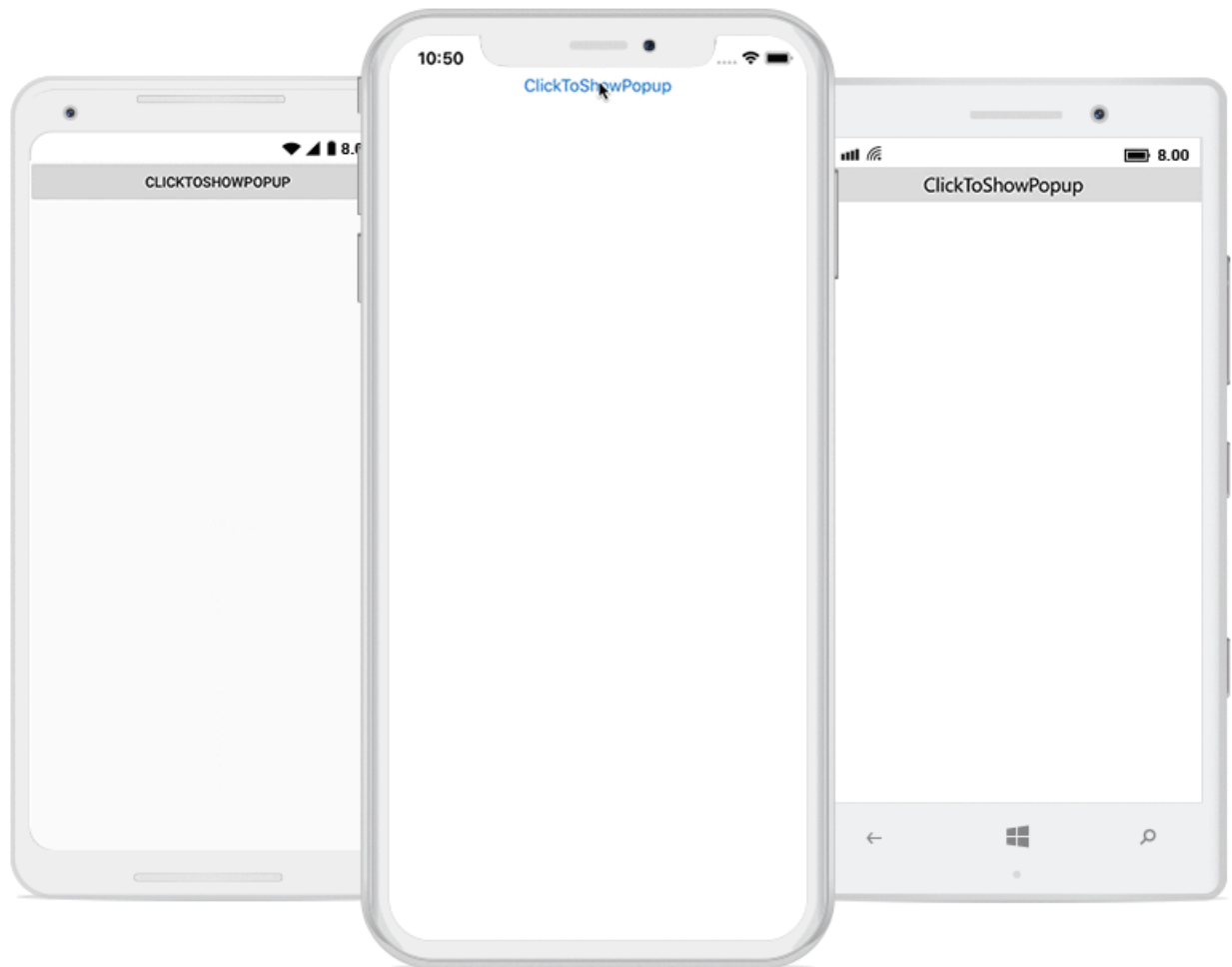
```
<sfPopup:SfPopupLayout x:Name="popUpLayout">  
  <sfPopup:SfPopupLayout.PopupView>  
    <sfPopup:PopupView AnimationMode="SlideOnBottom" />  
  </sfPopup:SfPopupLayout.PopupView>  
</sfPopup:SfPopupLayout>
```

#### C#

```
//MainPage.cs  
public MainPage()  
{  
  ....  
}
```



```
InitializeComponent();  
popupLayout.PopupView.AnimationMode = AnimationMode.SlideOnBottom;  
....  
}
```



None

Animation will not be applied.

#### XML

```
<sfPopup:SfPopupLayout x:Name="popUpLayout">  
  <sfPopup:SfPopupLayout.PopupView>  
    <sfPopup:PopupView AnimationMode="None" />  
  </sfPopup:SfPopupLayout.PopupView>  
</sfPopup:SfPopupLayout>
```

#### C#

```
//MainPage.cs  
public MainPage()  
{
```

```

.....
InitializeComponent();
popupLayout.PopupView.AnimationMode = AnimationMode.None;
.....
}

```

## Modal Window Popup

You can use popup layout as modal window by using the built-in Close icon and the [SfPopupLayout.StaysOpen](#) property prevents interaction with your application until you close the window.

**Modal:** Window loads under the parent window surrounded by an overlay which prevents clicking anywhere else on the screen apart from the control of the modal.

Modal does not require any action to open. It opens in the same window and gives callbacks when closing or opening the window.

Refer to the following code example in which the popup will close only if you click on Close icon.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popUpLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView HeightRequest="230"
HeaderTitle="Modal Window"
ShowFooter="False">
<sfPopup:PopupView.ContentTemplate>
<DataTemplate>
<Label Text="Window loads under the parent window surrounded by an
overlay which prevents clicking anywhere else on the screen
apart from the control of the modal. Modal opens in the same
window. It also does not require any user action to open, and
give callbacks when closing or opening the modal."
WidthRequest="260"
BackgroundColor="White"
HorizontalOptions="FillAndExpand"
/>
</DataTemplate>
</sfPopup:PopupView.ContentTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="mainLayout">
<Button x:Name="clickToShowPopup" Text="ClickToShowPopup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand" />
</StackLayout>
</sfPopup:SfPopupLayout.Content>

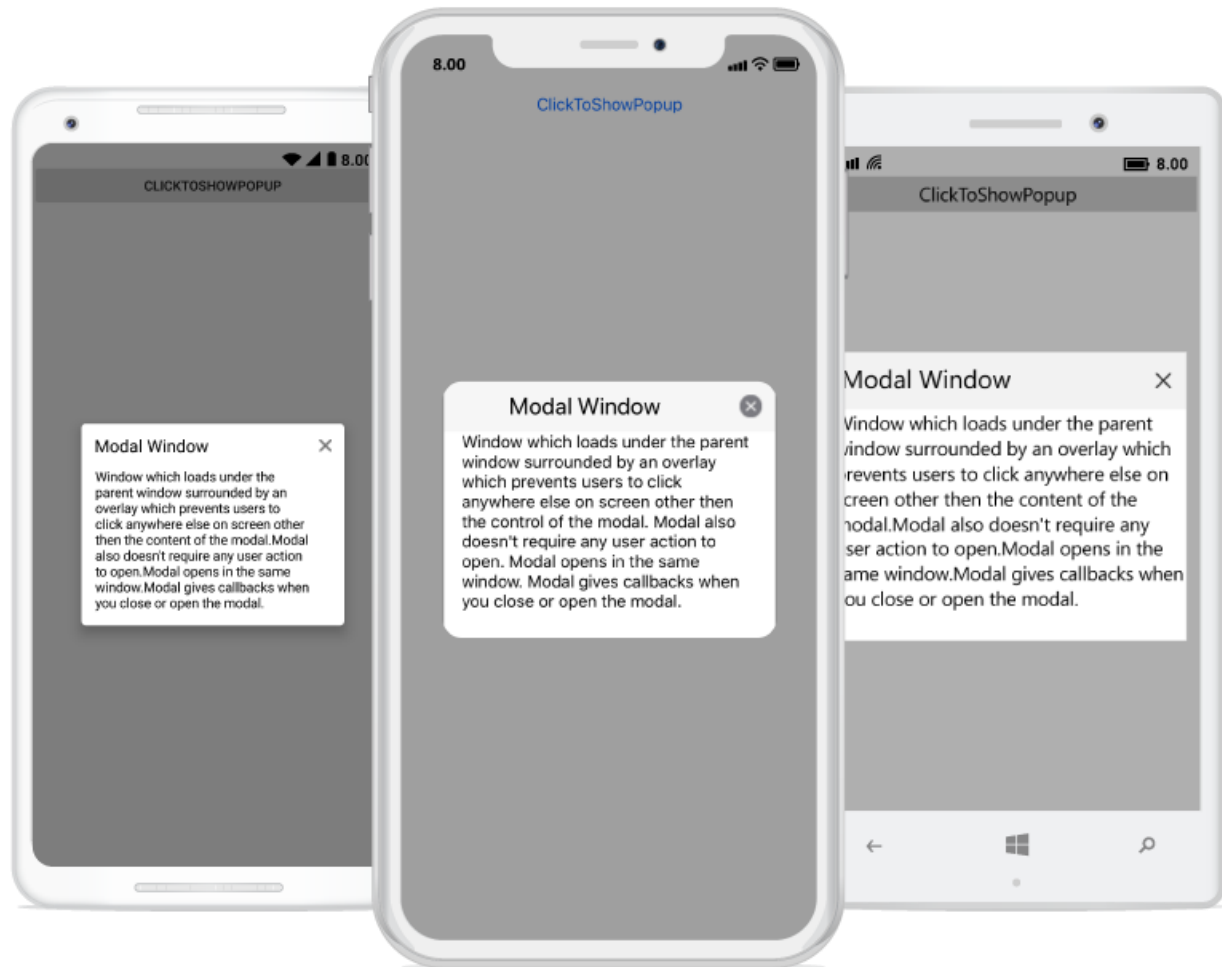
```

```
</sfPopup:SfPopupLayout>
</ContentPage>
```

## C#

```
using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        DataTemplate contentTemplateView;
        Label popupContent;
        public MainPage()
        {
            InitializeComponent();
            contentTemplateView = new DataTemplate(() =>
            {
                popupContent = new Label();
                popupContent.Text = "Window loads under the parent window surrounded by an overlay which prevents clicking anywhere else on the screen apart from the control of the modal. Modal opens in the same window. It also does not require any user action to open, and give callbacks when closing or opening the modal.";
                popupContent.WidthRequest = 260;
                popupContent.BackgroundColor = Color.White;
                popupContent.HorizontalOptions = LayoutOptions.FillAndExpand;
                return popupContent;
            });
            popupLayout.PopupView.HeightRequest = 230;
            popupLayout.PopupView.HeaderTitle = "Modal Window";
            popupLayout.PopupView.ContentTemplate = contentTemplateView;
            popupLayout.PopupView.ShowFooter = false;
            clickToShowPopup.Clicked += ClickToShowPopup_Clicked;
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            // Below code ensures that the popup doesn't collapse when user interacts outside the popup.
            popupLayout.StaysOpen = true;
            popupLayout.PopupView.ShowCloseButton = true;
            popupLayout.IsOpen = true;
        }
    }
}
```

Executing the above codes renders the following output in iOS, Android and Windows Phone devices respectively.



## Popup Events And Commands

There are four built-in events in the SfPopupLayout control namely:

- Opening
- Opened
- Closing
- Closed

### Opening event

The [SfPopupLayout.Opening](#) event will be fired whenever opening the PopupView in the application. It can cancel popup opening with [CancelEventArgs](#) that contains the following property:

- [Cancel](#): Popup opening is based on this value.

### XML

```
<sfPopup:SfPopupLayout x:Name="popupLayout" Opening="PopupLayout_Opening"/>
```

### C#

```

public MainPage()
{
    ....
    InitializeComponent();
    popupLayout.Opening += PopupLayout_Opening;
    ....
}
private void PopupLayout_Opening(object sender,
System.ComponentModel.CancelEventArgs e)
{
    e.Cancel = true;
}

```

### Opened event

The [SfPopupLayout.Opened](#) event will be fired whenever displaying the PopupView in the application.

You can execute your own set of codes once the popup is opened, and visible in the application in its respective event handler.

### XML

```

<sfPopup:SfPopupLayout x:Name="popupLayout" Opened="PopupLayout_Opened"/>

```

### C#

```

public MainPage()
{
    ....
    InitializeComponent();
    popupLayout.Opened += PopupLayout_Opened;
    ....
}
private void PopupLayout_Opened(object sender, EventArgs e)
{
    // Codes that needs to be executed once popup is visible in the screen.
}

```

### Closing event

The [SfPopupLayout.Closing](#) event will be fired whenever closing the PopupView in the application. It can cancel popup closing with `CancelEventArgs` that contains the following property:

- [Cancel](#): Popup opening is based on this value.

### XML

```

<sfPopup:SfPopupLayout x:Name="popupLayout" Closing="PopupLayout_Closing"/>

```

### C#

```

public MainPage()
{
    ....
    InitializeComponent();
}

```

```
popupLayout.Closing += PopupLayout_Closing;
....
}
private void PopupLayout_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
{
e.Cancel = true;
}
```

### Closed event

The [SfPopupLayout.Closed](#) event will be fired whenever dismissing the PopupView from the view.

You can execute your own set of codes once the popup is completely closed in its respective event handler.

### XML

```
<sfPopup:SfPopupLayout x:Name="popupLayout" Closed="PopupLayout_Closed"/>
```

### C#

```
public MainPage()
{
....
InitializeComponent();
popupLayout.Closed += PopupLayout_Closed;
....
}
private void PopupLayout_Closed(object sender, EventArgs e)
{
// Codes that needs to be executed once popup is completely closed.
}
```

### Accept command

The [SfPopupLayout.PopupView.AcceptCommand](#) will be fired when clicking the Accept button in the popup footer.

To handle the Accept button, follow the procedure:

- Derive a class from `ICommand`, and implement the necessary interface.
- Return true in the `CanExecute()` override method to close the popup, and fire the `Execute()` method.
- Return false to prevent popup from closing, and `Execute()` method is not fired.

### XML

```
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView AppearanceMode="TwoButton" />
</sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>
```

**C#**

```

public MainPage()
{
    ....
    InitializeComponent();
    popupLayout.PopupView.AppearanceMode = AppearanceMode.TwoButton;
    popupLayout.PopupView.AcceptCommand = new AcceptButtonCustomCommand();
    ....
}
//Accept Button Event handler
public class AcceptButtonCustomCommand : ICommand
{
    public event EventHandler CanExecuteChanged;
    public bool CanExecute(object parameter)
    {
        return false;
    }
    public void Execute(object parameter)
    {
        // You can write your set of codes that needs to be executed
    }
}

```

## Decline command

The [SfPopupLayout.PopupView.DeclineCommand](#) will be fired when clicking the Decline button in the popup footer.

To handle the Decline button, follow the procedure:

- Derive a class from `ICommand`, and implement the necessary interface.
- Return true in the `CanExecute()` override method to close the popup, and fire the `Execute()` method.
- Return false to prevent popup from closing, and `Execute()` method is not fired.

**XML**

```

<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView AppearanceMode="TwoButton" />
</sfPopup:SfPopupLayout.PopupView>
</sfPopup:SfPopupLayout>

```

**C#**

```

public MainPage()
{
    ....
    InitializeComponent();
    popupLayout.PopupView.AppearanceMode = AppearanceMode.TwoButton;
    popupLayout.PopupView.DeclineCommand = new DeclineButtonCustomCommand();
    ....
}
//Decline Button Event handler

```

```
public class DeclineButtonCustomCommand : ICommand
{
    public event EventHandler CanExecuteChanged;
    public bool CanExecute(object parameter)
    {
        return false;
    }
    public void Execute(object parameter)
    {
        // You can write your set of codes that needs to be executed
    }
}
```

## Localization

Localization is the process of translating application resources into different languages for specific cultures. SfPopupLayout uses the following static text that can be localized in application level:

- Title
- ACCEPT
- DECLINE
- Popup\_message

To localize the SfPopupLayout, follow the steps in application level:

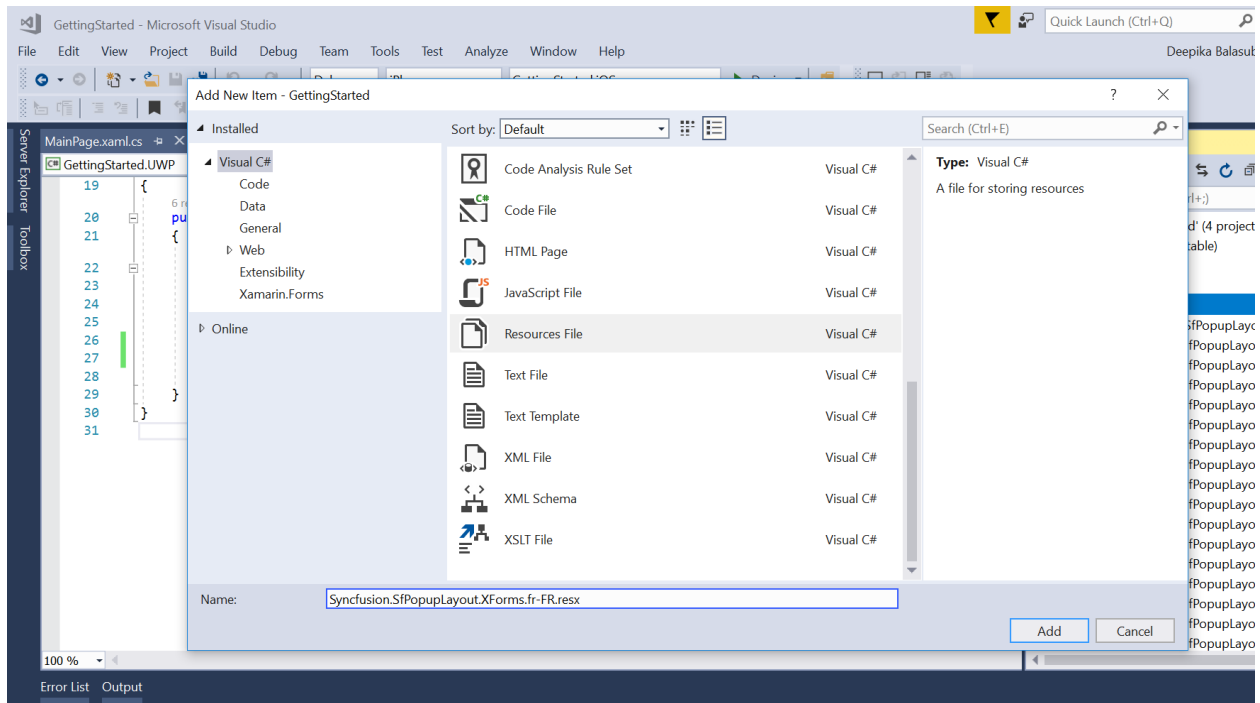
1. Add a .resx file.
2. Convert the platform specific language format to .NET format.
3. Apply the converted format.

### Add a .resx file

In the portable project of your application, add a .resx file inside the resources folder with **Build Action - > EmbeddedResource**. File name should be Syncfusion control's Namespace + language code format.

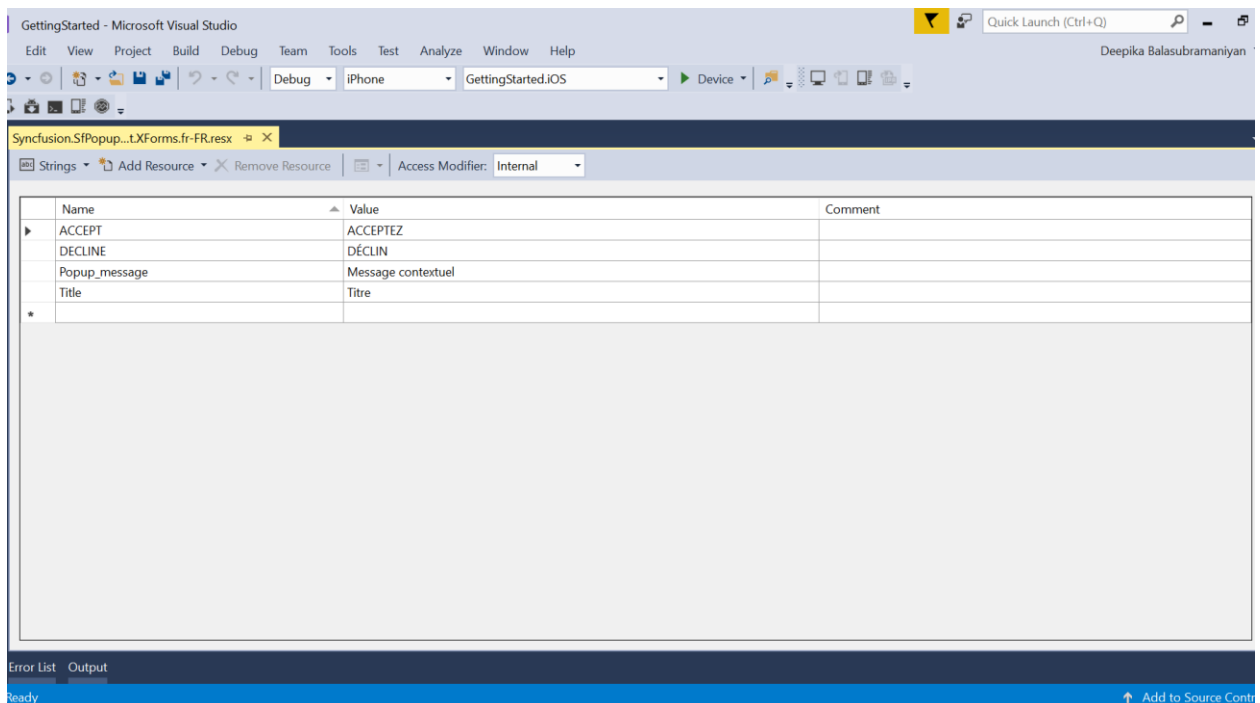
For example, to set the culture as French, the file should be named as **Syncfusion.SfPopupLayout.XForms.fr-FR.resx**.





Based on the language, set the appropriate equivalent text to the static text in the .resx file.

**Note:** You should create and add separate .resx files for the individual languages.



Convert the platform specific language format to .NET format

To get the localized text from the added .resx file, declare an interface named `ILocalize` in your PCL project and implement the interface in each platform renderer. This will query the language set in the device using platform specific code and convert to .NET format.

Refer to the following code snippet to declare the interface in PCL project.

### C#

```
public interface ILocalize
{
    CultureInfo GetCurrentCultureInfo();
    void SetLocale(CultureInfo cultureInfo);
}

public class PlatformCulture
{
    public PlatformCulture(string platformCultureString)
    {
        if (String.IsNullOrEmpty(platformCultureString))
            throw new ArgumentException("Expected culture identifier",
                "platformCultureString"); // in C# 6 use nameof(platformCultureString)
        PlatformString = platformCultureString.Replace("_", "-"); // .NET expects
        dash, not underscore
        var dashIndex = PlatformString.IndexOf("-", StringComparison.Ordinal);
        if (dashIndex > 0)
        {
            var parts = PlatformString.Split('-');
            LanguageCode = parts[0];
            LocaleCode = parts[1];
        }
        else
        {
            LanguageCode = PlatformString;
            LocaleCode = "";
        }
    }

    public string PlatformString
    {
        get; private set;
    }

    public string LanguageCode
    {
        get; private set;
    }

    public string LocaleCode
    {
        get; private set;
    }

    public override string ToString()
    {
        return PlatformString; ;
    }
}
```

Refer to the following code to implement the interface in Android renderer project.

### C#

```
public class Localize : ILocalize
{
    public void SetLocale(CultureInfo cultureInfo)
    {
    }
```

```

Thread.CurrentThread.CurrentCulture = cultureInfo;
Thread.CurrentThread.CurrentUICulture = cultureInfo;
}
public CultureInfo GetCurrentCultureInfo()
{
    var netLanguage = "en";
    var androidLocale = Java.Util.Locale.Default;
    netLanguage = AndroidToDotnetLanguage(androidLocale.ToString().Replace("_",
    "-"));
    // this gets called a lot - try/catch can be expensive so consider caching
    // or something
    CultureInfo cultureInfo = null;
    try
    {
        cultureInfo = new CultureInfo(netLanguage);
    }
    catch
    {
        // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
        // fallback to first characters, in this case "en"
        try
        {
            var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
            cultureInfo = new CultureInfo(fallback);
        }
        catch
        {
            // iOS language not valid .NET culture, falling back to English
            cultureInfo = new CultureInfo("en");
        }
    }
    return cultureInfo;
}
private string AndroidToDotnetLanguage(string androidLanguage)
{
    var netLanguage = androidLanguage;
    //certain languages need to be converted to CultureInfo equivalent
    switch (androidLanguage)
    {
        case "ms-BN": // "Malaysian (Brunei)" not supported .NET culture
        case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture
        case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture
            netLanguage = "ms"; // closest supported
            break;
        case "in-ID": // "Indonesian (Indonesia)" has different code in .NET
            netLanguage = "id-ID"; // correct code for .NET
            break;
        case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET
            culture
            netLanguage = "de-CH"; // closest supported
            break;
        // add more application-specific cases here (if required)
        // ONLY use cultures that have been tested and known to work
    }
    return netLanguage;
}
private string ToDotnetFallbackLanguage(PlatformCulture platformCulture)

```

```

{
    var netLanguage = platformCulture.LanguageCode; // use the first part of the
    identifier (two chars, usually);
    switch (platformCulture.LanguageCode)
    {
        case "gsw":
            netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
            break;
        // add more application-specific cases here (if required)
        // ONLY use cultures that have been tested and known to work
    }
    return netLanguage;
}

```

Refer to the following code to implement the interface in iOS renderer project.

### C#

```

public class Localize : ILocalize
{
    public void SetLocale(CultureInfo cultureInfo)
    {
        Thread.CurrentThread.CurrentCulture = cultureInfo;
        Thread.CurrentThread.CurrentUICulture = cultureInfo;
    }
    public CultureInfo GetCurrentCultureInfo()
    {
        var netLanguage = "en";
        if (NSLocale.PreferredLanguages.Length > 0)
        {
            var pref = NSLocale.PreferredLanguages[0];
            netLanguage = iOSToDotnetLanguage(pref);
        }
        // this gets called a lot - try/catch can be expensive so consider caching
        or something
        CultureInfo cultureInfo = null;
        try
        {
            cultureInfo = new CultureInfo(netLanguage);
        }
        catch
        {
            // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
            // fallback to first characters, in this case "en"
            try
            {
                var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
                cultureInfo = new CultureInfo(fallback);
            }
            catch
            {
                // iOS language not valid .NET culture, falling back to English
                cultureInfo = new CultureInfo("en");
            }
        }
    }
}

```

```

return cultureInfo;
}
private string iOSToDotnetLanguage(string iosLanguage)
{
var netLanguage = iosLanguage;
//certain languages need to be converted to CultureInfo equivalent
switch (iosLanguage)
{
case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture
case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture
netLanguage = "ms"; // closest supported
break;
case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET
culture
netLanguage = "de-CH"; // closest supported
break;
// add more application-specific cases here (if required)
// ONLY use cultures that have been tested and known to work
}
return netLanguage;
}
private string ToDotnetFallbackLanguage(PlatformCulture platCulture)
{
var netLanguage = platCulture.LanguageCode; // use the first part of the
identifier (two chars, usually);
switch (platCulture.LanguageCode)
{
//
case "pt":
netLanguage = "pt-PT"; // fallback to Portuguese (Portugal)
break;
case "gsw":
netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
break;
// add more application-specific cases here (if required)
// ONLY use cultures that have been tested and known to work
}
return netLanguage;
}
}
}

```

Implementation of the interface is not required for UWP project, since the resources automatically recognizes the selected language.

#### Apply the converted format

After setting the root/main page of the application in your App.Xaml.cs file of the PCL project, initialize a new instance of the `ResourceManager` class and set it to the [PopupLayoutResourceManager.Manager](#) property to look up into the resources with specified root name in the given assembly. Using `DependencyService`, call `SetLocale()` of the implemented interface with necessary language code as parameter.

#### C#

```

public partial class App : Application
{

```

```
public App()
{
    InitializeComponent();
    MainPage = new GettingStarted.MainPage();
    if (Device.RuntimePlatform == Device.iOS || Device.RuntimePlatform ==
        Device.Android)
    {
        PopupLayoutResourceManager.Manager = new
        ResourceManager("GettingStarted.Resources.Syncfusion.SfPopupLayout.XForms",
            this.GetType().GetTypeInfo().Assembly);
        // the ResourceManager class constructor has two parameters.
        // 1. ResXPath => Full path of the resx file in the application. Here in the
        // above line GettingStarted refers to the namespace of the Application
        // 2. Assembly => Application assembly (PCL)
        // Sets the required culture to the static texts in the control.
        DependencyService.Get<ILocalize>().SetLocale(new CultureInfo("fr-FR"));
    }
}
```

For Android and iOS, it is mandatory to implement the previous steps. However, to set the specific language to the application irrespective of the selected language in the device, use `CultureInfo.CurrentCulture` in a specific project of UWP platform.

Refer to the following code example to localize the text in UWP platform.

MainPage.Xaml.cs

### **C#**

```
public MainPage()
{
    this.InitializeComponent();
    SfPopupLayoutRenderer.Init();
    // Applying localization for UWP
    CultureInfo.CurrentCulture = new CultureInfo("fr");
    LoadApplication(new GettingStarted.App());
}
```



You can download the sample [here](#).

### AutomationId

SfPopupLayout supports built-in [AutomationId](#) for all its inner elements. These `AutomationId` values allow the automation framework to find and interact with the inner elements when the test scripts are run. A unique `AutomationId` is maintained for each inner element by prefixing the control's `AutomationId` with the inner element's Id.

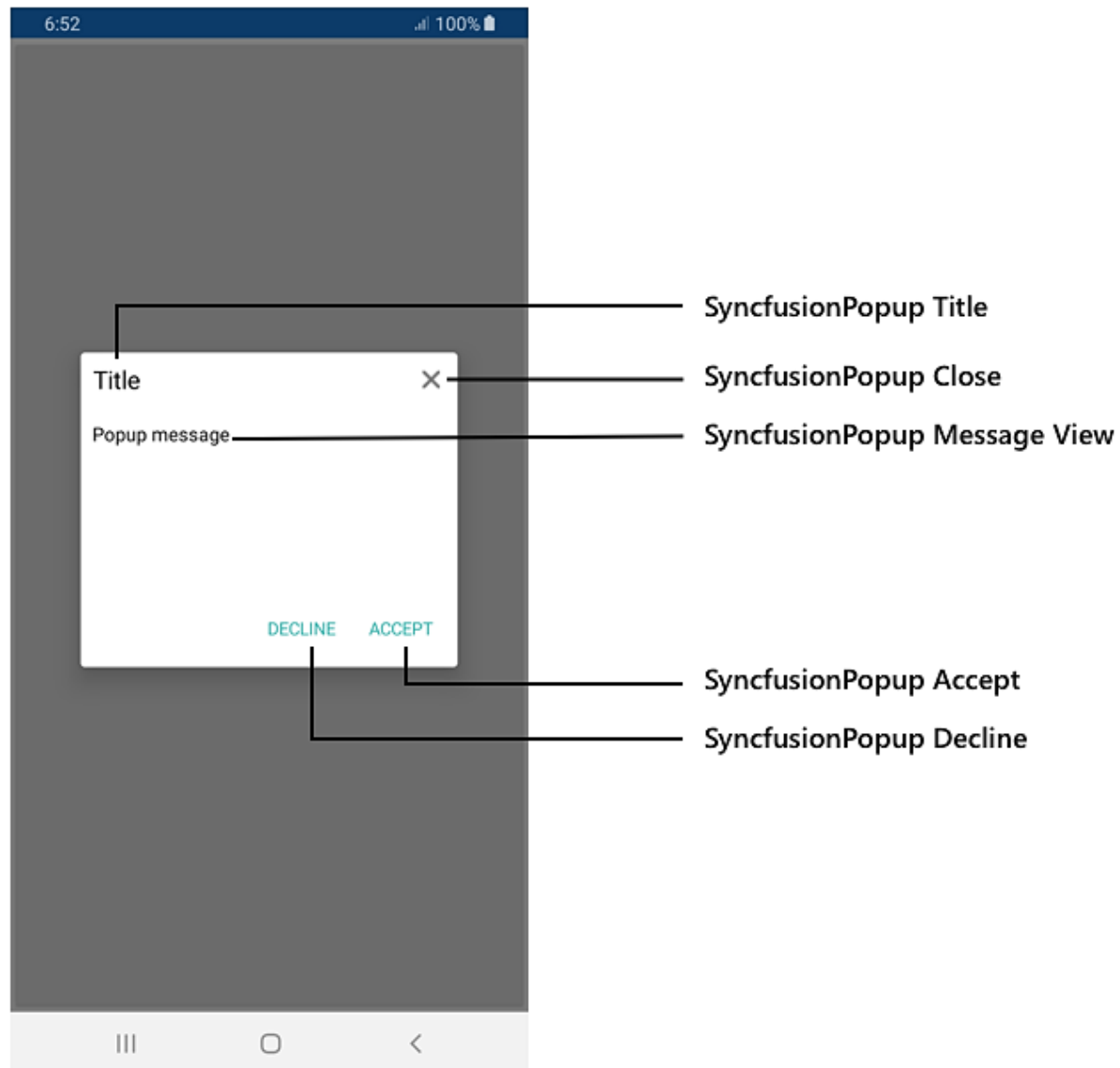
#### AutomationId for popup view inner elements

The below table illustrates the predefined automation values set internally which can be used to identify the PopupView's elements.

Element	Value
Header title	"Title"
Header close button	"Close"
PopupView content	"Message"
Accept button	"Accept"
Decline button	"Decline"

The following screenshot illustrates the `AutomationId` values of the inner elements of SfPopupView.





The following code snippet demonstrates how to set `AutomationId` to the `SfPopupLayout`.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms">
<sfPopup:SfPopupLayout x:Name="popupLayout" AutomationId="SyncfusionPopup">
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="mainLayout">
<Button x:Name="clickToShowPopup" Text="Click To Show Popup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked"/>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

```

</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>

```

**C#**

```

using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}

```

The following code snippet demonstrates how to access the inner elements of SfPopupView from the automation script.

**C#**

```

[Test]
[Description("SfPopupLayout Automation Id")]
public void SfPopupLayout_AutomationId()
{
    // To tap the Accept button
    App.Tap("SyncfusionPopup Accept");
    // To tap the Decline button
    App.Tap("SyncfusionPopup Decline");
    // To tap the Close button
    App.Tap("SyncfusionPopup Close");
    // To tap the Header Title label
    App.Tap("SyncfusionPopup Title");
    // To tap the MessageView
    App.Tap("SyncfusionPopup MessageView");
}

```

## AutomationId for template content

To access the template elements from script, set the **AutomationId** directly to the template view from the sample level.

The following code snippet explains how to set **AutomationId** for the template content.

**XML**

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"

```

```

x:Class="GettingStarted.MainPage"
Padding="0,40,0,0"
xmlns:sfPopup="clr-
namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XF
orms">
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:PopupView>
<sfPopup:PopupView.ContentTemplate>
<DataTemplate>
<Button Text="This is SfPopupLayout" BackgroundColor="SkyBlue"
AutomationId="TemplateButton"/>
</DataTemplate>
</sfPopup:PopupView.ContentTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout x:Name="mainLayout">
<Button x:Name="clickToShowPopup" Text="Click To Show Popup"
VerticalOptions="Start" HorizontalOptions="FillAndExpand"
Clicked="ClickToShowPopup_Clicked"/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>

```

**C#**

```

using Syncfusion.XForms.PopupLayout;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void ClickToShowPopup_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
    }
}

```

Refer to the following code snippet to access the template content of SfPopupView from the automation script.

**C#**

```

[Test]
[Description("SfPopupLayout Template Automation Id")]
public void Template_AutomationId()
{
    // To tap the Content template button
    App.Tap("TemplateButton");
}

```

## FAQ

Load the SfPopupLayout in GridTappedEvent of the SfDataGrid

The SfPopupLayout allows opening it in the GridTapped event of the SfDataGrid

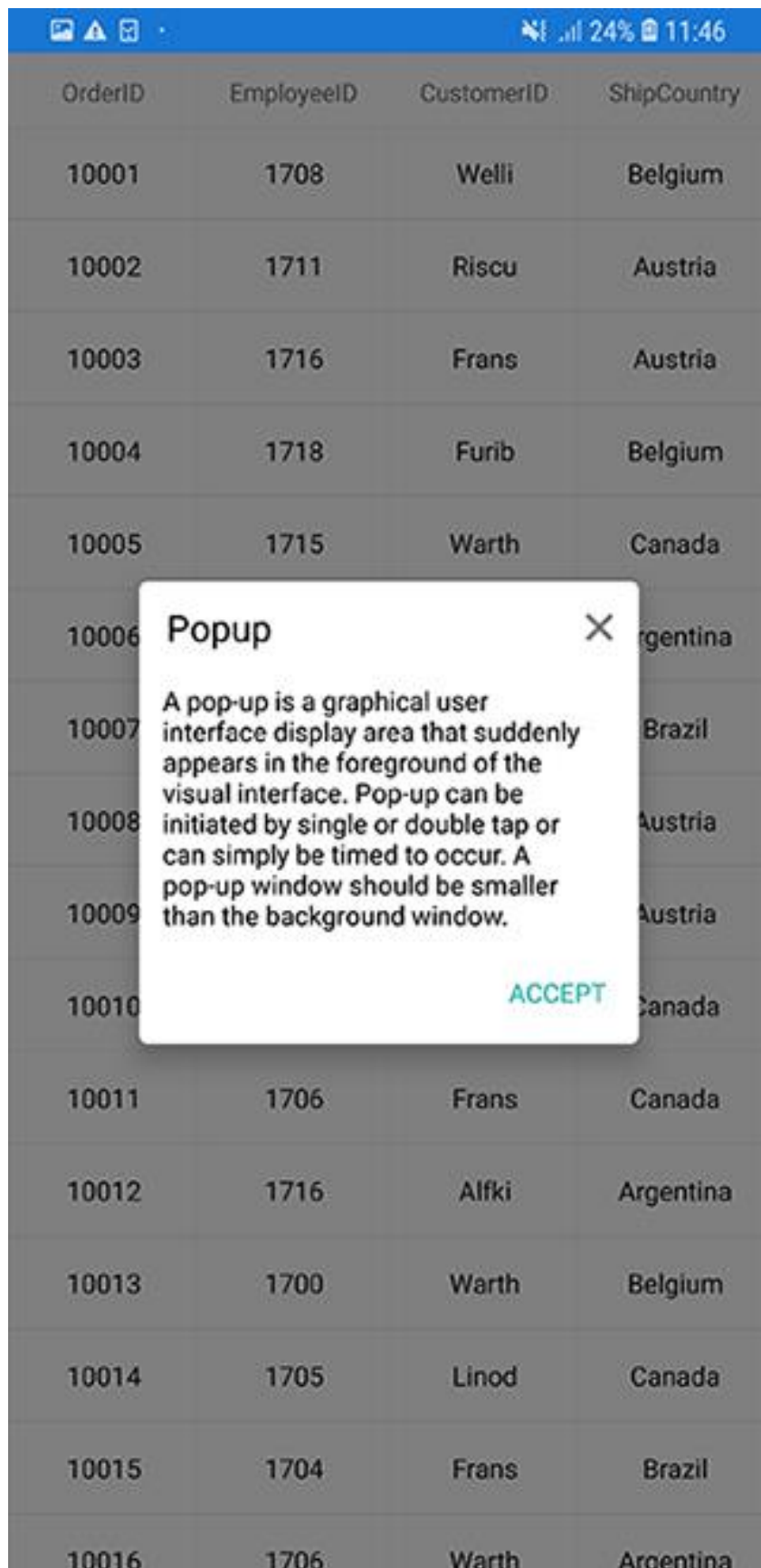
## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:PopupDemo"
xmlns:sfPopup="clr-
namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XF
orms"
xmlns:sfDataGrid="clr-
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms
"
x:Class="PopupDemo.MainPage">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView HeaderTitle="Popup">
<sfPopup:PopupView.ContentTemplate>
<DataTemplate>
<Label Text="A pop-up is a graphical user interface display area that
suddenly appears in the foreground of the visual interface. Pop-up can be
initiated by single or double tap or can simply be timed to occur. A pop-up
window should be smaller than the background window or interface; otherwise,
its a replacement interface."
BackgroundColor="White"
TextColor="Black"
/>
</DataTemplate>
</sfPopup:PopupView.ContentTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<sfDataGrid:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
GridTapped="DataGrid_GridTapped"
ColumnSizer="Star"
>
</sfDataGrid:SfDataGrid>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

## C#

```
using Syncfusion.XForms.PopupLayout;
using Syncfusion.SfDataGrid.XForms;
namespace GettingStarted
{
public partial class MainPage : ContentPage
```

```
{
SfDataGrid dataGrid;
ViewModel viewModel;
SfPopupLayout popupLayout;
public MainPage()
{
InitializeComponent();
dataGrid = new SfDataGrid();
viewModel = new ViewModel();
dataGrid.ItemsSource = viewModel.OrdersInfo;
dataGrid.GridTapped += DataGrid_GridTapped;
dataGrid.ColumnSizer = ColumnSizer.Star;
popupLayout = new SfPopupLayout();
popupLayout.PopupView.HeaderTitle = "Popup";
popupLayout.PopupView.ContentTemplate = new DataTemplate(() =>
{
return new Label()
{
Text = "A pop-up is a graphical user interface display area that suddenly
appears in the foreground of the visual interface. Pop-up can be initiated
by single or double tap or can simply be timed to occur. A pop-up window
should be smaller than the background window or interface; otherwise, its a
replacement interface.",
BackgroundColor = Color.White,
TextColor = Color.Black
};
});
popupLayout.Content = dataGrid;
this.Content = popupLayout;
}
private void DataGrid_GridTapped(object sender,
Syncfusion.SfDataGrid.XForms.GridTappedEventArgs e)
{
// Popup is opened at the Grid tapped event
popupLayout.Show();
}
}
```



Sample Link: You can download the above sample code by clicking [here](#).

Open SfPopupLayout in ItemTapped event of SfListView

The SfPopupLayout allows opening it in the ItemTapped event of the SfListView.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:PopupDemo"
xmlns:sfPopup="clr-
namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XF
orms"
xmlns:sfListView="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
x:Class="PopupDemo.MainPage">
<ContentPage.BindingContext>
<local:ContactsViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView WidthRequest="220" HeightRequest="120"
ShowFooter="False">
<sfPopup:PopupView.ContentTemplate>
<DataTemplate>
<Label Text="ListView item is tapped"
BackgroundColor="White"
TextColor="Black"
HorizontalTextAlignment="Center"
/>
</DataTemplate>
</sfPopup:PopupView.ContentTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<sfListView:SfListView x:Name="listView" ItemSpacing="5"
ItemsSource="{Binding Items}"
SelectionMode="None"
ItemTapped="ListView_ItemTapped">
<sfListView:SfListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid x:Name="grid" RowSpacing="1">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="50" />
<ColumnDefinition Width="200" />
<ColumnDefinition Width="50" />
</Grid.ColumnDefinitions>
<Image Source="{Binding ContactImage}"
VerticalOptions="Center"
HorizontalOptions="Center"
HeightRequest="50"/>
</Image>
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</sfListView:SfListView.ItemTemplate>
</sfListView:SfListView>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

```

<Label Grid.Column="1"
HorizontalTextAlignment="Center"
LineBreakMode="NoWrap"
Text="{Binding ContactName}"
FontSize="Medium" />
<Image Grid.Column="2"
Source="{Binding ContactType}"
VerticalOptions="End"
HorizontalOptions="End"
HeightRequest="50" />
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</sfListView:SfListView.ItemTemplate>
</sfListView:SfListView>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>

```

**C#**

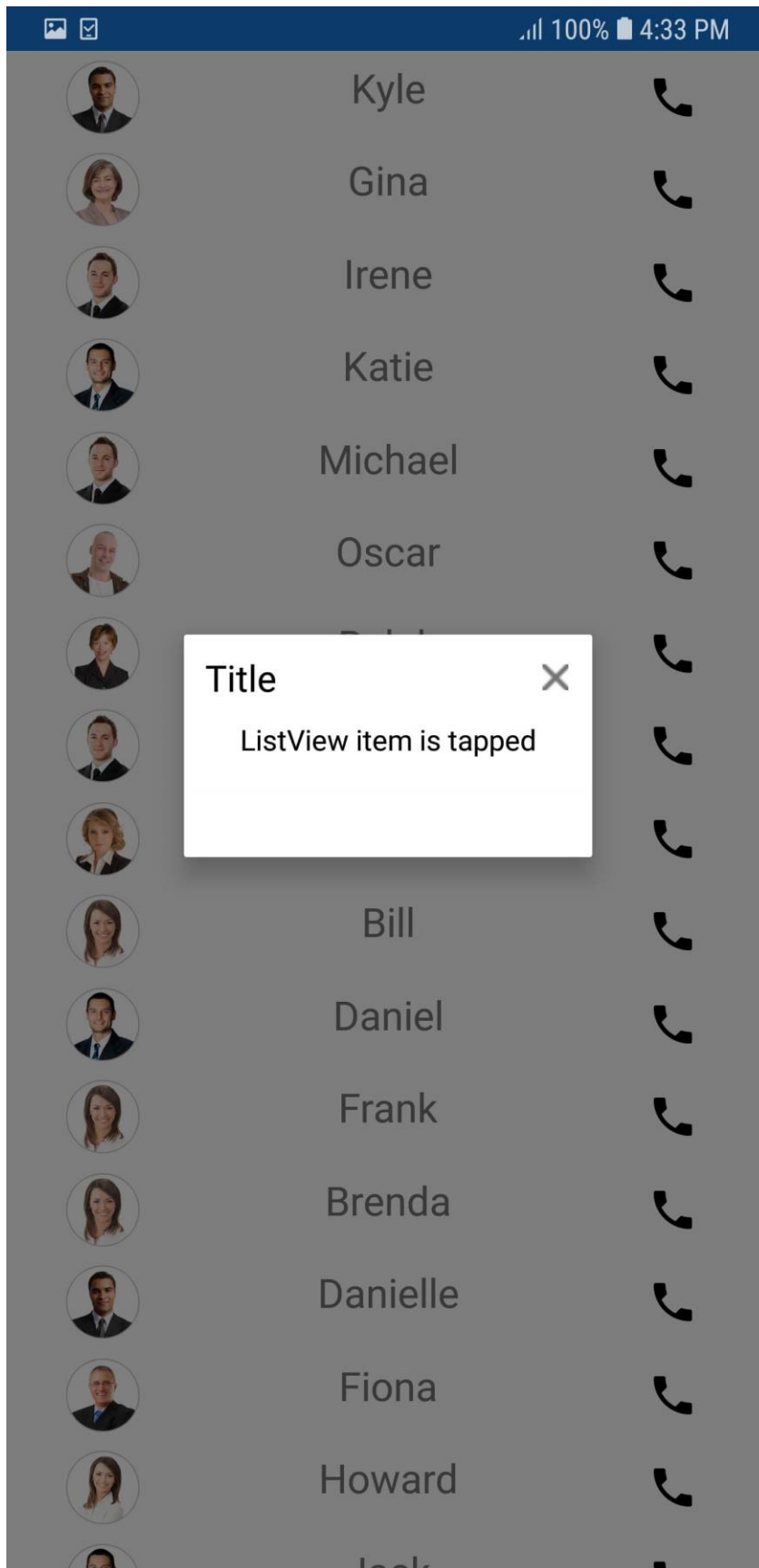
```

using Syncfusion.ListView.XForms;
using Syncfusion.XForms.PopupLayout;
namespace PopupDemo
{
    public partial class MainPage : ContentPage
    {
        SfListView listView;
        ContactsViewModel viewModel;
        SfPopupLayout popupLayout;
        public MainPage()
        {
            InitializeComponent();
            listView = new SfListView() { ItemSpacing = 5 };
            listView.ItemTemplate = new DataTemplate(() =>
            {
                ViewCell viewCell = new ViewCell();
                var grid = new Grid() { RowSpacing = 1 };
                grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = 50 });
                grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = 200 });
                grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = 50 });
                var contactImage = new Image()
                {
                    VerticalOptions = LayoutOptions.Center,
                    HorizontalOptions = LayoutOptions.Center,
                    HeightRequest = 50
                };
                contactImage.SetBinding(Image.SourceProperty, new Binding("ContactImage"));
                var contactName = new Label()
                {
                    HorizontalTextAlignment = TextAlignment.Center,
                    LineBreakMode = LineBreakMode.NoWrap,
                    FontSize = Font.SystemFontSize(NamedSize.Medium).FontSize,
                };
                contactName.SetBinding(Label.TextProperty, new Binding("ContactName"));
            });
        }
    }
}

```



```
var contactType = new Image()
{
    VerticalOptions = LayoutOptions.End,
    HorizontalOptions = LayoutOptions.End,
    HeightRequest = 50,
};
contactType.SetBinding(Image.SourceProperty, new Binding("ContactType"));
grid.Children.Add(contactImage, 0, 0);
grid.Children.Add(contactName, 1, 0);
grid.Children.Add(contactType, 2, 0);
viewCell.View = grid;
return viewCell;
});
viewModel = new ContactsViewModel();
listView.ItemsSource = viewModel.Items;
listView.SelectionMode = SelectionMode.None;
listView.ItemTapped += ListView_ItemTapped;
popupLayout = new SfPopupLayout();
popupLayout.PopupView.HeightRequest = 120;
popupLayout.PopupView.WidthRequest = 220;
popupLayout.PopupView.ShowFooter = false;
popupLayout.PopupView.ContentTemplate = new DataTemplate(() =>
{
    return new Label()
    {
        Text = "ListView item is tapped",
        BackgroundColor = Color.White,
        TextColor = Color.Black,
        HorizontalTextAlignment = TextAlignment.Center
    };
});
popupLayout.Content = listView;
this.Content = popupLayout;
}
private void ListView_ItemTapped(object sender,
Syncfusion.ListView.XForms.ItemTappedEventArgs e)
{
    popupLayout.Show();
}
}
```



Sample Link: You can download the above sample code by clicking [here](#).

### Show ListView as a popup

The SfPopupLayout allows loading the SfListView as a content of the popup. You have to set **WidthRequest** and **HeightRequest** properties for loading SfListView in the SfPopupLayout.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:PopupDemo"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms"
xmlns:sfListView="clr-namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
x:Class="PopupDemo.MainPage">
<ContentPage.BindingContext>
<local:ContactsViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView HeaderTitle="ListView">
<sfPopup:PopupView.ContentTemplate>
<DataTemplate>
<sfListView:SfListView x:Name="listView" ItemSpacing="5"
WidthRequest="350"
ItemsSource="{Binding Items}"
>
<sfListView:SfListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid x:Name="grid" RowSpacing="1">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="50" />
<ColumnDefinition Width="200" />
<ColumnDefinition Width="50" />
</Grid.ColumnDefinitions>
<Image Source="{Binding ContactImage}"
VerticalOptions="Center"
HorizontalOptions="Center"
HeightRequest="50"/>
<Label Grid.Column="1"
HorizontalTextAlignment="Center"
LineBreakMode="NoWrap"
Text="{Binding ContactName}"
FontSize="Medium" />
<Image Grid.Column="2"
Source="{Binding ContactType}"
VerticalOptions="End"
HorizontalOptions="End"
HeightRequest="50"/>

```

```

</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</sfListView:SfListView.ItemTemplate>
</sfListView:SfListView>
</DataTemplate>
</sfPopup:PopupView.ContentTemplate>
</sfPopup:PopupView>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout>
<Button Text="Click to show popup" Clicked="isOpenButton_Clicked"/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>

```

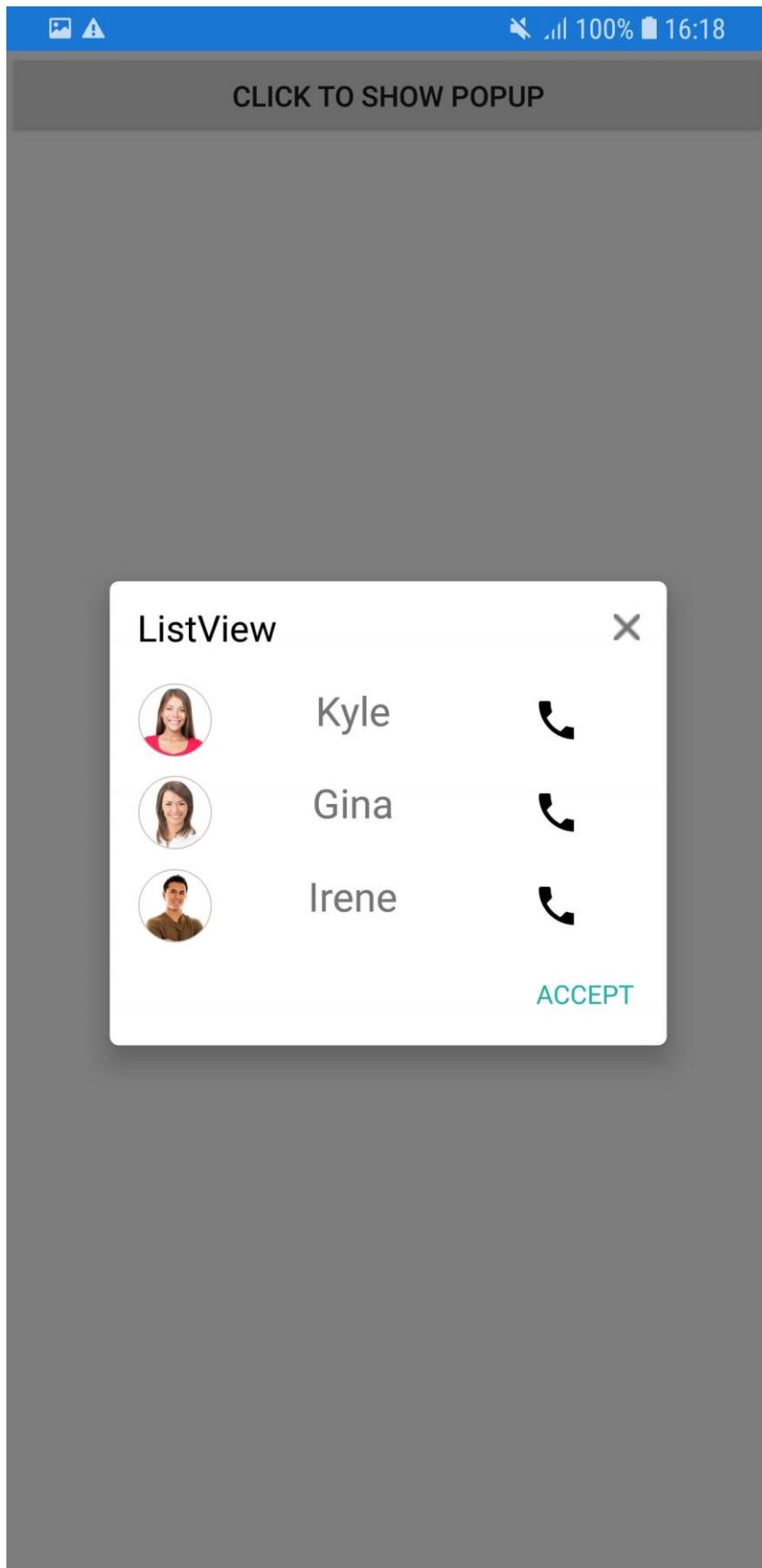
**C#**

```

using Syncfusion.ListView.XForms;
using Syncfusion.XForms.PopupLayout;
namespace PopupDemo
{
    public partial class MainPage : ContentPage
    {
        SfListView listView;
        ContactsViewModel viewModel;
        SfPopupLayout popupLayout;
        public MainPage()
        {
            InitializeComponent();
            listView = new SfListView() { ItemSpacing = 5 };
            listView.WidthRequest = 350;
            listView.ItemTemplate = new DataTemplate(() =>
            {
                ViewCell viewCell = new ViewCell();
                var grid = new Grid() { RowSpacing = 1 };
                grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = 50 });
                grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = 200 });
                grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = 50 });
                var contactImage = new Image()
                {
                    VerticalOptions = LayoutOptions.Center,
                    HorizontalOptions = LayoutOptions.Center,
                    HeightRequest = 50
                };
                contactImage.SetBinding(Image.SourceProperty, new Binding("ContactImage"));
                var contactName = new Label()
                {
                    HorizontalTextAlignment = TextAlignment.Center,
                    LineBreakMode = LineBreakMode.NoWrap,
                    FontSize = Font.SystemFontSize(NamedSize.Medium).FontSize,
                };
                contactName.SetBinding(Label.TextProperty, new Binding("ContactName"));
                var contactType = new Image()

```

```
{
    VerticalOptions = LayoutOptions.End,
    HorizontalOptions = LayoutOptions.End,
    HeightRequest = 50,
};
contactType.SetBinding(Image.SourceProperty, new Binding("ContactType"));
grid.Children.Add(contactImage, 0, 0);
grid.Children.Add(contactName, 1, 0);
grid.Children.Add(contactType, 2, 0);
viewCell.View = grid;
return viewCell;
});
viewModel = new ContactsViewModel();
listView.ItemsSource = viewModel.Items;
popupLayout = new SfPopupLayout();
popupLayout.PopupView.HeaderTitle = "ListView";
popupLayout.PopupView.ContentTemplate = new DataTemplate(() =>
{
    return listView;
});
StackLayout stackLayout = new StackLayout();
Button isOpenButton = new Button();
isOpenButton.Clicked += isOpenButton_Clicked;
isOpenButton.Text = "Click to show popup";
stackLayout.Children.Add(isOpenButton);
popupLayout.Content = stackLayout;
this.Content = popupLayout;
}
private void isOpenButton_Clicked(object sender, EventArgs e)
{
    popupLayout.Show();
}
}
```



Sample Link: You can download the above sample code by clicking [here](#).

Display popup when interacting with a switch

[SfPopupLayout.IsOpen](#) is a bindable property and hence can be binded to any property and based on its value the popup will open or close. In the below code example, we have binded the `IsOpen` property with the `IsToggled` property of the the switch and the popup will be opened or closed as the switch toggles.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SamplePopup"
xmlns:sfpopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms"
x:Class="SamplePopup.MainPage"
Padding="0,20,0,0">
<sfpopup:SfPopupLayout IsOpen="{Binding Source={x:Reference popupSwitch},
Path=IsToggled}">
<sfpopup:SfPopupLayout.Content>
<StackLayout Orientation="Horizontal">
<Label Text="On the switch to show popup" />
<Switch x:Name="popupSwitch" IsToggled="False" VerticalOptions="Start" />
</StackLayout>
</sfpopup:SfPopupLayout.Content>
</sfpopup:SfPopupLayout>
</ContentPage>
```

Display popup in MVVM

SfPopup can be used in MVVM architecture applications easily. In the below example [SfPopupLayout.IsOpen](#) property is binded to a property in the ViewModel based on which the popup is opened or closed. Refer the below code example to display popup in MVVM.

In the below code snippet, note that the Binding context is set for the page and the property (DisplayPopup) of the ViewModel is binded to the [SfPopupLayout.IsOpen](#).

#### XML

```
<ContentPage.BindingContext>
<local:ViewModel />
</ContentPage.BindingContext>
<sfpopup:SfPopupLayout x:Name="popup" IsOpen="{Binding DisplayPopup}">
<sfpopup:SfPopupLayout.Content>
<StackLayout x:Name="rootView">
<Button x:Name="isOpenButton" Text="Click to open popup" Command="{Binding
OpenPopupCommand}" />
</StackLayout>
</sfpopup:SfPopupLayout.Content>
</sfpopup:SfPopupLayout>
```

#### C#

```
// ViewModel.cs
public class ViewModel : INotifyPropertyChanged
```

```

{
private bool displayPopup;
public ICommand OpenPopupCommand { get; set; }
public bool DisplayPopup
{
get
{
return displayPopup;
}
set
{
displayPopup = value;
RaisePropertyChanged("DisplayPopup");
}
}
public ViewModel()
{
OpenPopupCommand = new Command(OpenPopup);
}
private void OpenPopup()
{
DisplayPopup = true;
}
#region INotifyPropertyChanged
public event PropertyChangedEventHandler PropertyChanged;
private void RaisePropertyChanged(string propertyName)
{
if(this.PropertyChanged != null)
{
this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
}
#endregion
}

```

### Load SfPopupLayout in Prism

Refer to the following code in which the SfPopupLayout is shown, when navigated to the next and previous pages via button click.

#### XML

```

// App.xaml
<prism:PrismApplication xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:prism="clr-namespace:Prism.Unity;assembly=Prism.Unity.Forms"
x:Class="PopupDemo.App">
</prism:PrismApplication>
//MainPage.xaml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:PopupDemo"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms"
x:Class="PopupDemo.MainPage">

```



```

<ContentPage.Content>
<StackLayout>
<Button Text="GO TO NEXT PAGE" VerticalOptions="Center"
HorizontalOptions="Center" Command="{Binding NavigateCommand}"/>
</StackLayout>
</ContentPage.Content>
</ContentPage>
// SecondPage.xaml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="PopupDemo.SecondPage">
<ContentPage.Content>
<StackLayout>
<Button Text="GO TO PREVIOUS PAGE" VerticalOptions="Center"
HorizontalOptions="Center" Command="{Binding NavigateCommand}"/>
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

## C#

```

// App.xaml.cs
using Prism.Unity;
namespace PopupDemo
{
public partial class App : PrismApplication
{
public App(IPlatformInitializer initializer = null) : base(initializer) { }
protected override void OnInitialized()
{
InitializeComponent();
NavigationService.NavigateAsync("NavigationPage/MainPage");
}
protected override void RegisterTypes()
{
Container.RegisterTypeForNavigation<NavigationPage>();
Container.RegisterTypeForNavigation<MainPage>();
Container.RegisterTypeForNavigation<SecondPage>();
}
}
}
// MainPageViewModel.cs
using Prism.Navigation;
using Syncfusion.XForms.PopupLayout;
using System.Windows.Input;
namespace PopupDemo
{
public class MainPageViewModel : INavigationAware
{
public INavigationService navigation_service;
public ICommand NavigateCommand { get; set; }
public MainPageViewModel(INavigationService navigationService)
{
navigation_service = navigationService;
NavigateCommand = new Command(Navigate);
}
}
}

```

```
}
private void Navigate()
{
    navigation_service.NavigateAsync("SecondPage");
}
public void OnNavigatedFrom(NavigationParameters parameters)
{
}
public void OnNavigatedTo(NavigationParameters parameters)
{
    if (App.Current.MainPage != null)
    {
        SfPopupLayout popup = new SfPopupLayout();
        popup.Show();
    }
}
public void OnNavigatingTo(NavigationParameters parameters)
{
}
}
// SecondPageViewModel.cs
using Prism.Navigation;
using Syncfusion.XForms.PopupLayout;
using System.Windows.Input;
namespace PopupDemo
{
    public class SecondPageViewModel : INavigationAware
    {
        public INavigationService navigation_service;
        public ICommand NavigateCommand { get; set; }
        public SecondPageViewModel(INavigationService navigationService)
        {
            navigation_service = navigationService;
            NavigateCommand = new Command(Navigate);
        }
        private void Navigate()
        {
            navigation_service.NavigateAsync("MainPage");
        }
        public void OnNavigatedFrom(NavigationParameters parameters)
        {
        }
        public void OnNavigatedTo(NavigationParameters parameters)
        {
            if (App.Current.MainPage != null)
            {
                SfPopupLayout popup = new SfPopupLayout();
                popup.Show();
            }
        }
        public void OnNavigatingTo(NavigationParameters parameters)
        {
        }
    }
}
```

Sample Link: You can download the above sample code by clicking [here](#).

### Change the close button icon

You can change the close button icon of the SfPopupLayout, please find the code example of the same below.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:PopupDemo"
xmlns:sfPopup="clr-namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XForms"
x:Class="PopupDemo.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<local:CustomStyle x:Key="customStyle" />
</ResourceDictionary>
</ContentPage.Resources>
<sfPopup:SfPopupLayout x:Name="popupLayout">
<sfPopup:SfPopupLayout.PopupView>
<sfPopup:PopupView PopupStyle="{StaticResource customStyle}"/>
</sfPopup:SfPopupLayout.PopupView>
<sfPopup:SfPopupLayout.Content>
<StackLayout>
<Button x:Name="isOpenButton" Text="Click to open popup"
Clicked="isOpenButton_Clicked"/>
</StackLayout>
</sfPopup:SfPopupLayout.Content>
</sfPopup:SfPopupLayout>
</ContentPage>
```

#### C#

```
namespace PopupDemo
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void isOpenButton_Clicked(object sender, EventArgs e)
        {
            popupLayout.Show();
        }
        public class CustomStyle : PopupStyle
        {
            public CustomStyle()
            {
                this.CloseButtonIcon =
                    ImageSource.FromResource("PopupDemo.Images.RedDown.png");
            }
        }
    }
}
```

```

}
}
}

```

Sample Link: You can download the above sample code by clicking [here](#).

How to close popup view

The [SfPopupLayout.IsOpen](#) or [SfPopupLayout.IsVisible](#) property is used to close the [SfPopupLayout](#)

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Popup_Demo"
x:Class="Popup_Demo.MainPage"
xmlns:sfPopup="clr-
namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XF
orms">
<ContentPage.Content>
<StackLayout x:Name="mainLayout">
<Button x:Name="clickToShowPopup" Text="click to show popup"/>
<Button x:Name="clickToClosePopup" Text="click to close popup"/>
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

### C#

```

using Syncfusion.XForms.PopupLayout;
using System;
using Xamarin.Forms;
namespace Popup_Demo
{
public partial class MainPage : ContentPage
{
SfPopupLayout popupLayout;
public MainPage()
{
InitializeComponent();
this.popupLayout = new SfPopupLayout();
this.popupLayout.StaysOpen = true;
this.popupLayout.PopupView.HeightRequest = 200;
this.popupLayout.PopupView.WidthRequest = 200;
this.clickToShowPopup.Clicked += ClickToShowPopup_Clicked;
this.clickToClosePopup.Clicked += ClickToClosePopup_Clicked;
}
private void ClickToShowPopup_Clicked(object sender, EventArgs e)
{
////// Opens SfPopupLayout.
this.popupLayout.Show(0, 0);
}
private void ClickToClosePopup_Clicked(object sender, EventArgs e)
{
////// Set IsOpen or IsVisible property as false to close the popup view
programmatically.
}
}

```

```

this.popupLayout.IsOpen = false;
this.popupLayout.IsVisible = false;
}
}
}

```

How to change popup view background color

The background color of popup view can be customized by setting the

[SfPopupLayout.PopupView.BackgroundColor](#)

property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Popup_Demo"
x:Class="Popup_Demo.MainPage"
xmlns:sfPopup="clr-
namespace:Syncfusion.XForms.PopupLayout;assembly=Syncfusion.SfPopupLayout.XF
orms">
<ContentPage.Content>
<StackLayout x:Name="mainLayout">
<Button x:Name="clickToShowPopup" Text="click to show popup"/>
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

### C#

```

using Syncfusion.XForms.PopupLayout;
using System;
using Xamarin.Forms;
namespace Popup_Demo
{
    public partial class MainPage : ContentPage
    {
        SfPopupLayout popupLayout;
        DataTemplate dataTemplate;
        public MainPage()
        {
            InitializeComponent();
            this.popupLayout = new SfPopupLayout();
            this.dataTemplate = new DataTemplate(() =>
            {
                Label label = new Label();
                label.Text = "Syncfusion";
                label.BackgroundColor = Color.Transparent;
                return label;
            });
            this.popupLayout.PopupView.HeaderTitle = "Company name";
            this.popupLayout.PopupView.ContentTemplate = dataTemplate;
            //// You can customize your color based on the requirement.
            this.popupLayout.PopupView.BackgroundColor = Color.LightBlue;
            this.clickToShowPopup.Clicked += ClickToShowPopup Clicked;

```

```

}
private void ClickToShowPopup_Clicked(object sender, EventArgs e)
{
    //// Opens SfPopupLayout.
    this.popupLayout.Show();
}
}
}

```

Sample Link: You can download the above sample code by clicking [here](#).

How to prevent the Popup from being closed when pressing the back navigation button

By default, the Popup will be closed whenever the back navigation button is pressed in Android devices.

To prevent the Popup from being closed when the back navigation button is pressed, set **False** to the **SfPopupLayout.ClosePopupOnBackButtonPressed** property.

#### XML

```

<sfPopup:SfPopupLayout x:Name="popUpLayout"
    ClosePopupOnBackButtonPressed="False">
</sfPopup:SfPopupLayout>

```

#### C#

```

popUpLayout.ClosePopupOnBackButtonPressed = false;

```

How to show overlay background always in Xamarin.Forms Popup?

By default, the overlay background will not be shown around the Popup if all the **WidthRequest**, **HeightRequest**, **StartX**, and **StartY** properties are set to **SfPopupLayout.PopupView**. For cases where the overlay background should always be shown around the Popup, set **True** to the **SfPopupLayout.ShowOverlayAlways** property.

#### XML

```

<sfPopup:SfPopupLayout x:Name="popUpLayout" ShowOverlayAlways="True">
</sfPopup:SfPopupLayout>

```

#### C#

```

popUpLayout.ShowOverlayAlways = true;

```

## Progress Bar

### Overview

The progress bar control for Xamarin.Forms provides a customizable visual to indicate the progress of a task.

### Key features

- Visualizes the progress in different shapes, such as rectangle and circle.
- Customizes the ranges with different colors in linear progress bar.

- Customizes the progress and tracks the thickness.
- Displays the custom content at the center of the circular progress bar.
- Visualizes the progress in segments.
- Customizes the angle of the circular progress bar.



## Getting Started

This section explains the steps required to work with the progress bar control for Xamarin.Forms.

### Adding SfProgressBar reference

You can add SfProgressBar reference using one of the following methods:

#### Method 1: Adding SfProgressBar reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfProgressBar). To add SfProgressBar to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfProgressBar](https://www.nuget.org/packages/Syncfusion.Xamarin.SfProgressBar), and then install it.

![Adding SfProgressBar reference from NuGet](overview\_images/Adding SfProgressBar reference.png)

---

**Note:** Install the same version of SfProgressBar NuGet in all the projects.

---

#### Method 2: Adding SfProgressBar reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfProgressBar control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfProgressBar assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfProgressBar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfProgressBar.XForms.Android.dll Syncfusion.SfProgressBar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfProgressBar.XForms.iOS.dll Syncfusion.SfProgressBar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfProgressBar.XForms.UWP.dll Syncfusion.SfProgressBar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the application on each platform with progress bar

To use the progress bar in an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch the progress bar in iOS, call the SfLinearProgressBarRenderer.Init() or SfCircularProgressBarRenderer.Init() in the FinishedLaunching overridden method of the AppDelegate class after the Xamarin.Forms framework has been initialized and before the LoadApplication is called as demonstrated in the following code sample.

#### C#



```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
...
global::Xamarin.Forms.Forms.Init();
// Add the below line if you are using SfLinearProgressBar.
Syncfusion.XForms.iOS.ProgressBar.SfLinearProgressBarRenderer.Init();
// Add the below line if you are using SfCircularProgressBar.
Syncfusion.XForms.iOS.ProgressBar.SfCircularProgressBarRenderer.Init();
LoadApplication(new App());
...
}
```

### Universal Windows Platform (UWP)

To launch the progress bar in UWP, initialize the progress bar assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with progress bar in **Release** mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
...
if (rootFrame == null)
{
List<Assembly> assembliesToInclude = new List<Assembly>();
// Add the below line if you are using SfLinearProgressBar.
assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.ProgressBar.SfLinearPro
gressRenderer).GetTypeInfo().Assembly);
// Add the below line if you are using SfCircularProgressBar.
assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.ProgressBar.SfCircularP
rogressBarRenderer).GetTypeInfo().Assembly);
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
}
...
}
```

### Android

The Android platform does not require any additional configuration to render the progress bar.

#### Initializing the progress bar

Import the progress bar namespace as demonstrated in the following code sample in your respective page.

#### XML

```
xmlns:progressBar="clr-
namespace:Syncfusion.XForms.ProgressBar;assembly=Syncfusion.SfProgressBar.XF
orms"
```

#### C#

```
using Syncfusion.XForms.ProgressBar;
```

The progress bar control has two variants: `SfLinearProgressBar` and `SfCircularProgressBar`. Each renders the progress in its own shape, such as rectangle and circle, respectively. Initialize both the progress bars with a progress value using the `Progress` property as demonstrated in the following code sample.

#### XML

```
<!--Using linear progress bar-->
<progressBar:SfLinearProgressBar Progress="75"/>
<!--Using circular progress bar-->
<progressBar:SfCircularProgressBar Progress="75"/>
```

#### C#

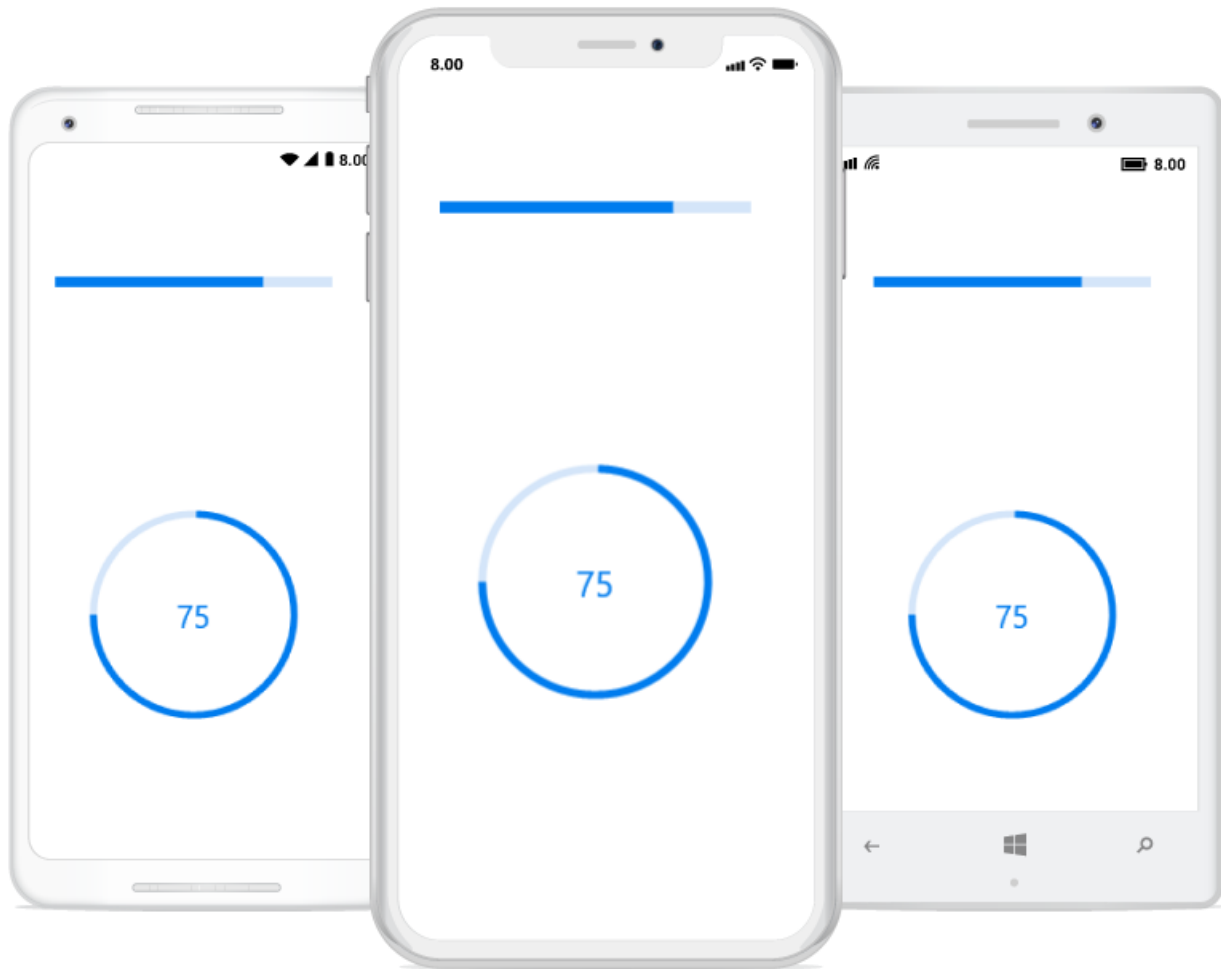
```
// Using linear progress bar.
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar { Progress =
75 };
// Using circular progress bar.
SfCircularProgressBar circularProgressBar = new SfCircularProgressBar {
Progress = 75 };
```

---

**Note:** By default, the value of progress should be specified between 0 and 100. To specify progress value between 0 and 1, specify the `Minimum` property to 0 and `Maximum` property to 1.

---

Run the project, and check if you get following output to make sure that the project has been configured properly to add the progress bar.



### Enabling indeterminate state

When the progress of a task cannot be shown determinately, you can enable the indeterminate state using the [IsIndeterminate](#) property to know that any progress is happening in the background.

#### XML

```
<!--Using linear progress bar-->  
<progressBar:SfLinearProgressBar IsIndeterminate="True"/>  
<!--Using circular progress bar-->  
<progressBar:SfCircularProgressBar IsIndeterminate="True"/>
```

#### C#

```
// Using linear progress bar.  
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar {  
    IsIndeterminate = true };  
// Using circular progress bar.  
SfCircularProgressBar circularProgressBar = new SfCircularProgressBar {  
    IsIndeterminate = true };
```

### Enable segments

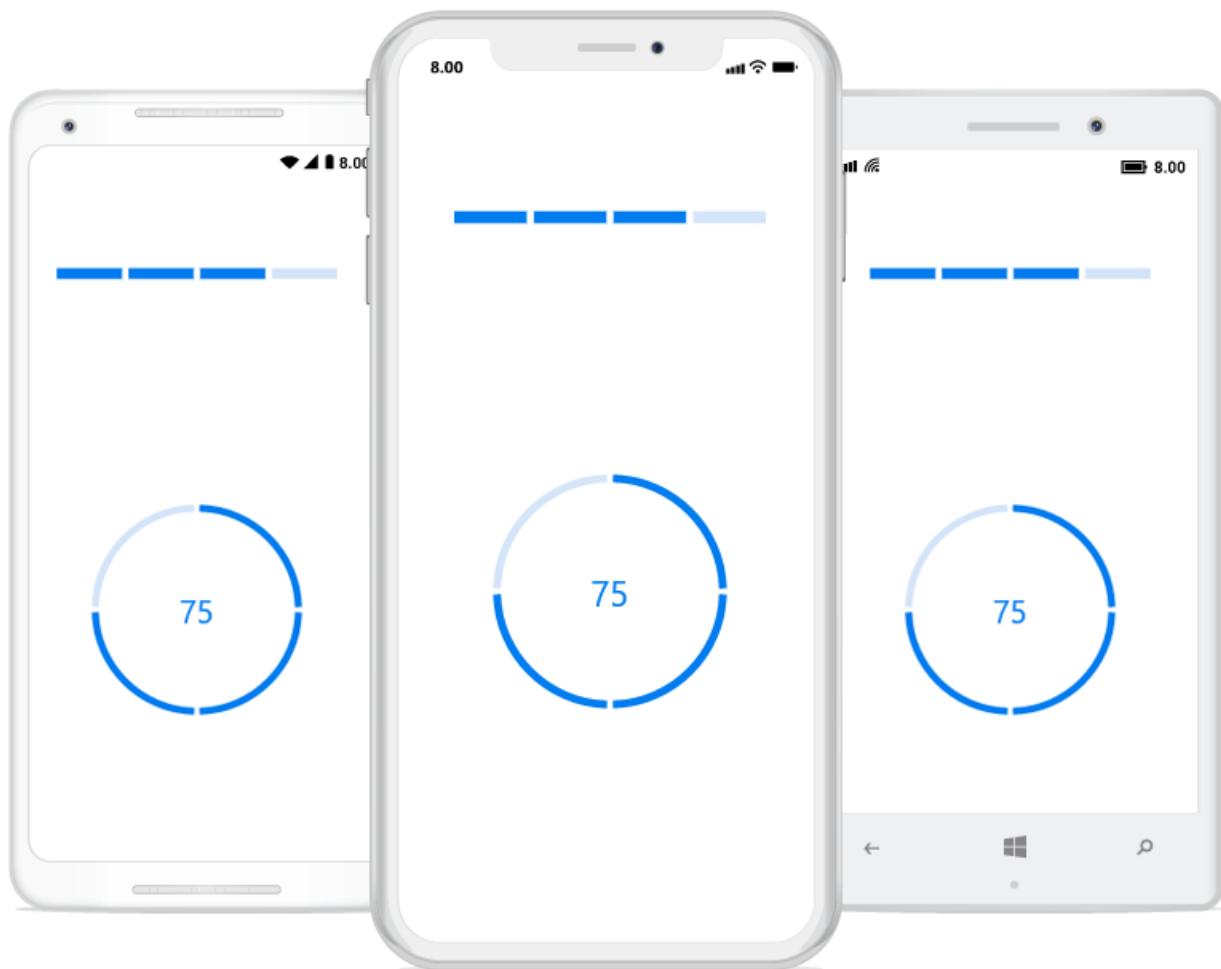
To visualize the progress of a multiple sequential task, split the progress bar into the multiple segments by defining the [SegmentCount](#) property as demonstrated in the following code sample.

#### XML

```
<!--Using linear progress bar-->  
<progressBar:SfLinearProgressBar SegmentCount="4" Progress="75"/>  
<!--Using circular progress bar-->  
<progressBar:SfCircularProgressBar SegmentCount="4" Progress="75"/>
```

#### C#

```
// Using linear progress bar.  
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar { Progress =  
75, SegmentCount = 4 };  
// Using circular progress bar.  
SfCircularProgressBar circularProgressBar = new SfCircularProgressBar {  
Progress = 75, SegmentCount = 4 };
```



## Apply colors

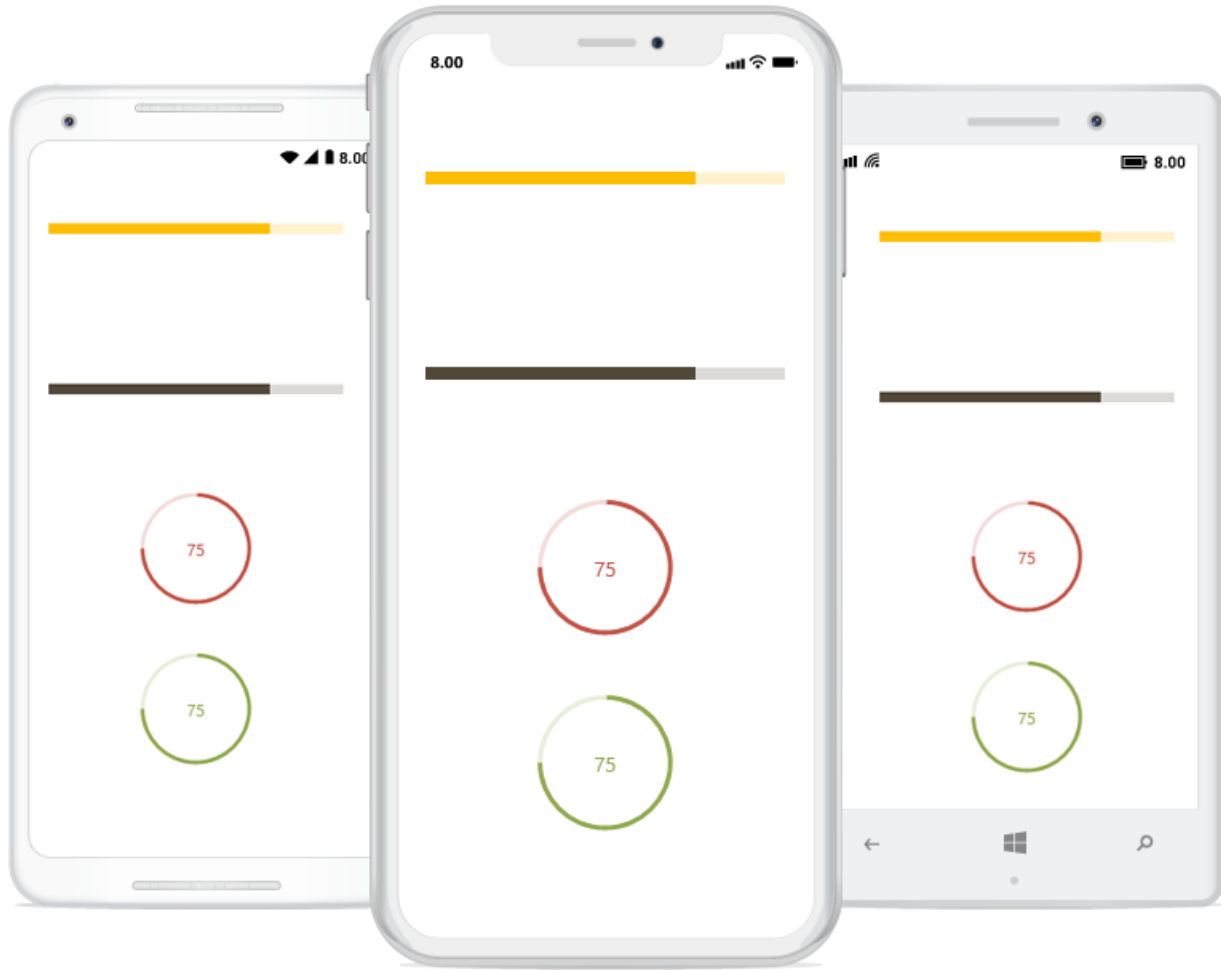
You can customize the color of the progress indicator and track by defining the [ProgressColor](#) and [TrackColor](#) properties, respectively.

### XML

```
<!--Using linear progress bar-->
<progressBar:SfLinearProgressBar Progress="75" TrackColor="#33ffbe06"
ProgressColor="#FFffbe06"/>
<progressBar:SfLinearProgressBar Progress="75" TrackColor="#3351483a"
ProgressColor="#FF51483a"/>
<!--Using circular progress bar-->
<progressBar:SfCircularProgressBar Progress="75" TrackColor="#33c15244"
ProgressColor="#FFc15244"/>
<progressBar:SfCircularProgressBar Progress="75" TrackColor="#3390a84e"
ProgressColor="#FF90a84e"/>
```

### C#

```
// Using linear progress bar.
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar{Progress =
75,TrackColor = Color.FromHex("#33ffbe06"),ProgressColor =
Color.FromHex("#FFffbe06")};
SfLinearProgressBar sfLinearProgressBar = new SfLinearProgressBar{Progress =
75,TrackColor = Color.FromHex("#3351483a"),ProgressColor =
Color.FromHex("#FF51483a")};
// Using circular progress bar.
SfCircularProgressBar circularProgressBar = new
SfCircularProgressBar{Progress = 75,TrackColor =
Color.FromHex("#33c15244"),ProgressColor = Color.FromHex("#FFc15244")};
SfCircularProgressBar sfCircularProgressBar = new
SfCircularProgressBar{Progress = 75,TrackColor =
Color.FromHex("#3390a84e"),ProgressColor = Color.FromHex("#FF90a84e")};
```



You can find the complete getting started sample here: [Getting started](#).

## States

You can configure the states of the progress bar control depending on the usage.

### Determinate

This is the default state. You can use it when the progress estimation is known.

### Indeterminate

By enabling the [IsIndeterminate](#) property, the state of the progress bar can be changed to indeterminate when the progress cannot be estimated or is not being calculated. It can be combined with determinate mode to know that the application is estimating progress before the actual progress starts.

### XML

```
<!--Using linear progress bar-->  
<progressBar:SfLinearProgressBar IsIndeterminate="True"/>  
<!--Using circular progress bar-->  
<progressBar:SfCircularProgressBar IsIndeterminate="True"/>
```

**C#**

```
// Using linear progress bar.
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar {
    IsIndeterminate = true};
// Using circular progress bar.
SfCircularProgressBar circularProgressBar = new SfCircularProgressBar {
    IsIndeterminate = true };
```

**Buffer**

The secondary task's progress can be defined using the [SecondaryProgress](#) property as demonstrated in the following code sample.

**XML**

```
<progressBar:SfLinearProgressBar x:Name="LinearProgressBar" Progress="25"
    SecondaryProgress="75"/>
```

**C#**

```
this.LinearProgressBar.Progress = 75;
this.LinearProgressBar.SecondaryProgress = 25;
```



## Range

Range represents the entire span of the progress bar and can be defined using the [Minimum](#) and [Maximum](#) properties. The default value of the range is 0 to 100.

The following code sample demonstrates how to customize the range as factor value to the progress bar.

### XML

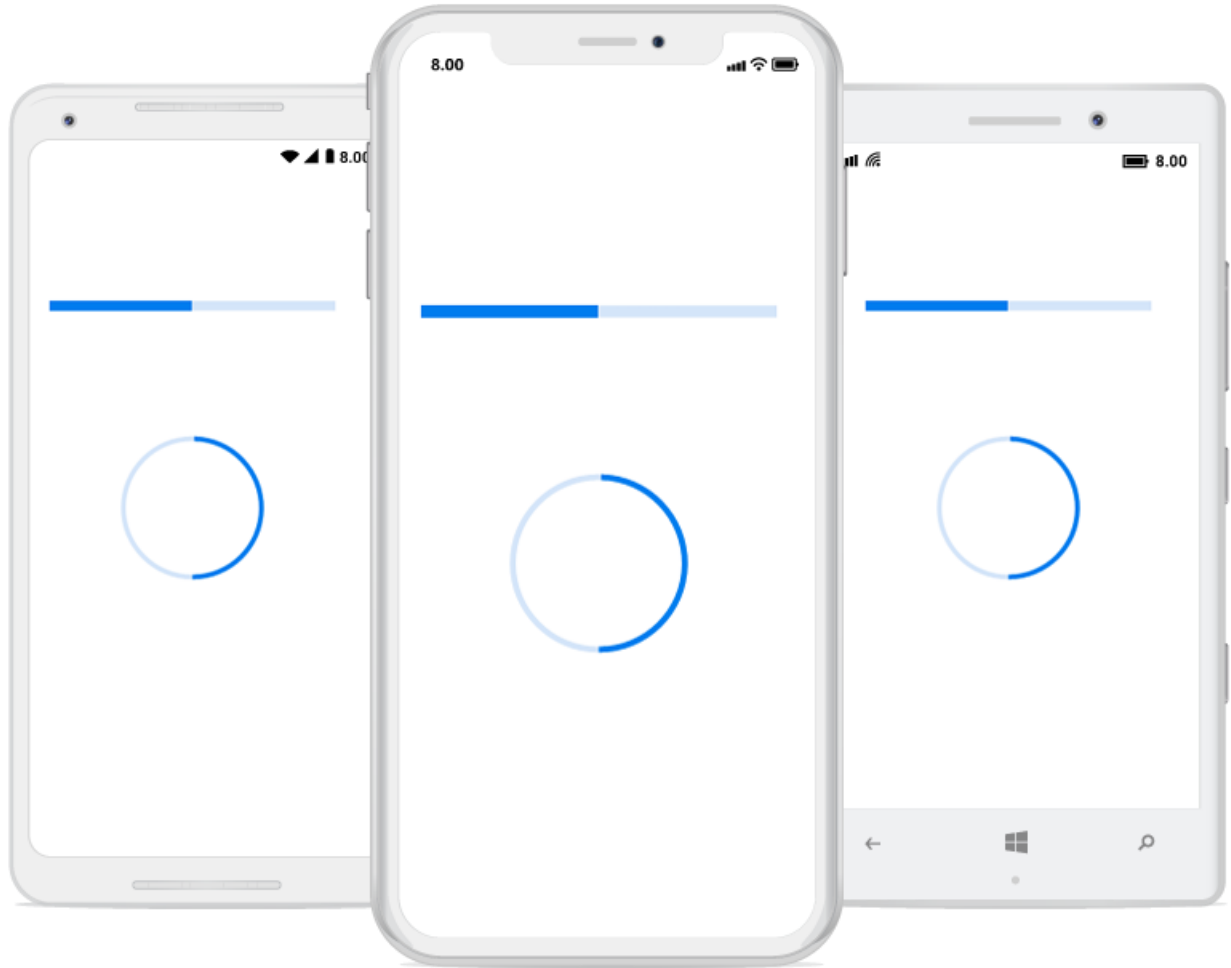
```
<!--Using linear progress bar-->
<progressBar:SfLinearProgressBar x:Name="LinearProgressBar" Minimum="0"
Progress="0.5" Maximum="1"/>
<!--Using circular progress bar-->
<progressBar:SfCircularProgressBar x:Name="CircularProgressBar" Minimum="0"
Progress="0.5" Maximum="1"/>
```

### C#

```
// Using linear progress bar.
this.LinearProgressBar.Minimum = 0;
this.LinearProgressBar.Maximum = 1;
this.LinearProgressBar.Progress = 0.5;
```



```
// Using circular progress bar.  
this.CircularProgressBar.Minimum = 0;  
this.CircularProgressBar.Maximum = 1;  
this.CircularProgressBar.Progress = 0.5;
```



## Custom Content

In the circular progress bar, you can add any view to the center using the [Content](#) property.

For example, you can include add, start, or pause button to control the progress. You can also add an image that indicates the actual task in progress or add custom text that conveys how far the task is completed.

The following code sample demonstrates how to add custom text content.

### XML

```
<progressBar:SfCircularProgressBar x:Name="CustomContentCircularProgressBar"  
Progress="23" HorizontalOptions="Center" >  
  <progressBar:SfCircularProgressBar.Content>  
    <Grid>  
      <Grid.RowDefinitions>  
        <RowDefinition Height="3*" />  
      </Grid.RowDefinitions>  
    </Grid>  
  </progressBar:SfCircularProgressBar.Content>  
</progressBar:SfCircularProgressBar>
```

```

<RowDefinition Height="*" />
</Grid.RowDefinitions>
<Label x:Name="CustomContentProgressBarLabel" Grid.Row="0"
TextColor="#007cee" Text="{Binding Progress,StringFormat='{0}%'}"
BindingContext="{x:Reference CustomContentCircularProgressBar}"
HorizontalTextAlignment="Center" VerticalTextAlignment="End">
</Label>
<Label Grid.Row="1" TextColor="#007cee" Text="used" VerticalOptions="Start"
Margin="0,-5,0,0" HorizontalTextAlignment="Center"
VerticalTextAlignment="Start">
</Label>
</Grid>
</progressBar:SfCircularProgressBar.Content>
</progressBar:SfCircularProgressBar>

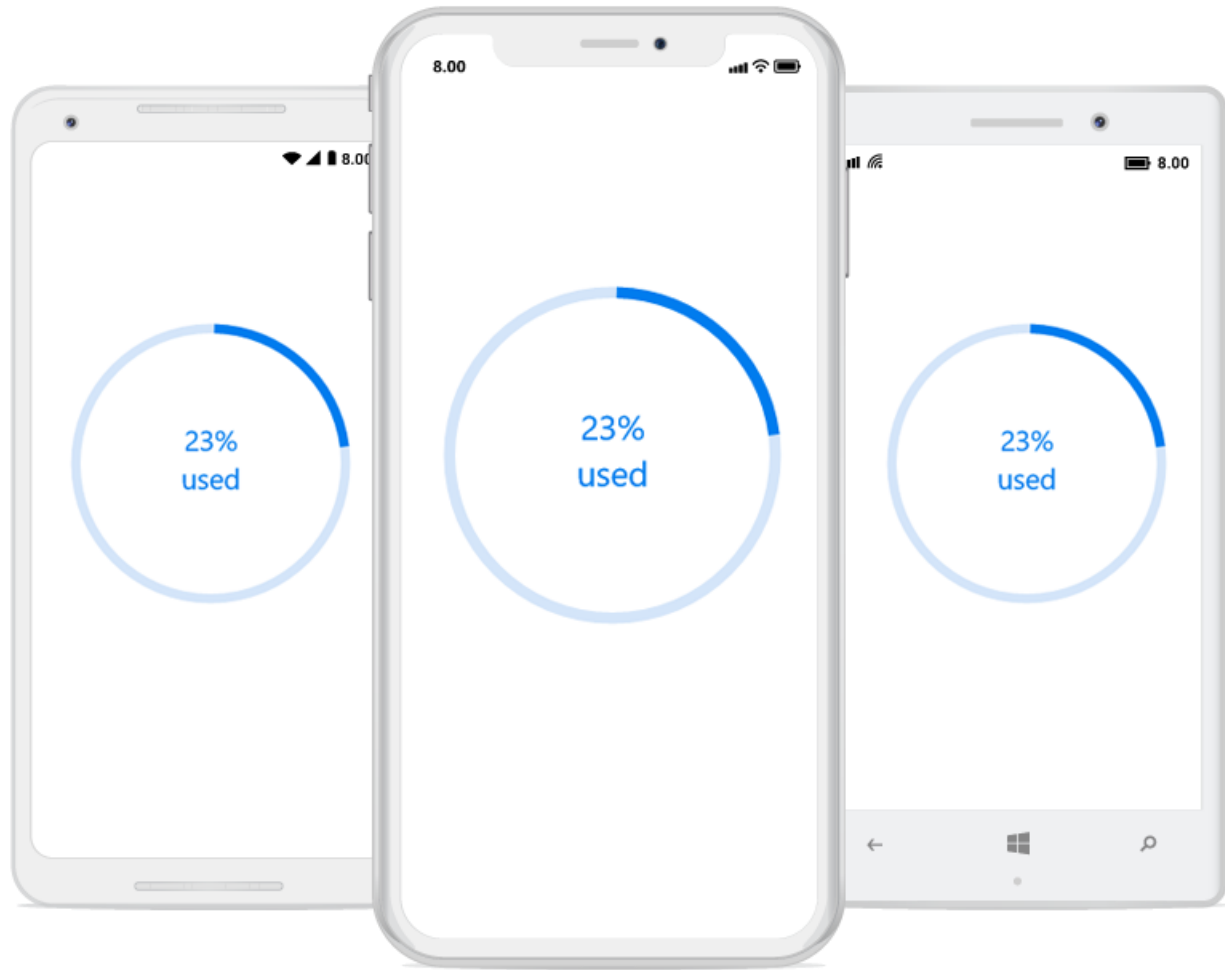
```

**C#**

```

SfCircularProgressBar circularProgressBar = new SfCircularProgressBar();
circularProgressBar.Progress = 23;
Grid grid = new Grid();
grid.RowDefinitions.Add(new RowDefinition() { Height = new GridLength(3,
GridUnitType.Star) });
grid.RowDefinitions.Add(new RowDefinition() { Height = new GridLength(1,
GridUnitType.Star) });
Label label = new Label();
label.BindingContext = circularProgressBar;
Binding binding = new Binding();
binding.Path = "Progress";
binding.StringFormat = "{0}%";
label.SetBinding(Label.TextProperty, binding);
label.HorizontalTextAlignment = TextAlignment.Center;
label.VerticalOptions = LayoutOptions.End;
label.FontSize = 10;
label.TextColor = Color.FromHex("007cee");
Grid.SetRow(label, 0);
grid.Children.Add(label);
Label textLabel = new Label();
textLabel.Text = "used";
textLabel.HorizontalTextAlignment = TextAlignment.Center;
textLabel.VerticalOptions = LayoutOptions.Start;
textLabel.FontSize = 10;
textLabel.TextColor = Color.FromHex("007cee");
Grid.SetRow(textLabel, 1);
grid.Children.Add(textLabel);
circularProgressBar.Content = grid;

```



By default, the progress value will be displayed at the center. You can hide the label in the circular progress bar by setting the [ShowProgressValue](#) property to false.

## Segments

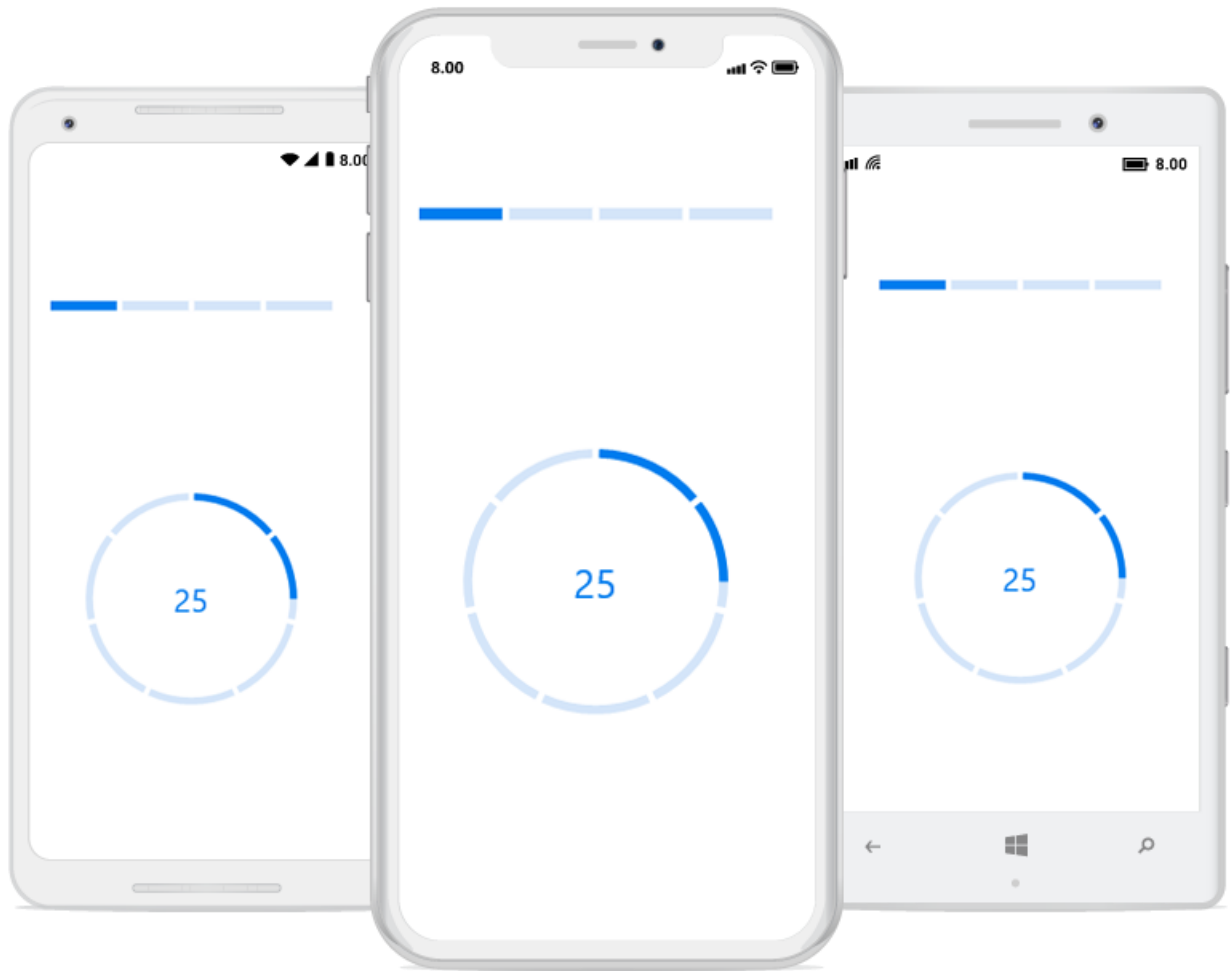
To visualize the progress of multiple sequential tasks, split the progress bar into multiple segments by setting the [SegmentsCount](#) property as demonstrated in the following code sample.

### XML

```
<!--Using linear progress bar-->
<progressBar:SfLinearProgressBar x:Name="LinearProgressBar" Progress="25"
SegmentCount="4" />
<!--Using circular progress bar-->
<progressBar:SfCircularProgressBar x:Name="CircularProgressBar"
Progress="25" SegmentCount="7" />
```

### C#

```
// Using linear progress bar.
this.LinearProgressBar.SegmentCount = 4;
// Using circular progress bar.
this.CircularProgressBar.SegmentCount = 7;
```



### Gap customization

You can also customize the default spacing between the segments using the [GapWidth](#) property as demonstrated in following code sample.

#### XML

```
<!--Using linear progress bar-->
<progressBar:SfLinearProgressBar x:Name="LinearProgressBar" Progress="25"
SegmentCount="4" GapWidth="5" />
<!--Using circular progress bar-->
<progressBar:SfCircularProgressBar x:Name="CircularProgressBar"
Progress="25" SegmentCount="7" GapWidth="10" />
```

#### C#

```
// Using linear progress bar.
this.LinearProgressBar.GapWidth = 5;
// Using circular progress bar.
this.CircularProgressBar.GapWidth = 10;
```



## Appearance

### Angle

The appearance of the circular progress bar can be customized to semi-circle, arc, etc. The start and end angles can be customized using the [StartAngle](#) and [EndAngle](#) properties.

The following code sample demonstrates how to change the appearance of the circular progress bar to semi-circle.

### XML

```
<progressBar:SfCircularProgressBar x:Name="CircularProgressBar"
    Progress="75" StartAngle="180" EndAngle="360" />
```

### C#

```
SfCircularProgressBar circularProgressBar = new SfCircularProgressBar();
circularProgressBar.Progress = 75;
circularProgressBar.StartAngle = 180;
circularProgressBar.EndAngle = 360;
```



### Range colors

You can visualize the multiple ranges with different colors that are mapped to each range to enhance the readability of progress.

The colors can be mapped to the specific ranges using the [RangeColors](#) property, which holds a collection of [RangeColor](#).

The following properties in the [RangeColor](#) are used to map the colors to a range:

- [Color](#): Represents the color to the specified range.
- [Start](#): Represents the start range of the color.
- [End](#): Represents the end range of the color.
- [IsGradient](#): Represents whether the gradient effect is applied to the color.

The following code sample demonstrates how to map the solid color range in the progress bar.

### XML

```
<!--Using linear progress bar-->  
<progressBar:SfLinearProgressBar Progress="100">  
  <progressBar:SfLinearProgressBar.RangeColors>  
    <progressBar:RangeColorCollection>
```

```

<progressBar:RangeColor Color="#00bdaf" Start="0" End="25"/>
<progressBar:RangeColor Color="#2f7ecc" Start="25" End="50"/>
<progressBar:RangeColor Color="#e9648e" Start="50" End="75"/>
<progressBar:RangeColor Color="#fbb78a" Start="75" End="100"/>
</progressBar:RangeColorCollection>
</progressBar:SfLinearProgressBar.RangeColors>
</progressBar:SfLinearProgressBar>
<!--Using circular progress bar-->
<progressBar:SfCircularProgressBar Progress="100" ProgressColor="#FF90a84e">
<progressBar:SfCircularProgressBar.RangeColors>
<progressBar:RangeColorCollection>
<progressBar:RangeColor Color="#00bdaf" Start="0" End="25"/>
<progressBar:RangeColor Color="#2f7ecc" Start="25" End="50"/>
<progressBar:RangeColor Color="#e9648e" Start="50" End="75"/>
<progressBar:RangeColor Color="#fbb78a" Start="75" End="100"/>
</progressBar:RangeColorCollection>
</progressBar:SfCircularProgressBar.RangeColors>
</progressBar:SfCircularProgressBar>

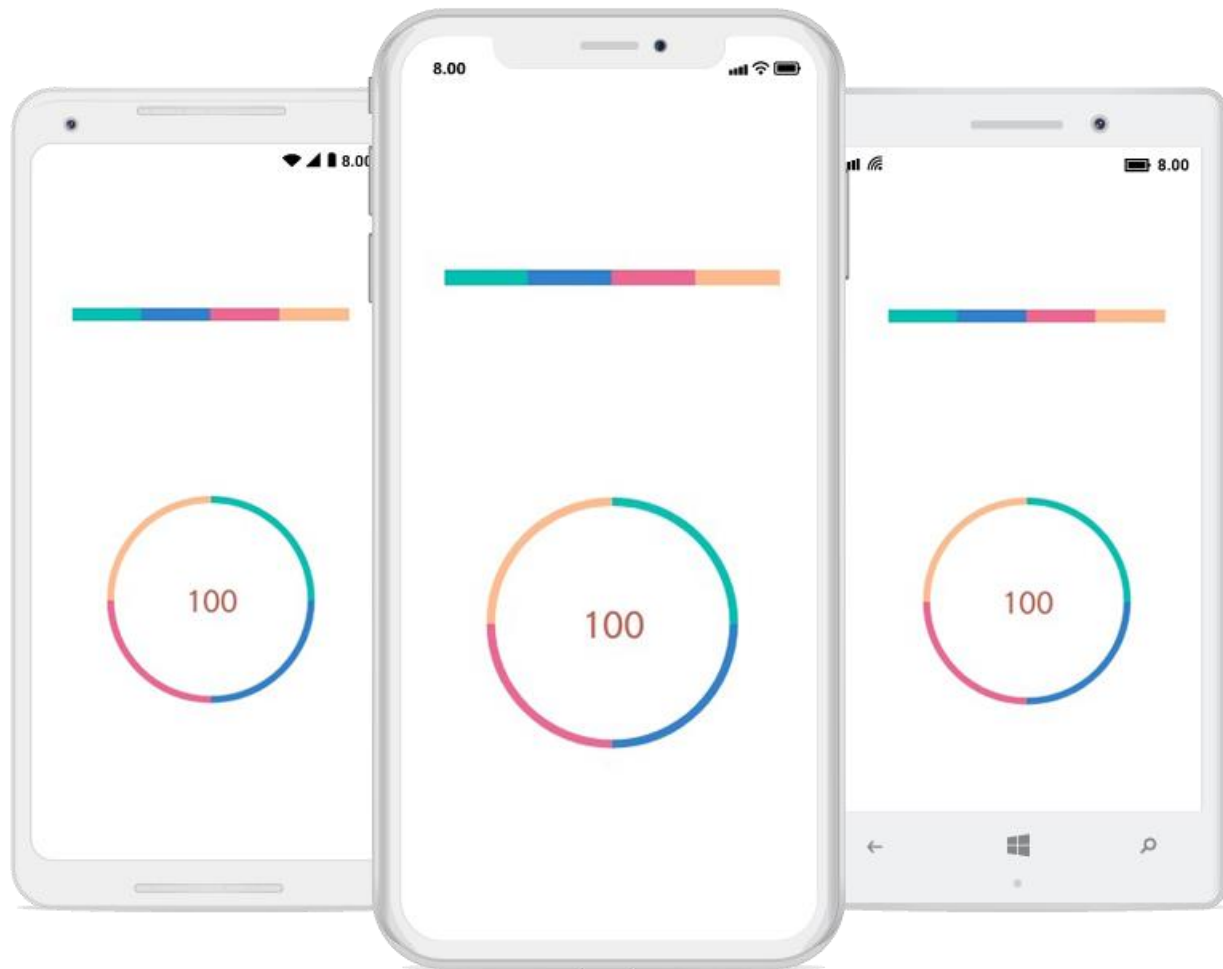
```

**C#**

```

// Using linear progress bar.
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar();
linearProgressBar.Progress = 100;
RangeColorCollection rangeColors = new RangeColorCollection();
rangeColors.Add(new RangeColor() { Color = Color.FromHex("00bdaf"), Start =
0, End = 25 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("2f7ecc"), Start =
25, End = 50 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("e9648e"), Start =
50, End = 75 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("fbb78a"), Start =
75, End = 100 });
linearProgressBar.RangeColors = rangeColors;
// Using circular progress bar.
SfCircularProgressBar circularProgressBar = new SfCircularProgressBar();
circularProgressBar.Progress = 100;
circularProgressBar.ProgressColor = Color.FromHex("FF90a84e");
RangeColorCollection rangeColors = new RangeColorCollection();
rangeColors.Add(new RangeColor() { Color = Color.FromHex("00bdaf"), Start =
0, End = 25 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("2f7ecc"), Start =
25, End = 50 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("e9648e"), Start =
50, End = 75 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("fbb78a"), Start =
75, End = 100 });
circularProgressBar.RangeColors = rangeColors;

```



The following code sample demonstrates how to apply gradient transition effect to the range colors in the linear progress bar.

#### XML

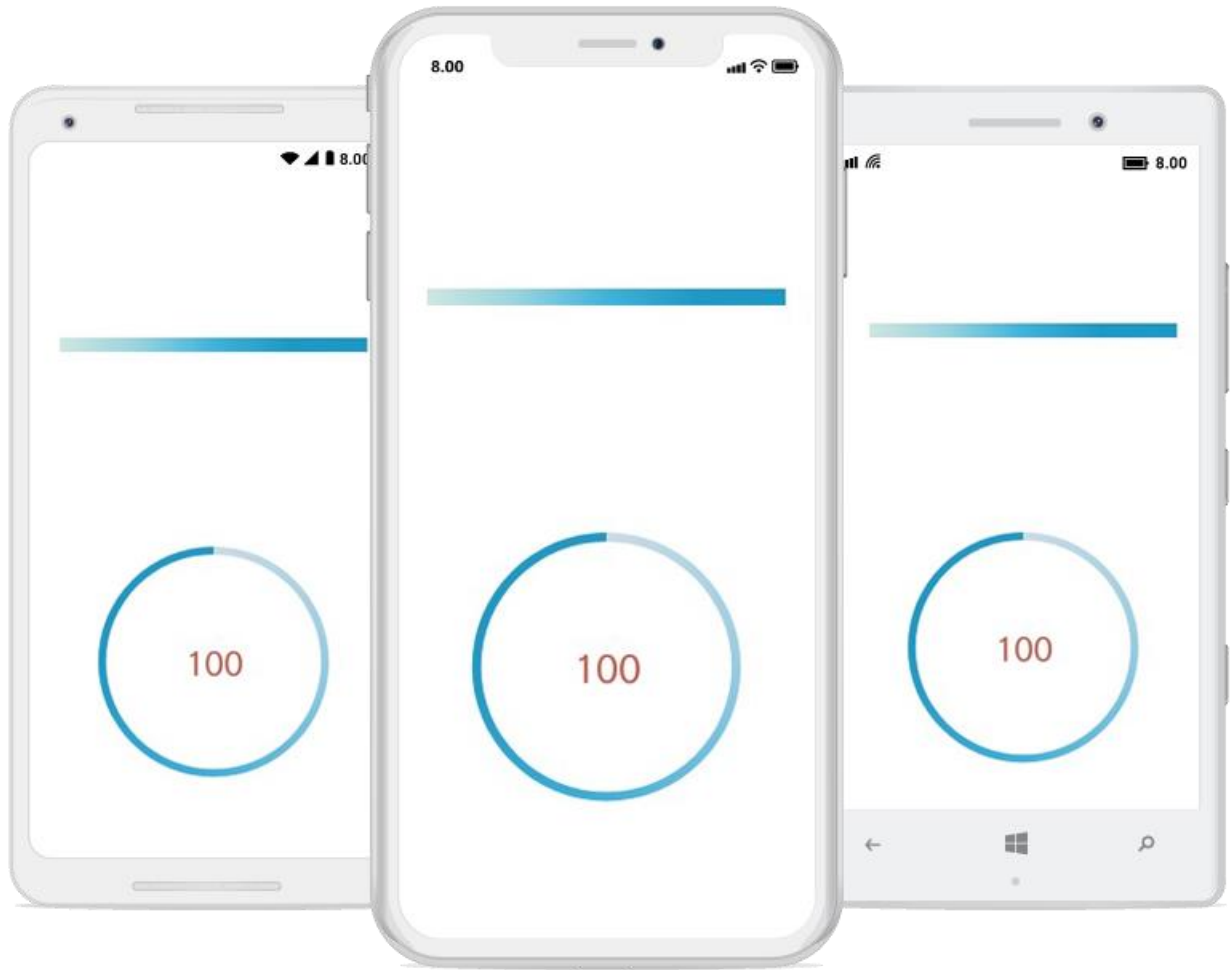
```
<progressBar:SfLinearProgressBar Progress="100" >
  <progressBar:SfLinearProgressBar.RangeColors>
    <progressBar:RangeColorCollection>
      <progressBar:RangeColor IsGradient="True" Color="#88A0D9EF" Start="0"
        End="25"/>
      <progressBar:RangeColor IsGradient="True" Color="#AA62C1E5" Start="25"
        End="50"/>
      <progressBar:RangeColor IsGradient="True" Color="#DD20A7DB" Start="50"
        End="75"/>
      <progressBar:RangeColor IsGradient="True" Color="#FF1C96C5" Start="75"
        End="100"/>
    </progressBar:RangeColorCollection>
  </progressBar:SfLinearProgressBar.RangeColors>
</progressBar:SfLinearProgressBar>
<progressBar:SfCircularProgressBar Progress="100" ProgressColor="#FF90a84e">
  <progressBar:SfCircularProgressBar.RangeColors>
    <progressBar:RangeColorCollection>
```



```
<progressBar:RangeColor IsGradient="True" Color="#88A0D9EF" Start="0"
End="25"/>
<progressBar:RangeColor IsGradient="True" Color="#AA62C1E5" Start="25"
End="50"/>
<progressBar:RangeColor IsGradient="True" Color="#DD20A7DB" Start="50"
End="75"/>
<progressBar:RangeColor IsGradient="True" Color="#FF1C96C5" Start="75"
End="100"/>
</progressBar:RangeColorCollection>
</progressBar:SfCircularProgressBar.RangeColors>
</progressBar:SfCircularProgressBar>
```

**C#**

```
// Using linear progress bar.
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar();
linearProgressBar.Progress = 100;
RangeColorCollection rangeColors = new RangeColorCollection();
rangeColors.Add(new RangeColor() { Color = Color.FromHex("88A0D9EF"),
IsGradient = true, Start = 0, End = 25 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("AA62C1E5"),
IsGradient = true, Start = 25, End = 50 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("DD20A7DB"),
IsGradient = true, Start = 50, End = 75 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("FF1C96C5"),
IsGradient = true, Start = 75, End = 100 });
linearProgressBar.RangeColors = rangeColors;
// Using circular progress bar.
SfCircularProgressBar circularProgressBar = new SfCircularProgressBar();
circularProgressBar.Progress = 100;
circularProgressBar.ProgressColor = Color.FromHex("FF90a84e");
RangeColorCollection rangeColors = new RangeColorCollection();
rangeColors.Add(new RangeColor() { Color = Color.FromHex("88A0D9EF"),
IsGradient = true, Start = 0, End = 25 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("AA62C1E5"),
IsGradient = true, Start = 25, End = 50 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("DD20A7DB"),
IsGradient = true, Start = 50, End = 75 });
rangeColors.Add(new RangeColor() { Color = Color.FromHex("FF1C96C5"),
IsGradient = true, Start = 75, End = 100 });
circularProgressBar.RangeColors = rangeColors;
```



### Thickness

#### *Linear progress bar*

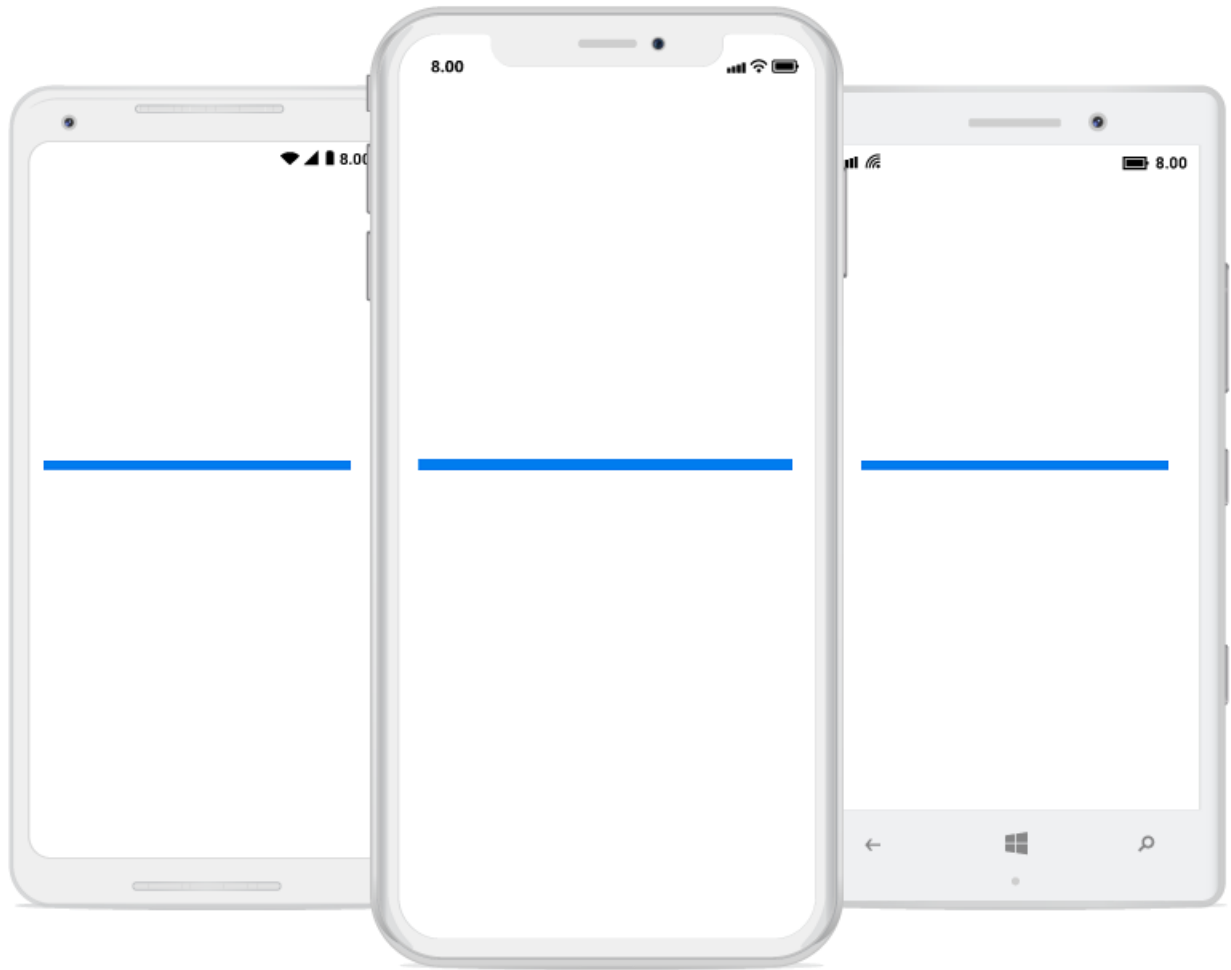
In the linear progress bar, the height of the track and padding of the progress indicator can be customized using the [TrackHeight](#) and [Padding](#) properties, respectively.

#### **XML**

```
<progressBar:SfLinearProgressBar Progress="100" TrackHeight="10"
Padding="2">
</progressBar:SfLinearProgressBar>
```

#### **C#**

```
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar();
linearProgressBar.Progress = 100;
linearProgressBar.TrackHeight = 10;
linearProgressBar.Padding = 2;
```



### *Circular progress bar*

The following properties are used to customize the appearance of the circular progress bar:

- [IndicatorOuterRadius](#): Defines the outer radius of the progress indicator.
- [IndicatorInnerRadius](#): Defines the inner radius of the progress indicator.
- [TrackOuterRadius](#): Defines the outer radius of the track indicator.
- [TrackInnerRadius](#): Defines the inner radius of the track indicator.

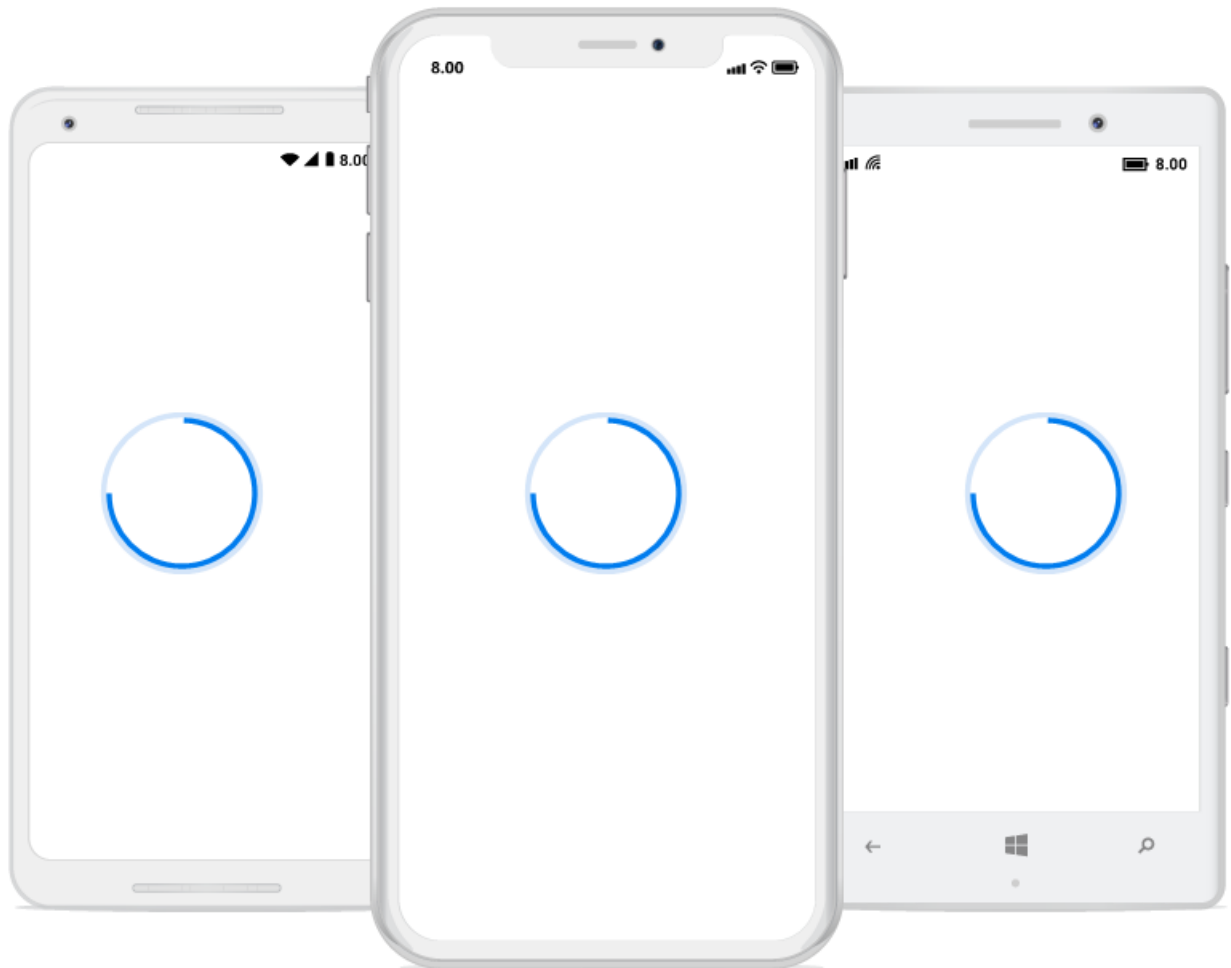
The following code sample demonstrates how to customize the appearance of circular progress bar.

### **XML**

```
<!--Circular progress bar with radius customization -->
<progressBar:SfCircularProgressBar x:Name="TrackOutsideProgressBar"
Grid.Column="0" Grid.Row="0"
Progress="75" Margin="0,10,0,0" IndicatorOuterRadius="0.7"
IndicatorInnerRadius="0.65" ShowProgressValue="False">
</progressBar:SfCircularProgressBar>
```

### **C#**

```
SfCircularProgressBar trackOutsideProgressBar = new SfCircularProgressBar();  
trackOutsideProgressBar.Progress = 75;  
trackOutsideProgressBar.IndicatorOuterRadius = 0.7;  
trackOutsideProgressBar.IndicatorInnerRadius = 0.65;  
trackOutsideProgressBar.ShowProgressValue = false;
```



### Corner radius

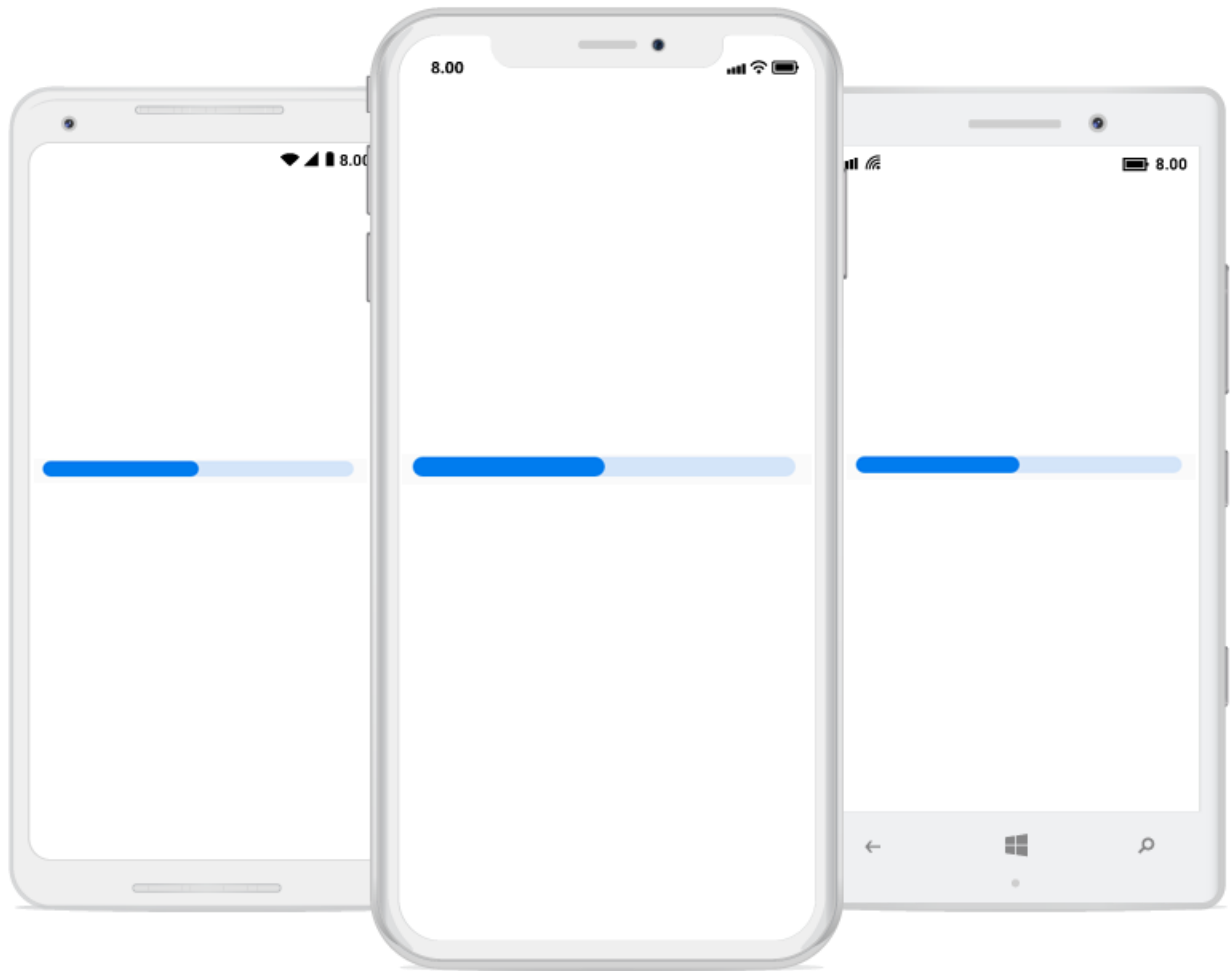
The [CornerRadius](#) property is used to customize the rounded edges in the linear progress bar as demonstrated in the following code sample.

#### XML

```
<progressBar:SfLinearProgressBar Progress="50" TrackHeight="10"  
CornerRadius="10">  
</progressBar:SfLinearProgressBar>
```

#### C#

```
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar();  
linearProgressBar.Progress = 50;  
linearProgressBar.CornerRadius = 10;
```



### Color customization

The following properties are used to customize the color in the progress bar:

- [ProgressColor](#): Represents the color of the progress indicator.
- [TrackColor](#): Represents the color of the track indicator.

The following code sample demonstrates the color customization in progress and track indicator.

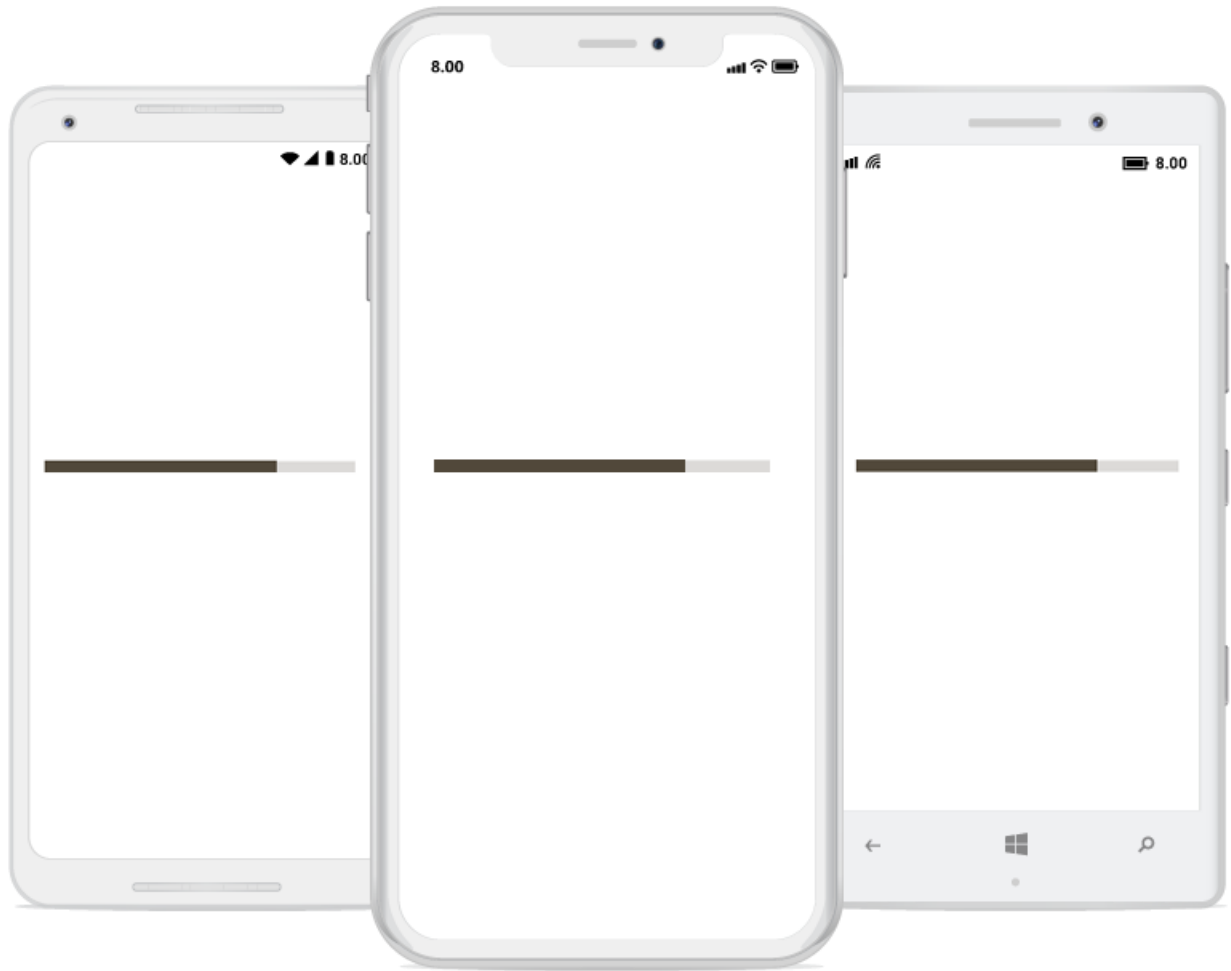
#### XML

```
<progressBar:SfLinearProgressBar Progress="75" TrackColor="#3351483a"
ProgressColor="#FF51483a">
</progressBar:SfLinearProgressBar>
```

#### C#

```
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar();
linearProgressBar.Progress = 75;
linearProgressBar.ProgressColor = Color.FromHex("FF51483a");
```

```
linearProgressBar.TrackColor = Color.FromHex("3351483a");
```



The linear progress bar provides support to customize the color for the secondary progress bar using the [SecondaryProgressColor](#) property as demonstrated in the following code sample.

#### XML

```
<progressBar:SfLinearProgressBar Progress="25" SecondaryProgress="75"  
SecondaryProgressColor="CornflowerBlue"></progressBar:SfLinearProgressBar>
```

#### C#

```
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar();  
linearProgressBar.Progress = 25;  
linearProgressBar.SecondaryProgress = 75;  
linearProgressBar.SecondaryProgressColor = Color.CornflowerBlue;
```



## Animation

The progress bar provides animation support to visualize the progress value changes in an interactive way.

The following properties are used to define the duration of animation for the specific states:

- [AnimationDuration](#): Represents animation duration of the determinate state's progress indicator.
- [SecondaryAnimationDuration](#): Represents animation duration of the determinate state's secondary progress indicator.
- [IndeterminateAnimationDuration](#): Represents animation duration of the indeterminate state's indicator.

## Easing effects

The [EasingEffect](#) property allows you specify the transfer function that controls animation speed when they run.

The following code sample demonstrates the [CubicInOut](#) easing function of the linear progress bar.

## XML

```
<progressBar:SfLinearProgressBar Progress="75" EasingEffect="CubicInOut">
</progressBar:SfLinearProgressBar>
```

**C#**

```
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar();
linearProgressBar.Progress = 75;
linearProgressBar.EasingEffect = EasingEffects.CubicInOut;
```

The [SetProgress\(\)](#) method in the progress bar is used to set progress value along with animation duration and easing effect applicable for the specific method call.

**C#**

```
void SetProgress(double progress, int animationDuration, Easing
easingEffect)
```

**Note:** The animationDuration and easingEffect parameters will not affect the configuration of the [AnimationDuration](#) and [EasingEffect](#) properties.

## Indeterminate Easing Effects

The [IndeterminateEasingEffect](#) property allows you to specify a transfer function for indeterminate state, which controls animation speed when they run.

The following code sample demonstrates the [BounceIn](#) easing function of the linear progress bar.

**XML**

```
<progressBar:SfLinearProgressBar IsIndeterminate="True"
IndeterminateEasingEffect="BounceIn">
</progressBar:SfLinearProgressBar>
```

**C#**

```
SfLinearProgressBar linearProgressBar = new SfLinearProgressBar();
linearProgressBar.IsIndeterminate = true;
linearProgressBar.IndeterminateEasingEffect =
IndeterminateEasingEffects.BounceIn;
```

## Events

## ValueChanged

This event is triggered when the progress value is changed. This event contains the following event argument.

- [Progress](#): Represents the progress value.

The following code sample demonstrates how to customize the color of a progress indicator based on progress using this event.

**XML**



```
<progressBar:SfLinearProgressBar Progress="100" x:Name="LinearProgressBar"
ValueChanged="ProgressBarBase_OnValueChanged">
</progressBar:SfLinearProgressBar>
```

**C#**

```
private void ProgressBarBase_OnValueChanged(object sender,
ProgressValueEventArgs e)
{
    if (e.Progress < 50)
    {
        this.LinearProgressBar.ProgressColor = Color.Red;
    }
    else if (e.Progress >= 50)
    {
        this.LinearProgressBar.ProgressColor = Color.Green;
    }
}
```

**ProgressCompleted**

This event is triggered when the progress attains the [Maximum](#) value. This event contains the following argument.

- [Progress](#): Represents the progress value.

The following code sample demonstrates how to customize the progress bar when progress reaches maximum using this event.

**XML**

```
<progressBar:SfCircularProgressBar x:Name="CircularProgressBar"
Minimum="100" Maximum="500" Progress="500"
ProgressCompleted="ProgressBarBase_OnProgressCompleted" >
<progressBar:SfCircularProgressBar.Content>
<Label Text="Start" FontSize="15" x:Name="Label"></Label>
</progressBar:SfCircularProgressBar.Content>
</progressBar:SfCircularProgressBar>
```

**C#**

```
private void ProgressBarBase_OnProgressCompleted(object sender,
ProgressValueEventArgs e)
{
    if (e.Progress.Equals(this.CircularProgressBar.Maximum))
    {
        // Changed the label text when progress reaches maximum value.
        this.Label.Text = "Completed";
    }
}
```

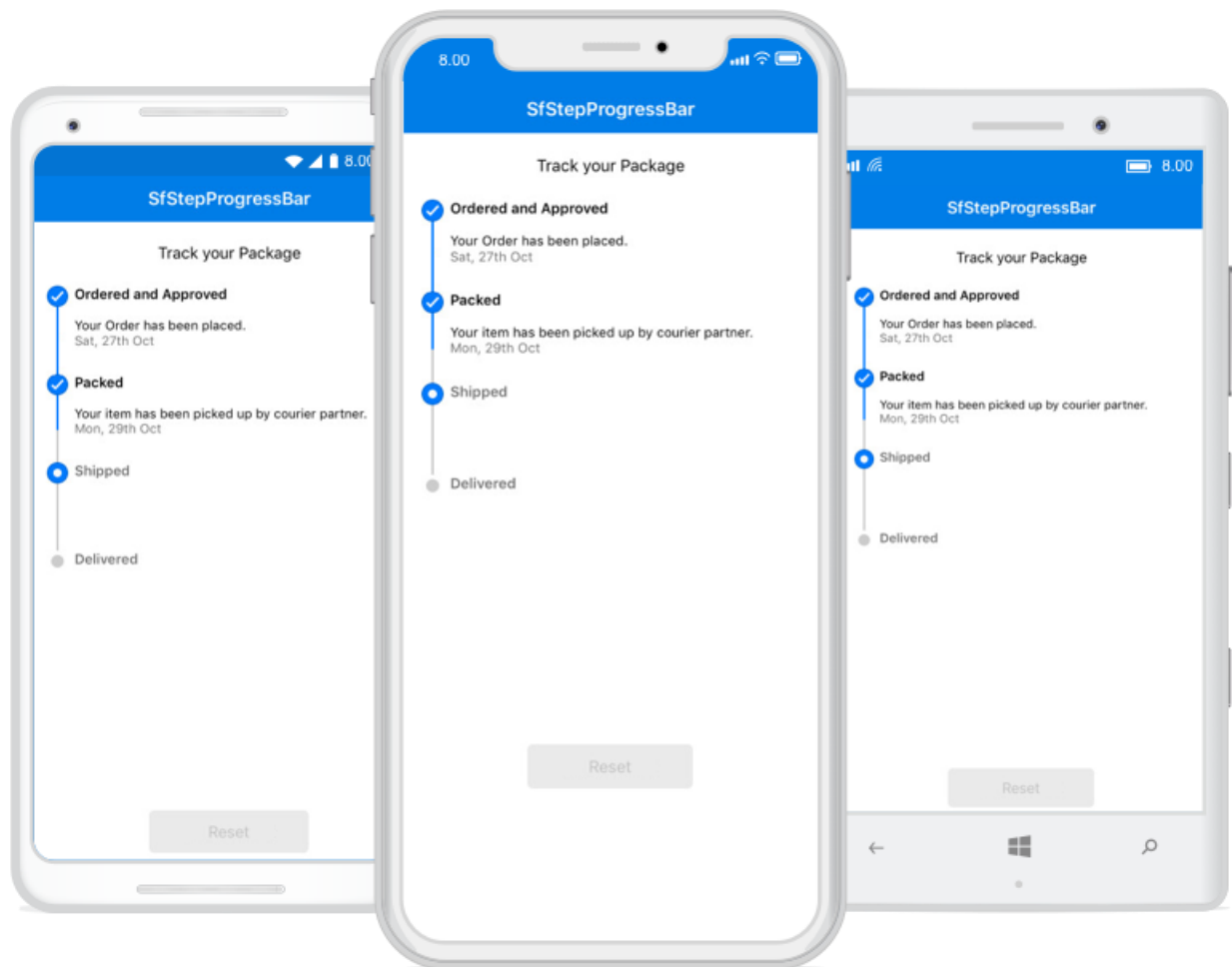
## Step Progress Bar

### Overview

The Xamarin StepProgressBar is a control that indicates the progress of a multiple step (state) process. This control can be used to track the progress of an online purchase, new user registration, live location tracking of buses, trains, flights, and more.

### Key features

- Display the progress in horizontal or vertical orientation.
- Provides three-state progress indication: Not Started, In Progress, and Completed.
- Customize the shape of the state or step as a circle or a square.
- Add descriptions to each state on both sides.
- Customize the step content with numbers, ticks, crosses, dots, or images.
- Allows initializing the state and progress value.



### Getting Started

This section explains the steps required to work with the StepProgressBar control for Xamarin.Forms.

### Adding SfStepProgressBar reference

You can add SfStepProgressBar reference using one of the following methods:

#### Method 1: Adding SfStepProgressBar reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfProgressBar). To add SfStepProgressBar to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfProgressBar](https://www.nuget.org/packages/Syncfusion.Xamarin.SfProgressBar), and then install it.

![Adding SfStepProgressBar reference from NuGet](overview\_images/Adding SfStepProgressBar reference.png)

---

**Note:** Install the same version of SfProgressBar NuGet in all the projects.

---

#### Method 2: Adding SfStepProgressBar reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the StepProgressBar control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfStepProgressBar assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from the NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfProgressBar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfProgressBar.XForms.Android.dll Syncfusion.SfProgressBar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfProgressBar.XForms.iOS.dll Syncfusion.SfProgressBar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfProgressBar.XForms.UWP.dll Syncfusion.SfProgressBar.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To learn more about obtaining Syncfusion components, refer to [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Refer the [Syncfusion license](#)

[key](#) to learn about registering Syncfusion license key in your Xamarin application to use Syncfusion components.

### Launching the application on each platform with StepProgressBar

To use the StepProgressBar in an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, this step is not required.

#### iOS

To launch the StepProgressBar in iOS, call the 'SfLinearProgressBarRenderer.Init()' and 'SfBorderRenderer.Init()' in the FinishedLaunching overridden method of the AppDelegate class after the Xamarin.Forms framework has been initialized and before the LoadApplication is called as demonstrated in the following code sample.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init();
    // Add the below line for using SfLinearProgressBar.
    Syncfusion.XForms.iOS.ProgressBar.SfLinearProgressBarRenderer.Init();
    // Add the below line for using SfBorder.
    Syncfusion.XForms.iOS.Border.SfBorderRenderer.Init();
    LoadApplication(new App());
    ...
}
```

#### Universal Windows Platform (UWP)

To launch the StepProgressBar in UWP, initialize the progress bar assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with progress bar in **Release** mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    if (rootFrame == null)
    {
        List<Assembly> assembliesToInclude = new List<Assembly>();
        // Add the below line for using SfLinearProgressBar.
        assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.ProgressBar.SfLinearProgressRenderer).GetTypeInfo().Assembly);
        // Add the below line for using SfBorder.
        assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.Border.SfBorderRenderer).GetTypeInfo().Assembly);
        Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    }
    ...
}
```

### Android

The Android platform does not require any additional configuration to render the `StepProgressBar`.

### Initializing the `StepProgressBar`

Import the progress bar namespace as demonstrated in the following code sample in your respective page.

#### XML

```
xmlns:progressBar="clr-namespace:Syncfusion.XForms.ProgressBar;assembly=Syncfusion.SfProgressBar.XForms"
```

#### C#

```
using Syncfusion.XForms.ProgressBar;
```

Then, initialize the `SfStepProgressBar` as shown in the following code:

#### XML

```
<Grid>
<progressBar:SfStepProgressBar HorizontalOptions="Center"
VerticalOptions="Center">
<progressBar:StepView PrimaryText="Step 1" />
<progressBar:StepView PrimaryText="Step 2" />
<progressBar:StepView PrimaryText="Step 3" />
<progressBar:StepView PrimaryText="Step 4" Status="InProgress" />
<progressBar:StepView PrimaryText="Step 5" />
</progressBar:SfStepProgressBar>
</Grid>
```

#### C#

```
public MainPage()
{
    InitializeComponent();
    Grid mainGrid = new Grid();
    // Create StepProgressBar control
    SfStepProgressBar stepProgressBar = new SfStepProgressBar();
    stepProgressBar.VerticalOptions = LayoutOptions.Center;
    stepProgressBar.HorizontalOptions = LayoutOptions.Center;
    stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 1" });
    stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 2" });
    stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 3" });
    stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 4", Status
= StepStatus.InProgress });
    stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 5" });
    mainGrid.Children.Add(stepProgressBar);
    this.Content = mainGrid;
}
```

The complete Getting Started sample is available in this [link](#).

## Orientation

The StepProgressBar control provides options to change the default orientation, so a multi-step process can be visualized in horizontal or vertical orientation.

### Horizontal

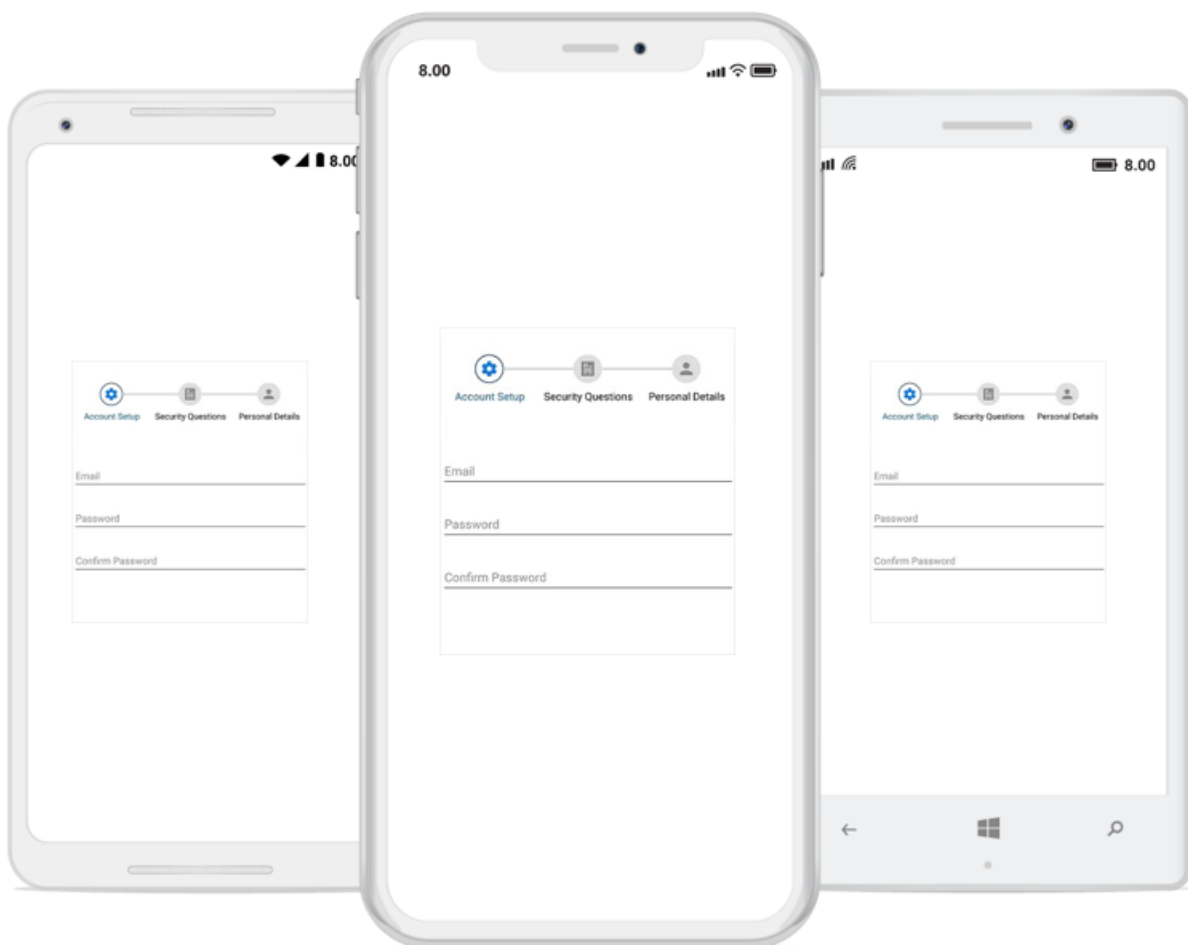
By default, StepProgressBar step views are displayed horizontally. You can also define the orientation as demonstrated in the following code example.

#### XML

```
<progressBar:SfStepProgressBar Orientation="Horizontal" />
```

#### C#

```
SfStepProgressBar stepProgress = new SfStepProgressBar();  
stepProgress.Orientation = StepOrientation.Horizontal;
```



### Vertical

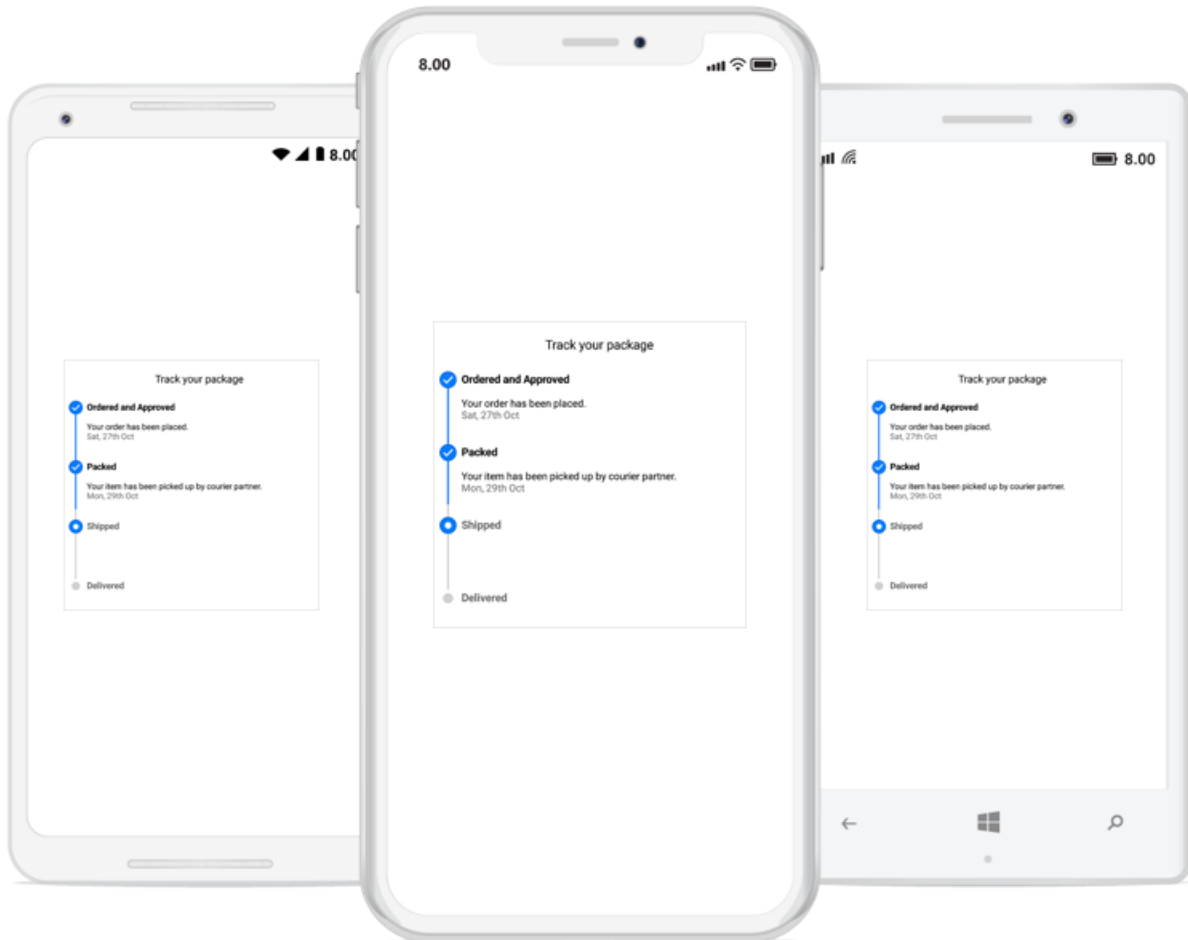
To view the step progress bar control vertically, you can define the vertical orientation as demonstrated in the following code example.

**XML**

```
<progressBar:SfStepProgressBar Orientation="Vertical" />
```

**C#**

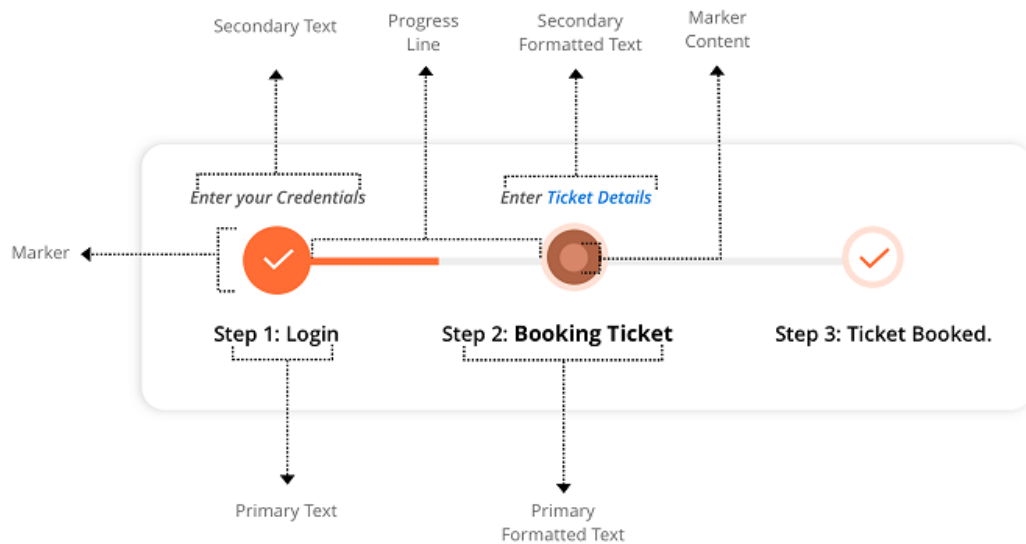
```
SfStepProgressBar stepProgress = new SfStepProgressBar();  
stepProgress.Orientation = StepOrientation.Vertical;
```

**Description**

Each step in a multi-step process has a different operation. To provide self-explanatory information about a step, description can be shown on either side. A primary description will be on the right or bottom of the step, and a secondary description will be on the left or top of the step.

**StepProgressBar overview**

The following overview image illustrates the major elements presented in StepProgressBar.



### Text

The primary and secondary description for a step view can be set using the `PrimaryText` and `SecondaryText` properties as demonstrated in the following code example.

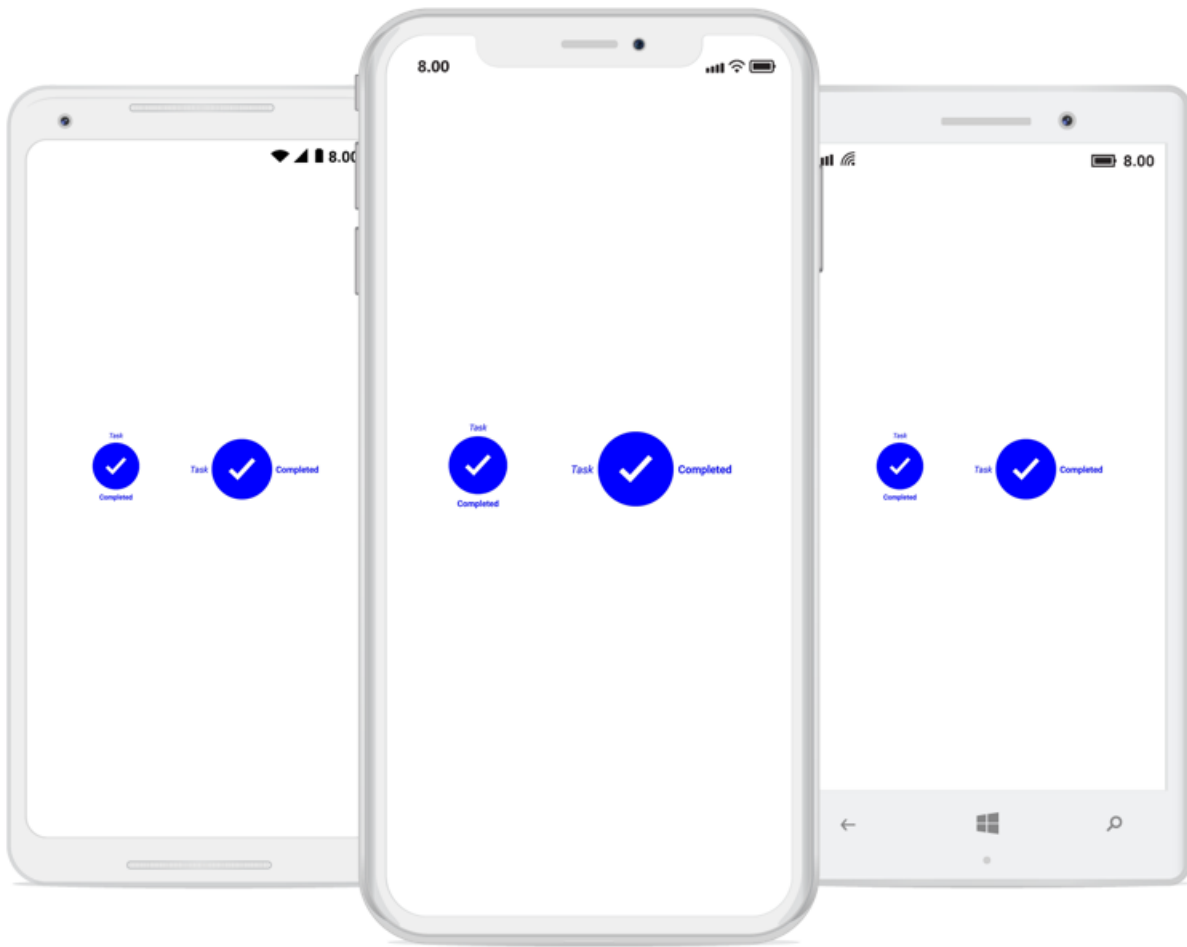
### XML

```
<progressBar:SfStepProgressBar Orientation="Horizontal"
HorizontalOptions="Center" VerticalOptions="Center">
  <progressBar:StepView PrimaryText="Completed" SecondaryText="Task"
  Status="Completed"/>
</progressBar:SfStepProgressBar>
```

### C#

```
SfStepProgressBar stepProgress = new SfStepProgressBar();
stepProgress.Orientation = StepOrientation.Vertical;
StepView step1 = new StepView();
step1.SecondaryText = "Task";
step1.PrimaryText = "Completed";
step1.Status = StepStatus.Completed;
stepProgress.Children.Add(step1);
```





### Formatted text

To customize the description with different formatting style, `PrimaryFormattedText` and `SecondaryFormattedText` can be used. The following code example explains how to set `PrimaryFormattedText` and `SecondaryFormattedText` to a step view.

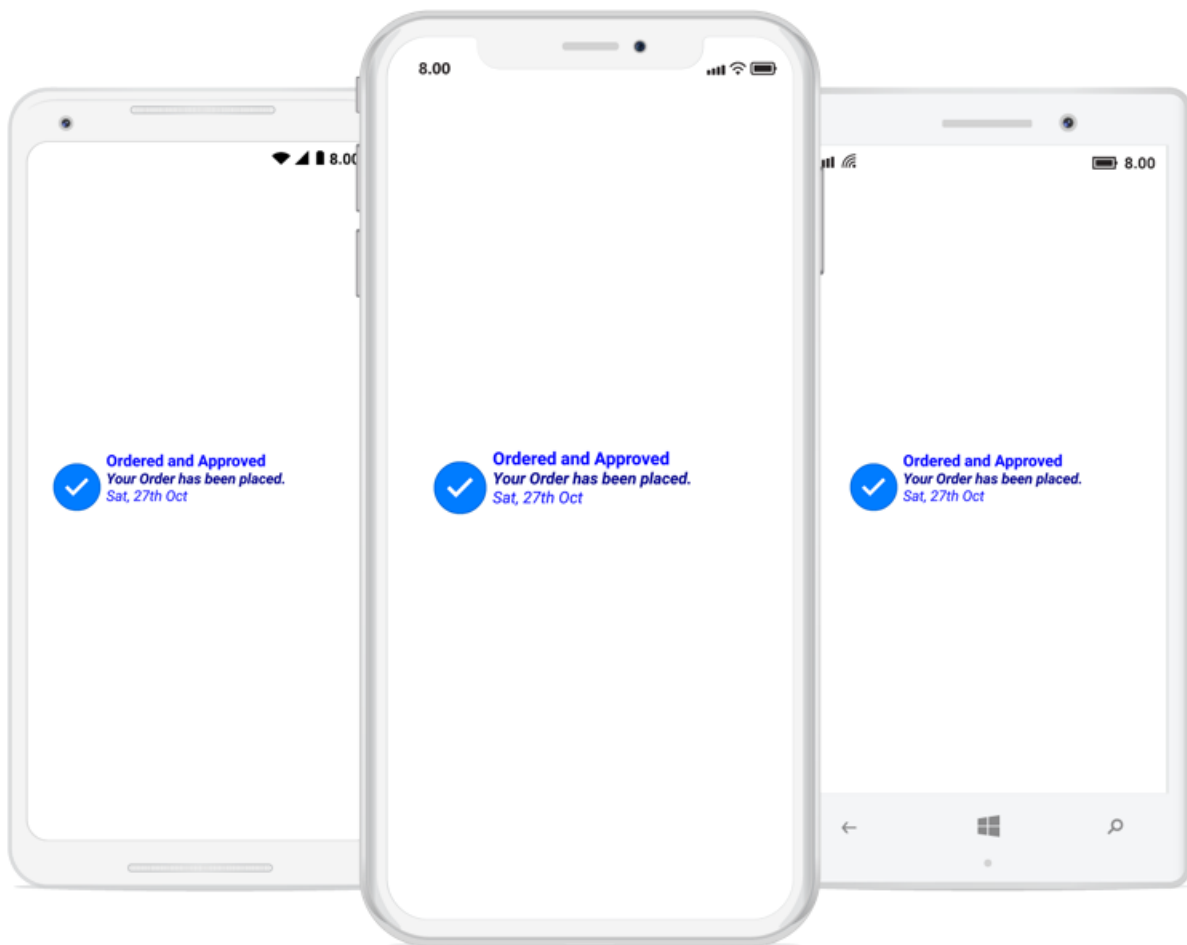
### XML

```
<progressBar:SfStepProgressBar Orientation="Vertical" >
  <progressBar:StepView Status="Completed">
    <progressBar:StepView.PrimaryFormattedText>
      <FormattedString>
        <Span x:Name="span1" ClassId="1" Text="Ordered and Approved" FontSize="13"
          FontAttributes="Bold" TextColor="Blue"/>
        <Span x:Name="span2" ClassId="2" Text="&#10;Your Order has been placed."
          FontAttributes="Italic,Bold" FontSize="12" TextColor="DarkBlue"/>
        <Span x:Name="span3" ClassId="3" Text="&#10;Sat, 27th Oct&#10;"
          FontSize="12" FontAttributes="Italic" TextColor="Blue"/>
      </FormattedString>
    </progressBar:StepView.PrimaryFormattedText>
  </progressBar:StepView>
</progressBar:SfStepProgressBar>
```

**C#**

```
SfStepProgressBar stepProgress = new SfStepProgressBar();
stepProgress.Orientation = StepOrientation.Vertical;
StepView step1 = new StepView();
step1.Status = StepStatus.Completed;
stepProgress.CompletedStepStyle.MarkerSize = 50;
stepProgress.CompletedStepStyle.MarkerContentSize = 25;
step1.PrimaryFormattedText = new FormattedString();
step1.PrimaryFormattedText.Spans.Add(new Span { Text = "Ordered and
Approved", FontSize = 13, FontAttributes = FontAttributes.Bold, TextColor =
Color.Blue });
step1.PrimaryFormattedText.Spans.Add(new Span { Text = "\nYour Order has
been placed", FontSize = 12, FontAttributes = FontAttributes.Italic |
FontAttributes.Bold, TextColor = Color.DarkBlue });
step1.PrimaryFormattedText.Spans.Add(new Span { Text = "\nSat, 27th Oct",
FontSize = 12, FontAttributes = FontAttributes.Italic, TextColor =
Color.Blue });
stepProgress.Children.Add(step1);
```

**Note:** To learn more about defining formats for different span in a text, refer to [FormattedString](#).



### Customize description

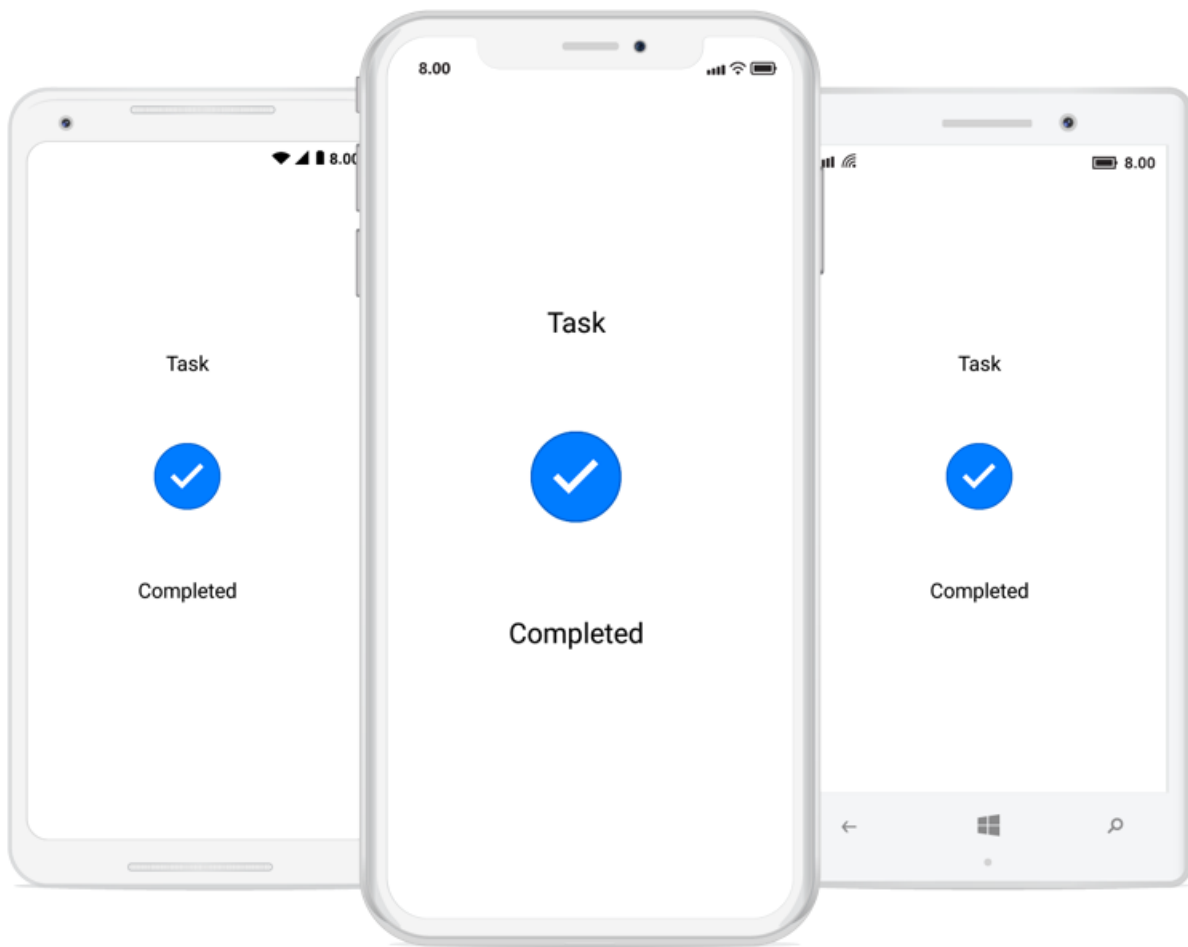
Using `TitleAlignment` and `TitleSpace`, the description alignment and space between the description and marker can be customized, respectively. The following code example explains how to customize the `TitleSpace` and `TitleAlignment` properties.

#### XML

```
<progressBar:SfStepProgressBar x:Name="stepProgress"
Orientation="Horizontal" TitleAlignment="Center" TitleSpace="50">
<progressBar:SfStepProgressBar.CompletedStepStyle>
<progressBar:StepStyle x:TypeArguments ="progressBar:CompletedStepState"
MarkerSize="50" MarkerContentSize="25" />
</progressBar:SfStepProgressBar.CompletedStepStyle>
<progressBar:StepView Status="Completed" PrimaryText="Completed"
SecondaryText="Task">
</progressBar:StepView>
</progressBar:SfStepProgressBar>
```

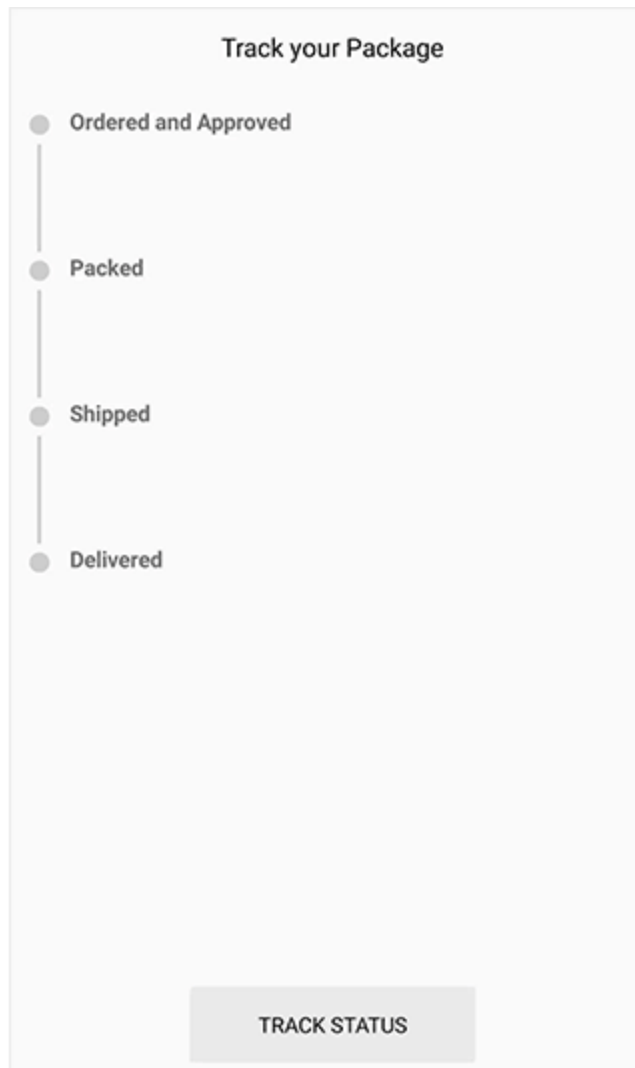
#### C#

```
SfStepProgressBar stepProgress = new SfStepProgressBar();
stepProgress.Orientation = StepOrientation.Horizontal;
stepProgress.TitleSpace = 50;
stepProgress.TitleAlignment = StepTitleAlignment.Center;
stepProgress.CompletedStepStyle.MarkerSize = 50;
stepProgress.CompletedStepStyle.MarkerContentSize = 25;
StepView step1 = new StepView();
step1.Status = StepStatus.Completed;
step1.SecondaryText = "Task";
step1.PrimaryText = "Completed";
stepProgress.Children.Add(step1);
```



### Status

A step has three statuses: not started, in progress, and completed. Based on the status, you can format a step with different styles, which means whenever the status of a step changes, the style of the visual will change synchronously. Refer the following GIF in which the status of step is updated dynamically.



The following code example explains how to set a status for a step view.

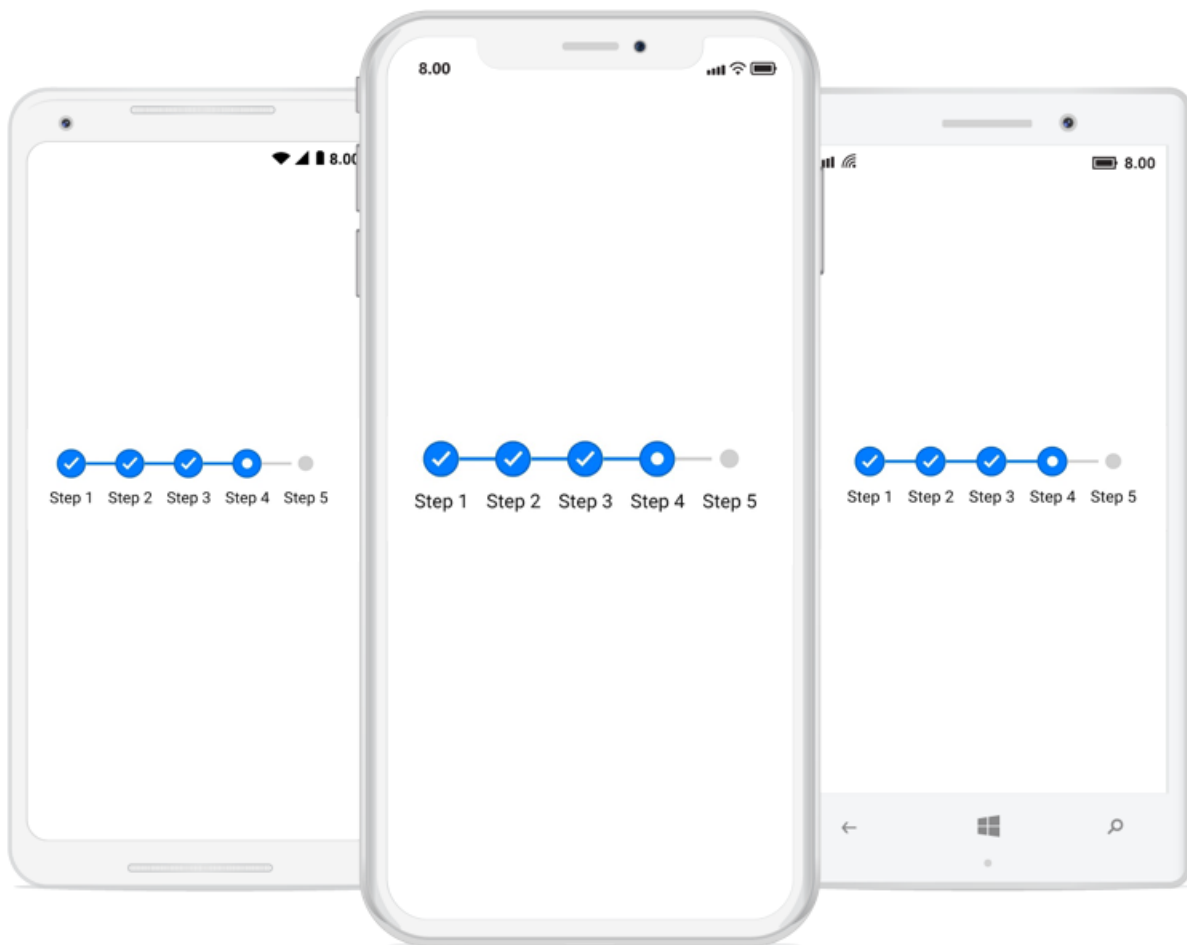
#### XML

```
<progressBar:SfStepProgressBar HorizontalOptions="Center"
VerticalOptions="Center">
  <progressBar:StepView PrimaryText="Step 1" />
  <progressBar:StepView PrimaryText="Step 2" />
  <progressBar:StepView PrimaryText="Step 3" />
  <progressBar:StepView PrimaryText="Step 4" Status="InProgress" />
  <progressBar:StepView PrimaryText="Step 5" />
</progressBar:SfStepProgressBar>
```

#### C#

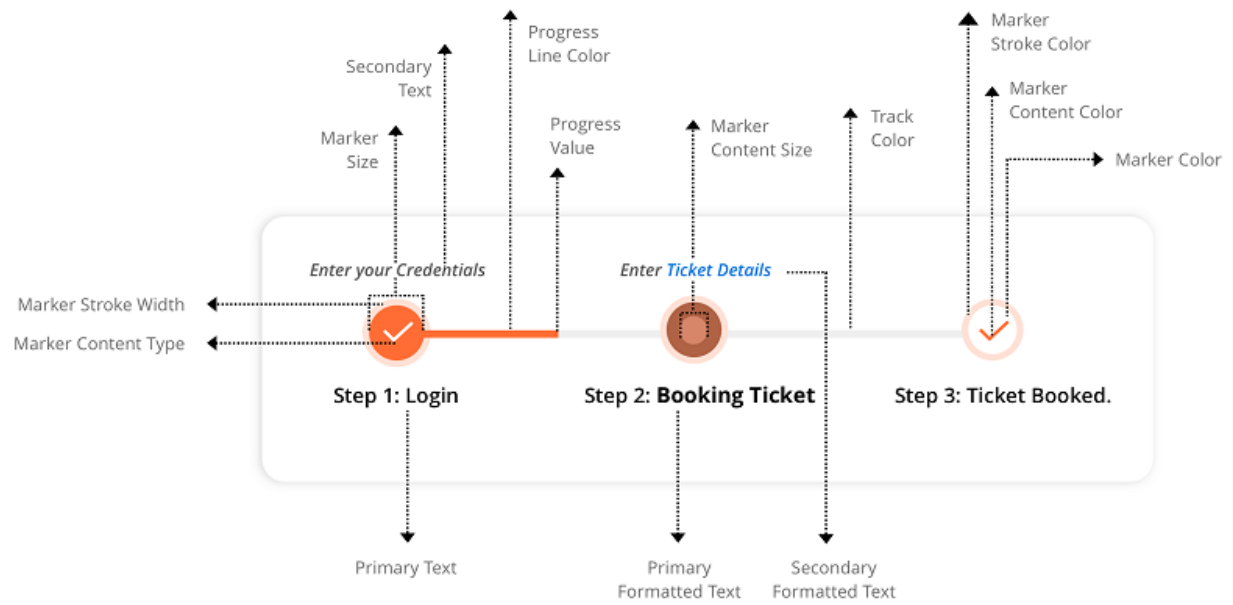
```
//Create StepProgressBar control
SfStepProgressBar stepProgressBar = new SfStepProgressBar();
stepProgressBar.VerticalOptions = LayoutOptions.Center;
stepProgressBar.HorizontalOptions = LayoutOptions.Center;
stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 1" });
stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 2" });
```

```
stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 3" });  
stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 4", Status  
= StepStatus.InProgress });  
stepProgressBar.Children.Add(new StepView() { PrimaryText = "Step 5" });
```



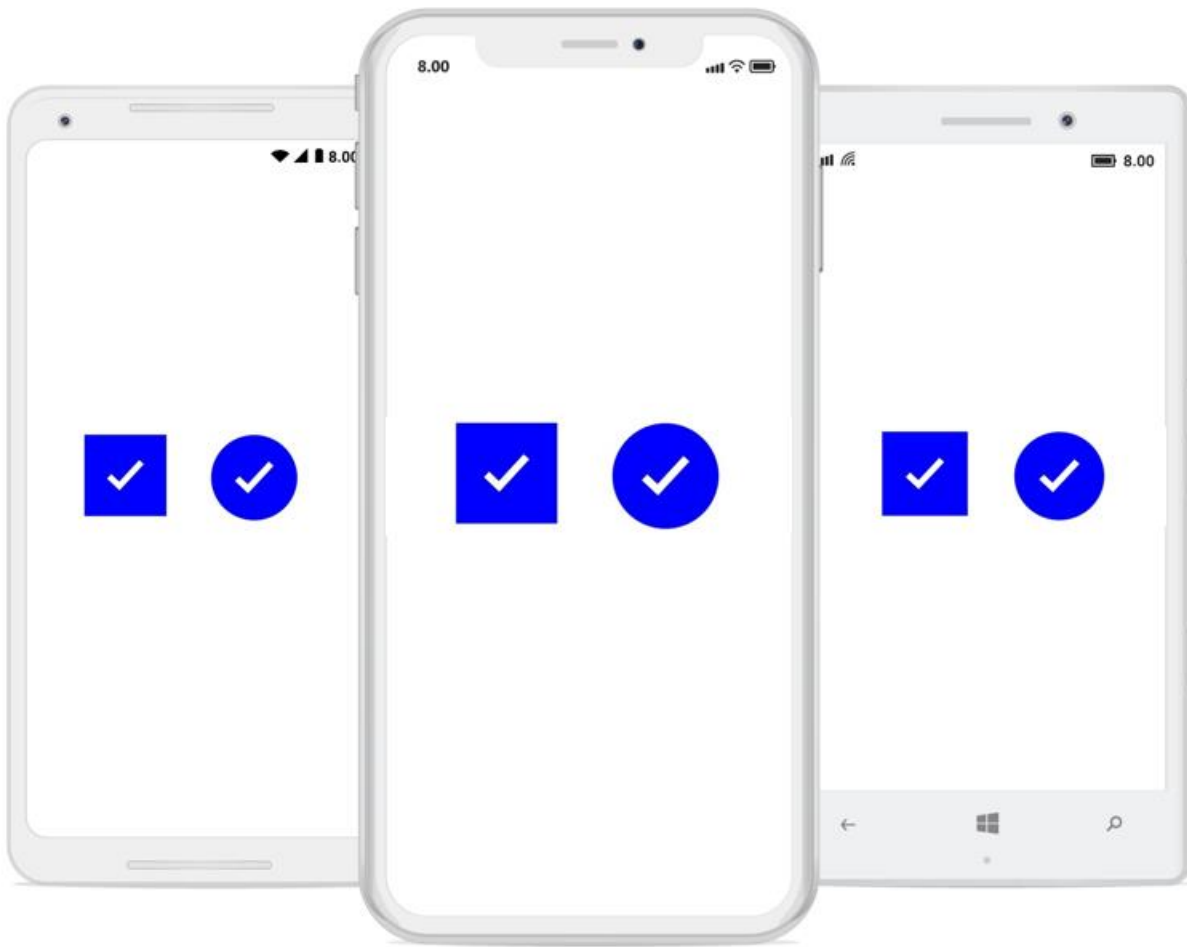
### Customizing Step view style based on their status

The Xamarin.Forms `StepProgressBar` control allows you to customize a step based on its status. Define an individual style for each status to achieve this. Marker color, marker shape type, marker content color, marker content type, marker stroke color, marker stroke width, marker size, marker content size, progress line color, and font can be defined for a style. The following overview image explains different properties available in `StepProgressBar`.



### Marker shape

The shape of a step marker can be a circle or a square.



#### *Content type*

You can customize the step content with numbers, ticks, crosses, dots or images.





The following code example explains how to customize a step view based on their status.

#### XML

```
<progressBar:SfStepProgressBar TitleAlignment="Start"
  BackgroundColor="Transparent" Orientation="Vertical" x:Name="stepProgress"
  Margin="16,16,0,0">
  <progressBar:SfStepProgressBar.NotStartedStepStyle>
  <progressBar:StepStyle x:TypeArguments="progressBar:NotStartedStepState"
  MarkerShapeType="Square" MarkerStrokeColor="Red" MarkerContentType="Cross"
  MarkerContentFillColor="Red"/>
  </progressBar:SfStepProgressBar.NotStartedStepStyle>
  <progressBar:SfStepProgressBar.InProgressStepStyle>
  <progressBar:StepStyle x:TypeArguments="progressBar:InProgressStepState"
  MarkerShapeType="Circle" MarkerContentType="None" MarkerFillColor="Black"/>
  </progressBar:SfStepProgressBar.InProgressStepStyle>
  <progressBar:SfStepProgressBar.CompletedStepStyle>
  <progressBar:StepStyle x:TypeArguments="progressBar:CompletedStepState"
  MarkerShapeType="Circle" MarkerContentType="Tick"
  MarkerContentFillColor="White" MarkerFillColor="Green"/>
  </progressBar:SfStepProgressBar.CompletedStepStyle>
  <progressBar:StepView x:Name="stepView1">
  <progressBar:StepView.PrimaryFormattedText>
```

```

<FormattedString>
<Span x:Name="span1" ClassId="1" Text="Ordered and Approved" FontSize="13"
FontAttributes="Bold"/>
<Span x:Name="span2" ClassId="2" Text="&#10;&#10;Your Order has been
placed." FontSize="12" />
<Span x:Name="span3" ClassId="3" Text="&#10;Sat, 27th Oct&#10;"
FontSize="12" />
</FormattedString>
</progressBar:StepView.PrimaryFormattedText>
</progressBar:StepView>
<progressBar:StepView x:Name="stepView2">
<progressBar:StepView.PrimaryFormattedText>
<FormattedString>
<Span x:Name="span21" ClassId="1" Text="Packed" FontSize="13"
FontAttributes="Bold" />
<Span x:Name="span22" ClassId="2" Text="&#10;&#10;Your item has been picked
up by courier partner." FontSize="12"/>
<Span x:Name="span23" ClassId="3" Text="&#10;Mon, 29th Oct" FontSize="12" />
</FormattedString>
</progressBar:StepView.PrimaryFormattedText>
</progressBar:StepView>
<progressBar:StepView x:Name="stepView3" Status="InProgress"
ProgressValue="50">
<progressBar:StepView.PrimaryFormattedText>
<FormattedString>
<Span x:Name="span31" ClassId="1" Text="Shipped" FontSize="13"
FontAttributes="Bold" />
</FormattedString>
</progressBar:StepView.PrimaryFormattedText>
</progressBar:StepView>
<progressBar:StepView x:Name="stepView4">
<progressBar:StepView.PrimaryFormattedText>
<FormattedString>
<Span x:Name="span41" ClassId="1" Text="Delivered" FontSize="13"
FontAttributes="Bold" />
</FormattedString>
</progressBar:StepView.PrimaryFormattedText>
</progressBar:StepView>
</progressBar:SfStepProgressBar>

```

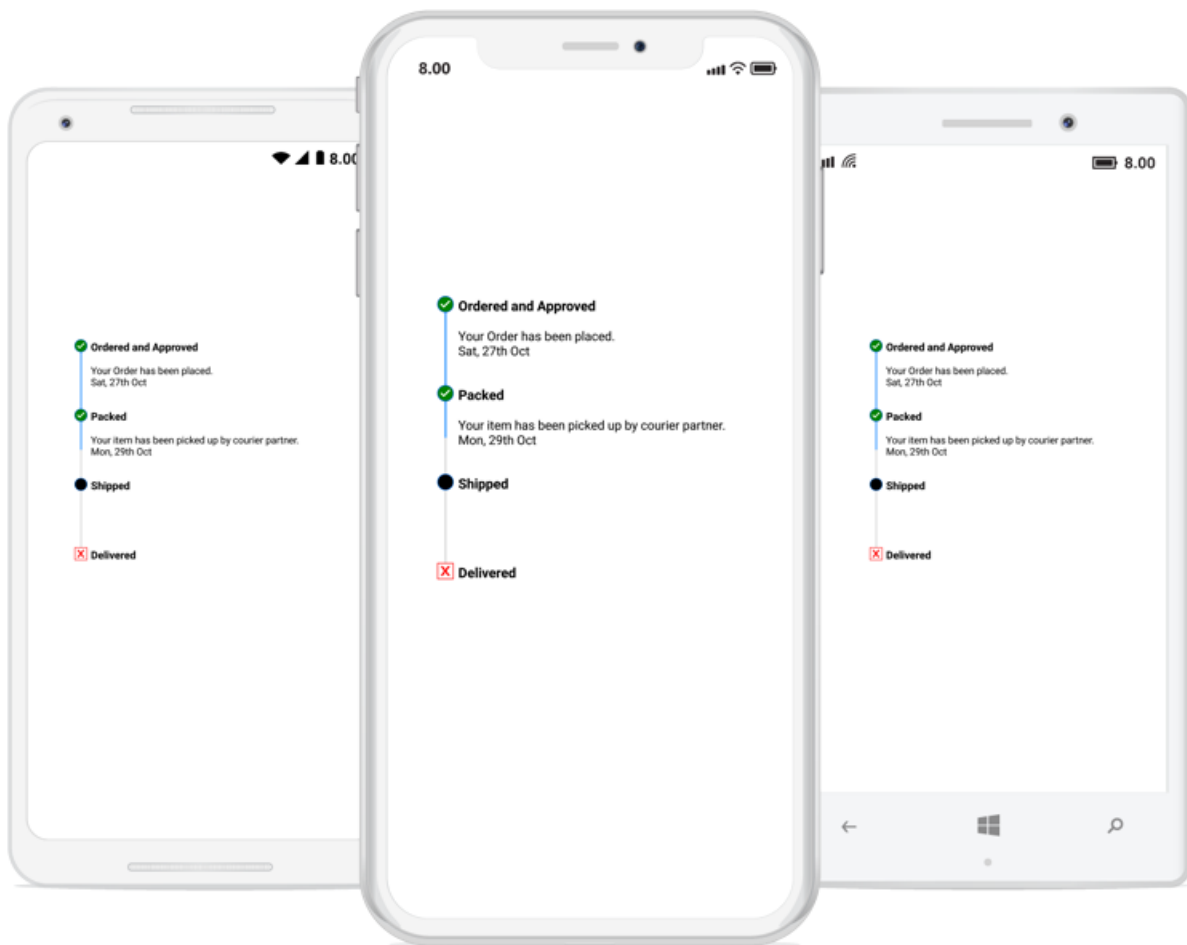
## C#

```

//Initialize the StepProgressBar
SfStepProgressBar stepProgress = new SfStepProgressBar();
stepProgress.Orientation = StepOrientation.Vertical;
stepProgress.TitleAlignment = StepTitleAlignment.Start;
//Modify NotStartedStepStyle
stepProgress.NotStartedStepStyle.MarkerShapeType = StepShapeType.Square;
stepProgress.NotStartedStepStyle.MarkerStrokeColor = Color.Red;
stepProgress.NotStartedStepStyle.MarkerContentType = StepContentType.Cross;
stepProgress.NotStartedStepStyle.MarkerContentFillColor = Color.Red;
//Modify InProgressStepStyle
stepProgress.InProgressStepStyle.MarkerContentType = StepContentType.None;
stepProgress.InProgressStepStyle.MarkerFillColor = Color.Black;
stepProgress.InProgressStepStyle.MarkerShapeType = StepShapeType.Circle;
//Modify CompletedStepStyle

```

```
stepProgress.CompletedStepStyle.MarkerShapeType = StepShapeType.Circle;
stepProgress.CompletedStepStyle.MarkerContentType = StepContentType.Tick;
stepProgress.CompletedStepStyle.MarkerContentFillColor = Color.White;
stepProgress.CompletedStepStyle.MarkerFillColor = Color.Green;
//Define StepView
StepView step1 = new StepView();
step1.PrimaryFormattedText = new FormattedString();
step1.PrimaryFormattedText.Spans.Add(new Span { Text = "Ordered and
Approved", FontSize = 13, FontAttributes = FontAttributes.Bold, TextColor =
Color.Blue });
step1.PrimaryFormattedText.Spans.Add(new Span { Text = "\nYour Order has
been placed", FontSize = 12, FontAttributes = FontAttributes.Italic |
FontAttributes.Bold, TextColor = Color.DarkBlue });
step1.PrimaryFormattedText.Spans.Add(new Span { Text = "\nSat, 27th Oct\n",
FontSize = 12, FontAttributes = FontAttributes.Italic, TextColor =
Color.Blue });
stepProgress.Children.Add(step1);
StepView step2 = new StepView();
step2.PrimaryFormattedText = new FormattedString();
step2.PrimaryFormattedText.Spans.Add(new Span { Text = "Packed", FontSize =
13, FontAttributes = FontAttributes.Bold, TextColor = Color.Blue });
step2.PrimaryFormattedText.Spans.Add(new Span { Text = "\nYour item has been
picked up by courier partner.", FontSize = 12, FontAttributes =
FontAttributes.Italic | FontAttributes.Bold, TextColor = Color.DarkBlue });
step2.PrimaryFormattedText.Spans.Add(new Span { Text = "\nMon, 29th Oct\n",
FontSize = 12, FontAttributes = FontAttributes.Italic, TextColor =
Color.Blue });
stepProgress.Children.Add(step2);
StepView step3 = new StepView();
step3.Status = StepStatus.InProgress;
step3.ProgressValue = 50;
step3.PrimaryFormattedText = new FormattedString();
step3.PrimaryFormattedText.Spans.Add(new Span { Text = "Shipped", FontSize =
13, FontAttributes = FontAttributes.Bold, TextColor = Color.Blue });
stepProgress.Children.Add(step3);
StepView step4 = new StepView();
step4.PrimaryFormattedText = new FormattedString();
step4.PrimaryFormattedText.Spans.Add(new Span { Text = "Delivered", FontSize
= 13, FontAttributes = FontAttributes.Bold, TextColor = Color.Blue });
stepProgress.Children.Add(step4);
```



## SfPullToRefresh

### Overview

SfPullToRefresh is a refresh control that allows you to interact and refresh the view loaded in it. The SfPullToRefresh control allows you to refresh the view upon performing the pull to refresh action. A progress indicator will be shown while start the pulling. The application will be refreshed once you have pulled down a certain distance and release the touch.

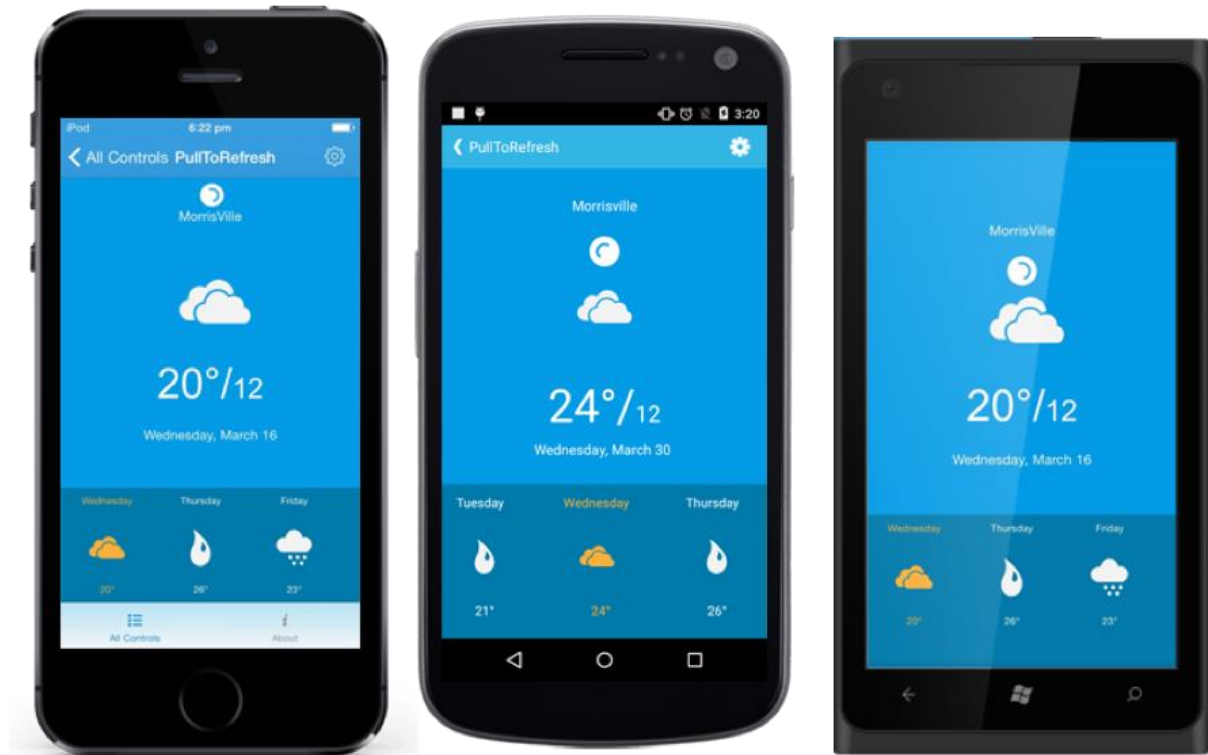
### Use Case Scenarios

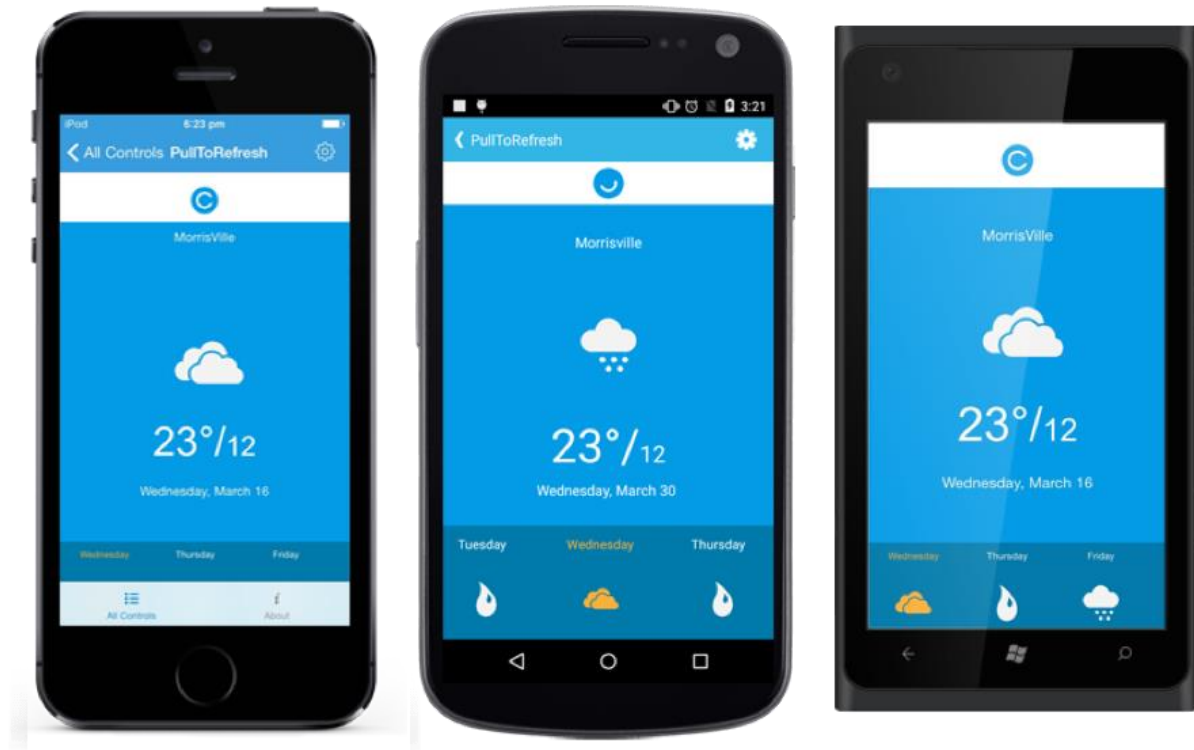
SfPullToRefresh controls are mainly used in applications where the user needs to refresh the content on demand. PullToRefresh controls are mainly used in applications where the user needs to refresh the content on demand. The main applications that use the PullToRefresh are listed below:

1. Facebook
2. Weather
3. Gmail
4. Live Score applications

## Key Features

- **TransitionMode:** Specifies the transition mode of the SfPullToRefresh.
- **Sizing:** Width, Height and progress stroke width customization support for SfPullToRefresh.
- **Color:** Background and progress stroke color customization support for SfPullToRefresh.
- **StartRefreshing():** Starts the programmatic refreshing.
- **EndRefreshing():** Ends the programmatic refreshing.





## Getting Started

This section provides a quick overview for working with SfPullToRefresh for Xamarin.Forms.

### Assembly deployment

After installing Essential Studio for Xamarin, you can find all the required assemblies in the installation folders,

{Syncfusion Essential Studio Installed location}\Essential Studio\{{ site.releaseversion }}\Xamarin\lib

Eg: C:\Program Files (x86)\Syncfusion\Essential Studio\{{ site.releaseversion }}\Xamarin\lib

---

**Note:** Assemblies can be found in unzipped package location in Mac

---

### NuGet configuration

To install the required NuGet for the SfPullToRefresh control in the application, configure the NuGet packages of the Syncfusion components.

Refer to the following KB to configure the NuGet package of the Syncfusion components:

[How to configure package source and install Syncfusion NuGet packages in an existing project?](#)

The following NuGet package should be installed to use the SfPullToRefresh control in the application.

Project	Required package
Xamarin.Forms	Syncfusion.Xamarin.SfPullToRefresh

### Adding SfPullToRefresh reference

You can add SfPullToRefresh reference using one of the following methods:

#### Method 1: Adding SfPullToRefresh reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfPullToRefresh). To add SfPullToRefresh to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfPullToRefresh](https://www.nuget.org/packages/Syncfusion.Xamarin.SfPullToRefresh), and then install it.

![Adding SfPullToRefresh reference from NuGet](SfPullToRefresh\_images/Adding SfPullToRefresh reference.png)

**Note:** When there is a mismatch between Syncfusion NuGet packages among your projects, `System.IO.FileLoadException` will occur. To overcome this exception, install the same version of SfPullToRefresh assemblies in all your projects.

#### Method 2: Adding SfPullToRefresh reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfPullToRefresh control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfPullToRefresh assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Core.XForms.dll Syncfusion.SfPullToRefresh.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.SfPullToRefresh.XForms.dll Syncfusion.SfPullToRefresh.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.SfPullToRefresh.XForms.dll Syncfusion.SfPullToRefresh.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.SfPullToRefresh.XForms.dll Syncfusion.SfPullToRefresh.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

---

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

**Note:** When there is a mismatch of Xamarin NuGet packages between your sample and SfPullToRefresh assemblies, an error `Could not load type Xamarin.Forms.ElementTemplate` will occur. Please refer to `ReadMe` to know the software requirements of Syncfusion controls.

---

### Launching the SfPullToRefresh on each platform

To use SfPullToRefresh inside an application, each platform application must initialize the SfPullToRefresh renderer. This initialization step varies from platform to platform and is discussed in the following sections.

#### Android

The Android launches the SfPullToRefresh without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

#### iOS

To launch the SfPullToRefresh in iOS, you need to call the `SfPullToRefreshRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework initialization and before the `LoadApplication` is called, as demonstrated in the following code example:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    SfPullToRefreshRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

#### Universal Windows Platform (UWP)

To launch the SfPullToRefresh in UWP, you need to call the `SfPullToRefreshRenderer.Init()` in the `MainPage` constructor before the `LoadApplication` is called, as demonstrated in the following code example:

#### C#

```
public MainPage()
{
    ...
    SfPullToRefreshRenderer.Init();
    LoadApplication (new App ());
    ...
}
```



### *ReleaseMode issue in UWP platform*

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the SfPullToRefresh assemblies in **App.xaml.cs** in UWP project as like in below code snippet.

#### **C#**

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfPullToRefreshRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

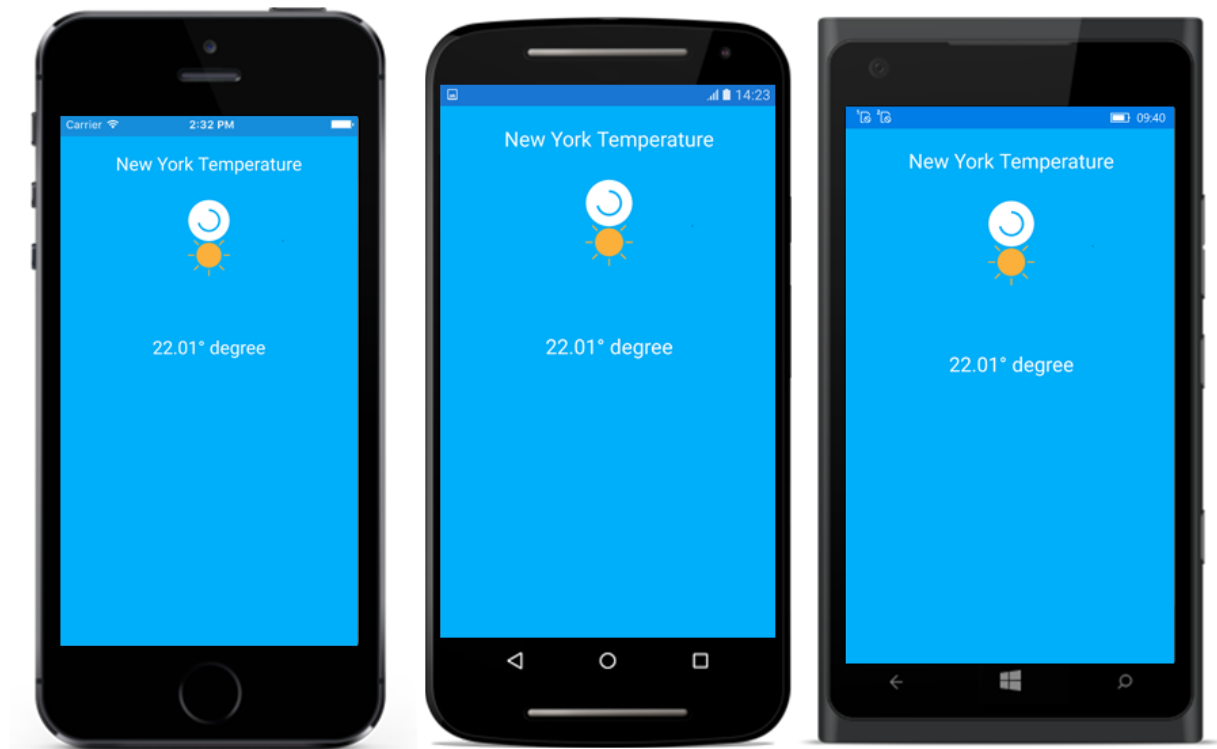
### Create a sample with SfPullToRefresh

This section explains how to create a SfPullToRefresh and configure it.

You can download the entire source code of this demo for Xamarin.Forms from [here](#).

#### Creating the project

Create a new BlankApp (Xamarin.Forms.Portable) application in Xamarin Studio or Visual Studio for Xamarin.Forms. This is how the final output will look like on iOS, Android and Windows Phone devices.



### Adding SfPullToRefresh in Xamarin.Forms

1. Add the required assembly references to the pcl and renderer projects as discussed in the [Assembly deployment](#) section.
2. Import SfPullToRefresh control namespace `Syncfusion.SfPullToRefresh.XForms`.
3. Set the any View as `PullableContent` of the `SfPullToRefresh`.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage"
xmlns:syncfusion="clr-namespace:Syncfusion.SfPullToRefresh.XForms;assembly=Syncfusion.SfPullToRefresh.XForms">
  <syncfusion:SfPullToRefresh x:Name="pullToRefresh"
    IsRefreshing="False"
    PullingThreshold="100"
    RefreshContentHeight="30"
    RefreshContentThreshold="30"
    RefreshContentWidth="30">
    <syncfusion:SfPullToRefresh.PullableContent>
      <StackLayout BackgroundColor="#00AFF9" Orientation="Vertical">
        <Label Text="New York Temperature" FontSize="Large" TextColor="White"
          HorizontalTextAlignment="Center" Margin="20"/>
      </StackLayout>
    </syncfusion:SfPullToRefresh.PullableContent>
  </syncfusion:SfPullToRefresh>
</ContentPage>
```

```

<Image WidthRequest="100" HorizontalOptions="Center" HeightRequest="100"
Margin="20" Source="GettingStarted/warmselected.png"/>
<Label x:Name="weatherData" FontSize="Large" TextColor="White"
HorizontalTextAlignment="Center" Margin="20"/>
</StackLayout>
</syncfusion:SfPullToRefresh.PullableContent>
</syncfusion:SfPullToRefresh>
</ContentPage>

```

**C#**

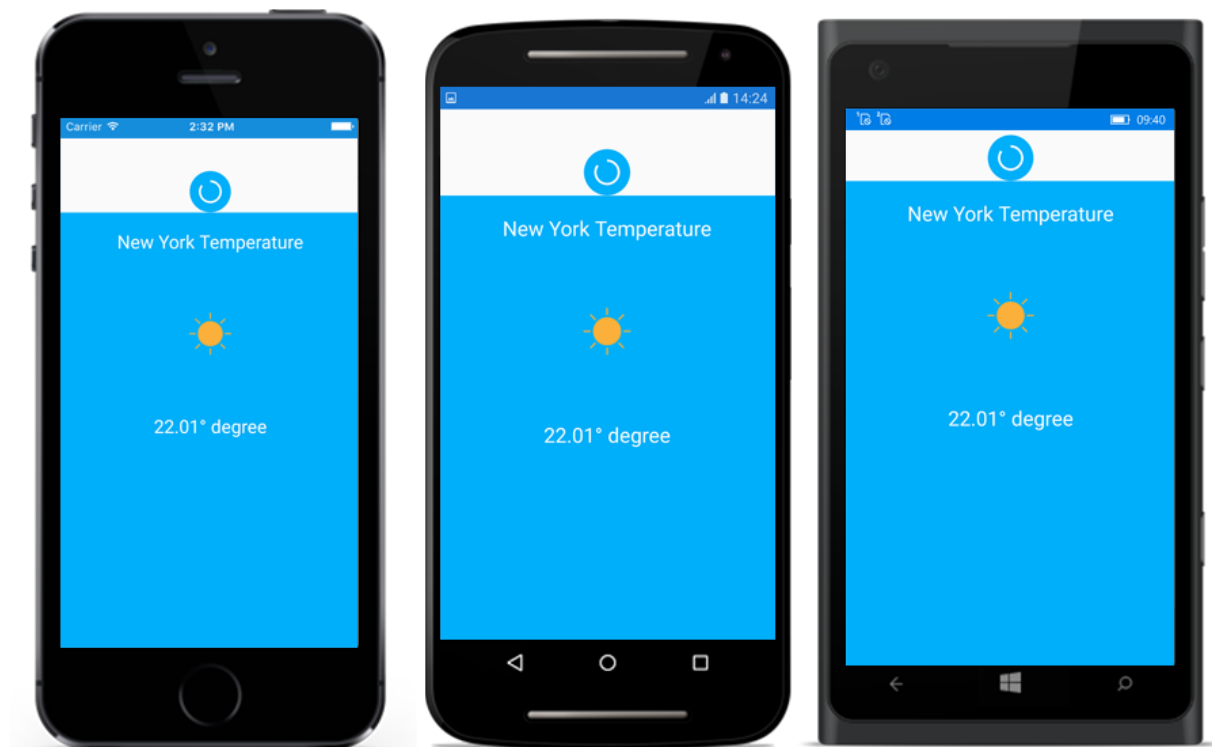
```

using Syncfusion.SfPullToRefresh.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        Random random = new Random();
        string[] temperatures = new string[] { "22°", "18°", "12°", "25°", "23°",
        "20°", "25°" };
        public MainPage()
        {
            InitializeComponent();
            this.BindingContext = this;
            weatherData.Text = temperatures[0].ToString()+" degree";
            pullToRefresh.Pulling += PullToRefresh_Pulling;
            pullToRefresh.Refreshing += PullToRefresh_Refreshing;
        }
        private async void PullToRefresh_Refreshing(object sender, EventArgs args)
        {
            pullToRefresh.IsRefreshing = true;
            await Task.Delay(2000);
            int number = random.Next(0, 6);
            new WeatherData(temperatures[number]);
            weatherData.Text = temperatures[number].ToString() + " degree";
            pullToRefresh.IsRefreshing = false;
        }
        private void PullToRefresh_Pulling(object sender, PullingEventArgs args)
        {
            args.Cancel = false;
            var progress = args.Progress;
        }
    }
    Model class:
    public class WeatherData: INotifyPropertyChanged
    {
        private string _temperature;
        public WeatherData(string temperature)
        {
            Temperature = temperature;
        }
        public string Temperature
        {
            get
            {

```

```
return _temperature;
}
set
{
    _temperature = value;
    RaisePropertyChanged("Temperature");
}
}
#region INotifyPropertyChanged implementation
public event PropertyChangedEventHandler PropertyChanged;
private void RaisePropertyChanged(String Name)
{
    if (PropertyChanged != null)
        this.PropertyChanged(this, new PropertyChangedEventArgs(Name));
}
#endregion
}
```

If we run the above sample with `TransitionMode` as `Push`, the output will look like on iOS, Android and Windows Phone devices as shown below.



## Features

### PullableContent

Gets or sets the content of the refresh view. `PullableContent` is the main view of the `SfPullToRefresh` control on which the desired items can be placed.

### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh"
    PullingThreshold="120"
    RefreshContentHeight="30"
    RefreshContentThreshold="30"
    RefreshContentWidth="30">
    <syncfusion:SfPullToRefresh.PullableContent>
    <Label x:Name="Monthlabel"
        TextColor="White"
        HorizontalTextAlignment="Center"
        VerticalTextAlignment="Start" />
    </syncfusion:SfPullToRefresh.PullableContent>
</syncfusion:SfPullToRefresh>
```

### TransitionMode

The **TransitionMode** property specifies the mode of the animations. It has the following two modes:

- **SlideOnTop**
- **Push**

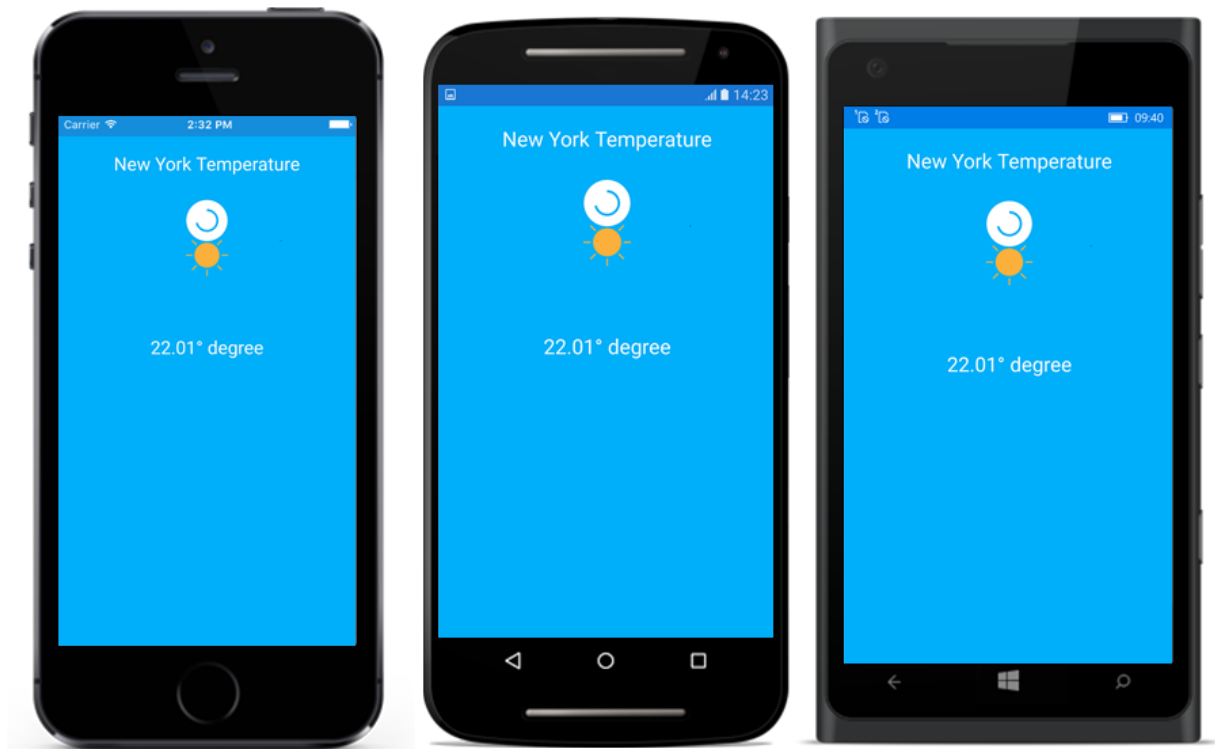
The default transition is **SlideOnTop** that draws the RefreshContent on top of the **PullableContent**.

### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh"
    TransitionMode="SlideOnTop" />
```

### C#

```
pullToRefresh.TransitionMode = TransitionType.SlideOnTop;
```



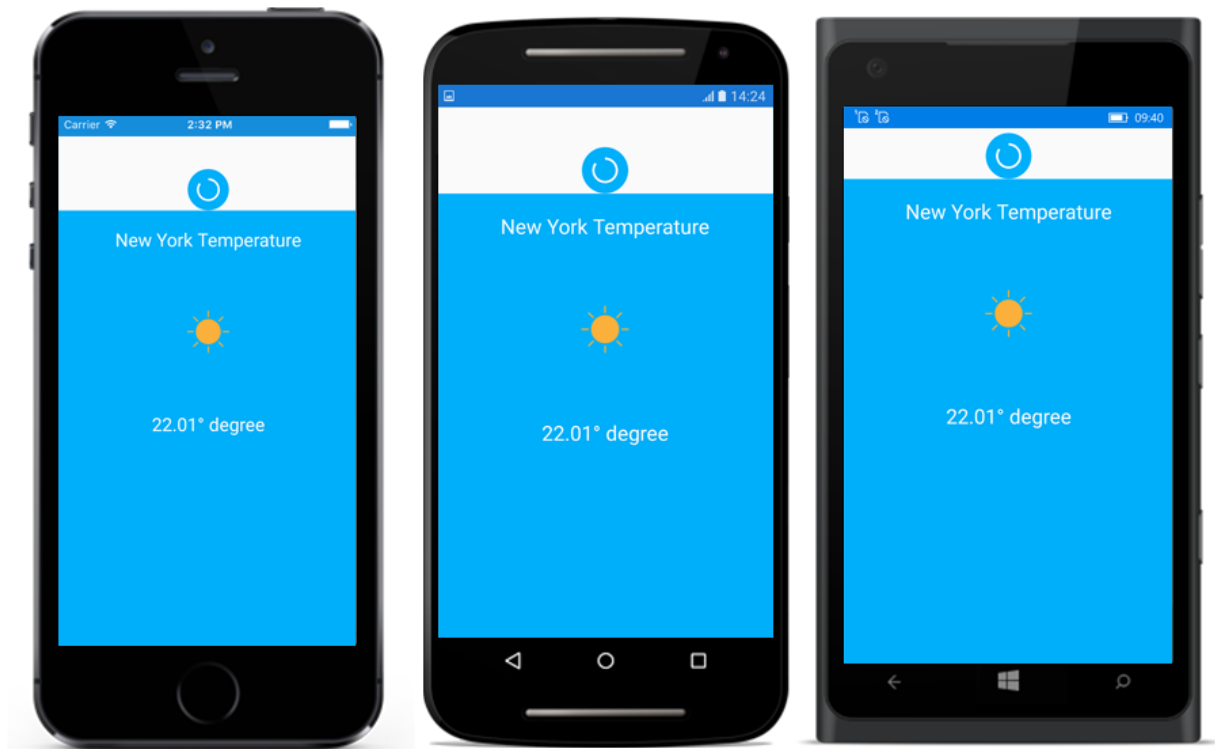
The following code example shows how to set `TransitionMode` as `Push` to `SfPullToRefresh`. This transition moves the refresh content and main content simultaneously.

#### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh" TransitionMode="Push" />
```

#### C#

```
pullToRefresh.TransitionMode = TransitionType.Push;
```



### RefreshContentThreshold

Gets or sets the refresh content threshold value that indicates progress indicator starting position in view

#### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh"
RefreshContentThreshold="50"/>
```

#### C#

```
pullToRefresh.RefreshContentThreshold = 50d;
```

### PullingThreshold

Gets or sets the value for the refresh content threshold, this indicate progress indicator maximum pulling position in view.

#### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh" PullingThreshold="200"/>
```

#### C#

```
pullToRefresh.PullingThreshold = 200d;
```

### IsRefreshing

Get or set the state for refreshing the view. View will get refresh while `IsRefreshing` property is set `true` and View refreshing will be stopped when you set `IsRefreshing` is `false`.

#### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh" IsRefreshing = "True"/>
```

#### C#

```
pullToRefresh.IsRefreshing = true;
```

### ProgressBackgroundColor

Get or set the background color to the progress indicator.

#### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh" ProgressBackgroundColor = "White"/>
```

#### C#

```
pullToRefresh.ProgressBackgroundColor = Color.White;
```

### ProgressStrokeColor

Get or set the color to the progress indicator stroke

#### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh" ProgressStrokeColor = "Blue"/>
```

#### C#

```
pullToRefresh.ProgressStrokeColor = Color.Blue;
```

### ProgressStrokeWidth

Get or set the width to the progress indicator stroke.

#### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh" ProgressStrokeWidth="5"/>
```

#### C#

```
pullToRefresh.ProgressStrokeWidth = 5d;
```

### RefreshContentWidth

Get or set the width to the refresh content.

#### XML



```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh"
RefreshContentWidth="50"/>
```

### C#

```
pullToRefresh.RefreshContentWidth = 50d;
```

### RefreshContentHeight

Get or set the width to the refresh content.

### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh"
RefreshContentHeight="50"/>
```

### C#

```
pullToRefresh.RefreshContentHeight = 50d;
```

### Programmatic Support

#### *StartRefreshing()*

StartRefreshing method is used to refresh the content without interaction in pullable content. When invoke this StartRefreshing() method, then Progress indicator will be shown.

### C#

```
pullToRefresh.StartRefreshing();
```

#### *EndRefreshing()*

EndRefreshing method is used to end the progress animation of SfPullToRefresh.

### C#

```
pullToRefresh.EndRefreshing();
```

### Host SfDataGrid as pullable content

SfPullToRefresh controls provides support for loading any custom control as pullable content. To host SfDataGrid inside the SfPullToRefresh, follow the below steps.

<ol>

<li> Add the required assembly references to the pcl and renderer projects as discussed in the Assembly deployment section of <https://help.syncfusion.com/xamarin/sfdatagrid/getting-started#assembly-deployment> and <https://help.syncfusion.com/xamarin/sfpulltorefresh/getting-started#assembly-deployment>.

<li> Import SfPullToRefresh and SfDataGrid control namespace as follows.</li>

<br/>

### XML

```
<xmlns:syncfusion="clr-  
namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms  
">  
<xmlns:pull="clr-  
namespace:Syncfusion.SfPullToRefresh.XForms;assembly=Syncfusion.SfPullToRefr  
esh.XForms">
```

## C#

```
using Syncfusion.SfPullToRefresh.XForms;  
using Syncfusion.SfDataGrid.XForms;
```

<br/>

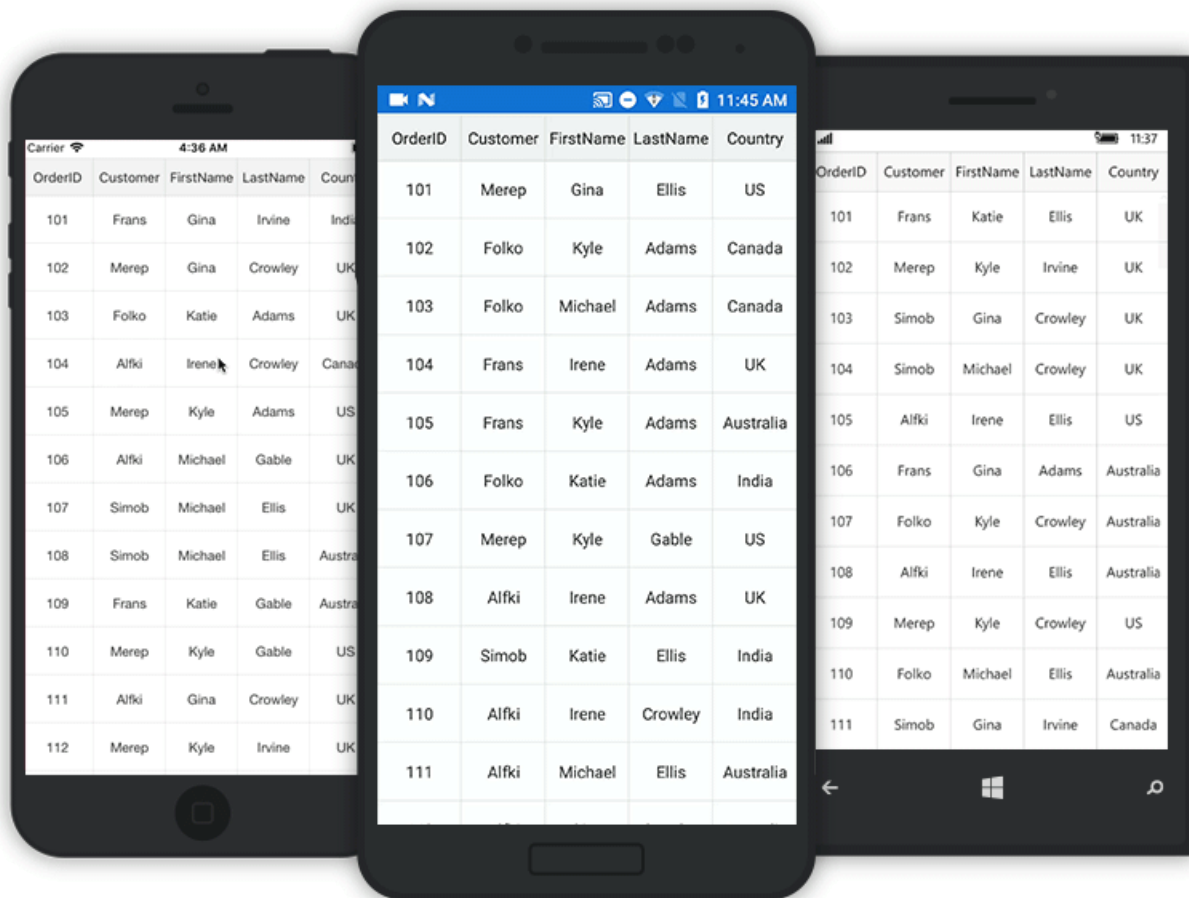
<li> Define SfDataGrid as PullableContent of the SfPullToRefresh.</li>

<li> Handle the pull to refresh events for refreshing the data. </li>

<li> Customize the required properties of SfDataGrid and SfPullToRefresh based on your requirement.</li>

</ol>

This is how the final output will look like on iOS, Android and Windows Phone devices when hosting a SfDatagrid control as pullable content.



## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:PullToRefreshSample"
x:Class="PullToRefreshSample.MainPage"
xmlns:syncfusion="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
xmlns:pull="clr-namespace:Syncfusion.SfPullToRefresh.XForms;assembly=Syncfusion.SfPullToRefresh.XForms">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<pull:SfPullToRefresh x:Name="pulltorefresh"
TransitionMode="SlideOnTop"
PullingThreshold="100"
RefreshContentHeight="30"
RefreshContentThreshold="40"
RefreshContentWidth="30">
<pull:SfPullToRefresh.PullableContent>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding OrdersInfo}"
```

```

AutoGenerateColumns="false">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID" />
<syncfusion:GridTextColumn MappingName="Customer"/>
<syncfusion:GridTextColumn MappingName="FirstName" />
<syncfusion:GridTextColumn MappingName="LastName" />
<syncfusion:GridTextColumn MappingName="Country" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
</pull:SfPullToRefresh.PullableContent>
</pull:SfPullToRefresh>
</ContentPage>

```

## C#

```

using Syncfusion.SfPullToRefresh.XForms;
using Xamarin.Forms;
using Syncfusion.SfDataGrid.XForms;
namespace PullToRefreshSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            pulltorefresh.Refreshing += PullToRefresh_Refreshing;
            dataGrid.VerticalOverScrollMode=VerticalOverScrollMode.None;
        }
        private async void PullToRefresh_Refreshing(object sender, EventArgs args)
        {
            await Task.Delay(new TimeSpan(0, 0, 3));
            viewModel.ItemsSourceRefresh();
            pulltorefresh.IsRefreshing = false;
        }
    }
    //Model class:
    public class OrderInfo
    {
        public OrderInfo()
        {
        }
        public string EmployeeID { get; internal set; }
        public object FirstName { get; internal set; }
        public object LastName { get; internal set; }
        public string Country { get; internal set; }
        public int OrderID { get; internal set; }
        public string Customer { get; internal set; }
    }
    //ViewModel Class:
    public class ViewModel
    {
        Random random = new Random();
        public ViewModel()
        {
        }
    }
}

```

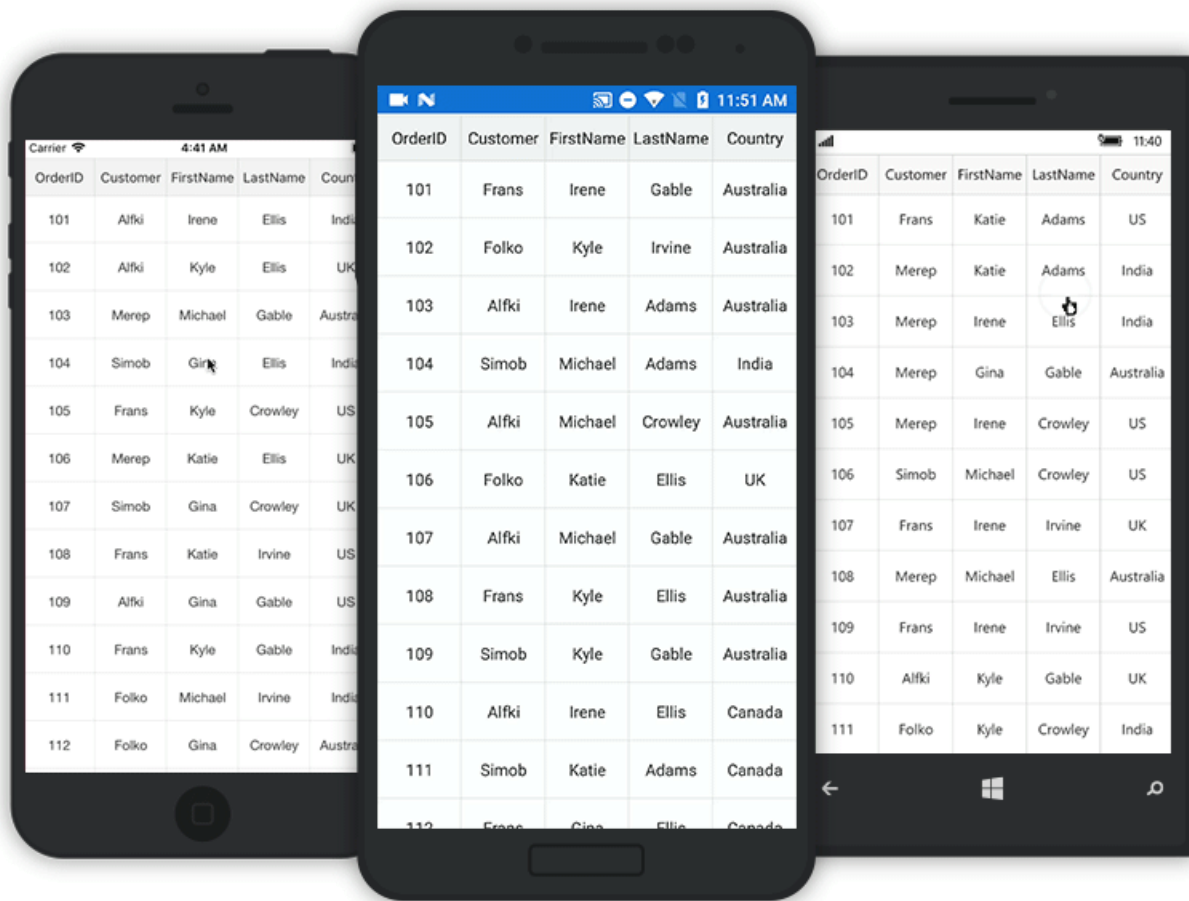
```
#region ItemsSourceRefresh
public void ItemsSourceRefresh()
{
    for (int i = 1; i <= 5; i++)
    {
        this.OrdersInfo.Insert(0, order.RefreshItemsSource(i));
    }
}
#endregion

//OrderInfoRepository Class:
public class OrderInfoRepository
{
    public OrderInfoRepository()
    {
    }
    private Random random = new Random();
    #region GetOrderDetails
    public ObservableCollection<OrderInfo> GetOrderDetails(int count)
    {
        ObservableCollection<OrderInfo> orderDetails = new
        ObservableCollection<OrderInfo>();
        for (int i = 101; i <= count + 100; i++)
        {
            var order = new OrderInfo()
            {
                OrderID = i,
                Customer = Customer[random.Next(5)],
                EmployeeID = random.Next(1700, 1800).ToString(),
                FirstName = FirstNames[random.Next(5)],
                LastName = LastNames[random.Next(5)],
                Country = country[random.Next(5)],
            };
            orderDetails.Add(order);
        }
        return orderDetails;
    }
    public OrderInfo RefreshItemsSource(int i)
    {
        var order = new OrderInfo()
        {
            OrderID = (i + random.Next(100, 110)),
            Customer = Customer[random.Next(5)],
            EmployeeID = random.Next(1700, 1800).ToString(),
            FirstName = FirstNames[random.Next(5)],
            LastName = LastNames[random.Next(5)],
            Country = country[random.Next(5)]
        };
        return order;
    }
}
#endregion

// Main DataSources
string[] FirstNames = new string[]
{ "Kyle", "Gina", "Irene", "Katie", "Michael" };
string[] LastNames = new string[]
{ "Adams", "Crowley", "Ellis", "Gable", "Irvine" };
string[] Customer = new string[] { "Alfie", "Frans", "Mered", "Folios", "Simon" };
```

```
string[] country = new string[] { "US", "Australia", "Canada", "UK", "India" };
}
```

If we run the above sample with TransitionMode as Push, the output will look like on iOS, Android and Windows Phone devices as shown below.



### Host SfListView as pullable content

To host SfListView inside the SfPullToRefresh which is used to update items in the list while performing the pull to refresh action.

<ol>

<li> Add the required assembly references to the pcl and renderer projects as discussed in the Assembly deployment section of [SfListView](https://help.syncfusion.com/xamarin/sflistview/getting-started#assembly-deployment) and [SfPullToRefresh](https://help.syncfusion.com/xamarin/sfpulltorefresh/getting-started#assembly-deployment).

<li> Import SfPullToRefresh control and SfListView control namespace as follows.</li>

<br/>

### XML

```
<xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XFormsassembly=Syncfusion.SfListView.XForms">
<xmlns:pull="clr-
namespace:Syncfusion.SfPullToRefresh.XFormsassembly=Syncfusion.SfPullToRefre
shXForms">
```

## C#

```
using Syncfusion.SfPullToRefresh.XForms;
using Syncfusion.ListView.XForms;
```

<br/>

<li> Define SfListView as PullableContent of the SfPullToRefresh.</li>

<li> Handle the pull to refresh events for refreshing the data. </li>

<li> Customize the required properties of SfListView and SfPullToRefresh based on your requirement.</li>

</ol>

This is how the final output will look like on iOS, Android and Windows Phone devices when hosting a SfListView control as pullable content.



## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```

xmlns:local="clr-namespace:ListviewPulltorefresh"
x:Class="ListviewPulltorefresh.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.ListView.XForms;assembly=Syncfusion.SfListView.XForms"
xmlns:pullToRefresh="clr-
namespace:Syncfusion.SfPullToRefresh.XForms;assembly=Syncfusion.SfPullToRefr
esh.XForms" >
<ContentPage.BindingContext>
<local:InboxRepository x:Name="ViewModel" />
</ContentPage.BindingContext>
<ContentPage.Content>
<pullToRefresh:SfPullToRefresh x:Name="pullToRefresh"
PullingThreshold="60"
ProgressStrokeWidth="5"
ProgressBackgroundColor="CornflowerBlue"
ProgressStrokeColor="White"
RefreshContentWidth="40"
RefreshContentHeight="40"
TransitionMode="Push"
IsRefreshing="False">
<pullToRefresh:SfPullToRefresh.PullableContent>
<syncfusion:SfListView x:Name="listView"
ItemSize="80"
ItemsSource="{Binding InboxItems}"
ItemSpacing="0,0,0,1"
BackgroundColor="LightGray"
SelectionMode="None">
<syncfusion:SfListView.ItemTemplate>
<DataTemplate>
<Grid ColumnSpacing="5" BackgroundColor="White" Padding="5" >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="80" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<StackLayout Grid.Column="0"
HeightRequest="70"
WidthRequest="70"
VerticalOptions="Center"
HorizontalOptions="Center"
BackgroundColor="{Binding BackgroundColor}">
<Grid HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand" >
<Label Text="{Binding DisplayString}"
TextColor="White"
FontSize="25"
HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand"/>
</Grid>
</StackLayout>
<Grid Grid.Column="1" RowSpacing="0">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Label Text="{Binding Sender}"
TextColor="Black"

```

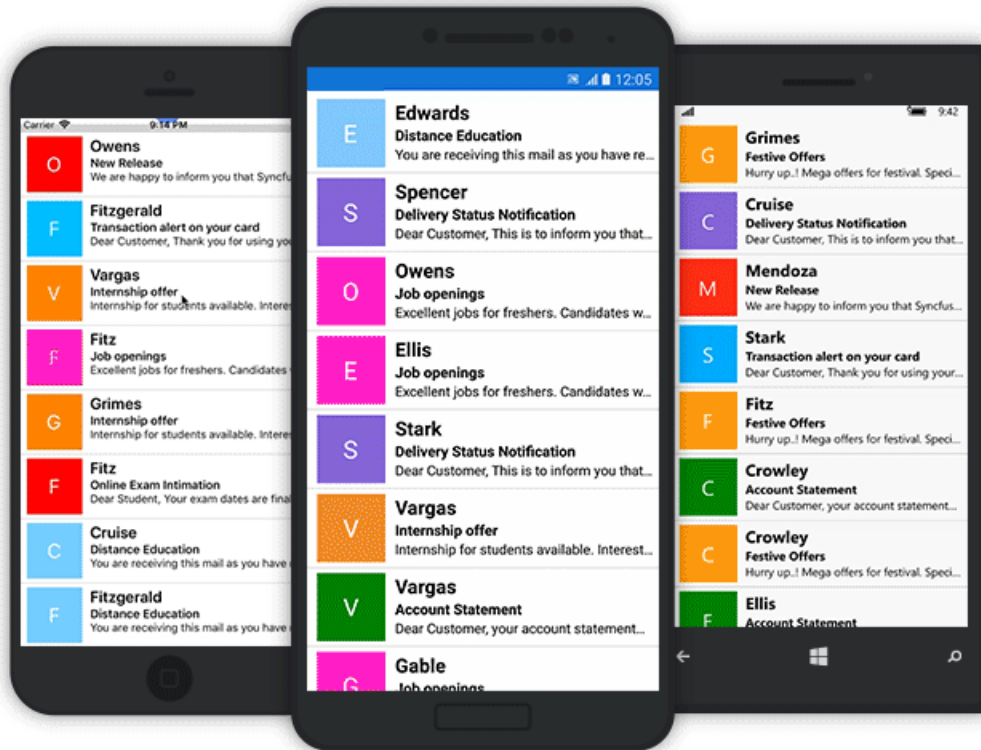


```
FontAttributes="Bold"
FontSize="20"/>
<Label Grid.Row="1"
FontSize = "15"
TextColor="Black"
FontAttributes="Bold"
Text="{Binding Subject}"
LineBreakMode="TailTruncation" />
<Label Grid.Row="2"
FontSize="14"
TextColor="Black"
Text="{Binding Details}"
LineBreakMode="TailTruncation" />
</Grid>
</Grid>
</DataTemplate>
</syncfusion:SfListView.ItemTemplate>
</syncfusion:SfListView>
</pullToRefresh:SfPullToRefresh.PullableContent>
</pullToRefresh:SfPullToRefresh>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using Syncfusion.SfPullToRefresh.XForms;
using Xamarin.Forms;
using Syncfusion.ListView.XForms;
namespace PullToRefreshSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            pullToRefresh.Refreshing += PullToRefresh_Refreshing; ;
        }
        private async void PullToRefresh_Refreshing(object sender, EventArgs e)
        {
            pullToRefresh.IsRefreshing = true;
            await Task.Delay(2000);
            ViewModel.RefreshItemSource();
            pullToRefresh.IsRefreshing = false;
        }
    }
}
```

If we run the above sample with TransitionMode as Push, the output will look like on iOS, Android and Windows Phone devices as shown below.



### Pulling and refreshing template

The SfPullToRefresh allows you set a template for pulling and refreshing the view. The pulling and refreshing a template can be set using the [SfPullToRefresh.PullingViewTemplate](#) and [SfPullToRefresh.RefreshingViewTemplate](#) properties, respectively.

Refer to the following code example in which a SfProgressBar is loaded in the pulling view template and refreshing view template.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SfPullToRefresh"
x:Class="SfPullToRefresh.MainPage"
xmlns:dataGrid="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
xmlns:pullToRefresh="clr-namespace:Syncfusion.SfPullToRefresh.XForms;assembly=Syncfusion.SfPullToRefresh.XForms">
<ContentPage.BindingContext>
<local:OrderInfoRepository x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<pullToRefresh:SfPullToRefresh x:Name="pullToRefresh"
RefreshContentHeight="50"
RefreshContentThreshold="40"
PullingThreshold="150"
```

```

RefreshContentWidth="50"
ProgressStrokeWidth="8"
TransitionMode="SlideOnTop"
IsRefreshing="False">
<pullToRefresh:SfPullToRefresh.PullableContent>
<dataGrid:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderInfoCollection}"
ColumnSizer="Star">
<dataGrid:SfDataGrid.Columns>
<dataGrid:GridTextColumn MappingName="OrderID" HeaderText="Order ID"/>
<dataGrid:GridTextColumn MappingName="CustomerID" HeaderText="Customer ID"/>
<dataGrid:GridTextColumn MappingName="Customer" HeaderText="Name"/>
<dataGrid:GridTextColumn MappingName="ShipCity" HeaderText="City"/>
<dataGrid:GridTextColumn MappingName="ShipCountry" HeaderText="Country"/>
</dataGrid:SfDataGrid.Columns>
</dataGrid:SfDataGrid>
</pullToRefresh:SfPullToRefresh.PullableContent>
</pullToRefresh:SfPullToRefresh>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

public partial class MainPage : ContentPage
{
    private SfCircularProgressBar progressbar;
    private SfBorder border;
    public MainPage()
    {
        InitializeComponent();
        this.progressbar = new SfCircularProgressBar();
        this.border = new SfBorder();
        this.border.BorderColor = Color.LightGray;
        this.border.BackgroundColor = Color.White;
        this.border.CornerRadius = 35;
        this.border.Content = this.progressbar;
        this.border.BorderWidth = 0.2;
        this.progressbar.SegmentCount = 10;
        this.progressbar.IndicatorInnerRadius = 0.5;
        this.progressbar.IndicatorOuterRadius = 0.7;
        this.progressbar.ShowProgressValue = true;
        this.progressbar.GapWidth = 0.5;
        this.progressbar.WidthRequest = 70;
        this.progressbar.HeightRequest = 55;
        this.progressbar.IndeterminateAnimationDuration = 750;
        var pullingTemplate = new DataTemplate(() =>
        {
            return new ViewCell { View = this.border };
        });
        this.pullToRefresh.Refreshing += this.PullToRefresh_Refreshing;
        this.pullToRefresh.Pulling += this.PullToRefresh_Pulling;
        this.pullToRefresh.PullingViewTemplate = pullingTemplate;
        this.pullToRefresh.RefreshingViewTemplate = pullingTemplate;
    }
    private async void pullToRefresh_Refreshing(object sender, EventArgs e)

```

```
{
    pullToRefresh.IsRefreshing = true;
    await Task.Delay(new TimeSpan(0, 0, 3));
    viewModel.ItemsSourceRefresh();
    pullToRefresh.IsRefreshing = false;
}

private void PullToRefresh_Pulling(object sender, PullingEventArgs e)
{
    this.progressBar.TrackInnerRadius = 0.8;
    this.progressBar.TrackOuterRadius = 0.1;
    this.progressBar.IsIndeterminate = false;
    this.progressBar.ProgressColor = Color.FromRgb(0, 124, 238);
    this.progressBar.TrackColor = Color.White;
    var absoluteProgress = Math.Abs(e.Progress);
    this.progressBar.Progress = absoluteProgress;
    this.progressBar.SetProgress(absoluteProgress, 1, Easing.CubicInOut);
}

private async void PullToRefresh_Refreshing(object sender, EventArgs e)
{
    this.pullToRefresh.IsRefreshing = true;
    await this.AnimateRefresh();
    this.progressBar.TrackInnerRadius = 0.1;
    this.progressBar.TrackOuterRadius = 0.9;
    this.viewModel.ItemsSourceRefresh();
    this.pullToRefresh.IsRefreshing = false;
}

private async Task AnimateRefresh()
{
    this.progressBar.Progress = 0;
    this.progressBar.IsIndeterminate = true;
    await Task.Delay(750);
    this.progressBar.ProgressColor = Color.Red;
    await Task.Delay(750);
    this.progressBar.ProgressColor = Color.Green;
    await Task.Delay(750);
    this.progressBar.ProgressColor = Color.Orange;
    await Task.Delay(750);
}
}
```

Order ID	Customer ID	Name	City	Country
1001	Maria Anders	ANATR	Mexico D.F.	Mexico
1002	Ana Trujillo	ANTON	Mexico D.F.	Mexico
1003	Ant Fuller	AROUT	London	UK
1004	Thomas Hardy	BERGS	Berlin	Sweden
1005	Lenny Lin	BLAUS	Mannheim	Germany
1006	John Carter	BLONP	Strasbourg	France
1007	Laura King	BOLID	Madrid	Spain
1008	Hanna Moos	BONAP	Marseille	France
1009	Lenny Lin	BOTTM	Tsawassen	Canada
1010	Hanna Moos	AROUT	London	UK
1011	John Carter	BLAUS	Mannheim	Germany
1012	Tim Adams	BLONP	Strasbourg	France
1013	Hanna Moos	AROUT	London	UK

You can download the sample code by clicking the following link: [Sample](#).

## Events

There are three built-in events in the PullToRefresh control namely:

1. Pulling
2. Refreshing
3. Refreshed

The Pulling event will be notified whenever the swipe gesture is started. This event will notify the listener each and every time until the refresh content height exceeds. When we release the gesture from pullable content, Refreshing event will be triggered. Now the refresh operation can be performed. Once the content is refreshed, we should set `SfPullToRefresh.IsRefreshing` to `false` to stop the animation. Once the animation is stopped the Refreshed event will be triggered to notify that the refreshing is completed.

### Pulling

Pulling event is triggered whenever you start pulling down on the `PullableContent` with `PullingEventArgs` that contains the following properties

- **Cancel** - You can cancel the pulling action based on the `Progress` value.
- **Progress** - Gets the progress completion value.

### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh"
    PullingEvent="PullToRefresh_Pulling" />
```

### C#

```
pullToRefresh.Pulling += PullToRefresh_Pulling;
private void PullToRefresh_Pulling(object sender, PullingEventArgs args)
{
    args.Cancel = false;
    var progress = args.Progress;
}
```

### Refreshing

Refreshing event is triggered once pointer is released. This event will occur till the `IsRefreshing` property is set as `false`.

### XML

```
<syncfusion:SfPullToRefresh x:Name="pullToRefresh"
    RefreshingEvent="PullToRefresh_Refreshing" />
```

### C#

```
pullToRefresh.Refreshing += PullToRefresh_Refreshing;
private async void PullToRefresh_Refreshing(object sender, EventArgs args)
{
    pullToRefresh.IsRefreshing = true;
    await Task.Delay(2000);
    pullToRefresh.IsRefreshing = false;
}
```

### Refreshed

Refreshed event is triggered once the `Refreshing` event is completed.

### XML

```
<syncfusion:SfPullToRefresh x:Name=" pullToRefresh"
    RefreshedEvent="PullToRefresh_Refreshed" />
```

### C#

```
pullToRefresh.Refreshed += PullToRefresh_Refreshed;
private void PullToRefresh_Refreshed(object sender, EventArgs args)
```

```
{  
}
```

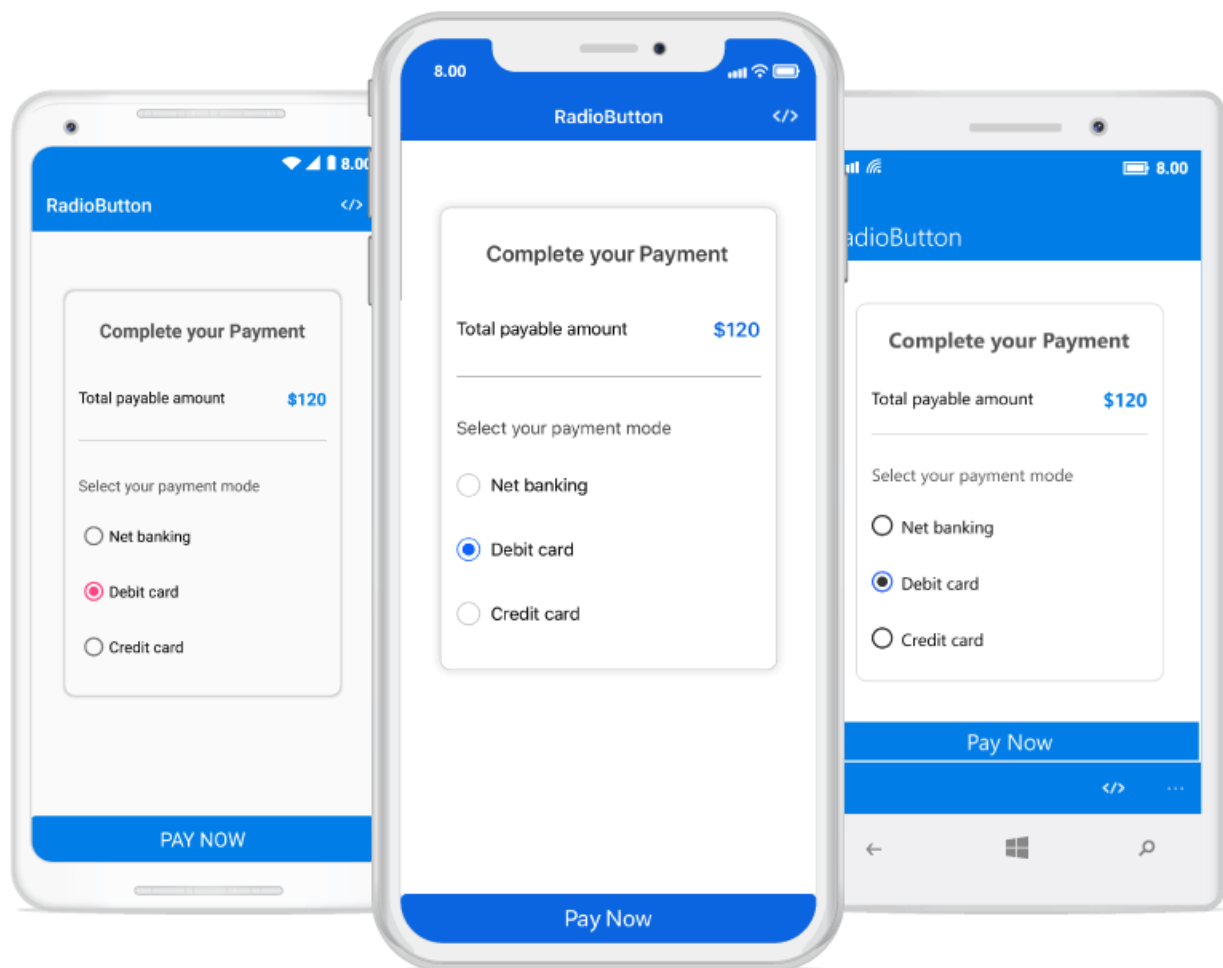
## SfRadioButton

### Overview

The radio button is a selection control that allows users to select one option from a set. The two states of radio button are checked and unchecked.

### Key features

- Allow users to select and clear the control by tapping.
- Supports radio button color and label text customization.



### Getting Started

This section explains the steps required to configure the **SfRadioButton** control in a real-time scenario and provides a walk-through on some of the customization features available in **SfRadioButton** control.

### Adding SfRadioButton reference

You can add SfRadioButton reference using one of the following methods:

#### Method 1: Adding SfRadioButton reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org). To add SfRadioButton to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Buttons](#), and then install it.

![Adding SfRadioButton reference from NuGet](Images/Adding SfRadioButton reference.png)

#### Note:

- Install the same version of SfRadioButton NuGet in all the projects.
- In addition, you need to install the [Syncfusion.Xamarin.Buttons.WPF]() package for Xamarin.Forms WPF platform only.

#### Method 2: Adding SfRadioButton reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfRadioButton control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfRadioButton assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Buttons.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
WPF	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.WPF.dll Syncfusion.Core.XForms.dll



	Syncfusion.Core.XForms.WPF.dll Syncfusion.Licensing.dll
--	--

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** After adding the assembly reference, an additional step is required for iOS and UWP projects. If you are adding the references from toolbox, this step is not needed.

#### *Additional step for iOS*

To launch **SfRadioButton** in iOS, call the **SfRadioButtonRenderer.Init()** in **FinishedLaunching** overridden method of **AppDelegate** class in iOS Project, as demonstrated in the following code example.

#### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    LoadApplication(new App());
    SfRadioButtonRenderer.Init();
    return base.FinishedLaunching(app, options);
}
```

#### *Additional step for UWP*

This step is required only if the application is deployed in Release mode with .NET native tool chain enabled. It is for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the **SfRadioButton** assembly at **OnLaunched** overridden method of the **App** class in UWP project is the suggested work around, as demonstrated in the following code example.

#### **C#**

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    .....
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies that your app uses
    assembliesToInclude.Add(typeof(SfRadioButtonRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    .....
}
```

*Additional step for WPF*

To launch the radio button in WPF, call the `SfRadioButtonRenderer.Init()` method in the `MainWindow` constructor of the `MainWindow` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

**C#**

```
public partial class MainWindow : FormsApplicationPage
{
    public MainWindow()
    {
        InitializeComponent();
        Forms.Init();
        Syncfusion.XForms.WPF.Buttons.SfRadioButtonRenderer.Init();
        LoadApplication(new App());
    }
}
```

*Create a Simple SfRadioButton*

The `SfRadioButton` control is configured entirely in C# code or by using XAML markup. The following steps explain how to create a `SfRadioButton` and configure its elements.

*Add namespace for referred assemblies***XML**

```
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
```

**C#**

```
using Syncfusion.XForms.Buttons;
```

*Refer SfRadioButton control with declared suffix name for Namespace***XML**

```
<?xml version="1.0" encoding="utf-8">
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<StackLayout>
<syncfusion:SfRadioButton x:Name="radioButton"/>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

**C#**

```
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace GettingStarted
```

```
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
StackLayout stackLayout = new StackLayout();
SfRadioButton radioButton = new SfRadioButton();
stackLayout.Children.Add(radioButton);
this.Content = stackLayout;
}
}
}
```

### Setting caption

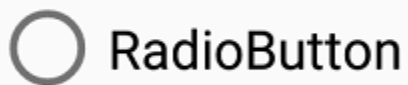
The radio button caption can be defined using the `Text` property of `SfRadioButton`. This caption normally describes the meaning of the radio button and it displays next to radio button.

#### XML

```
<syncfusion:SfRadioButton x:Name="radioButton" Text="RadioButton"/>
```

#### C#

```
SfRadioButton radioButton = new SfRadioButton();
radioButton.Text = "RadioButton";
```



This demo can be downloaded from this [link](#).

### Change the radio button state

The two different visual states of the `SfRadioButton` are:

- Checked
- Unchecked

You can change the state of the radio button using the `IsChecked` property of `SfRadioButton`. In the checked state, an inner circle is added to the visualization of radio button.

The radio buttons are used when there is a list of two or more options or group that are mutually exclusive and the user must select exactly one choice, such as “Select Gender” or “Choose the best option!”.

#### XML

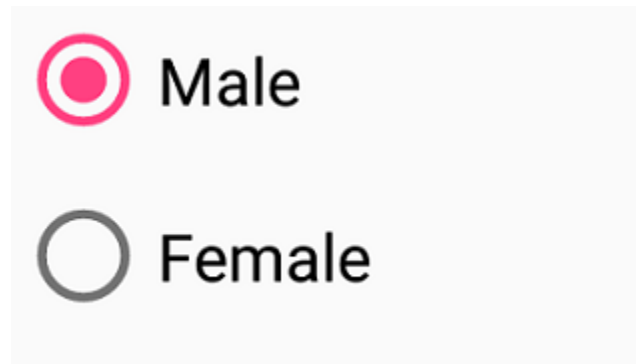
```
<syncfusion:SfRadioGroup x:Name="radioGroup">
<syncfusion:SfRadioButton x:Name="male" Text="Male" IsChecked="True"/>
```

```
<syncfusion:SfRadioButton x:Name="female" Text="Female"/>
</syncfusion:SfRadioGroup>
```

### C#

```
SfRadioGroup radioGroup = new SfRadioGroup();
SfRadioButton male = new SfRadioButton();
male.IsChecked = true;
male.Text = "Male";
SfRadioButton female = new SfRadioButton();
female.Text = "Female";
radioGroup.Children.Add(male);
radioGroup.Children.Add(female);
```

**Note:** SfRadioButtons are mutually exclusive among them when they are defined within SfRadioGroup.



This demo can be downloaded from this [link](#).

### Visual Customization

#### Customizing state color

The default state colors can be customized using the **CheckedColor** and **UncheckedColor** properties. The checked state color is updated to the **CheckedColor** property value when the state is changed to the checked. The unchecked state color is updated to the **UncheckedColor** property value when the state is changed to unchecked.

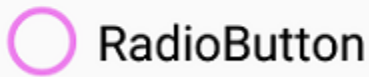
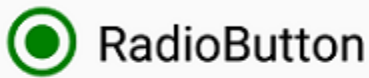
### XML

```
<syncfusion:SfRadioGroup x:Name="radioGroup">
<syncfusion:SfRadioButton x:Name="check" Text="RadioButton" IsChecked="True"
CheckedColor="Green"/>
<syncfusion:SfRadioButton x:Name="uncheck" Text="RadioButton"
UncheckedColor="Violet"/>
</syncfusion:SfRadioGroup>
```

### C#

```
SfRadioGroup radioGroup = new SfRadioGroup();
SfRadioButton check = new SfRadioButton();
check.Text = "RadioButton";
check.IsChecked = true;
```

```
check.CheckedColor = Color.Green;
SfRadioButton uncheck = new SfRadioButton();
uncheck.Text = "RadioButton";
uncheck.UncheckedColor = Color.Violet;
radioGroup.Children.Add(check);
radioGroup.Children.Add(uncheck);
```



#### Setting caption text appearance

You can customize the display text appearance of the **SfRadioButton** control using the following properties:

- **TextColor** : Changes the color of the text.
- **HorizontalTextAlignment**: Changes the horizontal alignment of the caption text.
- **FontFamily**: Changes the font family of the caption text.
- **FontAttributes**: Sets font attributes(bold/italic/none) of the caption text.
- **FontSize**: Sets font size of the caption text.

#### XML

```
<syncfusion:SfRadioButton x:Name="radioButton" Text="RadioButton"
IsChecked="True" TextColor="Violet" HorizontalTextAlignment="Center"
FontFamily="Arial" FontAttributes="Bold" FontSize="20"/>
```

#### C#

```
SfRadioButton radioButton = new SfRadioButton();
radioButton.Text = "RadioButton";
radioButton.IsChecked = true;
radioButton.TextColor = Color.Violet;
radioButton.HorizontalTextAlignment = TextAlignment.Center;
radioButton.FontFamily = "Arial";
radioButton.FontAttributes = FontAttributes.Bold;
radioButton.FontSize = 20;
```



RadioButton

### LineBreakMode

The `LineBreakMode` allows you to wrap or truncate the text. The default value of this property is `NoWrap`. The following other options are available in `LineBreakMode`:

- `NoWrap` - Avoids the text wrap.
- `WordWrap` - Wraps the text by words.
- `CharacterWrap` - Wraps the text by character.
- `HeadTruncation` - Truncates the text at the start.
- `MiddleTruncation` - Truncates the text at the center.
- `TailTruncation` - Truncates the text at the end.

This demo can be downloaded from this [link](#).

### Visual States

The visual of Radio Button can be customized using `VisualStates`. The [SfRadioButton](#) control contains the following two visual states:

- Checked
- Unchecked

### XML

```
<buttons:SfRadioButton Text="Radio Button">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="Checked">
        <VisualState.Setters>
          <Setter Property="TextColor" Value="Accent"/>
          <Setter Property="BackgroundColor" Value="#8bc5fb"/>
          <Setter Property="CheckedColor" Value="Accent"/>
        </VisualState.Setters>
      </VisualState>
      <VisualState x:Name="Unchecked">
        <VisualState.Setters>
          <Setter Property="TextColor" Value="#ea3737"/>
          <Setter Property="BackgroundColor" Value="#f6acac"/>
          <Setter Property="UncheckedColor" Value="#ea3737"/>
        </VisualState.Setters>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</buttons:SfRadioButton>
```

### C#

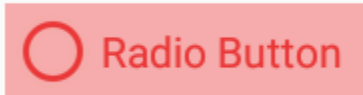
```
SfRadioButton radioButton = new SfRadioButton { Text = "Radio Button" };
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState checkedState = new VisualState
{
    Name = "Checked"
};
```

```
checkedState.Setters.Add(new Setter { Property =
SfRadioButton.TextColorProperty, Value = Color.Accent });
checkedState.Setters.Add(new Setter { Property =
SfRadioButton.BackgroundColorProperty, Value = Color.FromHex("#8bc5fb") });
checkedState.Setters.Add(new Setter { Property =
SfRadioButton.CheckedColorProperty, Value = Color.Accent });
VisualState uncheckedState = new VisualState
{
    Name = "Unchecked"
};
uncheckedState.Setters.Add(new Setter { Property =
SfRadioButton.TextColorProperty, Value = Color.FromHex("#ea3737") });
uncheckedState.Setters.Add(new Setter { Property =
SfRadioButton.BackgroundColorProperty, Value = Color.FromHex("#f6acac") });
uncheckedState.Setters.Add(new Setter { Property =
SfRadioButton.UncheckedColorProperty, Value = Color.FromHex("#ea3737") });
commonStateGroup.States.Add(checkedState);
commonStateGroup.States.Add(uncheckedState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(radioButton, visualStateGroupList);
```

#### Checked visual state:



#### Unchecked visual state:



## Grouping

### Group Key

The [GroupKey](#) in [SfRadioButton](#) allows you to group a set of radio buttons present inside any layout. By grouping in this way, you can select only one radio button that comes under same [GroupKey](#) at a time.

- [CheckedItem](#) - Gets the current checked item from radio group.

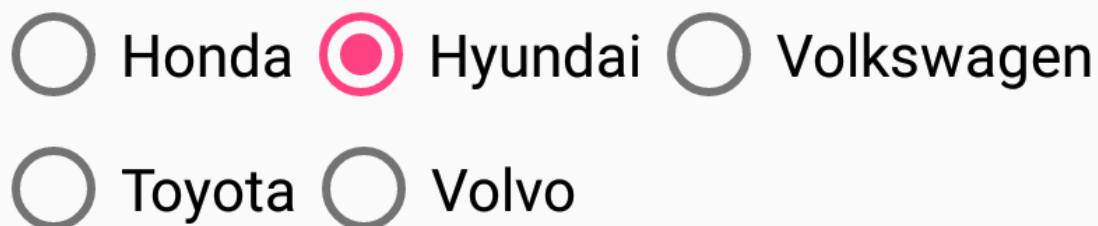
### XML

```
<ContentPage.Resources>
<syncfusion:SfRadioGroupKey x:Key="carBrand" />
</ContentPage.Resources>
<FlexLayout Wrap="Wrap" AlignItems="Start" AlignContent="Start">
<syncfusion:SfRadioButton Text="Honda" GroupKey="{StaticResource
carBrand}"/>
```

```
<syncfusion:SfRadioButton Text="Hyundai" GroupKey="{StaticResource
carBrand}"/>
<syncfusion:SfRadioButton Text="Volkswagen" GroupKey="{StaticResource
carBrand}"/>
<syncfusion:SfRadioButton Text="Toyota" GroupKey="{StaticResource
carBrand}"/>
<syncfusion:SfRadioButton Text="Volvo" GroupKey="{StaticResource
carBrand}"/>
</FlexLayout>
```

**C#**

```
SfRadioGroupKey carBrand = new SfRadioGroupKey();
SfRadioButton honda = new SfRadioButton();
honda.Text = "Honda";
honda.GroupKey = carBrand;
SfRadioButton hyundai = new SfRadioButton();
hyundai.Text = "Hyundai";
hyundai.GroupKey = carBrand;
SfRadioButton volkswagen = new SfRadioButton();
volkswagen.Text = "Volkswagen";
volkswagen.GroupKey = carBrand;
SfRadioButton toyota = new SfRadioButton();
toyota.Text = "Toyota";
toyota.GroupKey = carBrand;
SfRadioButton volvo = new SfRadioButton();
volvo.Text = "Volvo";
volvo.GroupKey = carBrand;
FlexLayout flexLayout = new FlexLayout()
{
    Wrap = FlexWrap.Wrap,
    AlignContent = FlexAlignContent.Start,
    AlignItems = FlexAlignItems.Start
};
flexLayout.Children.Add(honda);
flexLayout.Children.Add(hyundai);
flexLayout.Children.Add(volkswagen);
flexLayout.Children.Add(toyota);
flexLayout.Children.Add(volvo);
```

*CheckedChanged event*

The [CheckedChanged](#) event of [SfRadioGroupKey](#) occurs when a checked item is changed. The argument contains the following information:



- [PreviousItem](#) – Gets the previously checked radio button from group.
- [CurrentItem](#) – Gets the currently checked radio button from group.

### SfRadioGroup

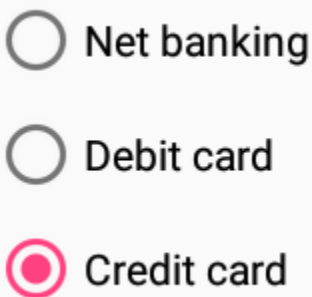
[SfRadioGroup](#) is a container that contains a set of radio buttons. When you select a radio button in a radio group, all other items will be deselected automatically. At a time, you can select only one radio button from the same radio group. It also contains the [CheckedChanged](#) event and the [CheckedItem](#) property.

#### XML

```
<syncfusion:SfRadioGroup>
<syncfusion:SfRadioButton Text="Net banking" />
<syncfusion:SfRadioButton Text="Debit card" />
<syncfusion:SfRadioButton Text="Credit card" />
</syncfusion:SfRadioGroup>
```

#### C#

```
SfRadioGroup radioGroup = new SfRadioGroup();
SfRadioButton netBanking = new SfRadioButton();
netBanking.Text = "Net banking";
SfRadioButton debitCard = new SfRadioButton();
debitCard.Text = "Debit card";
SfRadioButton creditCard = new SfRadioButton();
creditCard.Text = "Credit card";
radioGroup.Children.Add(netBanking);
radioGroup.Children.Add(debitCard);
radioGroup.Children.Add(creditCard);
```



### Event

#### StateChanged event

Occurs when the value(state) of the `IsChecked` property is changed by either touching the check box or setting the value to the `IsChecked` property using XAML or C# code. The event arguments are of type `StateChangedEventArgs` and expose the following property:

- `IsChecked`: The new value(state) of the `IsChecked` property.

#### XML

```

<syncfusion:SfRadioGroup x:Name="radioGroup">
<syncfusion:SfRadioButton x:Name="check" Text="Checked State"
IsChecked="True" StateChanged="RadioButton_StateChanged"/>
<syncfusion:SfRadioButton x:Name="uncheck" Text="Unchecked State"
StateChanged="RadioButton_StateChanged"/>
</syncfusion:SfRadioGroup>
private void RadioButton_StateChanged(object sender, StateChangedEventArgs
e)
{
if (e.IsChecked.HasValue && e.IsChecked.Value)
{
(sender as SfRadioButton).Text = "Checked State";
}
else if (e.IsChecked.HasValue && !e.IsChecked.Value)
{
(sender as SfRadioButton).Text = "Unchecked State";
}
}
}

```

**C#**

```

SfRadioGroup radioGroup = new SfRadioGroup();
SfRadioButton check = new SfRadioButton();
check.Text = "Checked State";
check.IsChecked = true;
check.StateChanged += RadioButton_StateChanged;
SfRadioButton uncheck = new SfRadioButton();
uncheck.Text = "Unchecked State";
uncheck.StateChanged += RadioButton_StateChanged;
radioGroup.Children.Add(check);
radioGroup.Children.Add(uncheck);
private void RadioButton_StateChanged(object sender, StateChangedEventArgs
e)
{
if (e.IsChecked.HasValue && e.IsChecked.Value)
{
(sender as SfRadioButton).Text = "Checked State";
}
else if (e.IsChecked.HasValue && !e.IsChecked.Value)
{
(sender as SfRadioButton).Text = "Unchecked State";
}
}
}


```



Checked State



Unchecked State

 Unchecked State

 Checked State

This demo can be downloaded from this [link](#).

## SfRadialMenu

### Overview

The Essential Xamarin SfRadialMenu displays a hierarchical menu in a circular layout, which is optimized for touch enabled devices. Typically, it is used as a context menu, and it can expose more menu items in the same space than traditional menus.

### Key features

Key features in SfRadialMenu:

- Drag — The SfRadialMenu can be floated over the layout to avoid obscuring the content behind it.
- Rotation — SfRadialMenu supports rotating items.
- FontIcon — The built-in icon font option helps users add vector images that prevent the control from experiencing any image glitches often faced with traditional image icons.
- Custom view — SfRadialMenu supports the custom view like image.
- Custom segments — Complete customization options for the menu and its items, such as coloring, sizing, placement, and shapes, using the segmentation option.
- Auto arrange — SfRadialMenu supports automatic item arrangement.
- Custom arrange — SfRadialMenu has options to place items as users needed.



## Getting Started

This section explains the steps required to launch the radial menu with hierarchical items that can be used as mobile phone system settings. This section covers only the minimal features that needed to get started with the radial menu.

### Adding SfRadialMenu reference

You can add SfRadialMenu reference using one of the following methods:

#### Method 1: Adding SfRadialMenu reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfRadialMenu to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfRadialMenu](#), and then install it.

![Adding SfRadialMenu reference from NuGet](images/Adding SfRadialMenu reference.png)

---

**Note:** Install the same version of SfRadialMenu NuGet in all the projects.

---

#### Method 2: Adding SfRadialMenu reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfRadialMenu control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfRadialMenu assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfRadialMenu.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfRadialMenu.Android.dll Syncfusion.SfRadialMenu.XForms.Android.dll Syncfusion.SfRadialMenu.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfRadialMenu.iOS.dll Syncfusion.SfRadialMenu.XForms.iOS.dll Syncfusion.SfRadialMenu.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfRadialMenu.UWP.dll Syncfusion.SfRadialMenu.XForms.UWP.dll Syncfusion.SfRadialMenu.XForms.dll Syncfusion.SfShared.UWP.dll Syncfusion.Core.XForms.dll

	Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
--	--

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

#### *Additional step for iOS*

To launch [SfRadialMenu](#) in iOS, call `SfRadialMenuRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class in the iOS project as demonstrated in the following code example.

#### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    SfRadialMenuRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

#### *Additional step for UWP*

This step is required only if the application is deployed in release mode with .NET native tool chain enabled. It is needed for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the [SfRadialMenu](#) assembly at the `OnLaunched` overridden method of the `App` class in UWP project is the suggested workaround. The following code example demonstrates how to initialize the [SfRadialMenu](#) assembly.

#### **C#**

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfRadialMenuRenderer).GetTypeInfo().Assembly);
    ;
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Creating a simple radial menu

The [SfRadialMenu](#) control is configured entirely in C# code or in XAML markup. The following steps explain how to create [SfRadialMenu](#) and configure its elements:

#### *Create the project*

Create a new BlankApp (Xamarin.Forms.Portable) application in Xamarin Studio or Visual Studio for Xamarin.Forms.

#### *Adding radial menu in Xamarin.Forms*

1. Add the required assembly references to the pcl and renderer projects.
2. Add namespace for the referred assemblies.

#### **XML**

```
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
```

#### **C#**

```
using Syncfusion.XForms.SfRadialMenu;
```

3. Set the radial menu control as content to the ContentPage.

#### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<radialMenu:SfRadialMenu x:Name="radialMenu"/>
</ContentPage.Content>
</ContentPage>
```

#### **C#**

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            SfRadialMenu radialMenu = new SfRadialMenu();
            this.Content = radialMenu;
        }
    }
}
```

```
}
}
```

*Adding radial menu with items*

### **XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu x:Name="radialMenu"
CenterButtonText="Edit"
CenterButtonFontSize="15">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Cut"
FontSize="15"/>
<radialMenu:SfRadialMenuItem Text="Copy"
FontSize="15"/>
<radialMenu:SfRadialMenuItem Text="Paste"
FontSize="15"/>
<radialMenu:SfRadialMenuItem Text="Crop"
FontSize="15"/>
<radialMenu:SfRadialMenuItem Text="Paint"
FontSize="15"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### **C#**

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu()
{
CenterButtonText = "Edit",
CenterButtonFontSize = 15
};
ObservableCollection<SfRadialMenuItem> itemCollection = new
ObservableCollection<SfRadialMenuItem>();
itemCollection.Add(new SfRadialMenuItem()
{
Text = "Cut",
FontSize = 15
});
}
```

```
});  
itemCollection.Add(new SfRadialMenuItem()  
{  
    Text = "Copy",  
    FontSize = 15  
});  
itemCollection.Add(new SfRadialMenuItem()  
{  
    Text = "Paste",  
    FontSize = 15  
});  
itemCollection.Add(new SfRadialMenuItem()  
{  
    Text = "Crop",  
    FontSize = 15  
});  
itemCollection.Add(new SfRadialMenuItem()  
{  
    Text = "Paint",  
    FontSize = 15  
});  
radialMenu.Items = itemCollection;  
this.Content = radialMenu;  
}  
}  
}
```



We have attached sample for reference. You can download the sample from the following link.

Sample link:[GettingStarted](#)

### Populating Items

This section explains the ways about populating items through radial menu item and item source with item template.

#### Through radial menu items

By passing a collection of `SfRadialMenuItem`, you can get the view of `SfRadialMenu` control. The radial menu item class provides various options to customize the items by giving custom views, font icons, and images. You can add radial menu items by hierarchy.

#### *Adding outer rim items of radial menu*

The following code snippet demonstrates how to add the outer rim items of radial menu.



## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

## C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
            });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
            });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}
```

### *Adding nested items of radial menu*

You can populate the nested levels of items within a menu to group similar actions based on their result. For example, you can group the clipboard operations by adding a clipboard as a main menu and cut, copy, and paste as its children.

The following code snippet demonstrates how to add the nested items of radial menu.

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu CenterButtonText="Edit"
CenterButtonFontSize="12">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold"
FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy"
FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo"
FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste"
FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12">
<radialMenu:SfRadialMenuItem.Items>
<radialMenu:SfRadialMenuItem Text="Font"
FontSize="12"
ItemWidth="50"/>
<radialMenu:SfRadialMenuItem Text="Gradient"
FontSize="12"
ItemWidth="50"/>
<radialMenu:SfRadialMenuItem Text="Highlight"
FontSize="12"
ItemWidth="50"/>
</radialMenu:SfRadialMenuItem.Items>
</radialMenu:SfRadialMenuItem>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

**C#**

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            string[] mainItem = new string[] { "Bold", "Copy", "Paste", "Undo", "Color"
        };
            string[] colorItem = new string[] { "Font", "Gradient", "Highlight" };
            SfRadialMenu radialMenu = new SfRadialMenu();
            // Adding radial menu outer rim items.
            for (int i = 0; i < 5; i++)
```

```

{
    SfRadialMenuItem mainMenuItems = new SfRadialMenuItem();
    mainMenuItems.Text = mainItem[i];
    mainMenuItems.FontSize = 12;
    radialMenu.Items.Add(mainMenuItems);
}
// Adding inner rim items.
for (int i = 0; i < 3; i++)
{
    SfRadialMenuItem colorSubMenuItem = new SfRadialMenuItem();
    colorSubMenuItem.Text = colorItem[i];
    colorSubMenuItem.FontSize = 12;
    colorSubMenuItem.ItemWidth = 50;
    radialMenu.Items[4].Items.Add(colorSubMenuItem);
}
}
}
}

```



### Through ItemsSource and ItemTemplate

Using ItemsSource, objects of any class can be given as items for SfRadialMenu. The views corresponding to the objects can be set using the ItemTemplate property. A simple usage of ItemTemplate and ItemsSource to display a default image and name of users is shown in the following code.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
<ContentPage.BindingContext>
<local:EmployeeViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<radialMenu:SfRadialMenu
x:Name="radial_Menu"
ItemsSource="{Binding EmployeeCollection}">

```

```

<radialMenu:SfRadialMenu.ItemTemplate>
<DataTemplate>
<StackLayout>
<Image Source="user.png"
HorizontalOptions="Center"
WidthRequest="15"/>
<Label Text="{Binding EmployeeName}"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"/>
</StackLayout>
</DataTemplate>
</radialMenu:SfRadialMenu.ItemTemplate>
</radialMenu:SfRadialMenu>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
}
}
public class EmployeeModel
{
private string employeeName;
public string EmployeeName
{
get { return employeeName; }
set { employeeName = value; }
}
}
public class EmployeeViewModel
{
private ObservableCollection<EmployeeModel> employeeCollection = new
ObservableCollection<EmployeeModel>();
public ObservableCollection<EmployeeModel> EmployeeCollection
{
get { return employeeCollection; }
set { employeeCollection = value; }
}
public EmployeeViewModel()
{
EmployeeCollection.Add(new EmployeeModel() { EmployeeName = "Alex" });
EmployeeCollection.Add(new EmployeeModel() { EmployeeName = "Lee" });
EmployeeCollection.Add(new EmployeeModel() { EmployeeName = "Ben" });
EmployeeCollection.Add(new EmployeeModel() { EmployeeName = "Carl" });
EmployeeCollection.Add(new EmployeeModel() { EmployeeName = "Yang" });
}
}

```

```
}
}
```

### Animation duration

Duration of animation in radial menu can be changed using the [AnimationDuration](#) property. It is used to change the speed of opening and closing of radial menu.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu AnimationDuration="1000">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu()
{
AnimationDuration = 1000
};
ObservableCollection<SfRadialMenuItem> itemCollection = new
ObservableCollection<SfRadialMenuItem>();
itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
});
itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
});
radialMenu.Items = itemCollection;
this.Content = radialMenu;
```

```

}
}
}

```

IsOpen

The [IsOpen](#) property indicates whether the radial menu is in open or close state.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu IsOpen="True">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

### C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                IsOpen = true
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
            });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
            });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}

```

```
}
}
```

### Selection color of radial menu

Selection color of an item can be changed using the [SelectionColor](#) property of radial menu.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu SelectionColor="#FFFF33">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                SelectionColor = Color.FromHex("#FFFF33")
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
            });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
            });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}
```

```
}

```

### Separator thickness and color in radial menu

Thickness of strip between the two items can be changed using the [SeparatorThickness](#) property and the color of strip can be changed using the [SeparatorColor](#) property.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu SeparatorThickness="5"
SeparatorColor="#FF1493">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu()
{
SeparatorThickness = 50,
SeparatorColor = Color.FromHex("#FF1493")
};
ObservableCollection<SfRadialMenuItem> itemCollection = new
ObservableCollection<SfRadialMenuItem>();
itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
});
itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
});
radialMenu.Items = itemCollection;
```



```

this.Content = radialMenu;
}
}
}

```

Rim color and rim radius in radial menu

The radius of rim can be changed using the [RimRadius](#) property and the color of rim can be changed using the [RimColor](#) property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu RimRadius="150"
RimColor="#FF1493">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

### C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu()
{
RimRadius = 150,
RimColor = Color.FromHex("#FF1493")
};
ObservableCollection<SfRadialMenuItem> itemCollection = new
ObservableCollection<SfRadialMenuItem>();
itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
});
itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });

```

```
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
});
radialMenu.Items = itemCollection;
this.Content = radialMenu;
}
}
}
```

### DisplayMemberPath

The control is populated with a list of employees, and the employee model contains two properties: ID and EmployeeName. So, it is necessary to intimate by which property it should display to the items. The [DisplayMemberPath](#) property specifies the property path.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<ContentPage.BindingContext>
<local:EmployeeViewModel/>
</ContentPage.BindingContext>
<radialMenu:SfRadialMenu
ItemsSource="{Binding EmployeeCollection}"
DisplayMemberPath="EmployeeName"/>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
}
}
public class Employee
{
private int id;
public int ID
{
get { return id; }
set { id = value; }
}
private string employeeName;
public string EmployeeName
```

```

{
    get { return employeeName; }
    set { employeeName = value; }
}
}
public class EmployeeViewModel
{
    private ObservableCollection<Employee> employeeCollection;
    public ObservableCollection<Employee> EmployeeCollection
    {
        get { return employeeCollection; }
        set { employeeCollection = value; }
    }
    public EmployeeViewModel()
    {
        employeeCollection = new ObservableCollection<Employee>();
        employeeCollection.Add(new Employee() { ID = 1, EmployeeName = "Eric" });
        employeeCollection.Add(new Employee() { ID = 2, EmployeeName = "James" });
        employeeCollection.Add(new Employee() { ID = 3, EmployeeName = "Jacob" });
        employeeCollection.Add(new Employee() { ID = 4, EmployeeName = "Lucas" });
    }
}
}

```

## SfRadialMenuItem Customization

The `SfRadialMenuItem` class provides various options to customize the items by giving Custom Views, FontIcons, and Images. You can add radial menu items by hierarchy. To add a `SfRadialMenuItem` with `SfRadialMenu`, create an instance of `SfRadialMenuItem`, and add it to the `Items` property, which is available in `SfRadialMenu`.

### Items

The `Items` property populates the items to the inner rim of `SfRadialMenu` when tapping the items of outer rim.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:RadialSample"
    xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
    x:Class="RadialSample.MainPage">
    <radialMenu:SfRadialMenu>
        <radialMenu:SfRadialMenu.Items>
            <radialMenu:SfRadialMenuItem Text="Copy">
                <radialMenu:SfRadialMenuItem.Items>
                    <radialMenu:SfRadialMenuItem Text="Paste"/>
                </radialMenu:SfRadialMenuItem.Items>
            </radialMenu:SfRadialMenuItem>
        </radialMenu:SfRadialMenu.Items>
    </radialMenu:SfRadialMenu>
</ContentPage>

```

## C#

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem() { Text = "Copy" });
            radialMenu.Items[0].Items.Add(new SfRadialMenuItem() { Text = "Paste" });
            this.Content = radialMenu;
        }
    }
}
```

## Text

The [Text](#) property provides text to the [SfRadialMenuItem](#).

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
    <radialMenu:SfRadialMenu>
        <radialMenu:SfRadialMenu.Items>
            <radialMenu:SfRadialMenuItem Text="Cut"/>
        </radialMenu:SfRadialMenu.Items>
    </radialMenu:SfRadialMenu>
</ContentPage>
```

## C#

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem() { Text = "Copy" });
            this.Content = radialMenu;
        }
    }
}
```

## ItemHeight

The [ItemHeight](#) changes the height of the [SfRadialMenuItem](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Cut"
ItemHeight="10"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                Text = "Cut",
                ItemHeight = 10
            });
            this.Content = radialMenu;
        }
    }
}
```

## ItemWidth

The [ItemWidth](#) property changes the width of the [SfRadialMenuItem](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
```

```
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Cut"
ItemWidth="10"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

## C#

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                Text = "Cut",
                ItemWidth = 10
            });
            this.Content = radialMenu;
        }
    }
}
```

## BackgroundColor

The [BackgroundColor](#) property changes the background color of the [SfRadialMenuItem](#).

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Cut"
BackgroundColor="Pink"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

**C#**

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                Text = "Cut",
                BackgroundColor = Color.Pink
            });
            this.Content = radialMenu;
        }
    }
}
```

**FontFamily**

The [FontFamily](#) property changes the font family of text in [SfRadialMenuItem](#).

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
    <radialMenu:SfRadialMenu>
        <radialMenu:SfRadialMenu.Items>
            <radialMenu:SfRadialMenuItem Text="Cut"
FontFamily="Times New Roman"/>
        </radialMenu:SfRadialMenu.Items>
    </radialMenu:SfRadialMenu>
</ContentPage>
```

**C#**

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
```

```
{
    Text = "Cut",
    FontFamily = "Times New Roman"
});
this.Content = radialMenu;
}
}
```

### FontSize

The [FontSize](#) property changes the text size in the [SfRadialMenuItem](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Cut"
FontSize="20"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                Text = "Cut",
                FontSize = 20
            });
            this.Content = radialMenu;
        }
    }
}
```

### FontAttribute

The [FontAttributes](#) property changes the font attributes of text in [SfRadialMenuItem](#).



**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Cut"
FontAttributes="Bold"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

**C#**

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                Text = "Cut",
                FontAttributes = FontAttributes.Bold
            });
            this.Content = radialMenu;
        }
    }
}
```

**Image**

The [Image](#) property provides image to the [SfRadialMenuItem](#).

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Image="user.png"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

```

</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

**C#**

```

using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                Image = "user.png"
            });
            this.Content = radialMenu;
        }
    }
}

```

**VisibleSegmentCount**

The [VisibleSegmentsCount](#) property provides the number of items to be displayed in radial menu item.

**XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
    <radialMenu:SfRadialMenu>
        <radialMenu:SfRadialMenu.Items>
            <radialMenu:SfRadialMenuItem VisibleSegmentsCount="3"/>
        </radialMenu:SfRadialMenu.Items>
    </radialMenu:SfRadialMenu>
</ContentPage>

```

**C#**

```

using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {

```

```
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu();
radialMenu.Items.Add(new SfRadialMenuItem()
{
    VisibleSegmentsCount = 3
});
this.Content = radialMenu;
}
}
```

### BackgroundImage

The [BackgroundImage](#) property provides the background image to the [SfRadialMenuItem](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem BackgroundImage="facebook.png"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                BackgroundImage = "facebook.png"
            });
            this.Content = radialMenu;
        }
    }
}
```

You can download the sample for reference from the following link: [Sample for adding BackgroundImage](#).

## FontIconText

The [FontIconText](#) property provides font icon to the [SfRadialMenuItem](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem FontIconText="&#xe734;"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                FontIconText = "\uE734"
            });
            this.Content = radialMenu;
        }
    }
}
```

## IconFontColor

The [IconFontColor](#) property changes the color of font icon in [SfRadialMenuItem](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
```

```
<radialMenu:SfRadialMenuItem FontIconText="&#xe734;"
IconFontColor="#313131"/>
</radialMenu:SfRadialMenuItem>
</radialMenu:SfRadialMenu>
</ContentPage>
```

**C#**

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                FontIconText = "\uE734",
                IconFontColor = Color.FromHex("#313131")
            });
            this.Content = radialMenu;
        }
    }
}
```

**IconFontSize**

The [IconFontSize](#) property changes the size of font icon in the [SfRadialMenuItem](#).

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
    <radialMenu:SfRadialMenu>
        <radialMenu:SfRadialMenu.Items>
            <radialMenu:SfRadialMenuItem FontIconText="&#xe734;"
IconFontColor="#313131"
IconFontSize="10"/>
        </radialMenu:SfRadialMenu.Items>
    </radialMenu:SfRadialMenu>
</ContentPage>
```

**C#**

```
using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample
```

```

{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu();
radialMenu.Items.Add(new SfRadialMenuItem()
{
FontIconText = "\uE734",
IconFontColor = Color.FromHex("#313131"),
IconFontSize = 10
});
this.Content = radialMenu;
}
}
}

```

### IconFontFamily

The [IconFontFamily](#) property changes the font family of font icon in the [SfRadialMenuItem](#).

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String"
x:Key="customfontfamily"
iOS="Segoe MDL2 Assets"
Android="radialmenu_Segoe MDL2 Assets.ttf"
UWP="radialmenu_Segoe MDL2_Assets.ttf#Segoe MDL2 Assets" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem FontIconText="&#xe734;"
IconFontColor="#313131"
IconFontSize="10"
IconFontFamily="{StaticResource customfontfamily}"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage.Content>
</ContentPage>

```

### C#

```

using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;

```

```

namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            radialMenu.Items.Add(new SfRadialMenuItem()
            {
                FontIconText = "\uE734",
                IconFontColor = Color.FromHex("#313131"),
                IconFontSize = 10,
                IconFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets"
                : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2
                Assets.ttf" : "radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets"
            });
            this.Content = radialMenu;
        }
    }
}

```

**Note:** <https://xamarinhelp.com/custom-fonts-xamarin-forms/> provides how to add the ttf file in each platform

View

The [View](#) provides custom view to the [SfRadialMenuItem](#).

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
    <radialMenu:SfRadialMenu>
        <radialMenu:SfRadialMenu.Items>
            <radialMenu:SfRadialMenuItem>
                <radialMenu:SfRadialMenuItem.View>
                    <Label Text="Cut"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"/>
                </radialMenu:SfRadialMenuItem.View>
            </radialMenu:SfRadialMenuItem>
        </radialMenu:SfRadialMenu.Items>
    </radialMenu:SfRadialMenu>
</ContentPage>

```

### C#

```

using Syncfusion.SfRadialMenu.XForms;
using Xamarin.Forms;
namespace RadialSample

```

```

{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu();
radialMenu.Items.Add(new SfRadialMenuItem()
{
View = new Label()
{
Text = "Cut",
HorizontalTextAlignment = TextAlignment.Center,
VerticalTextAlignment = TextAlignment.Center
}
});
this.Content = radialMenu;
}
}
}

```

## Layout types

There are two different layout types available for radial menu:

- Default
- Custom

Both the layout types divide the available space equally among all the children in the circular panel.

### Default

Number of segments in the panel is determined by children count in the level. Hence, segment count in each hierarchical level differs, radial menu items are arranged in the sequential order as added in the radial menu.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu LayoutType="Default">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```



**XML**

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                LayoutType = LayoutType.Default
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12 });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}

```

**Custom**

The number of segments in the panel is determined using the `VisibleSegmentsCount` property. Since the segment count in all the hierarchical levels are same, radial menu items are arranged in any order based on the `SegmentIndex` property.

**XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu LayoutType="Custom">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>

```

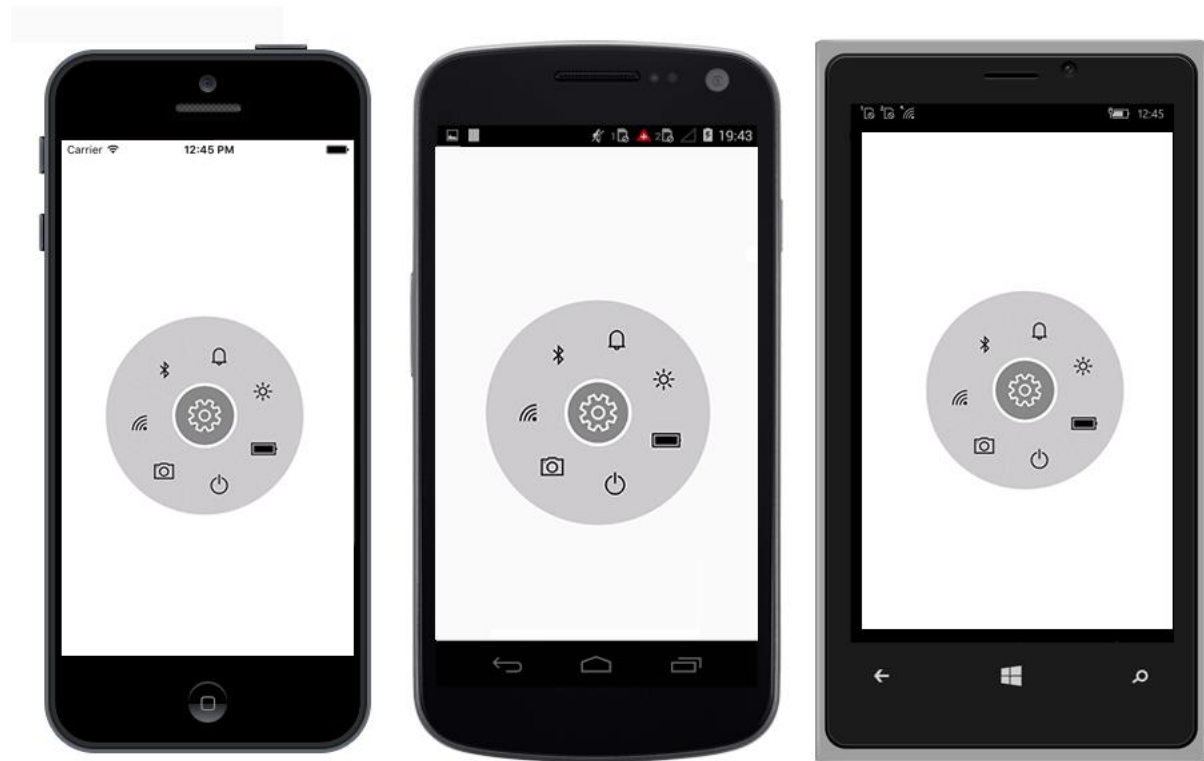
```
</radialMenu:SfRadialMenu>
</ContentPage>
```

### XML

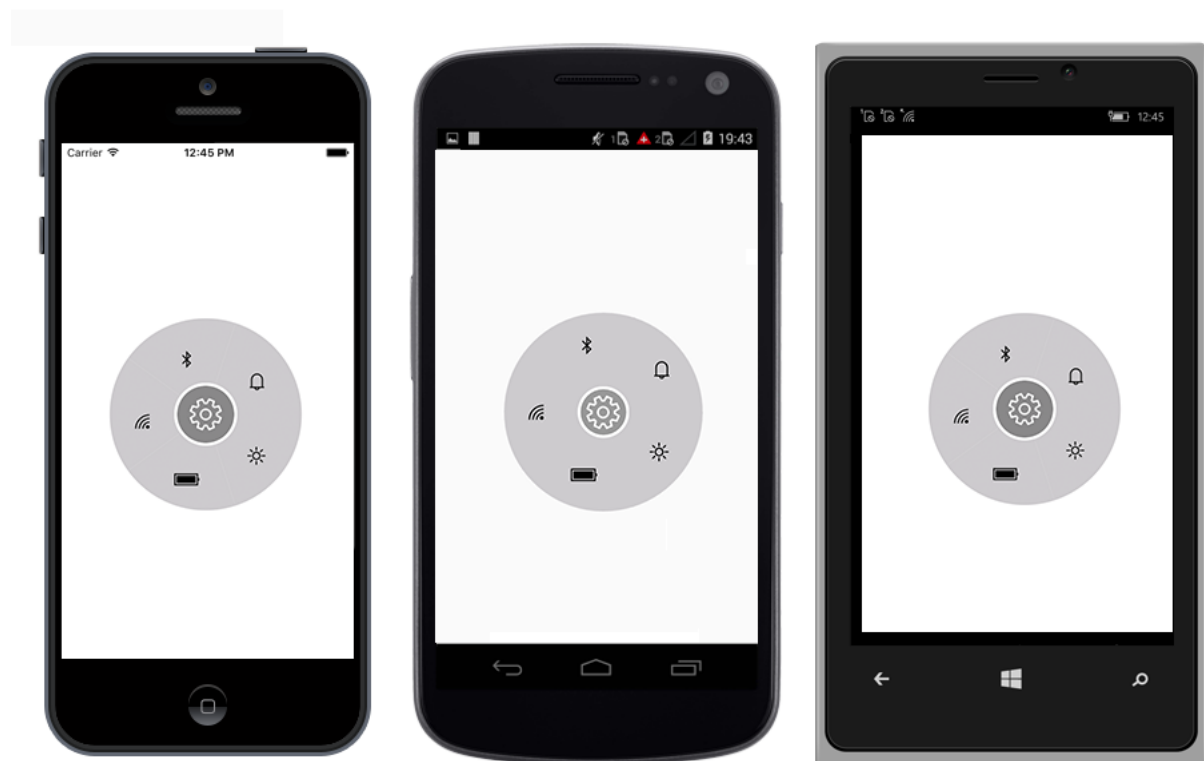
```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                LayoutType = LayoutType.Custom
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
            });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
            });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}
```

### [VisibleSegmentsCount](#)

The **VisibleSegmentsCount** property is used to specify the number of segments available in circular panel. When children count is greater than the value given in the VisibleSegmentsCount property, the overflowing children are not arranged in the panel. When children count is lesser than the VisibleSegmentsCount property, then remaining segments are left free.



If number of item count is higher than VisibleItemCount, excessive items will not be shown.



### SegmentIndex

**SegmentIndex** property is used to specify the index of the radial menu item in circular panel. Based on the index, the radial menu items are inserted in the segment. When the SegmentIndex is not specified for a RadialMenuItem the menu item is arranged in the next available free segment.

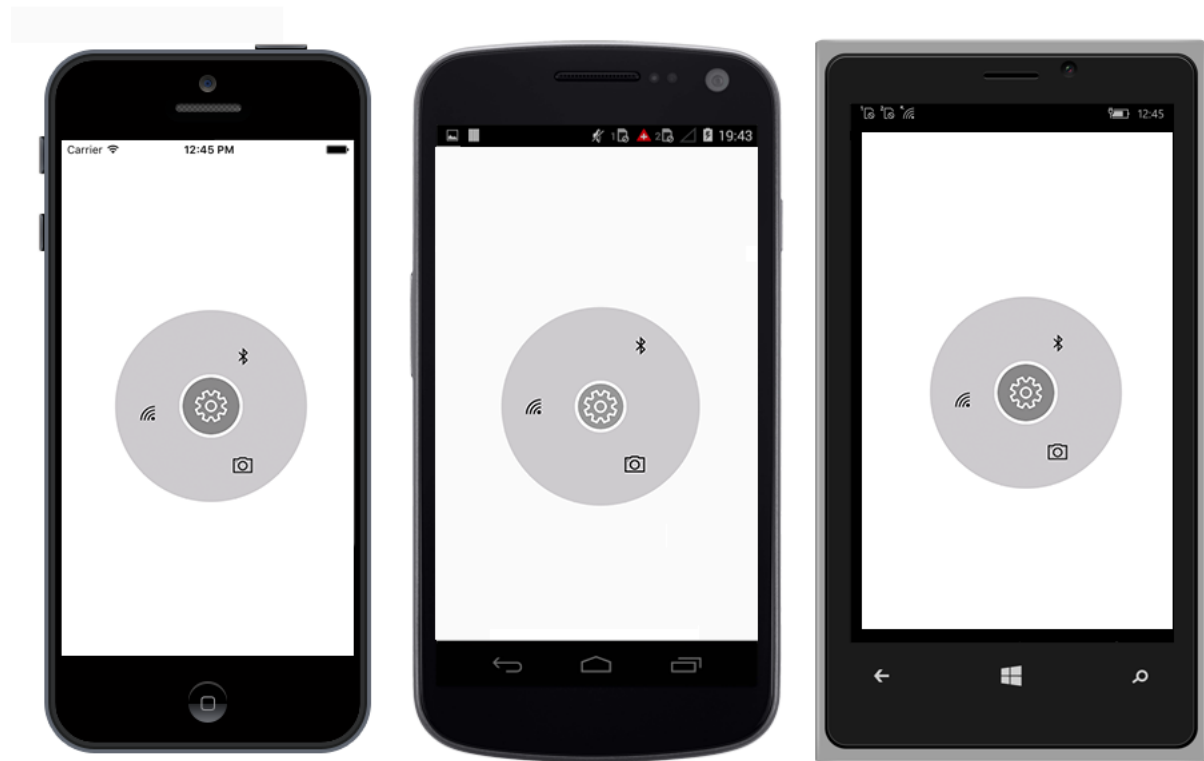
Code snippet for VisibleSegmentCount and SegmentIndex

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String"
x:Key="customfontfamily"
iOS="Segoe MDL2 Assets"
Android="radialmenu_Segoe MDL2 Assets.ttf"
UWP="radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets"/>
</ResourceDictionary>
</ContentPage.Resources>
<radialMenu:SfRadialMenu x:Name="radialMenu" LayoutType="Custom"
VisibleSegmentsCount="3">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem
FontIconText="&#xe734;"
IconFontColor="Orange"
SegmentIndex="1"
IconFontFamily="{StaticResource customfontfamily}">
</radialMenu:SfRadialMenuItem>
<radialMenu:SfRadialMenuItem
FontIconText="&#xe700;"
IconFontColor="White"
SegmentIndex="0"
IconFontFamily="{StaticResource customfontfamily}">
</radialMenu:SfRadialMenuItem>
<radialMenu:SfRadialMenuItem
FontIconText="&#xe72d;"
IconFontColor="Green"
SegmentIndex="2"
IconFontFamily="{StaticResource customfontfamily}">
</radialMenu:SfRadialMenuItem>
<radialMenu:SfRadialMenuItem
FontIconText="&#xe735;"
IconFontColor="#A52A2A"
IconFontFamily="{StaticResource customfontfamily}">
</radialMenu:SfRadialMenuItem>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

**C#**

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                LayoutType = LayoutType.Custom,
                VisibleSegmentsCount = 3
            };
            // Adding radial menu items
            string[] layer = new string[]
            {
                "\uE734",
                "\uE700",
                "\uE72d"
            };
            List<int> segmentIndex = new List<int>() { 1, 0, 2 };
            List<Color> iconColor = new List<Color>() { Color.White, Color.Orange,
            Color.Green };
            // Adding radial menu main menu items
            for (int i = 0; i < radialMenu.VisibleSegmentsCount; i++)
            {
                SfRadialMenuItem mainMenuItems = new SfRadialMenuItem();
                mainMenuItems.IconFontSize = 20;
                mainMenuItems.FontIconText = layer[i];
                mainMenuItems.IconFontFamily = Device.RuntimePlatform == "iOS" ? "Segoe MDL2
                Assets" : Device.RuntimePlatform == "Android" ? "radialmenu_Segoe MDL2
                Assets.ttf" : "radialmenu_Segoe MDL2_Assets.ttf#Segoe MDL2 Assets";
                mainMenuItems.IconFontColor = iconColor[i];
                mainMenuItems.SegmentIndex = segmentIndex[i];
                radialMenu.Items.Add(mainMenuItems);
            }
            this.Content = radialMenu;
        }
    }
}
```



## Placing and Dragging RadialMenu

You can place radial menu anywhere on its parent layout, and drag it with in the parent layout.

## Dragging RadialMenu

You can enable/disable dragging by using the `IsDragEnabled` property.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:RadialSample"
  xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
  x:Class="RadialSample.MainPage">
  <ContentPage.Resources>
    <ResourceDictionary>
      <OnPlatform x:TypeArguments="x:String" x:Key="customfontfamily" iOS="Segoe MDL2 Assets" Android="radialmenu_Segoe MDL2 Assets.ttf" WinPhone="radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets" />
    </ResourceDictionary>
  </ContentPage.Resources>
  <ContentPage.Content>
    <radialMenu:SfRadialMenu x:Name="radialMenu"
      IsDragEnabled="true"
      CenterButtonText="⚙️;"
      CenterButtonFontFamily="{StaticResource customfontfamily}"
      CenterButtonRadius="30"
```

```

CenterButtonFontSize="26"
CenterButtonText="White"
CenterButtonBorderColor="White"
CenterButtonBorderThickness="3">
</radialMenu:SfRadialMenu>
</ContentPage.Content>
</ContentPage>

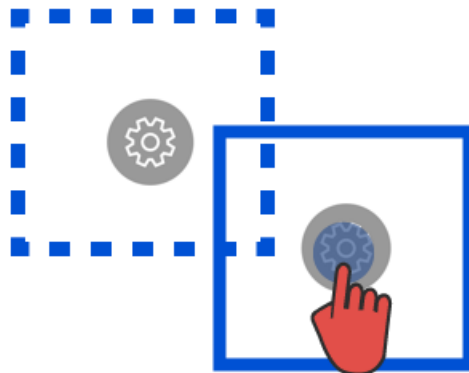
```

## C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                IsDragEnabled = true,
                CenterButtonText = "\uE713",
                CenterButtonFontFamily = Device.RuntimePlatform == "iOS" ? "Segoe MDL2 Assets" : Device.RuntimePlatform == "Android" ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets",
                CenterButtonRadius = 30,
                CenterButtonFontSize = 26,
                CenterButtonText = Color.White,
                CenterButtonBorderColor = Color.White,
                CenterButtonBorderThickness = 3
            };
            this.Content = radialMenu;
        }
    }
}

```



## DragEvents

SfRadialMenu provides for event for DragBegin and DragEnd in which the event get hooked when the RadialMenu is get dragged.

### DragBegin event

This event get triggered when RadialMenu is start to drag with `DragBeginEventArgs`.

- **Position** - Gets the Start position of the RadialMenu
- **Handled** - Gets and Sets the boolean value for enabling and disabling the dragging of RadialMenu.

To hook the `DragBegin` event, and to get the start position and restricts the dragging, follow the code example:

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String" x:Key="customfontfamily" iOS="Segoe
MDL2 Assets" Android="radialmenu_Segoe MDL2 Assets.ttf"
WinPhone="radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets" />
</ResourceDictionary>
</ContentPage.Resources>
<radialMenu:SfRadialMenu x:Name="radialMenu"
IsDragEnabled="true"
DragBegin="RadialMenu_DragBegin"
CenterButtonText="&#xe713;"
CenterButtonFontFamily="{StaticResource customfontfamily}"
CenterButtonRadius="30"
CenterButtonFontSize="26"
CenterButtonText="White"
CenterButtonBorderColor="White"
CenterButtonBorderThickness="3">
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
public partial class MainPage : ContentPage
{
public MainPage ()
```



```

{
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu()
{
IsDragEnabled = true,
CenterButtonText = "\uE713",
CenterButtonFontFamily = Device.RuntimePlatform == "iOS" ? "Segoe MDL2
Assets" : Device.RuntimePlatform == "Android" ? "radialmenu_Segoe MDL2
Assets.ttf" : "radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets",
CenterButtonRadius = 30,
CenterButtonFontSize = 26,
CenterButtonTextColors = Color.White,
CenterButtonBorderColor = Color.White,
CenterButtonBorderThickness = 3
};
radialMenu.DragBegin += RadialMenu_DragBegin;
this.Content = radialMenu;
}
private void RadialMenu_DragBegin(object sender, DragBeginEventArgs e)
{
e.Handled = true;
}
}
}

```

### Drag End

This event gets triggered when dragging is ended in RadialMenu with **DragEventArgs**.

- **OldValue** - Gets the Start position of the RadialMenu
- **NewValue** - Gets the end position of the RadialMenu
- **Handled** - Gets and Sets the boolean value for restricting the RadialMenu to move another position.

To hook the **DragEnd** event, and to get the start position, end position and restricts the movement of the RadialMenu, follow the code example:

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String" x:Key="customfontfamily" iOS="Segoe
MDL2 Assets" Android="radialmenu_Segoe MDL2 Assets.ttf"
WinPhone="radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets" />
</ResourceDictionary>
</ContentPage.Resources>
<radialMenu:SfRadialMenu x:Name="radialMenu"

```

```

IsDragEnabled="true"
DragEnd="radialMenu_DragEnd"
CenterButtonText="&#xe713;"
CenterButtonFontFamily="{StaticResource customfontfamily}"
CenterButtonRadius="30"
CenterButtonFontSize="26"
CenterButtonTextColors="White"
CenterButtonBorderColor="White"
CenterButtonBorderThickness="3">
</radialMenu:SfRadialMenu>
</ContentPage>

```

## C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                IsDragEnabled = true,
                CenterButtonText = "\uE713",
                CenterButtonFontFamily = Device.RuntimePlatform == "iOS" ? "Segoe MDL2 Assets" : Device.RuntimePlatform == "Android" ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets",
                CenterButtonRadius = 30,
                CenterButtonFontSize = 26,
                CenterButtonTextColors = Color.White,
                CenterButtonBorderColor = Color.White,
                CenterButtonBorderThickness = 3
            };
            radialMenu.DragEnd += radialMenu_DragEnd;
            this.Content = radialMenu;
        }
        private void radialMenu_DragEnd(object sender, DragEventArgs e)
        {
            e.Handled = true;
        }
    }
}

```

## Placing RadialMenu

You can place radial menu anywhere on its parent layout. Radial Menu's position is calculated based on parent layout's center point.

## XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

```

```

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String" x:Key="customfontfamily" iOS="Segoe
MDL2 Assets" Android="radialmenu_Segoe MDL2 Assets.ttf"
WinPhone="radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets" />
</ResourceDictionary>
</ContentPage.Resources>
<radialMenu:SfRadialMenu x:Name="radialMenu"
Point="100, 150"
CenterButtonText="&#xe713;"
CenterButtonFontFamily="{StaticResource customfontfamily}"
CenterButtonRadius="30"
CenterButtonFontSize="26"
CenterButtonTextColors="White"
CenterButtonBorderColor="White"
CenterButtonBorderThickness="3">
</radialMenu:SfRadialMenu>
</ContentPage>

```

**C#**

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonText = "\uE713",
                CenterButtonFontFamily = Device.RuntimePlatform == "iOS" ? "Segoe MDL2
Assets" : Device.RuntimePlatform == "Android" ? "radialmenu_Segoe MDL2
Assets.ttf" : "radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets",
                CenterButtonRadius = 30,
                CenterButtonFontSize = 26,
                CenterButtonTextColors = Color.White,
                CenterButtonBorderColor = Color.White,
                CenterButtonBorderThickness = 3,
                Point = new Point(100, 150)
            };
            this.Content = radialMenu;
        }
    }
}

```



### CenterButton Customization

The CenterButton or BackButton in radial menu is a view in the center of the radial menu. It performs the operations such as opening and closing the rim and navigating to next level items. The radial menu allows you to customize the CenterButton/BackButton with **FontIcon**, **Custom View**, and **Caption**.

### CenterButtonText and CenterButtonBackText

The [CenterButtonText](#) changes the text of the center button in [SfRadialMenu](#), and the [CenterButtonBackText](#) changes the text of the center back button in [SfRadialMenu](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu CenterButtonText="#xe700;"
CenterButtonBackText="#xe72b;">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
```

</ContentPage>

**C#**

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonText = "\uE700",
                CenterButtonBackText = "\uE72b"
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
            });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
            });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}
```

## CenterButtonTextColor and CenterButtonBackTextColor

The [CenterButtonTextColor](#) changes the text color of the center button in [SfRadialMenu](#), and the [CenterButtonBackTextColor](#) changes the text color of the center back button in [SfRadialMenu](#).

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu CenterButtonText="#xe700;"
CenterButtonBackText="#xe72b;"
CenterButtonText="#000000"
CenterButtonBackTextColor="#000000">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
```

```
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

**C#**

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonText = "\uE700",
                CenterButtonBackText = "\uE72b",
                CenterButtonTextColors = Color.FromHex("#000000"),
                CenterButtonBackTextColors = Color.FromHex("#000000")
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12 });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}
```

**CenterButtonBackgroundColor**

The [CenterButtonBackgroundColor](#) changes the background color of the center button in [SfRadialMenu](#).

**XML**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu CenterButtonBackgroundColor="#000000">
<radialMenu:SfRadialMenu.Items>
```

```
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

## C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonBackgroundColor = Color.FromHex("#000000")
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12 });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}
```

## CenterButtonRadius

The [CenterButtonRadius](#) changes the radius of the center button in [SfRadialMenu](#).

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:RadialSample"
  xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
  x:Class="RadialSample.MainPage">
  <radialMenu:SfRadialMenu CenterButtonRadius="5">
    <radialMenu:SfRadialMenu.Items>
      <radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
```

```

<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

**C#**

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonRadius = 5
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12 });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}

```

**CenterButtonFontFamily and CenterButtonBackFontFamily**

The [CenterButtonFontFamily](#) changes the font family of the center button in [SfRadialMenu](#), and the [CenterButtonBackFontFamily](#) changes the font family of the center back button in [SfRadialMenu](#).

**XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>

```



```

<OnPlatform x:TypeArguments="x:String"
x:Key="customfontfamily"
iOS="Segoe MDL2 Assets"
Android="radialmenu_Segoe MDL2 Assets.ttf"
UWP="radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets" />
</ResourceDictionary>
</ContentPage.Resources>
<radialMenu:SfRadialMenu CenterButtonText="⌂;"
CenterButtonBackText="⌂;"
CenterButtonTextColor="#000000"
CenterButtonBackTextColor="#000000"
CenterButtonFontFamily="{StaticResource customfontfamily}"
CenterButtonBackFontFamily="{StaticResource customfontfamily}">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

## C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonText = "\uE700",
                CenterButtonBackText = "\uE72b",
                CenterButtonTextColor = Color.FromHex("#000000"),
                CenterButtonBackTextColor = Color.FromHex("#000000"),
                CenterButtonFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets",
                CenterButtonBackFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets"
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
        }
    }
}

```

```
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
});
radialMenu.Items = itemCollection;
this.Content = radialMenu;
}
}
```

### CenterButtonFontSize and CenterButtonBackFontSize

The [CenterButtonFontSize](#) changes the font size of the center button in [SfRadialMenu](#), and the [CenterButtonBackFontSize](#) changes the font size of the center back button in [SfRadialMenu](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
  <ContentPage.Resources>
    <ResourceDictionary>
      <OnPlatform x:TypeArguments="x:String"
x:Key="customfontfamily"
iOS="Segoe MDL2 Assets"
Android="radialmenu_Segoe MDL2 Assets.ttf"
UWP="radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets" />
    </ResourceDictionary>
  </ContentPage.Resources>
  <radialMenu:SfRadialMenu CenterButtonText="⌂"
CenterButtonBackText="⌕"
CenterButtonTextColors="#000000"
CenterButtonBackTextColors="#000000"
CenterButtonFontFamily="{StaticResource customfontfamily}"
CenterButtonBackFontFamily="{StaticResource customfontfamily}"
CenterButtonFontSize="10"
CenterButtonBackFontSize="10">
    <radialMenu:SfRadialMenu.Items>
      <radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
      <radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
      <radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
      <radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
      <radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
    </radialMenu:SfRadialMenu.Items>
  </radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
```

```

{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonText = "\uE700",
                CenterButtonBackText = "\uE72b",
                CenterButtonText.Color = Color.FromHex("#000000"),
                CenterButtonBackText.Color = Color.FromHex("#000000"),
                CenterButtonFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets",
                CenterButtonBackFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets",
                CenterButtonFontSize = 10,
                CenterButtonBackFontSize = 10
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12 });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}

```

### CenterButtonFontAttributes

The [CenterButtonFontAttributes](#) changes the font attributes of the center button in [SfRadialMenu](#).

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:RadialSample"
    xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
    x:Class="RadialSample.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <OnPlatform x:TypeArguments="x:String"
                x:Key="customfontfamily"
                iOS="Segoe MDL2 Assets"
                Android="radialmenu_Segoe MDL2 Assets.ttf"
                UWP="radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets" />
        </ResourceDictionary>
    </ContentPage.Resources>
</ContentPage>

```

```

</ResourceDictionary>
</ContentPage.Resources>
<radialMenu:SfRadialMenu CenterButtonText="⌂;"
CenterButtonBackText="⌕;"
CenterButtonTextColor="#000000"
CenterButtonBackTextColor="#000000"
CenterButtonFontFamily="{StaticResource customfontfamily}"
CenterButtonBackFontFamily="{StaticResource customfontfamily}"
CenterButtonFontSize="10"
CenterButtonBackFontSize="10"
CenterButtonFontAttributes="Bold">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

## C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonText = "\uE700",
                CenterButtonBackText = "\uE72b",
                CenterButtonTextColor = Color.FromHex("#000000"),
                CenterButtonBackTextColor = Color.FromHex("#000000"),
                CenterButtonFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets",
                CenterButtonBackFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe_MDL2_Assets.ttf#Segoe MDL2 Assets",
                CenterButtonFontSize = 10,
                CenterButtonBackFontSize = 10,
                CenterButtonFontAttributes = FontAttributes.Bold
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
        }
    }
}

```

```
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
});
radialMenu.Items = itemCollection;
this.Content = radialMenu;
}
}
```

### CenterButtonBorderColor

The [CenterButtonBorderColor](#) changes the border color of the center button in [SfRadialMenu](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String"
x:Key="customfontfamily"
iOS="Segoe MDL2 Assets"
Android="radialmenu_Segoe MDL2 Assets.ttf"
UWP="radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets" />
</ResourceDictionary>
</ContentPage.Resources>
<radialMenu:SfRadialMenu CenterButtonText="#xe700;"
CenterButtonBackText="#xe72b;"
CenterButtonTextColors="#000000"
CenterButtonBackTextColors="#000000"
CenterButtonFontFamily="{StaticResource customfontfamily}"
CenterButtonBackFontFamily="{StaticResource customfontfamily}"
CenterButtonFontSize="10"
CenterButtonBackFontSize="10"
CenterButtonFontAttributes="Bold"
CenterButtonBorderColor="Black">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
```

```

{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu()
{
CenterButtonText = "\uE700",
CenterButtonBackText = "\uE72b",
CenterButtonTextColors = Color.FromHex("#000000"),
CenterButtonBackTextColors = Color.FromHex("#000000"),
CenterButtonFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets",
CenterButtonBackFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets",
CenterButtonFontSize = 10,
CenterButtonBackFontSize = 10,
CenterButtonFontAttributes = FontAttributes.Bold,
CenterButtonBorderColor = Color.Black
};
ObservableCollection<SfRadialMenuItem> itemCollection = new
ObservableCollection<SfRadialMenuItem>();
itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12 });
});
itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12 });
});
radialMenu.Items = itemCollection;
this.Content = radialMenu;
}
}
}

```

### CenterButtonBorderThickness

The [CenterButtonBorderThickness](#) changes the border thickness of the center button in [SfRadialMenu](#).

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XForms"
x:Class="RadialSample.MainPage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String"
x:Key="customfontfamily"
iOS="Segoe MDL2 Assets"

```

```

Android="radialmenu_Segoe MDL2 Assets.ttf"
UWP="radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets" />
</ResourceDictionary>
</ContentPage.Resources>
<radialMenu:SfRadialMenu CenterButtonText="&#xe700;"
CenterButtonBackText="&#xe72b;"
CenterButtonTextColor="#000000"
CenterButtonBackTextColor="#000000"
CenterButtonFontFamily="{StaticResource customfontfamily}"
CenterButtonBackFontFamily="{StaticResource customfontfamily}"
CenterButtonFontSize="10"
CenterButtonBackFontSize="10"
CenterButtonFontAttributes="Bold"
CenterButtonBorderColor="Black"
CenterButtonBorderThickness="2">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

## C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonText = "\uE700",
                CenterButtonBackText = "\uE72b",
                CenterButtonTextColor = Color.FromHex("#000000"),
                CenterButtonBackTextColor = Color.FromHex("#000000"),
                CenterButtonFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets",
                CenterButtonBackFontFamily = Device.RuntimePlatform == Device.iOS ? "Segoe MDL2 Assets" : Device.RuntimePlatform == Device.Android ? "radialmenu_Segoe MDL2 Assets.ttf" : "radialmenu_Segoe MDL2 Assets.ttf#Segoe MDL2 Assets",
                CenterButtonFontSize = 10,
                CenterButtonBackFontSize = 10,
                CenterButtonFontAttributes = FontAttributes.Bold,
                CenterButtonBorderColor = Color.Black,
                CenterButtonBorderThickness = 2
            };
        }
    }
}

```

```

ObservableCollection<SfRadialMenuItem> itemCollection = new
ObservableCollection<SfRadialMenuItem>();
itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
});
itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
});
radialMenu.Items = itemCollection;
this.Content = radialMenu;
}
}
}

```

### CenterButtonPlacement

The [CenterButtonPlacement](#) changes the placement of the center button in [SfRadialMenu](#)

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu CenterButtonPlacement="Center">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

#### C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                CenterButtonPlacement = SfRadialMenuCenterButtonPlacement.Center
            };

```



```

ObservableCollection<SfRadialMenuItem> itemCollection = new
ObservableCollection<SfRadialMenuItem>();
itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
});
itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
});
radialMenu.Items = itemCollection;
this.Content = radialMenu;
}
}
}

```

Center button view and center back button view

You can customize the center button using [CenterButtonView](#) and center back button using [CenterButtonBackView](#) in [SfRadialMenu](#).

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu x:Name="radialMenu">
<radialMenu:SfRadialMenu.CenterButtonView >
<Grid>
<StackLayout VerticalOptions="Center">
<Image Source="Beverage1.png"/>
</StackLayout>
</Grid>
</radialMenu:SfRadialMenu.CenterButtonView>
<radialMenu:SfRadialMenu.CenterButtonBackView>
<Grid>
<StackLayout VerticalOptions="Center">
<Image Source="Beverage2.png"/>
</StackLayout>
</Grid>
</radialMenu:SfRadialMenu.CenterButtonBackView>
</radialMenu:SfRadialMenu>
</ContentPage>

```

#### C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
public partial class MainPage : ContentPage

```

```

{
public MainPage()
{
InitializeComponent();
Grid centerButtonGrid = new Grid();
Grid centerButtonBackGrid = new Grid();
StackLayout centerButtonLayout = new StackLayout()
{
VerticalOptions = LayoutOptions.Center
};
StackLayout centerButtonBackLayout = new StackLayout()
{
VerticalOptions = LayoutOptions.Center
};
centerButtonLayout.Children.Add(new Image() { Source = "Beverage1.png" });
centerButtonGrid.Children.Add(centerButtonLayout);
centerButtonBackLayout.Children.Add(new Image() { Source = "Beverage2.png"
});
centerButtonBackGrid.Children.Add(centerButtonBackLayout);
SfRadialMenu radialMenu = new SfRadialMenu()
{
CenterButtonView = centerButtonGrid,
CenterButtonBackView = centerButtonBackGrid
};
}
}
}

```

### EnableCenterButtonAnimation

The [EnableCenterButtonAnimation](#) is used to either enable or disable animation of the center button in [SfRadialMenu](#).

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu EnableCenterButtonAnimation="True">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu()
            {
                EnableCenterButtonAnimation = true
            };
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
            });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
            });
            radialMenu.Items = itemCollection;
            this.Content = radialMenu;
        }
    }
}
```



## Events

Perform an action while navigating to next level

In radial menu, you can perform an action while navigating from one level to another level. The [Navigating](#) event occurs when navigating from one level to another level and the [Navigated](#) event occurs after navigating to another level.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu Navigating="SfRadialMenu_Navigating"
Navigated="SfRadialMenu_Navigated">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

## C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
public partial class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
SfRadialMenu radialMenu = new SfRadialMenu();
ObservableCollection<SfRadialMenuItem> itemCollection = new
ObservableCollection<SfRadialMenuItem>();
itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
});
itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
});
radialMenu.Items = itemCollection;
radialMenu.Navigating += SfRadialMenu_Navigating;
radialMenu.Navigated += SfRadialMenu_Navigated;
this.Content = radialMenu;
}
```

```
private void SfRadialMenu_Navigating(object sender, NavigatingEventArgs e)
{
    DisplayAlert("Alert", "ItemNavigating", "Ok");
}
private void SfRadialMenu_Navigated(object sender, NavigatedEventArgs e)
{
    DisplayAlert("Alert", "ItemNavigated", "Ok");
}
}
```

**Note:** You can cancel navigation using the **Cancel** event argument.

Perform an action while opening the radial menu

You can perform an action while opening the radial menu. The [Opening](#) event occurs when opening the radial menu and the [Opened](#) event occurs after opening the radial menu.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu Opening ="SfRadialMenu_Opening"
Opened="SfRadialMenu_Opened">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

### C#

```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
```

```

itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
});
itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
});
radialMenu.Items = itemCollection;
radialMenu.Opening += SfRadialMenu_Opening;
radialMenu.Opened += SfRadialMenu_Opened;
this.Content = radialMenu;
}
private void SfRadialMenu_Opening(object sender, OpeningEventArgs e)
{
    DisplayAlert("Alert", "ItemOpening", "Ok");
}
private void SfRadialMenu_Opened(object sender, OpenedEventArgs e)
{
    DisplayAlert("Alert", "ItemOpened", "Ok");
}
}
}

```

Perform an action while closing the radial menu

You can perform an action when closing the radial menu. The [Closing](#) event occurs when closing the radial menu and the [Closed](#) event occurs after closing the radial menu.

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu Closing="SfRadialMenu_Closing"
Closed="SfRadialMenu_Closed">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

#### C#

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{

```

```

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        SfRadialMenu radialMenu = new SfRadialMenu();
        ObservableCollection<SfRadialMenuItem> itemCollection = new
        ObservableCollection<SfRadialMenuItem>();
        itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
        itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
        itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
        });
        itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
        itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
        });
        radialMenu.Items = itemCollection;
        radialMenu.Closing += SfRadialMenu_Closing;
        radialMenu.Closed += SfRadialMenu_Closed;
        this.Content = radialMenu;
    }
    private void SfRadialMenu_Closing(object sender, ClosingEventArgs e)
    {
        DisplayAlert("Alert", "ItemClosing", "Ok");
    }
    private void SfRadialMenu_Closed(object sender, ClosedEventArgs e)
    {
        DisplayAlert("Alert", "ItemClosed", "Ok");
    }
}

```

Perform an action while tapping the center back button

You can perform an action when tapping the center back button of the radial menu. The [CenterButtonBackTapped](#) event occurs when tapping the center button of the radial menu.

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu
CenterButtonBackTapped="SfRadialMenu_CenterButtonBackTapped">
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>

```

**C#**

```

using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
            });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
            });
            radialMenu.Items = itemCollection;
            radialMenu.CenterButtonBackTapped += SfRadialMenu_CenterButtonBackTapped;
            this.Content = radialMenu;
        }
        private void SfRadialMenu_CenterButtonBackTapped(object sender,
            CenterButtonBackTappedEventArgs e)
        {
            {
                DisplayAlert("Alert", "CenterButtonTapped", "Ok");
            }
        }
    }
}

```

Perform an action while tapping the radial menu item

You can perform an action when tapping the radial menu item of the radial menu. The [ItemTapped](#) event occurs when tapping the items of the radial menu.

**XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RadialSample"
xmlns:radialMenu="clr-
namespace:Syncfusion.SfRadialMenu.XForms;assembly=Syncfusion.SfRadialMenu.XF
orms"
x:Class="RadialSample.MainPage">
<radialMenu:SfRadialMenu>
<radialMenu:SfRadialMenu.Items>
<radialMenu:SfRadialMenuItem Text="Bold" FontSize="12"
ItemTapped="SfRadialMenuItem_ItemTapped"/>
<radialMenu:SfRadialMenuItem Text="Copy" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Undo" FontSize="12"/>

```



```
<radialMenu:SfRadialMenuItem Text="Paste" FontSize="12"/>
<radialMenu:SfRadialMenuItem Text="Color" FontSize="12"/>
</radialMenu:SfRadialMenu.Items>
</radialMenu:SfRadialMenu>
</ContentPage>
```

## C#

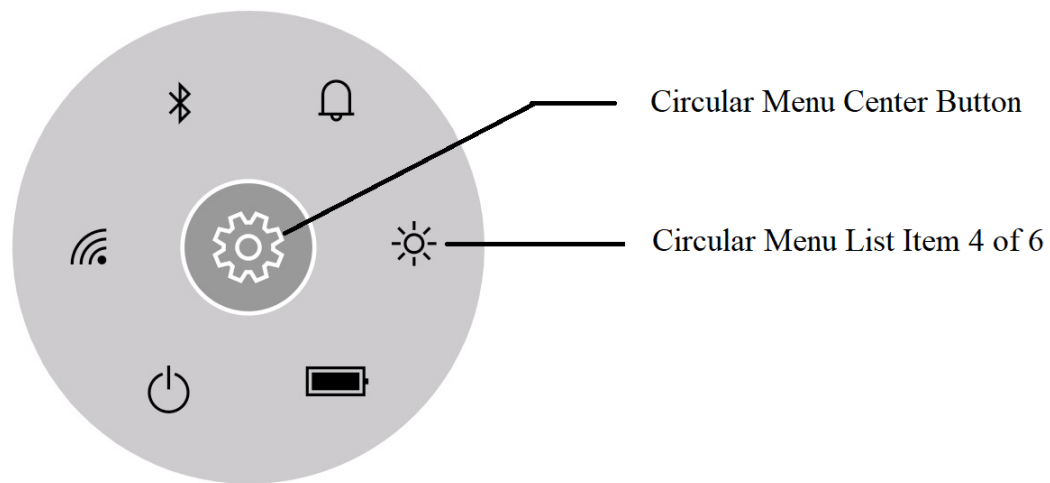
```
using Syncfusion.SfRadialMenu.XForms;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace RadialSample
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRadialMenu radialMenu = new SfRadialMenu();
            ObservableCollection<SfRadialMenuItem> itemCollection = new
            ObservableCollection<SfRadialMenuItem>();
            itemCollection.Add(new SfRadialMenuItem() { Text = "Bold", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Copy", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Paste", FontSize = 12
            });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Undo", FontSize = 12 });
            itemCollection.Add(new SfRadialMenuItem() { Text = "Color", FontSize = 12
            });
            radialMenu.Items = itemCollection;
            radialMenu.Items[0].ItemTapped += SfRadialMenuItem_ItemTapped;
            this.Content = radialMenu;
        }
        private void SfRadialMenuItem_ItemTapped(object sender,
            Syncfusion.SfRadialMenu.XForms.ItemTappedEventArgs e)
        {
            {
                DisplayAlert("Alert", "ItemTapped", "Ok");
            }
        }
    }
}
```

## AutomationId

The SfRadialMenu control has built-in AutomationId for inner elements. The AutomationId API allows the automation framework to find and interact with the inner elements of the SfRadialMenu control. To keep unique AutomationId, these inner elements AutomationIds are updated based on the control's AutomationId.

For example, if you set SfRadialMenu's AutomationId as "Circular Menu" and the first SfRadialMenuItem's AutomationId as "Circular Menu List", then the automation framework will interact with the center button as "Circular Menu Center Button" and the first radial item as "Circular Menu List Item 4 of 6" (6 denotes the total count).

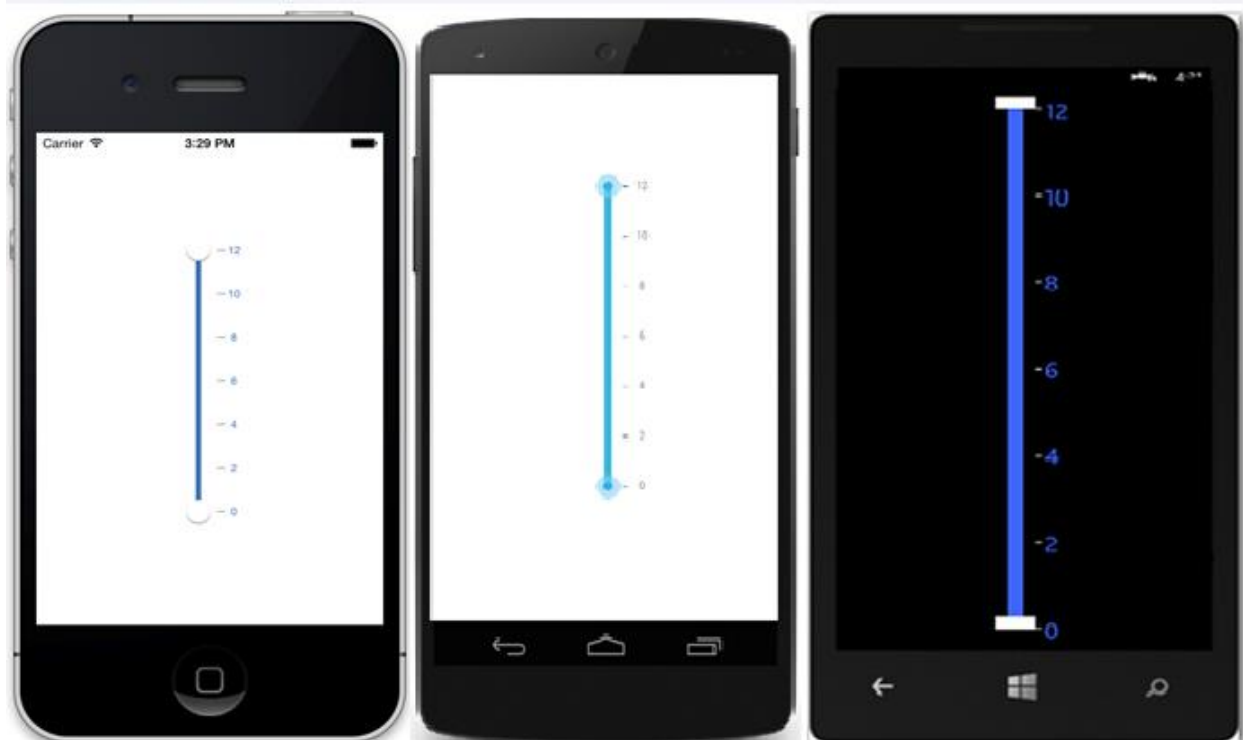
The following screenshot illustrates the AutomationIds of inner elements. The automation framework will interact with the Center Back Button as "Circular Menu Center Back Button".



## SfRangeSlider

### Overview

The range slider control for Xamarin.Forms allows you select a range of values within the specified minimum and maximum limits. The range can be selected by moving the thumb along track.



## Key features

- Provides support to select values as a range.
- Restricts values to choose within a minimum and maximum constraints.
- Supports to change the tick intervals in uniform pattern.
- Provides user-friendly customization support to customize ticks and labels.

## Getting Started

This section explains you the steps to configure a SfRangeSlider control in a real-time scenario and also provides a walk-through on some of the customization features available in SfRangeSlider control.

### Adding SfRangeSlider reference

You can add SfRangeSlider reference using one of the following methods:

#### Method 1: Adding SfRangeSlider reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfRangeSlider). To add SfRangeSlider to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfRangeSlider](https://www.nuget.org/packages/Syncfusion.Xamarin.SfRangeSlider), and then install it.

![Adding SfRangeSlider reference from nuget.org](images/Adding SfRangeSlider reference.png)

---

**Note:** Install the same version of SfRangeSlider NuGet in all the projects.

---

#### Method 2: Adding SfRangeSlider reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfRangeSlider control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfRangeSlider assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfRangeSlider.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfRangeSlider.Android.dll Syncfusion.SfRangeSlider.XForms.Android.dll Syncfusion.SfRangeSlider.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfRangeSlider.iOS.dll Syncfusion.SfRangeSlider.XForms.iOS.dll Syncfusion.SfRangeSlider.XForms.dll Syncfusion.Core.XForms.dll

	Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfInput.UWP.dll Syncfusion.SfShared.UWP.dll Syncfusion.SfRangeSlider.XForms.UWP.dll Syncfusion.SfRangeSlider.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** Currently an additional step is required for iOS project. You need to create an instance of RangeSlider custom renderer. If you are adding the references from toolbox, this step is not needed.

Create an instance of SfRangeSliderRenderer in the FinishedLaunching overridden method of AppDelegate class in the iOS Project as shown below.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    new SfRangeSliderRenderer ();
}
```

#### Additional step for UWP

This step is required only if the application is deployed in release mode with .NET native tool chain enabled. It is needed for resolving the known Framework issue “Custom controls not rendering in Release mode” in UWP platform. Initializing the SfRangeSlider assembly at the OnLaunched overridden method of the App class in UWP project is the suggested workaround. The following code example demonstrates how to initialize the SfRangeSlider assembly.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfRangeSliderRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
```

```
Xamarin.Forms.Forms.Init(e, assembliesToInclude);  
...  
}
```

- Adding namespace for the added assemblies.

### XML

```
<xmlns:range="clr-  
namespace:Syncfusion.SfRangeSlider.XForms;assembly=Syncfusion.SfRangeSlider.  
XForms"/>
```

### C#

```
using Syncfusion.SfRangeSlider.XForms;
```

- Now instantiate and add the SfRangeSlider control with a required optimal name.

### XML

```
<range:SfRangeSlider x:Name="rangeSlider"/>
```

### C#

```
SfRangeSlider rangeSlider=new SfRangeSlider();  
this.Content = rangeSlider;
```

### Set Range

SfRangeSlider provides option to set single thumb and double thumb. While setting the double thumb, each thumb value can be set using **RangeStart** and **RangeEnd** properties.

**Note:** The **ShowRange** property is used to switch between a single thumb and double thumb.

### XML

```
<range:SfRangeSlider x:Name="rangeSlider" RangeEnd="20" RangeStart="4"  
ShowRange="true"/>
```

### C#

```
SfRangeSlider rangeSlider=new SfRangeSlider();  
rangeSlider.RangeEnd=20;  
rangeSlider.RangeStart=4;  
rangeSlider.ShowRange=true;  
this.Content = rangeSlider;
```

### Restricting Values

SfRangeSlider provides option to restrict slider range between minimum and maximum values. Following code explains how to set the range using `Minimum` and `Maximum` properties in the SfRangeSlider.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" Minimum="0" Maximum="24"/>
```

#### C#

```
SfRangeSlider rangeSlider=new SfRangeSlider();  
rangeSlider.RangeEnd=20;  
rangeSlider.RangeStart=4;  
rangeSlider.ShowRange=true;  
rangeSlider.Minimum=0;  
rangeSlider.Maximum=24;  
this.Content = rangeSlider;
```

### Adding Snapping Mode

The movement of the thumb can be varied in different ways. This is achieved by setting the `SnapsTo` property.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" SnapsTo="Ticks"  
StepFrequency="6"/>
```

#### C#

```
SfRangeSlider rangeSlider=new SfRangeSlider();  
rangeSlider.RangeEnd=20;  
rangeSlider.RangeStart=4;  
rangeSlider.ShowRange=true;  
rangeSlider.Minimum=0;  
rangeSlider.Maximum=24;  
rangeSlider.SnapsTo=SnapsTo.Ticks;  
rangeSlider.StepFrequency=6;  
this.Content = rangeSlider;
```

The complete Getting Started sample is available in [this](#) link.

### Range

The SfRangeSlider control supports to select range of value by using two Thumbs.

#### Set Show Range

The `ShowRange` property should be set to true for displaying two thumbs in track with range of values.

**Note:** When this property is set to false, single thumb is displayed without any range

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" ShowRange="true"/>
```

### C#

```
rangeSlider.ShowRange= true;
```

### Set Range values

This section explains about setting Range start and end value.

#### RangeStart

Gets and sets the start value of the range.

### XML

```
<range:SfRangeSlider x:Name="rangeslider" RangeStart="4" />
```

### C#

```
rangeSlider.RangeStart=0;
```

#### RangeEnd

Gets and sets the end value of the range.

### XML

```
<range:SfRangeSlider x:Name="rangeslider" RangeEnd="20" />
```

### C#

```
rangeSlider.RangeEnd=10;
```

### ValueChangeMode

The ValueChangeMode property changes the value based on the touch of the SfRangeSlider control. It consists of the following two types

- Default
- OnThumbPress

**Note:** The default value of the ValueChangeMode property is Default.

#### Default

The value is updated when you touch inside the control.

### XML

```
<range:SfRangeSlider x:Name="rangeslider" ValueChangeMode="Default" />
```

### C#

```
rangeSlider.ValueChangeMode = ValueChangeMode.Default;
```

#### OnThumb

The value is updated when you touch or move the thumb/knob.

### XML

```
<range:SfRangeSlider x:Name="rangeslider" ValueChangeMode="OnThumb" />
```

### C#

```
rangeSlider.ValueChangeMode = ValueChangeMode.OnThumb;
```

### Value

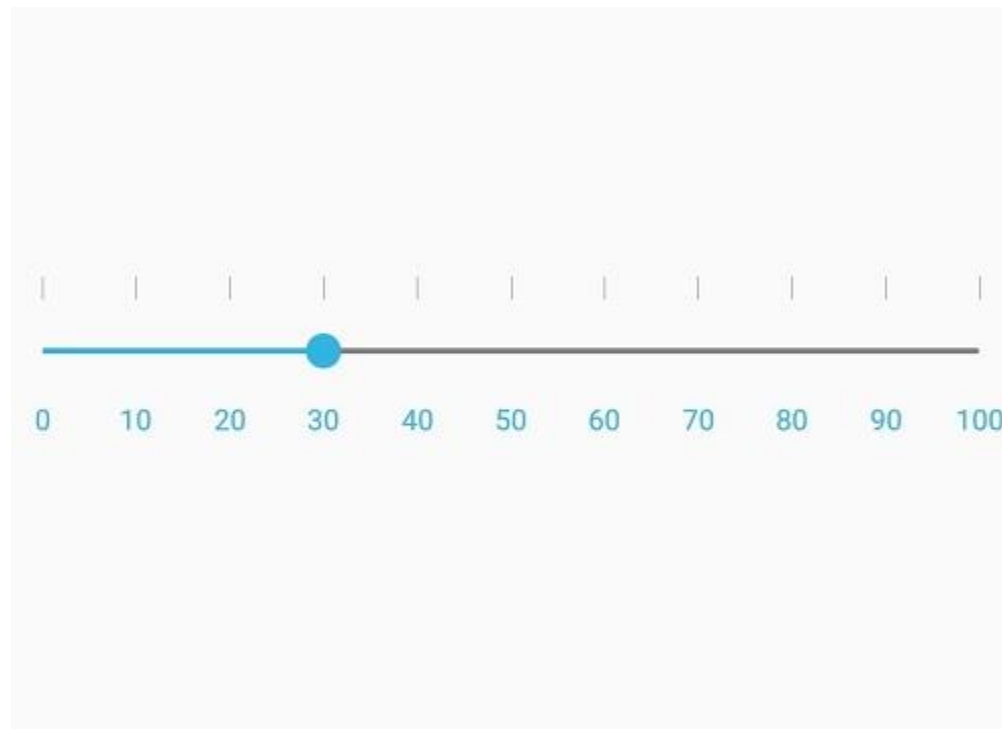
Gets or sets the range value, which ranges between Minimum and Maximum. The default value of RangeSlider is 0.

### XML

```
<range:SfRangeSlider x:Name="rangeslider" Minimum="0" Maximum="100" Value="30" />
```

### C#

```
rangeslider.Value = 30;
```



### Customizing ticks

Tick marks can be placed along the track in a uniform manner or its position can also be customized.

### TickPlacement

The **TickPlacement** property determines where to draw tick marks in relation to the track. Available options for this property are,



- BottomRight
- Inline
- None
- Outside
- TopLeft

**Note:** The default option is Inline.

#### *BottomRight*

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

#### **XML**

```
<range:SfRangeSlider x:Name="rangeslider" TickPlacement="BottomRight"/>
```

#### **C#**

```
rangeSlider.TickPlacement=TickPlacement.BottomRight;
```



#### *TopLeft*

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

#### **XML**

```
<range:SfRangeSlider x:Name="rangeslider" TickPlacement="TopLeft"/>
```

#### **C#**

```
rangeSlider.TickPlacement=TickPlacement.TopLeft;
```



#### *Inline*

Ticks are placed along the track.

#### **XML**

```
<range:SfRangeSlider x:Name="rangeslider" TickPlacement="Inline"/>
```

#### **C#**

```
rangeSlider.TickPlacement=TickPlacement.Inline;
```



### Outside

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

### XML

```
<range:SfRangeSlider x:Name="rangeslider" TickPlacement="Outside"/>
```

### C#

```
rangeSlider.TickPlacement=TickPlacement.Outside;
```



### Customizing tick color

The range slider control provides the `TickColor` property to customize the color of ticks in tick bar.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:range="clr-namespace:Syncfusion.SfRangeSlider.XForms;assembly=Syncfusion.SfRangeSlider.XForms"
x:Class="GettingStarted.RangeSliderSample">
<ContentPage.Content>
<range:SfRangeSlider x:Name="rangeslider" Orientation="Horizontal"
TickColor="#FFFFFF"/>
</ContentPage.Content>
</ContentPage>
```

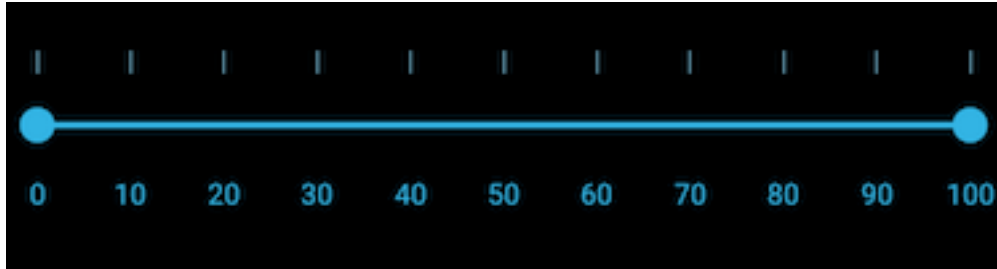
### C#

```
using Syncfusion.XForms.SfRangeSlider;
using Xamarin.Forms;
namespace GettingStarted
{
    /// <summary>
    /// Range slider sample.
    /// </summary>
    public partial class RangeSliderSample : ContentPage
    {
        public RangeSliderSample()
        {
            InitializeComponent();
            SfRangeSlider rangeSlider = new SfRangeSlider() { TickColor =
            Color.FromHex("#FFFFFF"), Orientation=Orientation.Horizontal};
```

```

this.Content = rangeSlider;
}
}
}

```



## TrackBar Customization

### Customizing track height

The thickness of track bar can be customized by setting the `TrackThickness` property of `SfRangeSlider`.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:TrackCustomization"
xmlns:slider="clr-
namespace:Syncfusion.SfRangeSlider.XForms;assembly=Syncfusion.SfRangeSlider.
XForms"
x:Class="TrackCustomization.MainPage">
<slider:SfRangeSlider Orientation="Horizontal"
TrackThickness="10"
RangeStart="0"
RangeEnd="2"/>
</ContentPage>

```

### C#

```

using Syncfusion.SfRangeSlider.XForms;
using Xamarin.Forms;
namespace TrackCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRangeSlider slider = new SfRangeSlider();
            slider.Orientation = Orientation.Horizontal;
            slider.RangeStart = 0;
            slider.RangeEnd = 2;
            slider.TrackThickness = 10;
            Content = slider;
        }
    }
}

```



### *Customizing dragged area height*

The thickness for the selected range or selected portion of track bar can be customized by setting the `TrackSelectionThickness` property of `SfRangeSlider`.

### **XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:TrackCustomization"
xmlns:slider="clr-
namespace:Syncfusion.SfRangeSlider.XForms;assembly=Syncfusion.SfRangeSlider.
XForms"
x:Class="TrackCustomization.MainPage">
<slider:SfRangeSlider Orientation="Horizontal"
TrackSelectionThickness="10"
RangeStart="10"
RangeEnd="30"/>
</ContentPage>
```

### **C#**

```
using Syncfusion.SfRangeSlider.XForms;
using Xamarin.Forms;
namespace TrackCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRangeSlider slider = new SfRangeSlider();
            slider.Orientation = Orientation.Horizontal;
            slider.TrackSelectionThickness = 10;
            slider.RangeStart = 10;
            slider.RangeEnd = 30;
            Content = slider;
        }
    }
}
```



### Customizing track color

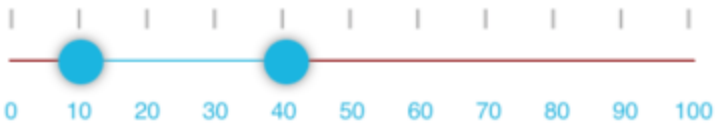
The color of track bar can be customized by setting the TrackColor property of SfRangeSlider.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:TrackCustomization"
xmlns:slider="clr-
namespace:Syncfusion.SfRangeSlider.XForms;assembly=Syncfusion.SfRangeSlider.
XForms"
x:Class="TrackCustomization.MainPage">
<slider:SfRangeSlider Orientation="Horizontal"
TrackColor="Maroon"
RangeStart="10"
RangeEnd="40"/>
</ContentPage>
```

### C#

```
using Syncfusion.SfRangeSlider.XForms;
using Xamarin.Forms;
namespace TrackCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRangeSlider slider = new SfRangeSlider();
            slider.Orientation = Orientation.Horizontal;
            slider.TrackColor = Color.Maroon;
            slider.RangeStart = 10;
            slider.RangeEnd = 40;
            Content = slider;
        }
    }
}
```



### Customizing dragged area color

The color for the selected range or selected portion of track bar can be customized by setting the `TrackSelectionColor` property of `SfRangeSlider`.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:TrackCustomization"
xmlns:slider="clr-
namespace:Syncfusion.SfRangeSlider.XForms;assembly=Syncfusion.SfRangeSlider.
XForms"
x:Class="TrackCustomization.MainPage">
<slider:SfRangeSlider Orientation="Horizontal"
TrackSelectionColor="Red"
RangeStart="10"
RangeEnd="40"/>
</ContentPage>
```

### C#

```
using Syncfusion.SfRangeSlider.XForms;
using Xamarin.Forms;
namespace TrackCustomization
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            SfRangeSlider slider = new SfRangeSlider();
            slider.Orientation = Orientation.Horizontal;
            slider.TrackSelectionColor = Color.Red;
            slider.RangeStart = 10;
            slider.RangeEnd = 40;
            Content = slider;
        }
    }
}
```



### Customizing knob color

The **KnobColor** property is used to change the knob color of SfRangeSlider.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" KnobColor="Red" Minimum="0"
Maximum="100"/>
```

#### C#

```
rangeslider.KnobColor = Color.Red;
```

### Customizing thumb size

The **ThumbSize** property is used to change the thumb size of SfRangeSlider.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" ThumbSize="3" Minimum="0"
Maximum="100"/>
```

#### C#

```
rangeslider.ThumbSize = 3;
```

## Formatting String

The Value label of the SfRangeSlider can be configured to display different formats like currency format, percent format etc. We can also customize the Value label with string formatting. We can customize using **FormatString** property which determines the format specifier by which the display Value has to be formatted.

### Formatting types

We have different formatting types such as currency format, exponential format, number format, percentage format etc. We can also add the text with Value using **FormatString**

#### XML

```
<StackLayout Margin="3">
<range:SfRangeSlider x:Name="rangeSlider1" FormatString="Money: {0:c}"
HeightRequest="90" WidthRequest="200" Minimum="0" Maximum="12"
RangeStart="0" RangeEnd="12" TickFrequency="2"/>
</StackLayout>
```

**C#**

```

StackLayout stack = new StackLayout();
SfRangeSlider rangeSlider2 = new SfRangeSlider();
rangeSlider2 .RangeEnd=12;
rangeSlider2 .RangeStart=0;
rangeSlider2 .TickFrequency = 2;
rangeSlider2 .HeightRequest="90";
rangeSlider2 .WidthRequest=200;
rangeSlider2 .FormatString = "{0:N2}"
rangeSlider2 .ShowRange=true;
rangeSlider2 .TrackHeight="4";
rangeSlider2 .Minimum=0;
rangeSlider2 .Maximum=12;
this.stack.Children.Add(rangeSlider2);
this.Content = stack;

```

**Culture Localization**

We have provided the support for changing the Culture when using Currency notation for the formatting type. For this we have to enable Currency format and set the desired culture to be shown.

**XML**

```

<range:SfRangeSlider x:Name="sfRangeSlider2" FormatString="c:{0:c2}"
HeightRequest="90" WidthRequest="200" Minimum="0" Maximum="12"
RangeStart="0" RangeEnd="12" TickFrequency="2"/>

```

**C#**

```

SfRangeSlider rangeSlider=new SfRangeSlider();
rangeSlider.RangeEnd=12;
rangeSlider.RangeStart=0;
rangeSlider.TickFrequency = 2;
rangeSlider.HeightRequest="90";
rangeSlider.WidthRequest=200;
rangeSlider.FormatString = "c:{0:C2}"
rangeSlider.Culture = new System.Globalization.CultureInfo("fr-FR");
rangeSlider.ShowRange=true;
rangeSlider.TrackHeight="4";
rangeSlider.Minimum=0;
rangeSlider.Maximum=12;
this.Content = rangeSlider;

```





## Selection Value Configuration

Various customization options are available to configure the selection value in SfRangeSlider.

### Set Minimum Value

Gets or sets the minimum possible value of the range. The thumb could not move beyond that value.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" Minimum="0"/>
```

#### C#

```
rangeSlider.Minimum=0;
```

### Set Maximum Value

Gets or sets the maximum possible value of the range. The thumb could not move after that value.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" Maximum="24"/>
```

#### C#

```
rangeSlider.Maximum=24;
```

### Set Tick Frequency

The `TickFrequency` property is used to decide the number of ticks to be displayed along the track based on Minimum and Maximum values.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" TickFrequency="4" />
```

#### C#

```
rangeSlider.TickFrequency=4;
```

---

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

---

### Set Interval between Snap Points.

The `StepFrequency` property is used to specify the interval between snap points.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" StepFrequency="4"/>
```

#### C#

```
rangeSlider.StepFrequency=4;
```

**Note:** When the **SnapsTo** property is set to **StepValues**, the **StepFrequency** property is enabled.

### Set Snapping Mode

The **SnapsTo** property determines whether the RangeSlider snaps to steps or ticks. Available options for this property are

- **StepValues** - The **StepFrequency** property will be used to specify the interval between snap points.
- **Ticks** - The **TickFrequency** property will be used to specify the interval between snap points
- **None** - The thumb is moved independent of any values.

**Note:** The default option is Ticks.

### XML

```
<range:SfRangeSlider x:Name="rangeslider" SnapsTo="Ticks"/>
```

### C#

```
rangeSlider.SnapsTo=SnapsTo.Ticks;
```

### Orientation

SfRangeSlider provides options to display values and slider to slide either horizontally or vertically.

**Note:** The default option is Vertical.

### Horizontal

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:Range="clr-namespace:Syncfusion.SfRangeSlider.XForms;assembly=Syncfusion.SfRangeSlider.XForms"
xmlns:Local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.RangeSliderSample">
  <ContentPage.Content>
    <Range:SfRangeSlider Orientation="Horizontal"/>
  </Range:SfRangeSlider>
</ContentPage.Content>
</ContentPage>
```

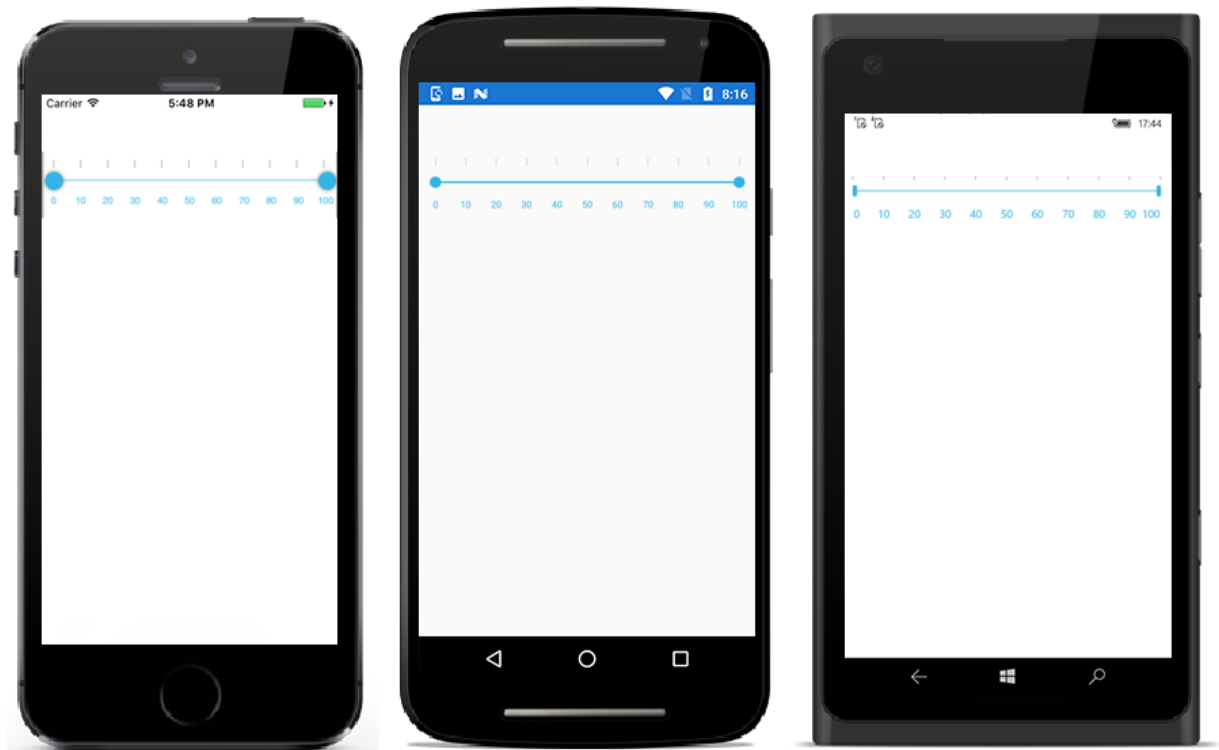
### C#

```
using System.Collections.ObjectModel;
using Syncfusion.XForms.SfRangeSlider;
using Xamarin.Forms;
namespace GettingStarted
{
    /// <summary>
```

```

/// Range slider sample.
/// </summary>
public partial class RangeSliderSample : ContentPage
{
    public RangeSliderSample()
    {
        InitializeComponent();
        SfRangeSlider rangeSlider = new SfRangeSlider();
        rangeSlider.Orientation=Orientation.Horizontal;
        this.Content = rangeSlider;
    }
}

```



## Vertical

### XML

```

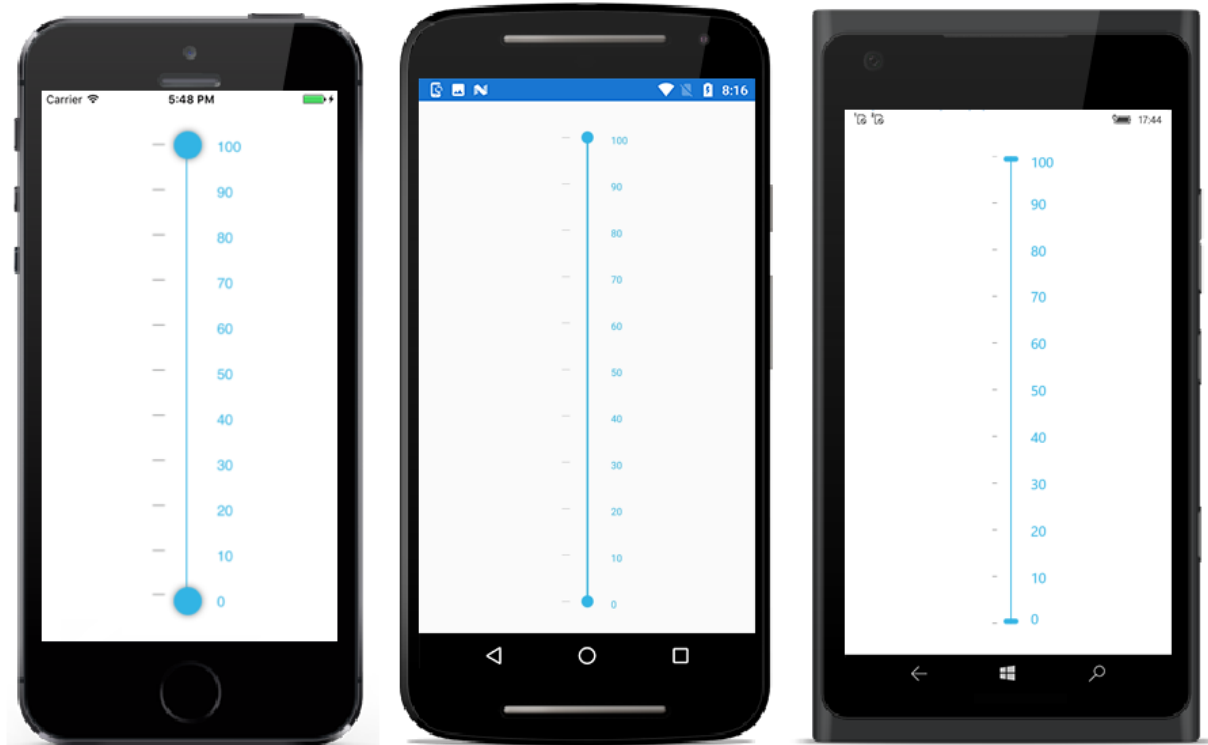
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:Range="clr-namespace:Syncfusion.SfRangeSlider.XForms;assembly=Syncfusion.SfRangeSlider.XForms"
xmlns:Local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.RangeSliderSample">
    <ContentPage.Content>
        <Range:SfRangeSlider Orientation="Vertical"/>
    </Range:SfRangeSlider>
</ContentPage.Content>

```

```
</ContentPage>
```

## C#

```
using System.Collections.ObjectModel;
using Syncfusion.XForms.SfRangeSlider;
using Xamarin.Forms;
namespace GettingStarted
{
    /// <summary>
    /// Range slider sample.
    /// </summary>
    public partial class RangeSliderSample : ContentPage
    {
        public RangeSliderSample()
        {
            InitializeComponent();
            SfRangeSlider rangeSlider = new SfRangeSlider();
            rangeSlider.Orientation=Orientation.Vertical;
            this.Content = rangeSlider;
        }
    }
}
```



## Customizing labels

SfRangeSlider provides option to show or hide the label and position customization.

### Show Value Label

This property allows us to display labels for the ticks. When it sets to true, it displays the label for all the ticks based on the `ValuePlacement` property.

**Note:** The default value of the `ShowValueLabel` property is false.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" ShowValueLabel="true"/>
```

#### C#

```
rangeSlider.ShowValueLabel= true;
```

### Set Custom Label

To display custom labels, `ShowCustomLabel` property should be set to true and need to populate the `CustomLabels` property with observable collection of items by specifying the custom labels for corresponding values.

#### XML

```
<range:SfRangeSlider x:Name="rangeslider" HeightRequest="400"  
ShowCustomLabel="true" CustomLabels="customCollection"/>
```

#### C#

```
SfRangeSlider rangeSlider;  
ObservableCollection<Items> customCollection;  
public RangeSliderPage ()  
{  
    customCollection = new ObservableCollection<Items> ();  
    customCollection.Add(new Items() {Label = "Min", Value= 0});  
    customCollection.Add(new Items() { Label = "Max", Value = 100 });  
    rangeSlider = new SfRangeSlider ();  
    rangeSlider.HeightRequest = 400;  
    rangeSlider.ShowCustomLabel = true;  
    rangeSlider.CustomLabels = customCollection  
}
```



### Value Placement

The `ValuePlacement` property describes the position of the Value relative to ticks.

Available options for this property are:

- BottomRight
- TopLeft

### XML

```
<range:SfRangeSlider x:Name="rangeslider" ValuePlacement="TopLeft"/>
```

### C#

```
rangeSlider.ValuePlacement=ValuePlacement.TopLeft;
```



### XML

```
<range:SfRangeSlider x:Name="rangeslider" ValuePlacement="BottomRight"/>
```

**C#**

```
rangeSlider.ValuePlacement=ValuePlacement.BottomRight;
```

**Label Placement**

The **LabelPlacement** property describes the position of the custom labels relative to ticks.

Available options for this property are:

- BottomRight
- TopLeft

**XML**

```
<range:SfRangeSlider x:Name="rangeslider" LabelPlacement="TopLeft"/>
```

**C#**

```
rangeSlider.LabelPlacement=LabelPlacement.TopLeft;
```

**Customizing label font**

The range slider control provides the **FontFamily**, **FontSize**, and **FontAttribute** properties to customize the value text and custom label text.

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:Range="clr-namespace:Syncfusion.SfRangeSlider.XForms;assembly=Syncfusion.SfRangeSlider.XForms"
xmlns:Local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.RangeSliderSample">
  <ContentPage.Content>
    <Range:SfRangeSlider FontFamily="Times New Roman" ShowCustomLabel="true"
FontAttribute="Italic" FontSize="12" Orientation="Horizontal">
      <Range:SfRangeSlider.CustomLabels>
        <Local:ObservableCollectionList>
          <Range:Items Label="Min" Value="0"/>
          <Range:Items Label="Mid" Value="50"/>
          <Range:Items Label="Max" Value="100"/>
        </Local:ObservableCollectionList>
      </Range:SfRangeSlider.CustomLabels>
    </Range:SfRangeSlider>
  </ContentPage.Content>
</ContentPage>
```

```

</Range:SfRangeSlider>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using System.Collections.ObjectModel;
using Syncfusion.XForms.SfRangeSlider;
using Xamarin.Forms;
namespace GettingStarted
{
    /// <summary>
    /// Range slider sample.
    /// </summary>
    public partial class RangeSliderSample : ContentPage
    {
        public RangeSliderSample()
        {
            InitializeComponent();
            SfRangeSlider rangeSlider = new SfRangeSlider() { FontFamily="Times New Roman", ShowCustomLabel=true, FontAttribute=FontAttributes.Italic, FontSize=12, Orientation=Orientation.Horizontal };
            rangeSlider.CustomLabels = new ObservableCollection<Items>()
            {
                new Items() { Value=0, Label="Min" },
                new Items() { Value=50, Label="Mid" },
                new Items() { Value=100, Label="Max" }
            };
            this.Content = rangeSlider;
        }
    }
}

```

**C#**

```

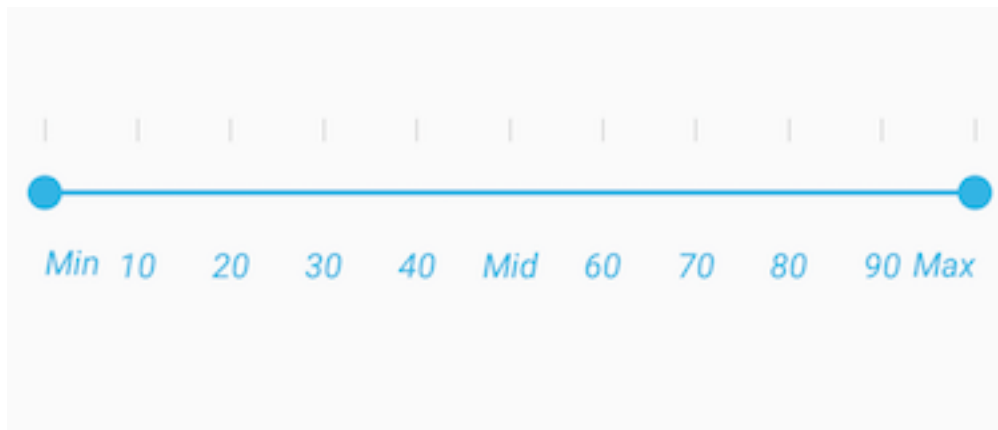
using System.Collections.ObjectModel;
using Syncfusion.XForms.SfRangeSlider;
using Xamarin.Forms;
namespace GettingStarted
{
    /// <summary>
    /// Observable collection of Items list.
    /// </summary>
    public class ObservableCollectionList : ObservableCollection<Items>
    {
    }

    /// <summary>
    /// Range slider sample.
    /// </summary>
    public partial class RangeSliderSample : ContentPage
    {
        public RangeSliderSample()
        {
            InitializeComponent();
        }
    }
}

```



```
}
```



### LabelColor

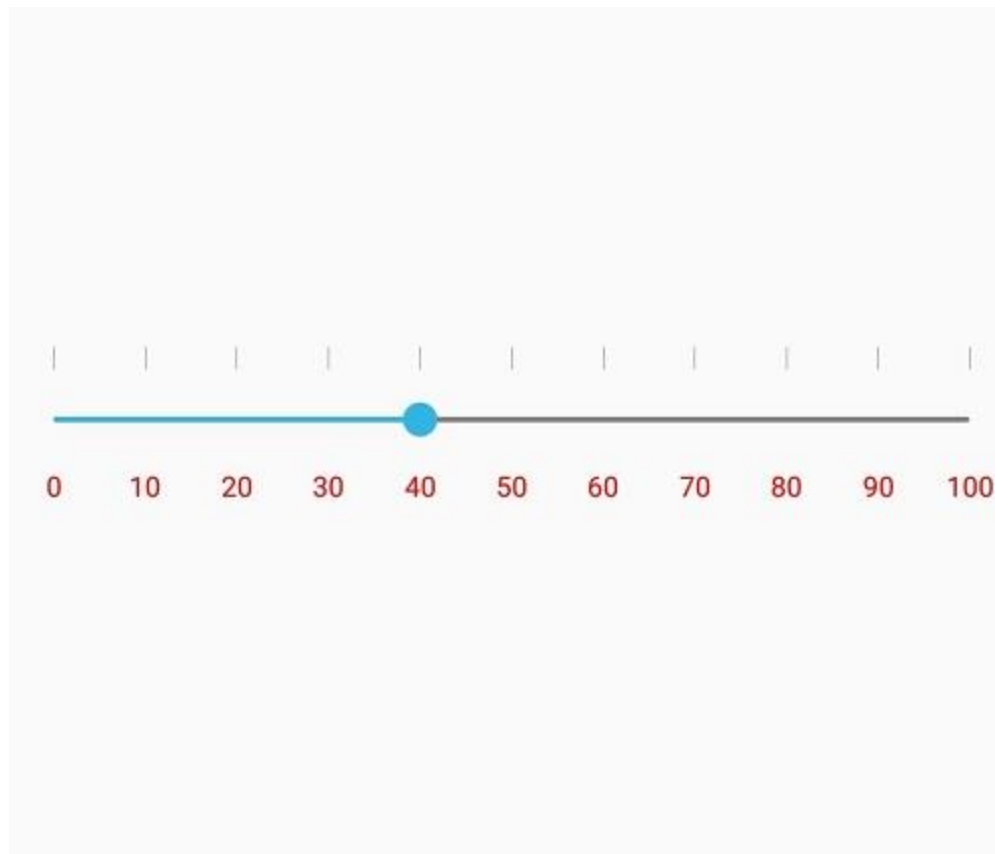
The **LabelColor** property used to change the color of the label.

### XML

```
<range:SfRangeSlider x:Name="rangeslider" LabelColor="Red" Minimum="0"
Maximum="100"/>
```

### C#

```
rangeslider.LabelColor = Color.Red;
```



### ToolTip Support

The ToolTip shows the current value based on thumb position.

### Set ToolTip Precision

The `ToolTipPrecision` property is used to define the precision of the value displayed in the ToolTip.

### XML

```
<range:SfRangeSlider x:Name="rangeslider" ToolTipPrecision="2"/>
```

### C#

```
rangeSlider.ToolTipPrecision = 2;
```

### Set ToolTip Placement

The position of the ToolTip in relation to the thumb can be controlled by the `ToolTipPlacement` property. It has the following options.

1. BottomRight
2. TopLeft
3. None

### *BottomRight*

The ToolTip will be placed either below the Thumb in horizontal orientation or right of the Thumb in vertical orientation.

#### **XML**

```
<range:SfRangeSlider x:Name="rangeslider" ToolTipPlacement="BottomRight"/>
```

#### **C#**

```
rangeSlider.ToolTipPlacement = ToolTipPlacement.BottomRight;
```

### *TopLeft*

the ToolTip will be placed either above the Thumb in horizontal orientation or left of the Thumb in vertical orientation.

#### **XML**

```
<range:SfRangeSlider x:Name="rangeslider" ToolTipPlacement="TopLeft"/>
```

#### **C#**

```
rangeSlider.ToolTipPlacement = ToolTipPlacement.TopLeft;
```

### *None*

ToolTip will be collapsed.

#### **XML**

```
<range:SfRangeSlider x:Name="rangeslider" ToolTipPlacement="None"/>
```

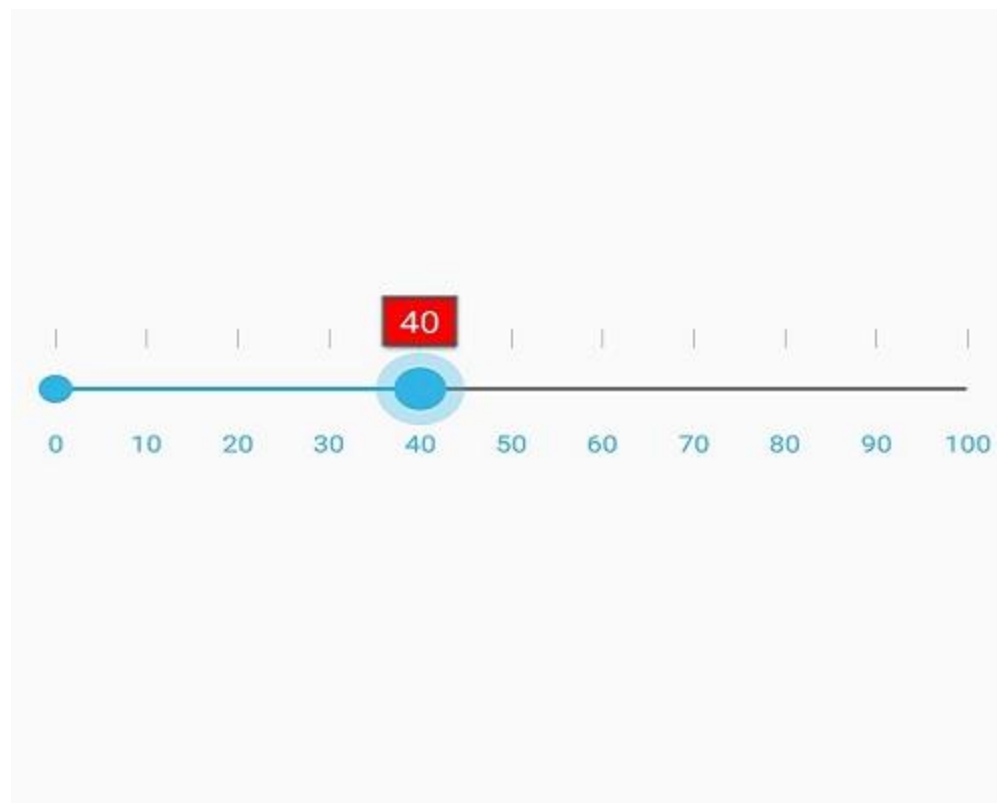
#### **C#**

```
rangeSlider.ToolTipPlacement = ToolTipPlacement.None;
```

### *Tooltip color*

**ToolTipTextColor** - Changes the text color of tooltip.

**ToolTipBackgroundColor** - Changes the background color of tooltip.



#### How to Perform an Action while Selecting a Value?

ValueChanged event will be triggered when value is changed with single thumb. ValueEventArgs has RangeSlider and RangeValue of the control.

Members	Description
RangeSlider	Displays the native control.
Value	Displays the value when moved with single thumb.

#### C#

```
rangeSlider.ValueChanging += (object sender, ValueEventArgs e) =>
{
    float range = e.Value;
    SfRangeSlider rangeSlider = e.RangeSlider;
};
```

#### How to Perform an Action when the Range Get Changing?

RangeChanging event will be triggered when either RangeStart or RangeEnd values are changed. RangeEventArgs has RangeStart and RangeEnd value of SfRangeSlider.

**Note:** ShowRange value must be true.

Members	Description
---------	-------------

RangeSlider	Displays the native control
Start	It shows the start value when moved with thumb
End	It shows the end value when moved with thumb

**C#**

```
rangeSlider.RangeChanging+= (object sender, RangeEventArgs e) =>
{
    float rangeStart = e.Start;
    float rangeEnd = e.End;
    SfRangeSlider rangeSlider = e.RangeSlider;
};
```

[How to get notifications when a thumb drag is started and completed?](#)

The **DragStarted** event is raised when a thumb is dragged. After the thumb releases the pointer capture, the **DragCompleted** event is raised. The **IsStartThumb** property of the **DragThumbEventArgs** returns a boolean value, which indicates the thumb used for performing drag operations.

Members	Description
IsStartThumb	Indicates the thumb used for performing drag operations.

**C#**

```
rangeSlider.DragStarted+=(object sender, DragThumbEventArgs e) =>
{
    //perform the operation
};
rangeSlider.DragCompleted+=(object sender, DragThumbEventArgs e) =>
{
    //perform the operation
};
```

[How to trigger the ThumbTouchDown event?](#)

The **ThumbTouchDown** event occurs when touching the thumb. The argument contains the state of the thumb.

**IsStartThumb** - Gets the state whether thumb touch down position is start or end. If the thumb touch down position is start, then **IsStartThumb** state is true. If it's end, then **IsStartThumb** state is false. It is a read only property.

**C#**

```
private void Rangeslider_ThumbTouchDown(object sender,
Syncfusion.SfRangeSlider.XForms.DragThumbEventArgs e)
{
    var isStartThumb = e.IsStartThumb;
}
```

### How to trigger the ThumbTouchUp event?

**IsStartThumb** - Gets the state whether thumb touch up position is start or end. If the thumb touch up position is start, then **IsStartThumb** state is true. If it's end, then **IsStartThumb** state is false. It is a read only property.

#### C#

```
private void Rangeslider_ThumbTouchUp(object sender,
Syncfusion.SfRangeSlider.XForms.DragThumbEventArgs e)
{
    var isStartThumb= e.IsStartThumb;
}
```

**Note:** **ThumbTouchDown** and **ThumbTouchUp** events applicable only for Android, iOS platform and not for UWP.

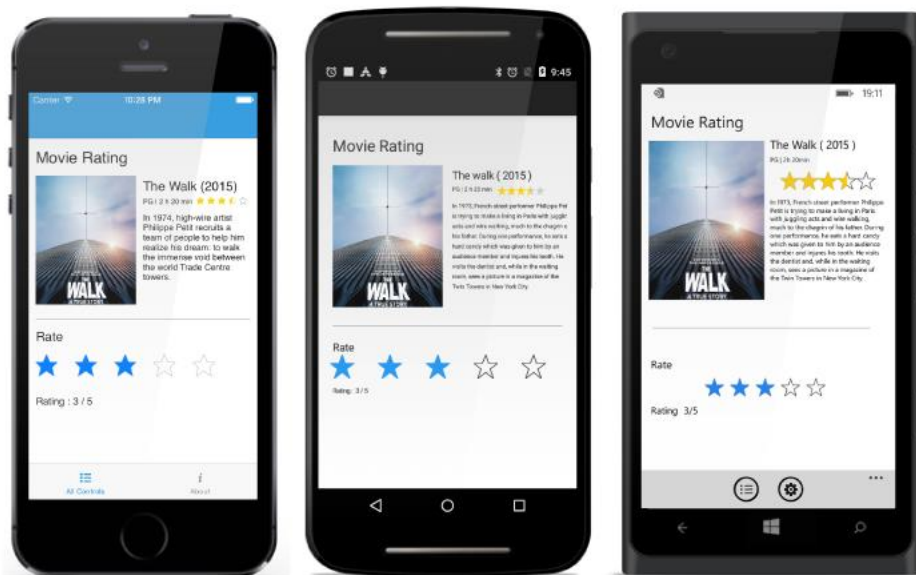
#### AutomationId

**SfRangeSlider** does not have support for **AutomationId** to access the thumb along track. Instead, you can change the **RangeStart**, **RangeEnd** using the co-ordinates and **Value** with the button click event with its **AutomationId**.

## SfRating

### Overview

The Essential Xamarin Rating control provides the number of stars that represents a rating. And also used to configure the item size, item spacing and the number of displayed items in the SfRating control. Essential Xamarin Rating control can be used in various scenarios like movie rating, rating the application etc.



## Key Features

- **Precision:** Options to decide the accuracy level of rating.
- **Item Count:** Support to determine the number of Rating items to be displayed.

## Getting Started

This section explains how to configure a SfRating control in a real-time scenario and also provides a walk-through on some of the customization features available in SfRating control.

### Adding SfRating reference

You can add SfRating reference using one of the following methods:

#### Method 1: Adding SfRating reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfRating). To add SfRating to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfRating](https://www.nuget.org/packages/Syncfusion.Xamarin.SfRating), and then install it.

![Adding SfRating reference from nuget](images/Adding SfRating reference.png)

---

**Note:** Install the same version of SfRating NuGet in all the projects.

---

#### Method 2: Adding SfRating reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfRating control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfRating assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfRating.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfRating.Android.dll Syncfusion.SfRating.XForms.Android.dll Syncfusion.SfRating.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfRating.iOS.dll Syncfusion.SfRating.XForms.iOS.dll Syncfusion.SfRating.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfInput.UWP.dll Syncfusion.SfShared.UWP.dll

	Syncfusion.SfRating.XForms.UWP.dll Syncfusion.SfRating.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
--	--

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** Currently an additional step is required for iOS project. You need to create an instance of the rating custom renderer. If you are adding the references from toolbox, this step is not needed.

Create an instance of SfRatingRenderer in FinishedLaunching overridden method of AppDelegate class in iOS Project as shown below

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    ...
    new SfRatingRenderer ();
    ...
}
```

#### *ReleaseMode issue in UWP platform*

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the SfRating assemblies in **App.xaml.cs** in UWP project as like in below code snippet.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfRatingRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```



The SfRating control is configured entirely in C# code or by using XAML markup. The following steps explains how to create a SfRating and configure its elements.

#### Adding namespace

Add the following namespace.

##### XML

```
<xmlns:rating="clr-namespace:Syncfusion.SfRating.XForms;assembly=Syncfusion.SfRating.XForms"/>
```

##### C#

```
using Syncfusion.SfRating.XForms;
```

#### Initialize Rating

Now, add the SfRating control with a required optimal name using the included namespace.

##### XML

```
<rating:SfRating x:Name="rating" />
```

##### C#

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    this.Content = rating;  
}
```

#### Set Number of Rating Items

The number of rating items to be displayed can be customized in the SfRating control. Users can create a rating application with 5 items as follows. The `ItemCount` property is used to define the number of rating items.

**Note:** The default value of ItemCount is 5.

##### XML

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

##### C#

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.ItemCount = 5;  
}
```

### Set Value

The display value can be set in the SfRating control, which is selected among the items. The following code example shows the display value of 3 with 5 rating items. The `Value` property is used to set display value.

**Note:** The default value of this property is 0.

#### XML

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

#### C#

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.Value = 3;  
}
```

### Precision

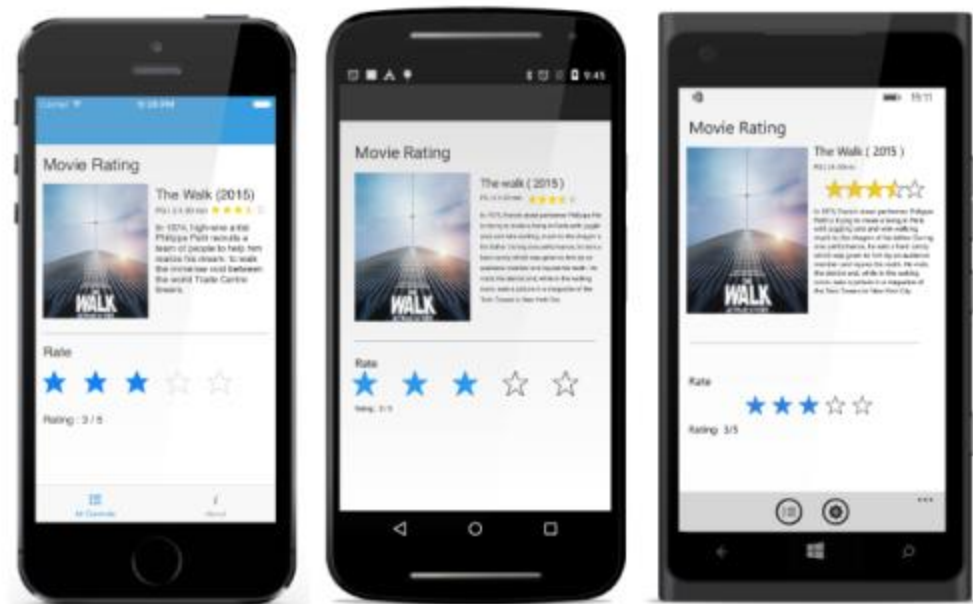
The SfRating control provides an option to rate the items in full, half, and exact values. This can be set using the `Precision` property. By default, the precision mode is `Standard`.

#### XML

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

#### C#

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.Precision = Precision.Standard;  
}
```



The complete Getting Started sample is available in this [documentation](#).

### Precision Mode

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

#### Standard

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

#### XML

```
<rating:SfRating x:Name="rating" Precision="Standard" />
```

#### C#

```
SfRating rating;
public MainPage()
{
    InitializeComponent();
    rating = new SfRating();
    rating.Precision = Precision.Standard;
}
```



#### Half

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

## XML

```
<rating:SfRating x:Name="rating" Precision="Half" />
```

## C#

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.Precision = Precision.Half;  
}
```



### Exact

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

## XML

```
<rating:SfRating x:Name="rating" Precision="Exact" />
```

## C#

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.Precision = Precision.Exact;  
}
```



## Tooltip

Tooltip provides additional information about objects that are unfamiliar to users and are not directly displayed in UI. In the Xamarin.Forms SfRating control, tooltip shows the data of **Value**. It will be displayed when the mouse is hovered over the rating items and will be disappeared when a rating item is selected.

### Set Tooltip Placement

Using **ToolTipPlacement** property, We can define where the ToolTip need to be displayed. ToolTipPlacement having the following three types of placement.

- \*BottomRight,
- \*None,
- \*TopLeft.

**Note:** By default, this property value is set to None.

#### *BottomRight*

The Tooltip will display on bottom of the SfRating control.

#### **XML**

```
<rating:SfRating TooltipPlacement="BottomRight" />
```

#### **C#**

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.TooltipPlacement = TooltipPlacement.BottomRight;  
}
```



#### *TopLeft*

The Tooltip will be displayed on top of the SfRating control.

#### **XML**

```
<rating:SfRating TooltipPlacement="TopLeft" />
```

#### **C#**

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.TooltipPlacement = TooltipPlacement.TopLeft;  
}
```



#### *None*

When we set `ToolTipPlacement` as None, The ToolTip will be disappear.

#### **XML**

```
<rating:SfRating TooltipPlacement="None" />
```

#### **C#**

```
SfRating rating;  
public MainPage()
```

```
{
    InitializeComponent();
    rating = new SfRating();
    rating.TooltipPlacement = TooltipPlacement.None;
}
```



### Set ToolTip Precision

The **ToolTipPrecision** property sets the number precisions to be displayed after decimal point in ToolTip.

**Note:** The default value of ToolTip precision is 1.

### XML

```
<rating:SfRating TooltipPrecision="6" />
```

### C#

```
SfRating rating;
public MainPage()
{
    InitializeComponent();
    rating = new SfRating();
    rating.ToolTipPrecision = 6;
}
```



### Appearance and Styling

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

### Set Size

The **ItemSize** property sets the size of the rating items.

**Note:** By default, property value is 50.

### XML

```
<rating:SfRating ItemSize="20" />
```

### C#

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

```
rating.ItemSize = 20;  
}
```



#### Set Number of Items

The `ItemCount` property sets the number of rating items to be displayed.

**Note:** The default property value is 5.

#### XML

```
<rating:SfRating ItemCount="4" />
```

#### C#

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.ItemCount = 4;  
}
```



#### Set Space between Items

The `ItemSpacing` property sets the spacing between the rating items.

**Note:** By default, property value is 5.

#### XML

```
<rating:SfRating ItemSpacing="20" />
```

#### C#

```
SfRating rating;  
public MainPage()  
{  
    InitializeComponent();  
    rating = new SfRating();  
    rating.ItemSpacing = 20;  
}
```



#### Rating Settings

This section explains about various rating settings available in the SfRating control.

**XML**

```
<rating:SfRating Value="3">
  <rating:SfRating.RatingSettings>
    <rating:SfRatingSettings RatedFill="Red" UnRatedFill="Blue"
      RatedStrokeWidth="3" UnRatedStrokeWidth="2" />
  </rating:SfRating.RatingSettings>
</rating:SfRating>
```

**C#**

```
SfRating rating;
SfRatingSettings ratingSettings;
public MainPage()
{
    InitializeComponent();
    rating = new SfRating();
    ratingSettings = new SfRatingSettings();
    ratingSettings.RatedFill = Color.Red;
    ratingSettings.UnRatedFill = Color.Blue;
    ratingSettings.RatedStrokeWidth = 3;
    ratingSettings.UnRatedStrokeWidth = 2;
    rating.RatingSettings = ratingSettings;
}
```

**Restrict User Selection**

SfRating control provides support for changeable or unchangeable values for Rating control. This is achieved by the **ReadOnly** property. When this property is set to True, the Rating value becomes unchangeable. By default, this property value is set to False.

**C#**

```
rating.ReadOnly=true;
```

**XML**

```
<rating:SfRating x:Name="rating" ReadOnly="true" />
```

**Appearance Customization**

We can customize the rated and unrated items Color, Stroke width and Stroke color using the following properties of SfRatingSettings.

- RatedFill
- UnRatedFill
- RatedStroke
- UnRatedStroke
- RatedStrokeWidth
- UnRatedStrokeWidth



### Set Fill Color

SfRating control has support to set the fill color for the selected and unselected items.

#### *Selected Items*

The **RatedFill** property fills the rated item with the specified solid color in the SfRating control.

#### **XML**

```
<rating:SfRating Value="3">
  <rating:SfRating.RatingSettings>
    <rating:SfRatingSettings RatedFill="Red"/>
  </rating:SfRating.RatingSettings>
</rating:SfRating>
```

#### **C#**

```
SfRating rating= new SfRating();
rating.Value = 3;
SfRatingSettings ratingSettings = new SfRatingSettings();
ratingSettings.RatedFill = Color.Red;
rating.RatingSettings = ratingSettings;
```



#### *Unselected Items*

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

#### **XML**

```
<rating:SfRating Value="3">
  <rating:SfRating.RatingSettings>
    <rating:SfRatingSettings UnRatedFill="Gray"/>
  </rating:SfRating.RatingSettings>
</rating:SfRating>
```

#### **C#**

```
SfRating rating= new SfRating();
rating.Value = 3;
SfRatingSettings ratingSettings = new SfRatingSettings();
ratingSettings.UnRatedFill = Color.Gray;
rating.RatingSettings = ratingSettings;
```



### Set Stroke Color

SfRating control has support to set the stroke color for the selected and unselected items.

#### *Selected Items*

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

**XML**

```
<rating:SfRating Value="3">
  <rating:SfRating.RatingSettings>
    <rating:SfRatingSettings RatedStroke="Black"/>
  </rating:SfRating.RatingSettings>
</rating:SfRating>
```

**C#**

```
SfRating rating= new SfRating();
rating.Value = 3;
SfRatingSettings ratingSettings = new SfRatingSettings();
ratingSettings.RatedStroke=Color.Black;
rating.RatingSettings = ratingSettings;
```

*Unselected Items*

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

**XML**

```
<rating:SfRating Value="3">
  <rating:SfRating.RatingSettings>
    <rating:SfRatingSettings UnRatedStroke="Black"/>
  </rating:SfRating.RatingSettings>
</rating:SfRating>
```

**C#**

```
SfRating rating= new SfRating();
rating.Value = 3;
SfRatingSettings ratingSettings = new SfRatingSettings();
ratingSettings.UnRatedStroke = Color.Black;
rating.RatingSettings = ratingSettings;
```

*Set Stroke Width*

SfRating control has support to set the stroke width for the selected and unselected items.

*Selected Items*

The **RatedStrokeWidth** property sets the stroke width for the rated item with the specified value in the SfRating control.

**XML**

```
<rating:SfRating Value="3">
  <rating:SfRating.RatingSettings>
    <rating:SfRatingSettings RatedStrokeWidth="3"/>
  </rating:SfRating.RatingSettings>
</rating:SfRating>
```

```
</rating:SfRating.RatingSettings>
</rating:SfRating>
```

**C#**

```
SfRating rating= new SfRating();
rating.Value = 3;
SfRatingSettings ratingSettings = new SfRatingSettings();
ratingSettings.RatedStrokeWidth=3;
rating.RatingSettings = ratingSettings;
```

*Unselected Items*

The `UnRatedStrokeWidth` property sets the stroke width for the unrated item with the specified value in the SfRating control.

**XML**

```
<rating:SfRating Value="3">
  <rating:SfRating.RatingSettings>
    <rating:SfRatingSettings UnRatedStrokeWidth="3"/>
  </rating:SfRating.RatingSettings>
</rating:SfRating>
```

**C#**

```
SfRating rating= new SfRating();
rating.Value = 3;
SfRatingSettings ratingSettings = new SfRatingSettings();
ratingSettings.UnRatedStrokeWidth=3;
rating.RatingSettings = ratingSettings;
```

*Custom Views*

SfRating Items control provides support to add custom views.

**Note:** To use custom views in Xamarin.Forms UWP platform, you need to set `ItemCount` value.

*Add SfRating items*

To customize the view of rating items, create and set custom view as `SelectedView` and `UnSelectedView` of `SfRatingItem`. Refer to the following code snippet to create a custom view.

**XML**

```
<rating:SfRating EnableCustomView="true">
  <rating:SfRating.Items>
    <rating:SfRatingItem/>
  </rating:SfRating.Items>
</rating:SfRating>
```

**C#**

```
SfRating rating= new SfRating();  
SfRatingItem ratingItem = new SfRatingItem();  
rating.Items = ratingItem;  
rating.EnableCustomView = true;
```

## Set selected view

The **SelectedView** property is used to apply the given SelectedView to selected rating item.

**XML**

```
<rating:SfRatingItem>  
<rating:SfRatingItem.SelectedView>  
<Image Source="Angry_selected.png"/>  
</rating:SfRatingItem.SelectedView>  
</rating:SfRatingItem>
```

**C#**

```
ratingItem.SelectedView = new Image() { Source = "Angry_selected.png",  
Aspect = Aspect.Fill };
```

## Set unselected view

The **UnSelectedView** property is used to apply the given UnSelectedView to unselected rating item.

**XML**

```
<rating:SfRatingItem>  
<rating:SfRatingItem.UnSelectedView>  
<Image Source="Angry_Unselected.png"/>  
</rating:SfRatingItem.UnSelectedView>  
</rating:SfRatingItem>
```

**C#**

```
ratingItem.UnSelectedView = new Image() { Source = "Angry_Unselected.png",  
Aspect = Aspect.Fill };
```

## Add Items

The **Items** property is used to hold the collection of SfRatingItem.

---

**Note:** SfRatingItem keeps both selected and unselected view respectively.

---

**C#**

```
ObservableCollection<SfRatingItem> ratingItemList = new  
ObservableCollection<SfRatingItem>();  
ratingItemList.Add(ratingItem);  
rating.Items = ratingItemList;
```

### Enable custom items

When the `EnableCustomItems` property is enabled, the custom items added in the rating items will be displayed.

### XML

```
<rating:SfRating x:Name="rating" EnableCustomView="True" ItemCount="5">
  <rating:SfRating.Items>
    <collection:ObservableCollection x:TypeArguments="rating:SfRatingItem">
      <rating:SfRatingItem>
        <rating:SfRatingItem.SelectedView>
          <Image Source="Angry_selected.png"/>
        </rating:SfRatingItem.SelectedView>
        <rating:SfRatingItem.UnSelectedView>
          <Image Source="Angry_Unselected.png"/>
        </rating:SfRatingItem.UnSelectedView>
      </rating:SfRatingItem>
      <rating:SfRatingItem>
        <rating:SfRatingItem.SelectedView>
          <Image Source="UnHappy_selected.png"/>
        </rating:SfRatingItem.SelectedView>
        <rating:SfRatingItem.UnSelectedView>
          <Image Source="UnHappy_Unselected.png"/>
        </rating:SfRatingItem.UnSelectedView>
      </rating:SfRatingItem>
      <rating:SfRatingItem>
        <rating:SfRatingItem.SelectedView>
          <Image Source="Neutral_selected.png"/>
        </rating:SfRatingItem.SelectedView>
        <rating:SfRatingItem.UnSelectedView>
          <Image Source="Neutral_Unselected.png"/>
        </rating:SfRatingItem.UnSelectedView>
      </rating:SfRatingItem>
      <rating:SfRatingItem>
        <rating:SfRatingItem.SelectedView>
          <Image Source="Happy_selected.png"/>
        </rating:SfRatingItem.SelectedView>
        <rating:SfRatingItem.UnSelectedView>
          <Image Source="Happy_Unselected.png"/>
        </rating:SfRatingItem.UnSelectedView>
      </rating:SfRatingItem>
      <rating:SfRatingItem>
        <rating:SfRatingItem.SelectedView>
          <Image Source="Excited_selected.png"/>
        </rating:SfRatingItem.SelectedView>
        <rating:SfRatingItem.UnSelectedView>
          <Image Source="Excited_Unselected.png"/>
        </rating:SfRatingItem.UnSelectedView>
      </rating:SfRatingItem>
    </collection:ObservableCollection>
  </rating:SfRating.Items>
</rating:SfRating>
```

### C#

```
SfRating rating;
```

```

public MainPage()
{
    InitializeComponent();
    rating = new SfRating();
    rating.EnableCustomView = true;
    rating.ItemCount = 5;
    SfRatingItem angry = new SfRatingItem();
    angry.SelectedView = new Image() { Source = "Angry_selected.png" };
    angry.UnSelectedView = new Image() { Source = "Angry_Unselected.png" };
    SfRatingItem unhappy = new SfRatingItem();
    unhappy.SelectedView = new Image() { Source = "UnHappy_selected.png" };
    unhappy.UnSelectedView = new Image() { Source = "UnHappy_Unselected.png" };
    SfRatingItem neutral = new SfRatingItem();
    neutral.SelectedView = new Image() { Source = "Neutral_selected.png" };
    neutral.UnSelectedView = new Image() { Source = "Neutral_Unselected.png" };
    SfRatingItem happy = new SfRatingItem();
    happy.SelectedView = new Image() { Source = "Happy_selected.png" };
    happy.UnSelectedView = new Image() { Source = "Happy_Unselected.png" };
    SfRatingItem excited = new SfRatingItem();
    excited.SelectedView = new Image() { Source = "Excited_selected.png" };
    excited.UnSelectedView = new Image() { Source = "Excited_Unselected.png" };
    ObservableCollection<SfRatingItem> ratingItemList = new
    ObservableCollection<SfRatingItem>();
    customItems.Add(angry);
    customItems.Add(unhappy);
    customItems.Add(neutral);
    customItems.Add(happy);
    customItems.Add(excited);
    rating.Items = ratingItemList;
    this.Content = rating;
}

```



### View Range Selection

When using CustomView in SfRating, Only one item will be rated. If you need to change the view of all rated CustomView items, Use the `EnableViewRangeSelection` boolean property.

**Note:** The `EnableViewRangeSelection` property is used only for CustomViews.

### XML

```

<rating:SfRating x:Name="rating" EnableCustomView="True" ItemCount="5"
EnableViewRangeSelection="True" />

```

### C#

```

SfRating rating;
public MainPage()

```

```
{  
InitializeComponent();  
-----  
rating.EnableCustomView = true;  
rating.EnableViewRangeSelection = true;  
-----  
}
```



### AutomationId

The SfRating control has built-in **AutomationId** for inner elements. The **AutomationId** API allows the automation framework to find and interact with the inner elements of the SfRating control. To keep unique AutomationId, these inner elements' AutomationIds are updated based on the control's AutomationId.

For example, if you set SfRating's **AutomationId** as "Feedback" and you select the fourth item, then the automation framework will interact with the SfRating as "Feedback Rating 4". When the rating is added as custom view, the Automation framework will interact with the element's **AutomationId**.

---

**Note:** AutomationId support works only when the precision mode of SfRating is set as Standard.

---



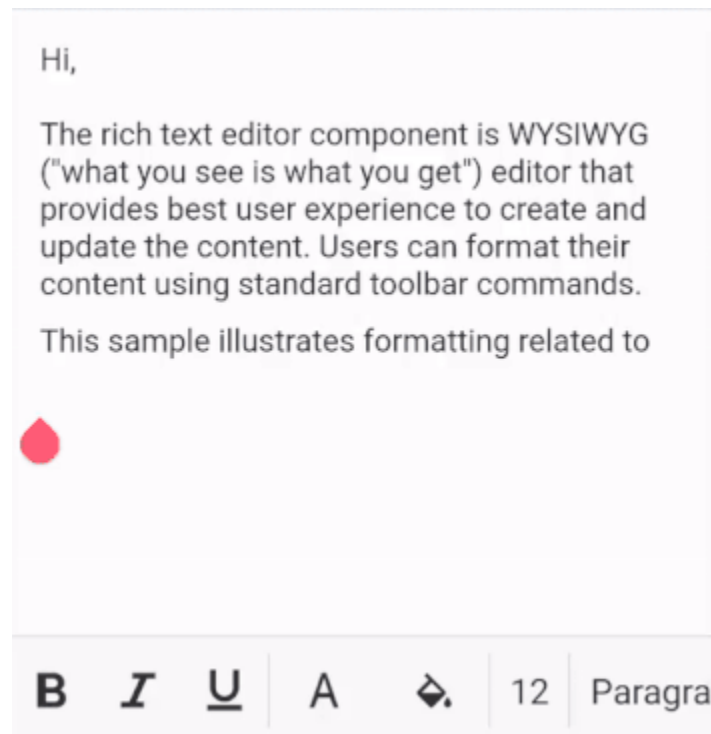
## SfRichTextEditor

### Overview

The Xamarin Rich Text Editor control is a WYSIWYG editor that provides a great user experience for composing or editing rich text content from your Xamarin.Forms applications. Users can format their content using standard toolbar commands.

## Key features

- Applies formatting such as bold, italics, and underline.
- Applies font color and background color to the content.
- Customizes the text size and selection.
- Creates bulleted and numbered lists.



## Getting Started

This section explains the steps required to work with the Rich Text Editor control for Xamarin.Forms.

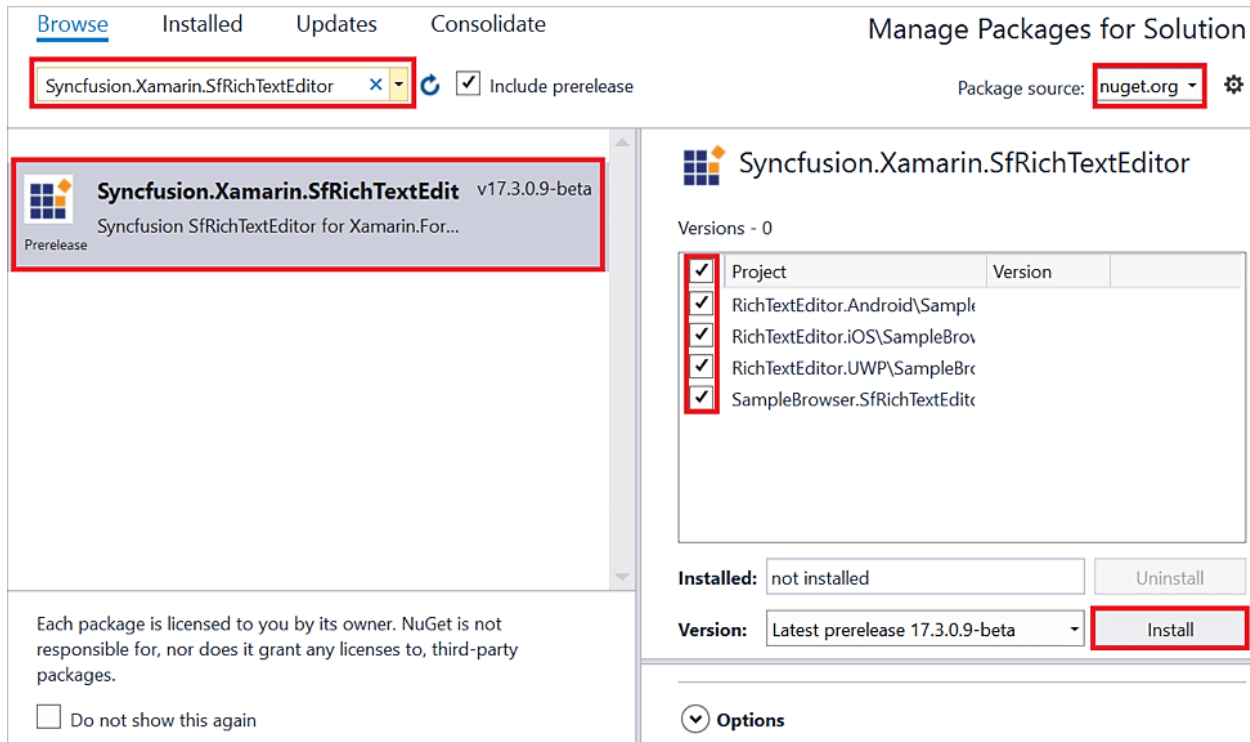
### Adding SfRichTextEditor reference

You can add Rich Text Editor reference using one of the following methods:

#### Method 1: Adding SfRichTextEditor reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfRichTextEditor). To add Rich Text Editor to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfRichTextEditor](https://www.nuget.org/packages/Syncfusion.Xamarin.SfRichTextEditor) and then install it.





**Note:** Install the same version of SfRichTextEditor NuGet in all the projects.

### Method 2: Adding SfRichTextEditor reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the Rich Text Editor control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfRichTextEditor assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from the NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfRichTextEditor.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfRichTextEditor.XForms.Android.dll Syncfusion.SfRichTextEditor.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfRichTextEditor.XForms.iOS.dll Syncfusion.SfRichTextEditor.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll

UWP	Syncfusion.SfRichTextEditor.XForms.UWP.dll Syncfusion.SfRichTextEditor.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
-----	--

**Note:** To learn more about obtaining Syncfusion components, refer to [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Refer to the [Syncfusion license key](#) to learn about registering Syncfusion license key in your Xamarin application to use Syncfusion components.

### Launching the application on each platform with Rich Text Editor

To use the Rich Text Editor in an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, this step is not required.

#### iOS

To launch the Rich Text Editor in iOS, call the 'SfRichTextEditorRenderer.Init()' in the FinishedLaunching overridden method of the AppDelegate class after the Xamarin.Forms framework has been initialized and before the LoadApplication is called as demonstrated in the following code sample.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary
options)
{
    ...
    global::Xamarin.Forms.Forms.Init();
    // Add the below line for using SfRichTextEditor.
    Syncfusion.XForms.iOS.RichTextEditor.SfRichTextEditorRenderer.Init();
    LoadApplication(new App());
    ...
}
```

#### Universal Windows Platform (UWP)

To launch the Rich Text Editor in UWP, initialize the Rich Text Editor assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with Rich Text Editor in **Release** mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    if (rootFrame == null)
    {
        List<Assembly> assembliesToInclude = new List<Assembly>();
        // Add the below line for using SfRichTextEditor.
    }
}
```

```

assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.RichTextEditor.SfRichTextEditorRenderer).GetTypeInfo().Assembly);
// Add the below line for using SfButton.
assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.Buttons.SfButtonRenderer).GetTypeInfo().Assembly);
// Add the below line for using SfComboBox.
assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.ComboBox.SfComboBoxRenderer).GetTypeInfo().Assembly);
// Add the below line for using SfBorder.
assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.Border.SfBorderRenderer).GetTypeInfo().Assembly);
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
}
...
}

```

### Android

The Android platform does not require any additional configuration to render the Rich Text Editor.

### Initializing the Rich Text Editor

Import the Rich Text Editor namespace as demonstrated in the following code sample in your respective page.

#### XML

```

xmlns:richtexteditor="clr-namespace:Syncfusion.XForms.RichTextEditor;assembly=Syncfusion.SfRichTextEditor.XForms"

```

#### C#

```

using Syncfusion.XForms.RichTextEditor;

```

Then, initialize the SfRichTextEditor as shown in the following code:

#### XML

```

<StackLayout>
<richtexteditor:SfRichTextEditor VerticalOptions="FillAndExpand" Text= "The rich text editor component is WYSIWYG editor that provides the best user experience to create and update the content" />
</StackLayout>

```

#### C#

```

public MainPage()
{
InitializeComponent();
StackLayout stack = new StackLayout();
SfRichTextEditor editor = new SfRichTextEditor
{
VerticalOptions = LayoutOptions.FillAndExpand,
Text = "The rich text editor component is WYSIWYG editor that provides the best user experience to create and update the content"
};
}

```

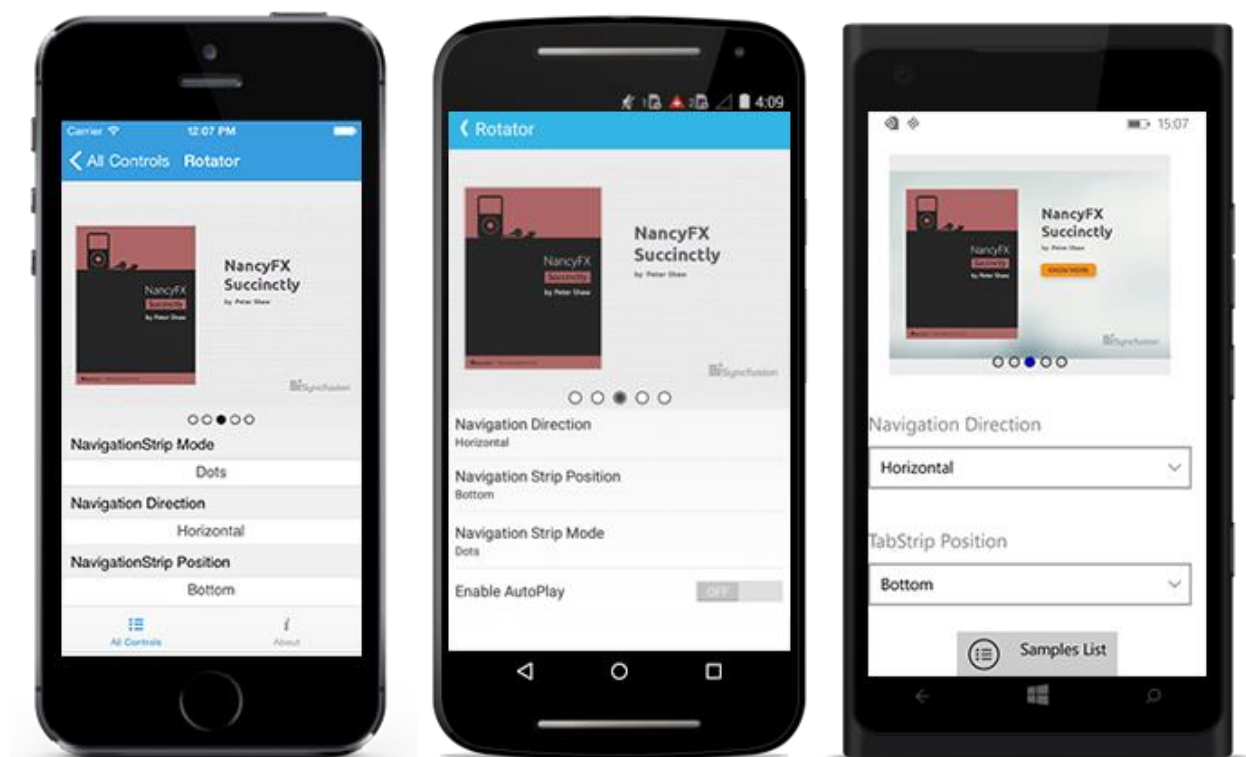
```
stack.Children.Add(editor);  
this.Content = stack;  
}
```

The complete Getting Started sample is available in this [link](#)

## SfRotator

### Overview

The SfRotator is a data control used to display image data and navigate through them. The images can be selected either by Thumbnail or by Dots support.



### Key Features

- **Modes** - Options to navigate data using dots or thumbnail navigation modes.
- **Position** - Support to decide the placement position of dots or thumbnail items in any of the four sides.
- **Autoplay and Items Looping** - Options for auto playing items and navigate items in loop.

### Getting Started

This section explains you the steps to configure a SfRotator control in a real-time scenario and also provides a walk-through on some of the customization features available in SfRotator control.

#### Adding SfRotator reference

You can add SfRotator reference using one of the following methods:

**Method 1: Adding SfRotator reference from nuget.org**

Syncfusion Xamarin components are available in [nuget.org](https://www.syncfusion.com/nuget-packages). To add SfRotator to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfRotator](#), and then install it.

![Adding SfRotator reference from nuget](images/Adding SfRotator reference.png)

---

**Note:** Install the same version of SfRotator NuGet in all the projects.

---

**Method 2: Adding SfRotator reference from toolbox**

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfRotator control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

**Method 3: Adding SfRotator assemblies manually from the installed location**

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfRotator.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfRotator.Android.dll Syncfusion.SfRotator.XForms.Android.dll Syncfusion.SfRotator.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfRotator.iOS.dll Syncfusion.SfRotator.XForms.iOS.dll Syncfusion.SfRotator.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfRotator.UWP.dll Syncfusion.SfRotator.XForms.UWP.dll Syncfusion.SfRotator.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Launching the SfRotator on each platform

To use SfRotator inside an application, each platform application must initialize the SfRotator renderer. This initialization step varies from platform to platform and is discussed in the following sections.

#### Android and UWP

The Android and UWP launches the SfRotator without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

#### iOS

To launch SfRotator in iOS, need to create an instance of SfRotatorRenderer in FinishedLaunching overridden method of AppDelegate class in iOS Project as shown below.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    new SfRotatorRenderer();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### ReleaseMode issue in UWP platform

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the SfRotator assemblies in **App.xaml.cs** in UWP project as like in below code snippet.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfRotatorRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Create a Simple SfRotator

The SfRotator control is configured entirely in C# code or by using XAML markup. The following steps explain on how to create a SfRotator and configure its elements,

- Adding namespace for the added assemblies.

**XML**

```
xmlns:rotator="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
```

**C#**

```
using Syncfusion.SfRotator.XForms;
```

- Now add the SfRotator control with a required optimal name by using the included namespace.

**XML**

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:GettingStarted"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="GettingStarted.RotatorControlPage">
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator" />
</ContentPage.Content>
</ContentPage>
```

**C#**

```
using Syncfusion.SfRotator.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class RotatorControlPage : ContentPage
    {
        public RotatorControlPage()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            this.Content = rotator;
        }
    }
}
```

**Add Rotator Items**

We can populate the rotator's items by using any one of the following ways,

- Through SfRotatorItem
- Through ItemTemplate

*Through SfRotatorItem*

By passing the list of **SfRotatorItem**, we can get the view of SfRotator control. In that we can pass Images as well as Item content.

The following code example illustrates to add list of Images in Rotator ,

### C#

```
using Syncfusion.SfRotator.XForms;
using System.Collections.Generic;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        SfRotator rotator = new SfRotator();
        StackLayout stackLayout = new StackLayout();
        public Rotator()
        {
            InitializeComponent();
            stackLayout.HeightRequest = 300;
            List<SfRotatorItem> collectionOfItems = new List<SfRotatorItem>();
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie1.png" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie2.png" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie3.png" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie4.png" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie5.png" });
            rotator.DataSource = collectionOfItems;
            stackLayout.Children.Add(rotator);
            this.Content = stackLayout;
        }
    }
}
```



The following code example illustrates to add list of items through ItemContent API in Rotator ,

### C#

```
using Syncfusion.SfRotator.XForms;
```



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        SfRotator rotator;
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            List<SfRotatorItem> collectionOfItems = new List<SfRotatorItem>();
            collectionOfItems.Add(new SfRotatorItem() { ItemContent = new
            Xamarin.Forms.Button() { Text = "RotatorButton", TextColor = Color.White,
            BackgroundColor = Color.FromHex("#7E6E6B"), FontSize = 12 } });
            collectionOfItems.Add(new SfRotatorItem() { ItemContent = new Label() { Text
            = "RotatorLabel", BackgroundColor = Color.FromHex("#7E6E6B"), FontSize = 12
            } });
            collectionOfItems.Add(new SfRotatorItem() { ItemContent = new Image() {
            Source = "image1.png", Aspect = Aspect.AspectFit } });
            rotator.DataSource = collectionOfItems;
            this.Content = rotator;
        }
    }
}
```



#### *Through ItemTemplate*

ItemTemplate property of SfRotator control is used to customize the contents of SfRotator items. ItemTemplate provides common template with different data. SfRotator items can be populated with a collection of image data. This collection includes Arrays, Lists and DataTables.

#### **C#**

```
// Model Class for Rotator.
public RotatorModel(string imageString)
{
    Image = imageString;
}
private String _image;
public String Image
{
    get { return _image; }
    set { _image = value; }
}
```

Create and populate Rotator collection as follows

#### **C#**

```
// ViewModel class for Rotator.
public RotatorViewModel()
{
    ImageCollection.Add(new RotatorModel("movie1.png"));
    ImageCollection.Add(new RotatorModel("movie2.png"));
    ImageCollection.Add(new RotatorModel("movie3.png"));
    ImageCollection.Add(new RotatorModel("movie4.png"));
    ImageCollection.Add(new RotatorModel("movie5.png"));
}
```

```

}
private List<RotatorModel> imageCollection = new List<RotatorModel>();
public List<RotatorModel> ImageCollection
{
    get { return imageCollection; }
    set { imageCollection = value; }
}

```

## XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
NavigationDirection="Horizontal"
NavigationStripMode="Dots"
BackgroundColor="#ececec">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            var ImageCollection = new List<RotatorModel> {

```

```
new RotatorModel ("movie1.png"),
new RotatorModel ("movie2.png"),
new RotatorModel ("movie3.png"),
new RotatorModel ("movie4.png"),
new RotatorModel ("movie5.png")
};
var itemTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var nameLabel = new Image();
    nameLabel.SetBinding(Image.SourceProperty, "Image");
    grid.Children.Add(nameLabel);
    return grid;
});
rotator.ItemTemplate = itemTemplate;
rotator.ItemsSource = ImageCollection;
this.Content = rotator;
}
}
public class RotatorModel
{
    public RotatorModel(string imageString)
    {
        Image = imageString;
    }
    private String _image;
    public String Image
    {
        get { return _image; }
        set { _image = value; }
    }
}
}
```

- Set the `BindingContext` for the items collection.

### XML

```
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
```

### C#

```
rotator.BindingContext = new RotatorViewModel();
```



**Information:** Rotator's Images are placed within the application folder for Android, iOS and UWP with build action Android Resource, Bundled Resource and Content respectively.

**Note:** In addition, rotator provides a support to load the Images from URL and SD Card location.

### Setting Navigation Mode

SfRotator provides option to display the navigating items either in Thumbnail or Dots mode. The navigation mode for navigating the items can be decided using `NavigationMode` property.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
  <ContentPage.BindingContext>
    <local:RotatorViewModel/>
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <syncfusion:SfRotator x:Name="rotator"
      NavigationDelay="2000"
      ItemsSource="{Binding ImageCollection}"
      SelectedIndex="2"
      NavigationDirection="Horizontal"
      NavigationStripMode="Thumbnail"
      BackgroundColor="#ececec">
      <syncfusion:SfRotator.ItemTemplate>
        <DataTemplate>
          <Image Source="{Binding Image}"/>
        </DataTemplate>
      </syncfusion:SfRotator.ItemTemplate>
    </syncfusion:SfRotator>
  </ContentPage.Content>
</ContentPage>
```

```

</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

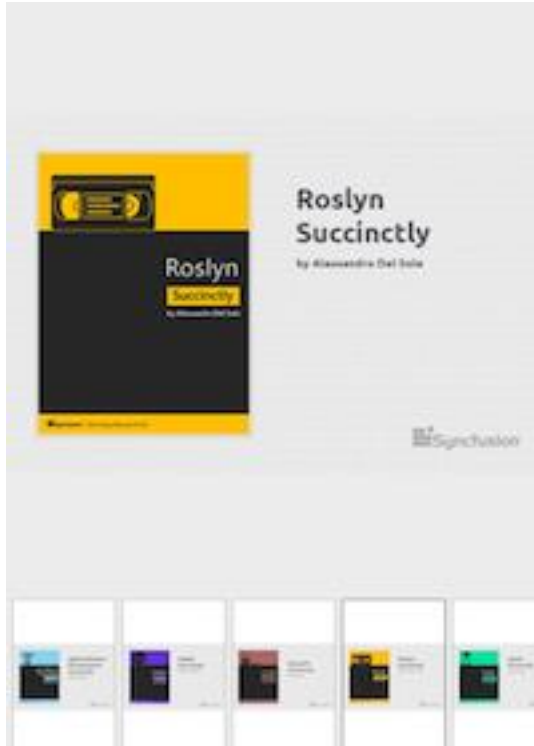
## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            var ImageCollection = new List<RotatorModel> {
                new RotatorModel ("movie1.png"),
                new RotatorModel ("movie2.png"),
                new RotatorModel ("movie3.png"),
                new RotatorModel ("movie4.png"),
                new RotatorModel ("movie5.png")
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            rotator.ItemTemplate = itemTemplate;
            rotator.NavigationStripMode = NavigationStripMode.Thumbnail;
            rotator.ItemsSource = ImageCollection;
            this.Content = rotator;
        }
    }
    public class RotatorModel
    {
        public RotatorModel(string imageString)
        {
            Image = imageString;
        }
        private String _image;
        public String Image
        {
            get { return _image; }
        }
    }
}

```

```
set { _image = value; }
}
}
}
```



### Customizing Position

The placement position of navigation strip items such as Thumbnail or Dots can be customized in SfRotator. This can be specified using `NavigationStripPosition` property.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
NavigationDirection="Horizontal"
NavigationStripMode="Dots"
BackgroundColor="#ececec"
NavigationStripPosition="Top">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
```

```

<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            var ImageCollection = new List<RotatorModel> {
                new RotatorModel ("movie1.png"),
                new RotatorModel ("movie2.png"),
                new RotatorModel ("movie3.png"),
                new RotatorModel ("movie4.png"),
                new RotatorModel ("movie5.png")
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            rotator.ItemTemplate = itemTemplate;
            rotator.NavigationStripMode = NavigationStripMode.Dots;
            rotator.NavigationStripPosition = NavigationStripPosition.Top;
            rotator.ItemsSource = ImageCollection;
            this.Content = rotator;
        }
    }
    public class RotatorModel
    {
        public RotatorModel(string imageString)
        {
            Image = imageString;
        }
        private String _image;
        public String Image

```



```
{  
get { return _image; }  
set { _image = value; }  
}  
}  
}
```



You can find the complete getting started sample from this [link](#).

## Populating Data

SfRotator control supports binding to different data sources such as IList Data Source, Observable Collection Data Source.

### Through Binding

This section explains about setting Item Source and applying custom template to the data.

#### Create a Model with Data

SfRotator items can be populated with a collection of image data. You can assign a collection to it. Collections include arrays, Lists and DataTables. For example you may want to create a Rotator model with Image as follows.

#### C#

```
// Model Class for Rotator.  
public RotatorModel(string imageString)  
{  
    Image = imageString;  
}  
private String _image;  
public String Image  
{  
    get { return _image; }  
    set { _image = value; }  
}
```

Create and populate SfRotator collection as follows

### C#

```
// ViewModel class for Rotator.
public RotatorViewModel()
{
    ImageCollection.Add(new RotatorModel("movie1.png"));
    ImageCollection.Add(new RotatorModel("movie2.png"));
    ImageCollection.Add(new RotatorModel("movie3.png"));
    ImageCollection.Add(new RotatorModel("movie4.png"));
    ImageCollection.Add(new RotatorModel("movie5.png"));
}
private List<RotatorModel> imageCollection = new List<RotatorModel>();
public List<RotatorModel> ImageCollection
{
    get { return imageCollection; }
    set { imageCollection = value; }
}
```

Assigning collection to ItemSource ,

### XML

```
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
```

### C#

```
rotator.BindingContext = new RotatorViewModel();
```

### *Binding the Data with Custom Template*

SfRotator provides support to add a custom view as RotatorItems by designing a view inside its ItemTemplate. This template will be applied for all its items and its data will be binded.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
NavigationDirection="Horizontal"
NavigationStripMode="Dots"
BackgroundColor="#ececec">
<syncfusion:SfRotator.ItemTemplate>
```

```

<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class Rotator : ContentPage
{
public Rotator()
{
InitializeComponent ();
SfRotator rotator = new SfRotator();
var ImageCollection = new List<RotatorModel> {
new RotatorModel ("movie1.png"),
new RotatorModel ("movie2.png"),
new RotatorModel ("movie3.png"),
new RotatorModel ("movie4.png"),
new RotatorModel ("movie5.png")
};
var itemTemplate = new DataTemplate(() =>
{
var grid = new Grid();
var nameLabel = new Image();
nameLabel.SetBinding(Image.SourceProperty, "Image");
grid.Children.Add(nameLabel);
return grid;
});
rotator.ItemTemplate = itemTemplate;
rotator.ItemsSource = ImageCollection;
this.Content = rotator;
}
}
public class RotatorModel
{
public RotatorModel(string imageString)
{
Image = imageString;
}
private String _image;
public String Image
{

```

```

get { return _image; }
set { _image = value; }
}
}
}

```

### Through Rotator Item

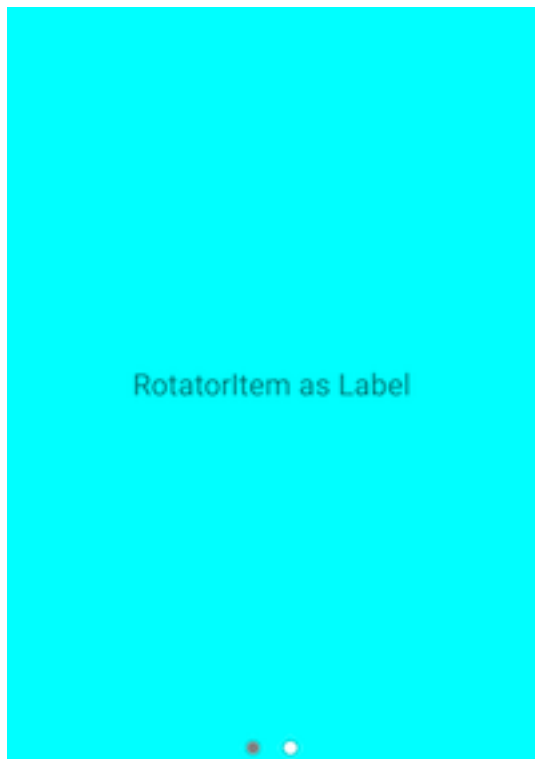
The ItemTemplate provides common template with different data, whereas if different views for every items is needed, it can also be provided using `ItemContent` property in `SfRotatorItem` class.

### C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace RangeSlider
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            // Rotator Item as Label
            SfRotator rotator = new SfRotator();
            SfRotatorItem itemLabel = new SfRotatorItem();
            List<SfRotatorItem> rotatorItem = new List<SfRotatorItem>();
            Label label = new Label();
            label.Text = "RotatorItem as Label";
            label.BackgroundColor = Color.Aqua;
            label.FontSize = 20;
            label.VerticalTextAlignment = TextAlignment.Center;
            label.HorizontalTextAlignment = TextAlignment.Center;
            label.VerticalOptions = LayoutOptions.Center;
            itemLabel.ItemContent = label;
            rotatorItem.Add(itemLabel);
            // Rotator Item as Image
            SfRotatorItem itemImage = new SfRotatorItem();
            Image image = new Image();
            image.Source = ImageSource.FromFile("movie1.png");
            image.Aspect = Aspect.AspectFit;
            image.VerticalOptions = LayoutOptions.Center;
            image.HeightRequest = 400;
            image.WidthRequest = 400;
            itemImage.ItemContent = image;
            rotatorItem.Add(itemImage);
            rotator.ItemsSource = rotatorItem;
            this.Content = rotator;
        }
    }
}

```



And also rotator provides a support to display only the Image data with `Image` property in `SfRotatorItem` class.

### C#

```
public partial class RotatorControlPage : ContentPage
{
    public RotatorControlPage()
    {
        InitializeComponent();
        SfRotator rotator = new SfRotator();
        StackLayout stackLayout = new StackLayout();
        public Rotator()
        {
            InitializeComponent();
            stackLayout.HeightRequest = 300;
            List<SfRotatorItem> collectionOfItems = new List<SfRotatorItem>();
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie1.png" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie2.png" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie3.png" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie4.png" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie5.png" });
            rotator.DataSource = collectionOfItems;
            stackLayout.Children.Add(rotator);
            this.Content = stackLayout;
        }
    }
}
```



Similarly every item can be created and customized in case of different Rotator item view is needed.

### DataTemplateSelector

SfRotator supports DataTemplateSelector which you can choose a DataTemplate based on the data object.

### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<DataTemplate x:Key="DefaultTemplate">
<Grid>
<Image Source="{Binding Image}" HorizontalOptions="Center"
VerticalOptions="Center"/>
</Grid>
</DataTemplate>
<DataTemplate x:Key="SpecificTemplate">
<Grid>
<Label Text="Not Available" FontSize="Large" HorizontalOptions="Center"
VerticalOptions="Center" FontAttributes="Italic" FontFamily="Calbiri"/>
<Image Source="{Binding Image}" Opacity="0.5" HorizontalOptions="Center"
VerticalOptions="Center"/>
</Grid>
</DataTemplate>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<Grid >
<syncfusion:SfRotator x:Name="sfRotator"
ItemsSource="{Binding ImageCollection}" >
<syncfusion:SfRotator.ItemTemplate>
<local:DataTemplateViewModel DefaultTemplate="{StaticResource
DefaultTemplate}" SpecificTemplate="{StaticResource SpecificTemplate}"/>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</Grid>
</ContentPage.Content>
```

</ContentPage>

**C#**

```
public partial class MainPage : ContentPage
{
    DataTemplate defaultTemplate;
    DataTemplate specifictempalte;
    public MainPage()
    {
        InitializeComponent();
        this.BindingContext = new ViewModel();
        SfRotator rotator = new SfRotator();
        defaultTemplate = new DataTemplate(() =>
        {
            Grid grid = new Grid();
            Image image = new Image();
            image.SetBinding(Image.SourceProperty, "Image");
            grid.Children.Add(image);
            return grid;
        });
        specifictempalte = new DataTemplate(() =>
        {
            Grid grid = new Grid();
            Image image = new Image();
            Label label = new Label();
            image.SetBinding(Image.SourceProperty, "Image");
            image.Opacity = 0.5;
            label.Text = "Not Available";
            label.FontSize = 50;
            label.HorizontalOptions = LayoutOptions.Center;
            label.VerticalOptions = LayoutOptions.Center;
            grid.Children.Add(image);
            grid.Children.Add(label);
            return grid;
        });
        var ImageCollection = new List<Model> {
            new Model ("movie1.png"),
            new Model ("movie2.png"),
            new Model ("movie3.png"),
            new Model ("movie4.png"),
            new Model ("movie5.png")
        };
        rotator.NavigationDirection = NavigationDirection.Horizontal;
        rotator.NavigationStripMode = NavigationStripMode.Thumbnail;
        rotator.BackgroundColor = Color.White;
        rotator.ItemsSource = ImageCollection;
        rotator.ItemTemplate = new DataTemplateViewModel { DefaultTemplate =
            defaultTemplate, SpecificTemplate = specifictempalte};
        this.Content = rotator;
    }
}
```

### OnSelectTemplate

The OnSelectTemplate is a overridden method to return a particular DataTemplate, which shown in the following code:

#### C#

```
public class DataTemplateViewModel : DataTemplateSelector
{
    private DataTemplate defaulttemplate;
    public DataTemplate DefaultTemplate
    {
        get { return defaulttemplate; }
        set { defaulttemplate = value; }
    }
    private DataTemplate specifictemplate;
    public DataTemplate SpecificTemplate
    {
        get { return specifictemplate; }
        set { specifictemplate = value; }
    }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject container)
    {
        var message = item as Model;
        if (message == null)
            return null;
        return message.Image.ToString() == "movie2.png" ? SpecificTemplate :
            DefaultTemplate;
    }
}
```

The following screenshot illustrates the output of above code.



We have attached sample for reference. You can download the sample from the following link.

Sample link: [DataTemplateSelectorSample](#)

### Adding Looping and Delays

Looping and delay can be enabled in SfRotator control and also we can customize the Text and Navigation direction.



### Toggle AutoPlay

The `EnableAutoPlay` property specifies whether the items should navigate automatically based on `NavigationDelay` property, when the property value is set to true.

**Note:** By default, the property value is set to false.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
NavigationDirection="Horizontal"
NavigationStripMode="Thumbnail"
BackgroundColor="#ececec"
EnableAutoPlay="true"
NavigationStripPosition="Bottom">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class Rotator : ContentPage
{
public Rotator()
{
InitializeComponent();
SfRotator rotator = new SfRotator();
var ImageCollection = new List<RotatorModel> {
new RotatorModel ("movie1.png"),
```

```

new RotatorModel ("movie2.png"),
new RotatorModel ("movie3.png"),
new RotatorModel ("movie4.png"),
new RotatorModel ("movie5.png")
};
var itemTemplate = new DataTemplate(() =>
{
var grid = new Grid();
var nameLabel = new Image();
nameLabel.SetBinding(Image.SourceProperty, "Image");
grid.Children.Add(nameLabel);
return grid;
});
rotator.ItemTemplate = itemTemplate;
rotator.EnableAutoPlay = true;
rotator.ItemsSource = ImageCollection;
this.Content = rotator;
}
}
public class RotatorModel
{
public RotatorModel(string imageString)
{
Image = imageString;
}
private String _image;
public String Image
{
get { return _image; }
set { _image = value; }
}
}
}

```

### Setting Navigation Delay

The `NavigationDelay` property specifies the delay duration while switching to next navigation item, when `EnableAutoPlay` property is enabled.

**Note:** The property value should be in milliseconds.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"

```

```

NavigationDirection="Horizontal"
NavigationStripMode="Thumbnail"
BackgroundColor="#ececec"
EnableAutoPlay="true"
NavigationStripPosition="Bottom">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class Rotator : ContentPage
{
public Rotator()
{
InitializeComponent ();
SfRotator rotator = new SfRotator();
var ImageCollection = new List<RotatorModel> {
new RotatorModel ("movie1.png"),
new RotatorModel ("movie2.png"),
new RotatorModel ("movie3.png"),
new RotatorModel ("movie4.png"),
new RotatorModel ("movie5.png")
};
var itemTemplate = new DataTemplate(() =>
{
var grid = new Grid();
var nameLabel = new Image();
nameLabel.SetBinding(Image.SourceProperty, "Image");
grid.Children.Add(nameLabel);
return grid;
});
rotator.ItemTemplate = itemTemplate;
rotator.EnableAutoPlay = true;
rotator.NavigationDelay = 2000;
rotator.ItemsSource = ImageCollection;
this.Content = rotator;
}
}
public class RotatorModel

```

```
{
public RotatorModel(string imageString)
{
Image = imageString;
}
private String _image;
public String Image
{
get { return _image; }
set { _image = value; }
}
}
}
```

### Looping Items

The `EnableLooping` property specifies whether the items should navigate to first item once it reaches the last item and vice-versa.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
NavigationDirection="Horizontal"
NavigationStripMode="Thumbnail"
BackgroundColor="#ececec"
EnableAutoPlay="true"
EnableLooping="true"
NavigationStripPosition="Bottom">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            var ImageCollection = new List<RotatorModel> {
                new RotatorModel ("movie1.png"),
                new RotatorModel ("movie2.png"),
                new RotatorModel ("movie3.png"),
                new RotatorModel ("movie4.png"),
                new RotatorModel ("movie5.png")
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            rotator.ItemTemplate = itemTemplate;
            rotator.EnableAutoPlay = true;
            rotator.NavigationDelay = 2000;
            rotator.EnableLooping = true;
            rotator.ItemsSource = ImageCollection;
            this.Content = rotator;
        }
    }
    public class RotatorModel
    {
        public RotatorModel(string imageString)
        {
            Image = imageString;
        }
        private String _image;
        public String Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }
}

```

### Enable swiping

To restrict the user interaction, the `EnableSwiping` property of `SfRotator` can be set to `false`.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
ItemsSource="{Binding ImageCollection}"
BackgroundColor="#ececec"
EnableSwiping="false">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class Rotator : ContentPage
{
public Rotator()
{
InitializeComponent ();
SfRotator rotator = new SfRotator();
var ImageCollection = new List<RotatorModel> {
new RotatorModel ("movie1.png"),
new RotatorModel ("movie2.png"),
new RotatorModel ("movie3.png"),
new RotatorModel ("movie4.png"),
new RotatorModel ("movie5.png")
};
var itemTemplate = new DataTemplate(() =>
{
var grid = new Grid();
var nameLabel = new Image();
nameLabel.SetBinding(Image.SourceProperty, "Image");
grid.Children.Add(nameLabel);
```

```

return grid;
});
rotator.ItemTemplate = itemTemplate;
rotator.EnableSwiping = false;
rotator.ItemsSource = ImageCollection;
this.Content = rotator;
}
}
public class RotatorModel
{
public RotatorModel(string imageString)
{
Image = imageString;
}
private String _image;
public String Image
{
get { return _image; }
set { _image = value; }
}
}
}
}

```

## Navigation Modes

The `NavigationStripMode` property specifies the appearance of navigation bar items. The image data can be selected either by Thumbnail or by Dots navigation modes.

- **Thumbnail** - The slider items will be loaded in thumbnail view additionally. When a thumbnail item is clicked, the slider will switch to the corresponding image data.

## XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RangeSlider"
xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
NavigationDirection="Horizontal"
NavigationStripMode="Thumbnail"
BackgroundColor="#ececec">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>

```

```

</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            var ImageCollection = new List<RotatorModel> {
                new RotatorModel ("movie1.png"),
                new RotatorModel ("movie2.png"),
                new RotatorModel ("movie3.png"),
                new RotatorModel ("movie4.png"),
                new RotatorModel ("movie5.png")
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            rotator.ItemTemplate = itemTemplate;
            rotator.NavigationStripMode = NavigationStripMode.Thumbnail;
            rotator.ItemsSource = ImageCollection;
            this.Content = rotator;
        }
    }
    public class RotatorModel
    {
        public RotatorModel(string imageString)
        {
            Image = imageString;
        }
        private String _image;
        public String Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }
}

```



```
}
}
```



- **Dots** - The slider items will be loaded in dots view additionally. When a dots item is clicked, the slider will switch to the corresponding image data.

## XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
NavigationDirection="Horizontal"
NavigationStripMode="Dots"
BackgroundColor="#ececec">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
```

```

</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            var ImageCollection = new List<RotatorModel> {
                new RotatorModel ("movie1.png"),
                new RotatorModel ("movie2.png"),
                new RotatorModel ("movie3.png"),
                new RotatorModel ("movie4.png"),
                new RotatorModel ("movie5.png")
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            rotator.ItemTemplate = itemTemplate;
            rotator.ItemsSource = ImageCollection;
            this.Content = rotator;
        }
    }
    public class RotatorModel
    {
        public RotatorModel(string imageString)
        {
            Image = imageString;
        }
        private String _image;
        public String Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }
}

```

```
}
}
```



### Items / Dot Strip Positions

The `NavigationStripPosition` specifies the placement position of the navigation bar items such as thumbnail or dots relative to the slider area.

There are four available positions,

- **Bottom** - Sets the position of the navigation bar items to the bottom.
- **Left** - Sets the position of the navigation bar items to the left.
- **Top** - Sets the position of the navigation bar items to the top.
- **Right** - Sets the position of the navigation bar items to the right.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
```

```

SelectedIndex="2"
NavigationDirection="Horizontal"
NavigationStripMode="Dots"
BackgroundColor="#ececec"
NavigationStripPosition="Bottom">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class Rotator : ContentPage
{
public Rotator()
{
InitializeComponent ();
SfRotator rotator = new SfRotator();
var ImageCollection = new List<RotatorModel> {
new RotatorModel ("movie1.png"),
new RotatorModel ("movie2.png"),
new RotatorModel ("movie3.png"),
new RotatorModel ("movie4.png"),
new RotatorModel ("movie5.png")
};
var itemTemplate = new DataTemplate(() =>
{
var grid = new Grid();
var nameLabel = new Image();
nameLabel.SetBinding(Image.SourceProperty, "Image");
grid.Children.Add(nameLabel);
return grid;
});
rotator.ItemTemplate = itemTemplate;
rotator.NavigationStripMode = NavigationStripMode.Dots;
rotator.NavigationStripPosition = NavigationStripPosition.Bottom;
rotator.ItemsSource = ImageCollection;
this.Content = rotator;
}
}
public class RotatorModel

```

```
{
    public RotatorModel(string imageString)
    {
        Image = imageString;
    }
    private String _image;
    public String Image
    {
        get { return _image; }
        set { _image = value; }
    }
}
}
```



## Sliding Direction

The `NavigationDirection` property specifies the direction in which items should be navigated in SfRotator control.

### Horizontal

Rotator Items can be navigated in horizontal direction.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:RangeSlider"
    xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
    x:Class="RangeSlider.Rotator">
    <ContentPage.BindingContext>
```

```

<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
NavigationDirection="Horizontal"
NavigationStripMode="Thumbnail"
BackgroundColor="#ecec"
EnableAutoPlay="true"
EnableLooping="true"
NavigationStripPosition="Bottom">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class Rotator : ContentPage
{
public Rotator()
{
InitializeComponent ();
SfRotator rotator = new SfRotator();
var ImageCollection = new List<RotatorModel> {
new RotatorModel ("movie1.png"),
new RotatorModel ("movie2.png"),
new RotatorModel ("movie3.png"),
new RotatorModel ("movie4.png"),
new RotatorModel ("movie5.png")
};
var itemTemplate = new DataTemplate(() =>
{
var grid = new Grid();
var nameLabel = new Image();
nameLabel.SetBinding(Image.SourceProperty, "Image");
grid.Children.Add(nameLabel);
return grid;
});
}
}

```

```

rotator.ItemTemplate = itemTemplate;
rotator.NavigationStripMode = NavigationStripMode.Thumbnail;
rotator.NavigationDirection = NavigationDirection.Horizontal;
rotator.ItemsSource = ImageCollection;
this.Content = rotator;
}
}
public class RotatorModel
{
    public RotatorModel(string imageString)
    {
        Image = imageString;
    }
    private String _image;
    public String Image
    {
        get { return _image; }
        set { _image = value; }
    }
}
}

```

## Vertical

Rotator Items can be navigated in vertical direction.

## XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:RangeSlider"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="RangeSlider.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
NavigationDelay="2000"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
NavigationDirection="Vertical"
NavigationStripMode="Thumbnail"
BackgroundColor="#ececec"
EnableAutoPlay="true"
EnableLooping="true"
NavigationStripPosition="Bottom">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

**C#**

```
using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            var ImageCollection = new List<RotatorModel> {
                new RotatorModel ("movie1.png"),
                new RotatorModel ("movie2.png"),
                new RotatorModel ("movie3.png"),
                new RotatorModel ("movie4.png"),
                new RotatorModel ("movie5.png")
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            rotator.ItemTemplate = itemTemplate;
            rotator.NavigationStripMode = NavigationStripMode.Thumbnail;
            rotator.NavigationDirection = NavigationDirection.Vertical;
            rotator.ItemsSource = ImageCollection;
            this.Content = rotator;
        }
    }

    public class RotatorModel
    {
        public RotatorModel(string imageString)
        {
            Image = imageString;
        }
        private String _image;
        public String Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }
}
```



## Header Visibility

The `IsTextVisible` property can be used to enable the text area visibility in bottom area of SfRotator for providing additional information of items. `IsTextVisible` property is used to change the visibility of the Text panel that is displayed when SfRotatorItem collection is set and will have no effect when setting Item template.

**Note:** By default, the property value is false.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
BackgroundColor="#ececec"
IsTextVisible="true">
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>
```

### C#

```
using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        SfRotator rotator = new SfRotator();
        StackLayout stackLayout = new StackLayout();
        public Rotator()
        {
            InitializeComponent();
            stackLayout.HeightRequest = 300;
            List<SfRotatorItem> collectionOfItems = new List<SfRotatorItem>();
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie1.png", ItemText =
            "Agile Software" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie2.png", ItemText =
            "Delphi Succinctly" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie3.png", ItemText =
            "NancyFX Succinctly" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie4.png", ItemText =
            "Roslyn Succinctly" });
            collectionOfItems.Add(new SfRotatorItem() { Image = "movie5.png", ItemText =
            "Spark Succinctly" });
        }
    }
}
```

```

rotator.DataSource = collectionOfItems;
rotator.IsTextVisible = true;
rotator.DotPlacement = DotPlacement.Outside;
stackLayout.Children.Add(rotator);
this.Content = stackLayout;
}
}
}

```



## Placement Modes

By default, the rotator control displays the dots of each rotator item. It can be changed to any of the following types:

- Default
- None
- Outside

## DotsPlacement

The Display Type of Rotator can be modified using the DotsPlacement Mode. The “None” type can be used when the dots are not needed for the rotator control.

## XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"

```

```

ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
BackgroundColor="#ececec"
NavigationStripPosition="Bottom"
DotPlacement="None" >
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class Rotator : ContentPage
{
public Rotator()
{
InitializeComponent ();
SfRotator rotator = new SfRotator();
var ImageCollection = new List<RotatorModel> {
new RotatorModel ("movie1.png"),
new RotatorModel ("movie2.png"),
new RotatorModel ("movie3.png"),
new RotatorModel ("movie4.png"),
new RotatorModel ("movie5.png")
};
var itemTemplate = new DataTemplate(() =>
{
var grid = new Grid();
var nameLabel = new Image();
nameLabel.SetBinding(Image.SourceProperty, "Image");
grid.Children.Add(nameLabel);
return grid;
});
rotator.ItemTemplate = itemTemplate;
rotator.DotPlacement = DotPlacement.None;
rotator.ItemsSource = ImageCollection;
this.Content = rotator;
}
}
public class RotatorModel
{

```

```

public RotatorModel(string imageString)
{
    Image = imageString;
}
private String _image;
public String Image
{
    get { return _image; }
    set { _image = value; }
}
}
}

```



### Customization

The Rotator control supports to customize the dots border color, selected dots color and unselected dots color.

### DotsBorder Color

The [DotsBorderColor] property is used to customize the color of dots border in SfRotator.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
BackgroundColor="#ececec"
NavigationStripPosition="Bottom"
DotsBorderColor="Aqua" >
<syncfusion:SfRotator.ItemTemplate>

```

```

<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class Rotator : ContentPage
{
public Rotator()
{
InitializeComponent ();
SfRotator rotator = new SfRotator();
var ImageCollection = new List<RotatorModel> {
new RotatorModel ("movie1.png"),
new RotatorModel ("movie2.png"),
new RotatorModel ("movie3.png"),
new RotatorModel ("movie4.png"),
new RotatorModel ("movie5.png")
};
var itemTemplate = new DataTemplate(() =>
{
var grid = new Grid();
var nameLabel = new Image();
nameLabel.SetBinding(Image.SourceProperty, "Image");
grid.Children.Add(nameLabel);
return grid;
});
rotator.ItemTemplate = itemTemplate;
rotator.DotsBorderColor = Color.Aqua;
rotator.ItemsSource = ImageCollection;
this.Content = rotator;
}
}
public class RotatorModel
{
public RotatorModel(string imageString)
{
Image = imageString;
}
private String _image;
public String Image

```

```
{
    get { return _image; }
    set { _image = value; }
}
}
```



### Selected Dot Color

The [SelectedDotColor] property is used to customize the color of selected dots in SfRotator.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
    <ContentPage.BindingContext>
        <local:RotatorViewModel/>
    </ContentPage.BindingContext>
    <ContentPage.Content>
        <syncfusion:SfRotator x:Name="rotator"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
BackgroundColor="#ecec"
NavigationStripPosition="Bottom"
DotsBorderColor="Aqua"
SelectedDotColor="Blue">
            <syncfusion:SfRotator.ItemTemplate>
                <DataTemplate>
                    <Image Source="{Binding Image}"/>
                </DataTemplate>
            </syncfusion:SfRotator.ItemTemplate>
        </syncfusion:SfRotator>
    </ContentPage.Content>
```

&lt;/ContentPage&gt;

**C#**

```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            var ImageCollection = new List<RotatorModel> {
                new RotatorModel ("movie1.png"),
                new RotatorModel ("movie2.png"),
                new RotatorModel ("movie3.png"),
                new RotatorModel ("movie4.png"),
                new RotatorModel ("movie5.png")
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            rotator.ItemTemplate = itemTemplate;
            rotator.DotsBorderColor = Color.Aqua;
            rotator.SelectedDotColor = Color.Blue;
            rotator.ItemsSource = ImageCollection;
            this.Content = rotator;
        }
    }
    public class RotatorModel
    {
        public RotatorModel(string imageString)
        {
            Image = imageString;
        }
        private String _image;
        public String Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }
}

```

```
}

```



### Unselected Dot Color

The `[UnselectedDotColor]` property is used to customize the color of unselected dots in SfRotator.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.Rotator">
<ContentPage.BindingContext>
<local:RotatorViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfRotator x:Name="rotator"
ItemsSource="{Binding ImageCollection}"
SelectedIndex="2"
BackgroundColor="#ecec"
NavigationStripPosition="Bottom"
DotsBorderColor="Aqua"
SelectedDotColor="Blue"
UnselectedDotColor="Gray">
<syncfusion:SfRotator.ItemTemplate>
<DataTemplate>
<Image Source="{Binding Image}"/>
</DataTemplate>
</syncfusion:SfRotator.ItemTemplate>
</syncfusion:SfRotator>
</ContentPage.Content>
</ContentPage>
```

### C#



```

using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator rotator = new SfRotator();
            var ImageCollection = new List<RotatorModel> {
                new RotatorModel ("movie1.png"),
                new RotatorModel ("movie2.png"),
                new RotatorModel ("movie3.png"),
                new RotatorModel ("movie4.png"),
                new RotatorModel ("movie5.png")
            };
            var itemTemplate = new DataTemplate(() =>
            {
                var grid = new Grid();
                var nameLabel = new Image();
                nameLabel.SetBinding(Image.SourceProperty, "Image");
                grid.Children.Add(nameLabel);
                return grid;
            });
            rotator.ItemTemplate = itemTemplate;
            rotator.DotsBorderColor = Color.Aqua;
            rotator.SelectedDotColor = Color.Blue;
            rotator.UnselectedDotColor = Color.Gray;
            rotator.ItemsSource = ImageCollection;
            this.Content = rotator;
        }
    }
    public class RotatorModel
    {
        public RotatorModel(string imageString)
        {
            Image = imageString;
        }
        private String _image;
        public String Image
        {
            get { return _image; }
            set { _image = value; }
        }
    }
}

```



## Loading URL Images

This section describes how to load the online images in SfRotator Control.

### C#

```
using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Rotator
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Rotator : ContentPage
    {
        public Rotator()
        {
            InitializeComponent();
            SfRotator sfRotator = new SfRotator();
            sfRotator.ItemsSource = GetDataSource();
            var imageTemplate = new DataTemplate(() =>
            {
                Image image = new Image();
                image.SetBinding(Image.SourceProperty, "Image");
                return image;
            });
            sfRotator.ItemTemplate = imageTemplate;
            this.Content = sfRotator;
        }
        List<CustomData> GetDataSource()
        {
            List<CustomData> list = new List<CustomData>();
```

```
list.Add(new
CustomData("https://cdn.syncfusion.com/content/images/Images/Camtasia_Succin
ctly.png?v=22022017060923"));
list.Add(new
CustomData("https://cdn.syncfusion.com/content/images/Images/SQL_Queries_Suc
cinctly.png?v=04022017014551"));
list.Add(new
CustomData("https://upload.wikimedia.org/wikipedia/commons/0/0c/Cow_female_b
lack_white.jpg"));
list.Add(new
CustomData("https://cdn.syncfusion.com/content/images/Images/Keystonejs_Succ
inctly.png?v=22022017060923"));
list.Add(new
CustomData("https://cdn.syncfusion.com/content/images/Images/sql_server_for_
c_sharp_developers_succinctly_cover_img.png?v=22022017060923"));
list.Add(new
CustomData("https://cdn.syncfusion.com/content/images/Images/sql_server_for_
c_sharp_developers_succinctly_cover_img.png?v=22022017060923"));
return list;
}
}
}
```

## C#

```
// Custom Data
using System;
using Xamarin.Forms;
namespace Rotator
{
    public class CustomData : ContentPage
    {
        public CustomData(string image)
        {
            Image = image;
        }
        public string Image
        {
            get;
            set;
        }
        public CustomData()
        {
        }
    }
}
```



## Events

### Selection Changed

The SelectionChanged event is used to notify when the selection is changed by swiping or dynamically setting the SelectedIndex property of SfRotator.

### ItemTapped

The ItemTapped event will be triggered whenever tapping the item.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Rotator"
xmlns:rotator="clr-
namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"
x:Class="Rotator.MainPage">
<rotator:SfRotator.BindingContext>
<local:RotatorViewModel/>
</rotator:SfRotator.BindingContext>
<ContentPage.Content>
<Grid HorizontalOptions="FillAndExpand" VerticalOptions="Fill">
<rotator:SfRotator x:Name="rotator"
ItemTapped="Rotator_ItemTapped"
SelectedIndexChanged="Rotator_SelectedIndexChanged"
ItemsSource="{Binding ImageCollection}"
VerticalOptions="Start">
<rotator:SfRotator.ItemTemplate>
<DataTemplate>
<StackLayout>
<Image Source="{Binding Image}" />
```

```

</StackLayout>
</DataTemplate>
</rotator:SfRotator.ItemTemplate>
</rotator:SfRotator>
</Grid>
</ContentPage.Content>
</ContentPage>

```

## C#

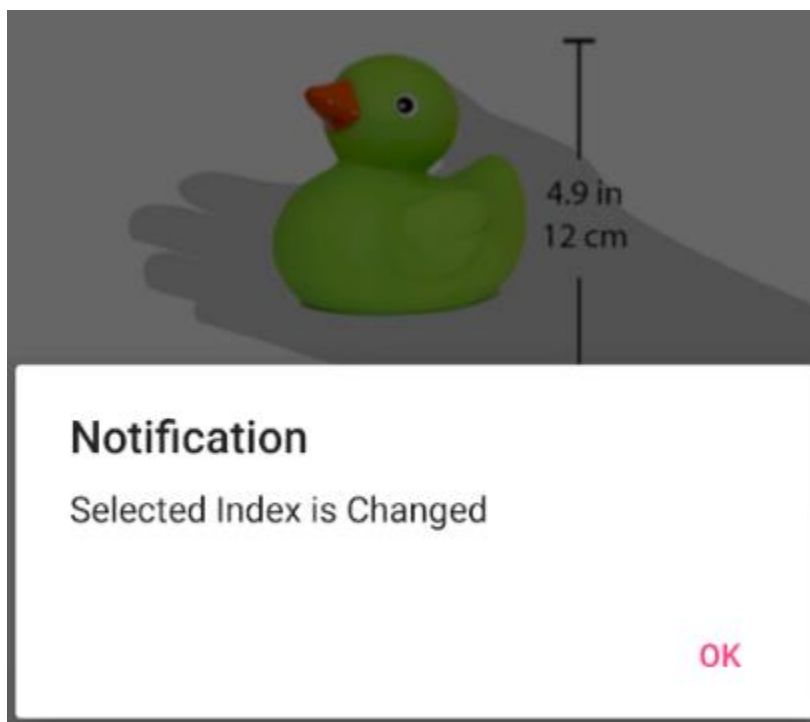
```

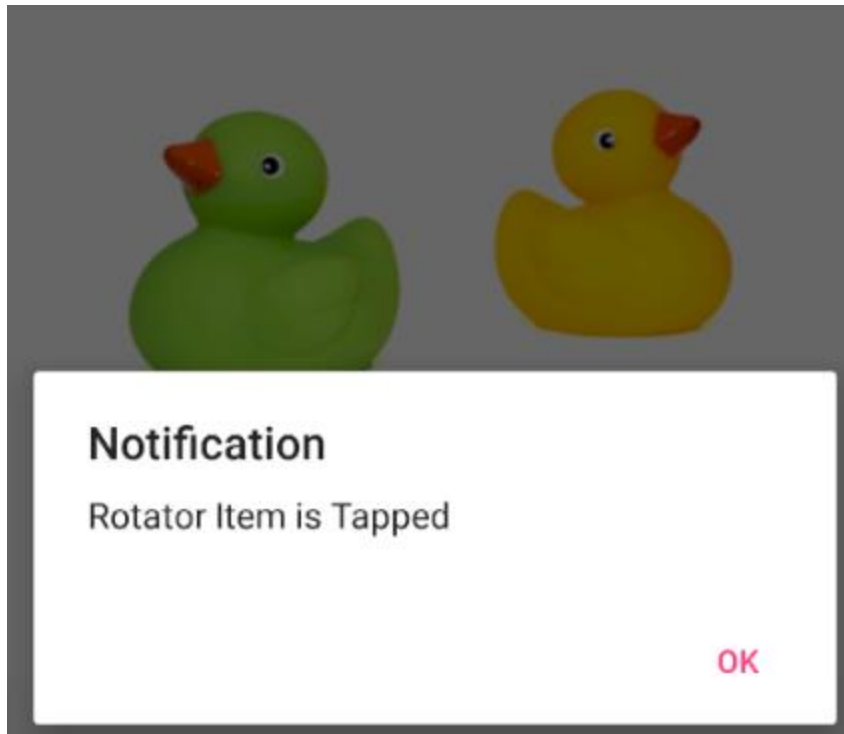
using Syncfusion.SfRotator.XForms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace Rotator
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void Rotator_ItemTapped(object sender, EventArgs e)
        {
            DisplayAlert("Notification", "Rotator Item is Tapped", "Ok");
        }
        private void Rotator_SelectedIndexChanged(object sender,
            SelectedIndexChangedEventArgs e)
        {
            DisplayAlert("Notification", "Selected Index is Changed", "Ok");
        }
        public class RotatorModel
        {
            public RotatorModel(string imageString)
            {
                Image = imageString;
            }
            private String _image;
            public String Image
            {
                get { return _image; }
                set { _image = value; }
            }
        }
        RotatorViewModel Class:
        using System;
        using System.Collections.Generic;
        using System.Collections.ObjectModel;
        using System.Text;
        namespace Rotator
        {

```

```
public class RotatorViewModel
{
    public RotatorViewModel()
    {
        ImageCollection.Add(new RotatorModel("movie1.png"));
        ImageCollection.Add(new RotatorModel("image1.png"));
        ImageCollection.Add(new RotatorModel("movie3.png"));
        ImageCollection.Add(new RotatorModel("movie4.png"));
        ImageCollection.Add(new RotatorModel("movie5.png"));
    }
    private ObservableCollection<RotatorModel> imageCollection = new
    ObservableCollection<RotatorModel>();
    public ObservableCollection<RotatorModel> ImageCollection
    {
        get { return imageCollection; }
        set { imageCollection = value; }
    }
}
```

The following screenshot illustrates the output of above code.





### AutomationId

The SfRotator control has built-in **AutomationId** for inner elements. The **AutomationId** API allows the automation framework to find and interact with the inner elements of the SfRotator control. To keep unique **AutomationId**, these inner elements' **AutomationIds** are updated based on the control's **AutomationId**.

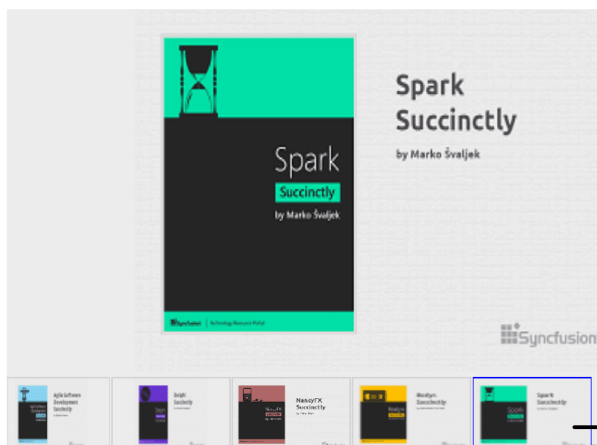
For example, if you set SfRotator's **AutomationId** as "Book Display Rotator" and you want to select the fifth index, then the automation framework will interact with the Thumbnail as "Book Display Rotator Index 5 of 5".

The automation framework will interact only to the Dots and Thumbnail **NavigationStripMode**. You cannot interact with the rotator item when you want to select the index that is not visible in the view.

---

**Note:** You cannot provide **AutomationId** when the rotator item is populated with custom template.

---

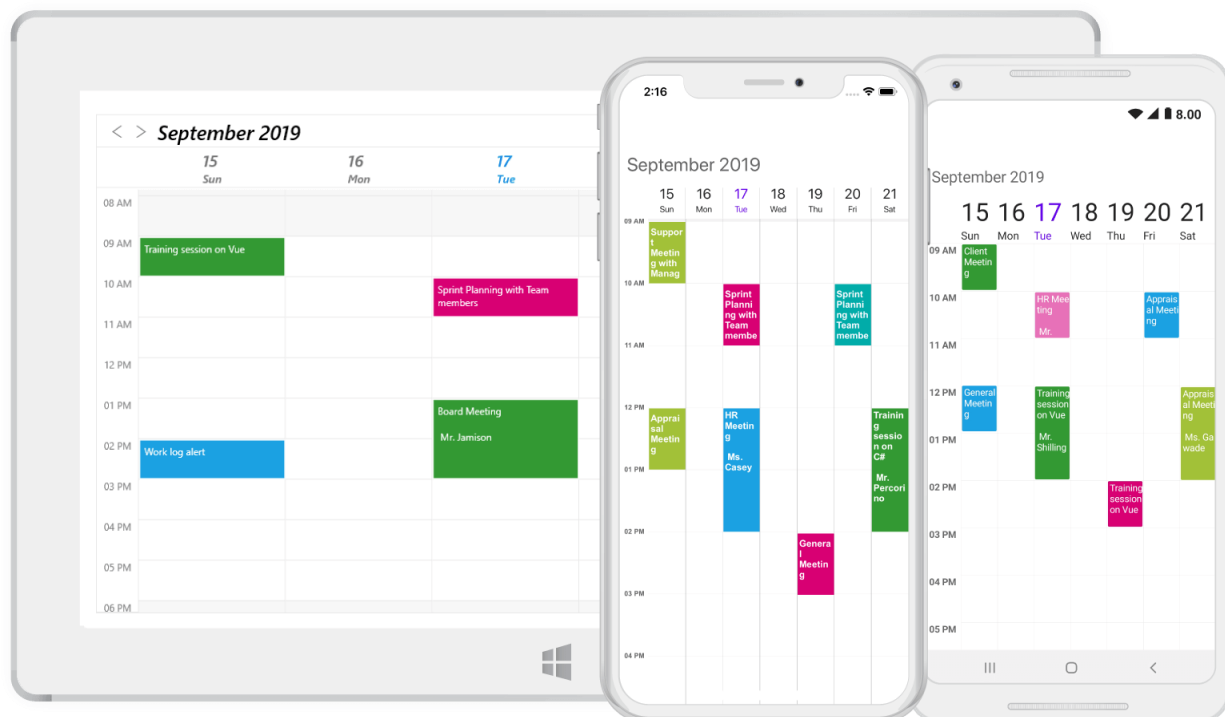


Book Display Rotator Item 5 of 5

## SfSchedule

### Overview

Essential Schedule for Xamarin.Forms provides all the common scheduling functionalities to create and manage appointments as well as exposes a gesture friendly UI to perform all common operations likes selection, navigation, etc. **Essential Schedule** is a perfect solution for developers looking to add advanced, feature rich **Schedule** to their applications.





## Key features

**Built-in Views** — Schedule provides five different types of views such as Day, WorkWeek, Week, Timeline and Month.

**Recurrence Appointment** — Recurring appointments can be created with Daily, weekly, monthly, and yearly recurrence patterns which is supported in iCal standard.

**Customization** — Control has simple APIs allowing for elegant customizations. You can edit the look to match the rest of your application.

**Localization** — The built-in content of the user interface can be changed according to culture that is needed. Also, it has built-in culture support for basic items such as day and month text representations.

## Getting Started

This section provides you an overview for working with SfSchedule for Xamarin.Forms and also provides a walk through to configure SfSchedule control in real time scenario.

### Adding SfSchedule reference

You can add SfSchedule reference using one of the following methods:

#### Method 1: Adding SfSchedule reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfSchedule). To add SfSchedule to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfSchedule](https://www.nuget.org/packages/Syncfusion.Xamarin.SfSchedule), and then install it.

![Adding SfSchedule reference from NuGet](GettingStarted\_images/Adding SfSchedule reference.png)

**Note:** When there is mismatch between the Syncfusion NuGet packages among the projects, `System.IO.FileLoadException` will occur. To overcome this exception, install the same version of the SfSchedule assemblies in all the projects.

#### Method 2: Adding SfSchedule reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfSchedule control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfSchedule assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfSchedule.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfSchedule.Android.dll Syncfusion.SfSchedule.XForms.Android.dll Syncfusion.SfSchedule.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll

iOS	Syncfusion.SfSchedule.iOS.dll Syncfusion.SfSchedule.XForms.iOS.dll Syncfusion.SfSchedule.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfSchedule.XForms.UWP.dll Syncfusion.SfSchedule.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
WPF	Syncfusion.SfSchedule.XForms.WPF.dll Syncfusion.SfSchedule.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.WPF.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

#### *Launching SfSchedule on each platform*

To use SfSchedule inside an application, each platform application must initialize the SfScheduleRenderer. This initialization step varies from platform to platform and it is discussed in the following sections.

#### *Android*

Android application launches SfSchedule without any initialization.

**Note:** If you are adding the references from toolbox, this step is not needed.

#### *iOS*

To launch SfSchedule in iOS, you need to call `SfScheduleRenderer.Init()` in the FinishedLaunching overridden method of the AppDelegate class after the Xamarin.Forms Framework initialization.

#### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    Syncfusion.SfSchedule.XForms.iOS.SfScheduleRenderer.Init();
    LoadApplication(new App());
}
```

#### *Universal Windows Platform (UWP)*

UWP application launches SfSchedule without any initialization.

### ReleaseMode issue in UWP platform

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in **Release Mode**.

The above problem can be resolved by initializing the SfSchedule assemblies in **App.xaml.cs** in UWP project as like in below code snippet.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.SfSchedule.XForms.UWP.SfScheduleRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
}
```

### Windows Presentation Foundation (WPF)

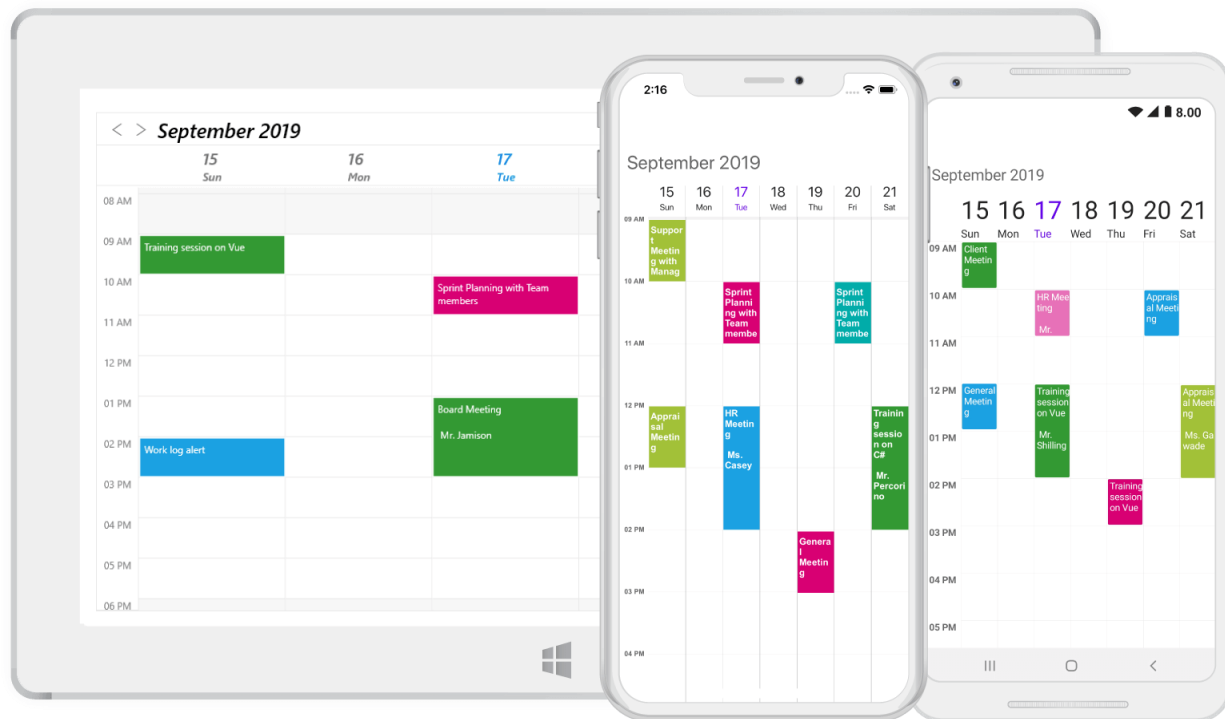
To launch the schedule in WPF SfScheduleRenderer.Init() method in the MainWindow constructor of the MainWindow class after the Xamarin.Forms Framework has been initialized and before the LoadApplication method is called as demonstrated in the following code sample.

#### C#

```
// In App.xaml.cs
public partial class MainWindow : FormsApplicationPage
{
    public MainWindow()
    {
        InitializeComponent();
        Forms.Init();
        Syncfusion.SfSchedule.XForms.WPF.SfScheduleRenderer.Init();
        LoadApplication(new App());
    }
}
```

### Create a simple application with SfSchedule

This section explains how to create a simple application using SfSchedule control. SfSchedule control can be configured entirely in C# code or by using XAML markup. This is how the final output will look like on iOS, Android and Windows Phone devices.



You can download the entire source code of this demo for Xamarin.Forms from here [ScheduleGettingStarted](#)

This section provides a walks through to create **MeetingRoomScheduler** using our Schedule control.

- [Creating a new project](#)
- [Adding SfSchedule to the project](#)
- [Changing Schedule Views](#)
- [Binding data to SfSchedule control](#)

### Creating a new project

Create a new Blank App (Xamarin.Forms) application in Xamarin Studio or Visual Studio.

Add the required assembly references to the .NET Standard and renderer projects as discussed in the [Assembly Configuration](#) section.

Import SfSchedule control namespace **Syncfusion.SfSchedule.XForms**.

### Adding SfSchedule to the project

Set the SfSchedule control as content to the ContentPage.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
  x:Class="GettingStarted.Sample"
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted">
```

```
xmlns:syncfusion="clr-
namespace:Syncfusion.SfSchedule.XForms;assembly=Syncfusion.SfSchedule.XForms
">
<ContentPage.Content>
<syncfusion:SfSchedule x:Name="schedule" />
</ContentPage.Content>
</ContentPage>
```

## C#

```
using Syncfusion.SfSchedule.XForms;
using Xamarin.Forms;
namespace GettingStarted
{
    public class App : Application
    {
        SfSchedule schedule;
        public App()
        {
            InitializeComponent();
            //Creating new instance for SfSchedule
            schedule = new SfSchedule();
            MainPage = new ContentPage { Content = schedule };
        }
    }
}
```

You can download the source code for rendering of schedule for Xamarin.Forms from here [ScheduleProject](#)

## Changing Schedule Views

SfSchedule control provides five different types of views to display dates and it can be assigned to the control by using [ScheduleView](#) property. By default the control is assigned with [DayView](#). Current date will be displayed initially for all the Schedule views.

Schedule control will be rendered with [Sunday](#) as the first day of the week, but you can customize to any day by using [FirstDayOfWeek](#) property of [SfSchedule](#).

## XML

```
<syncfusion:SfSchedule x:Name="schedule" FirstDayOfWeek="3"/>
```

## C#

```
//setting first day of the week
schedule.FirstDayOfWeek = (int) DayOfWeek.Tuesday;
```

September 2019						
Tue	Wed	Thu	Fri	Sat	Sun	Mon
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

#### Day View

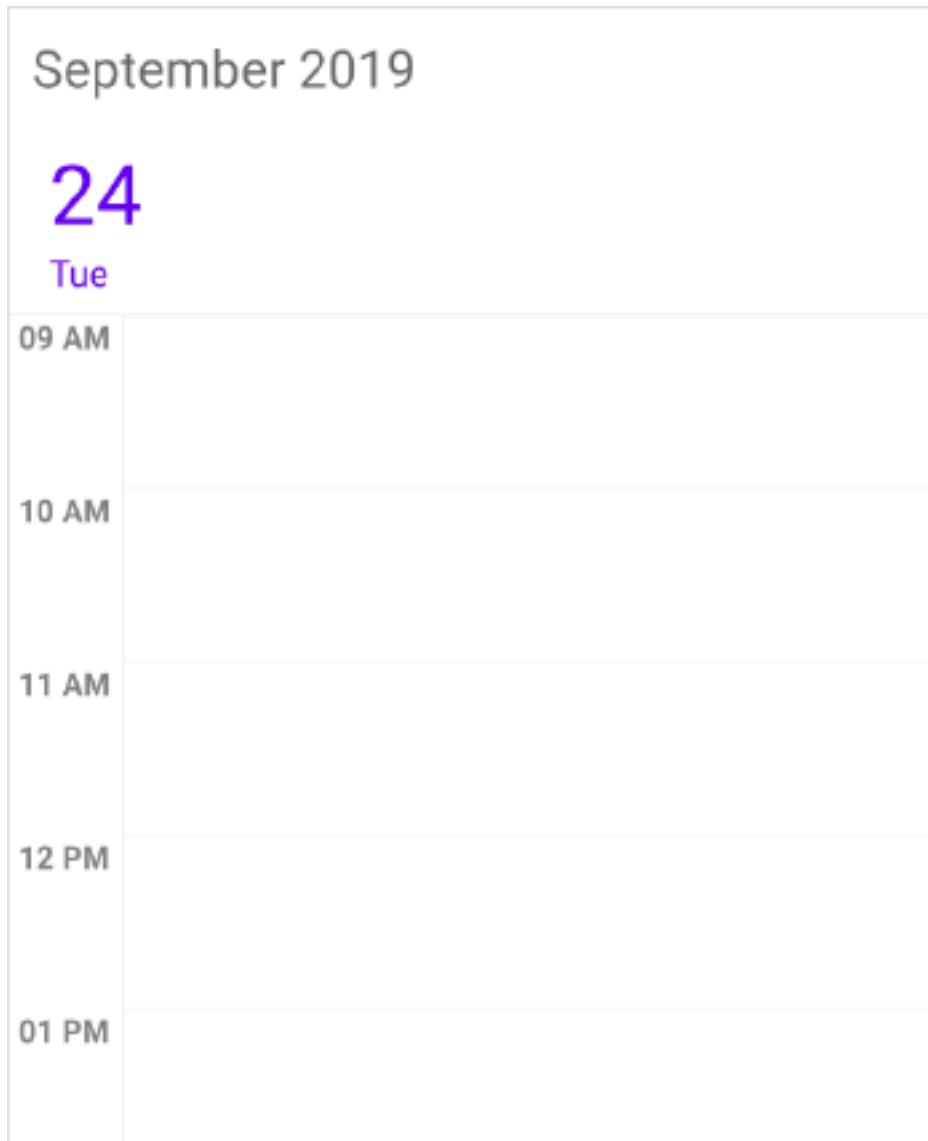
DayView is used to display a single day, current day will be visible by default. Appointments on a specific day will be arranged in respective timeslots based on its duration.

#### XML

```
<syncfusion:SfSchedule x:Name="schedule" ScheduleView="DayView" />
```

#### C#

```
schedule.ScheduleView = ScheduleView.DayView;
```



#### [Week View](#)

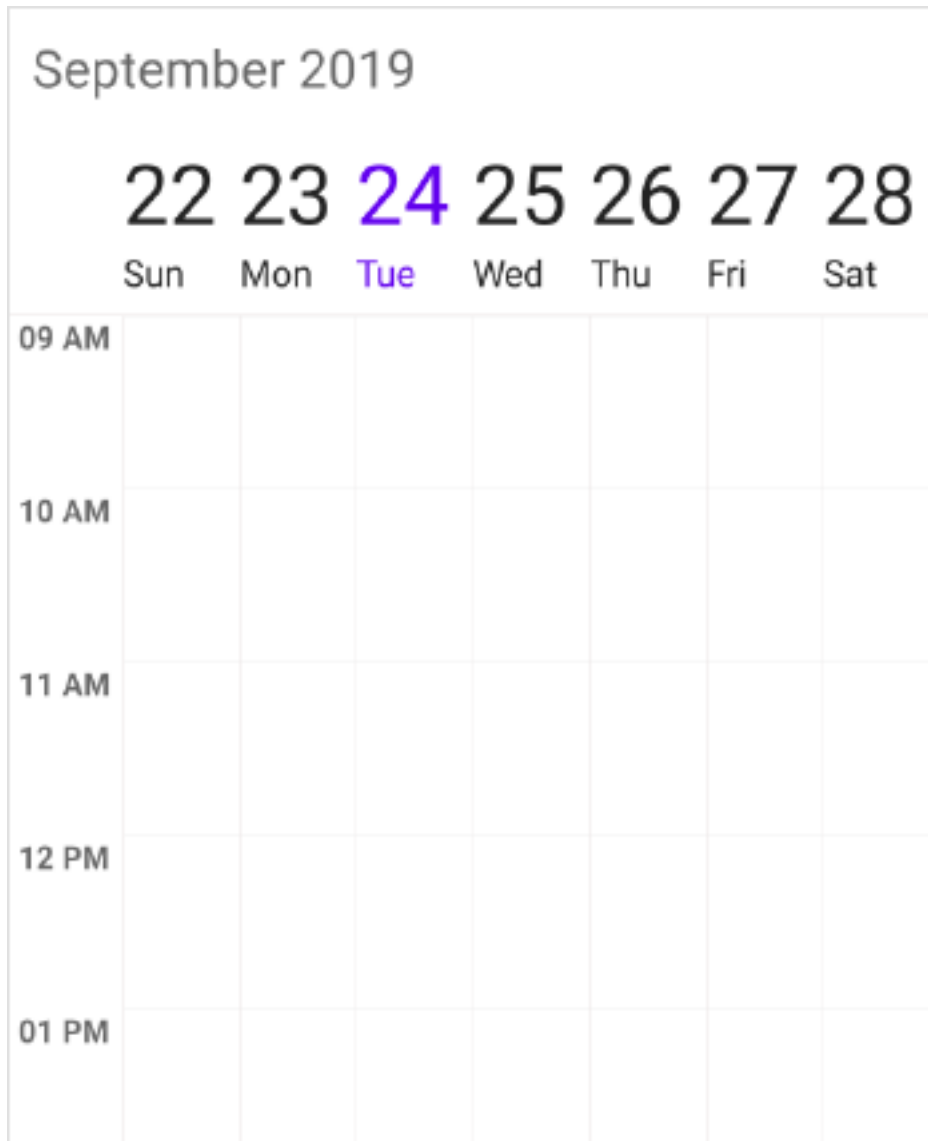
**WeekView** is to view all days of a particular week. Appointments will be arranged based on the dates on the week in respective timeslots.

#### **XML**

```
<syncfusion:SfSchedule x:Name="schedule" ScheduleView="WeekView" />
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.WeekView;
```



#### [Work Week View](#)

**WorkWeekView** is to view only working days of a particular week. By default, Saturday and Sunday are the non-working days. You can be customize it with any days of a Week. Appointments arranged in timeslots based on its duration with respective day of the week.

#### **XML**

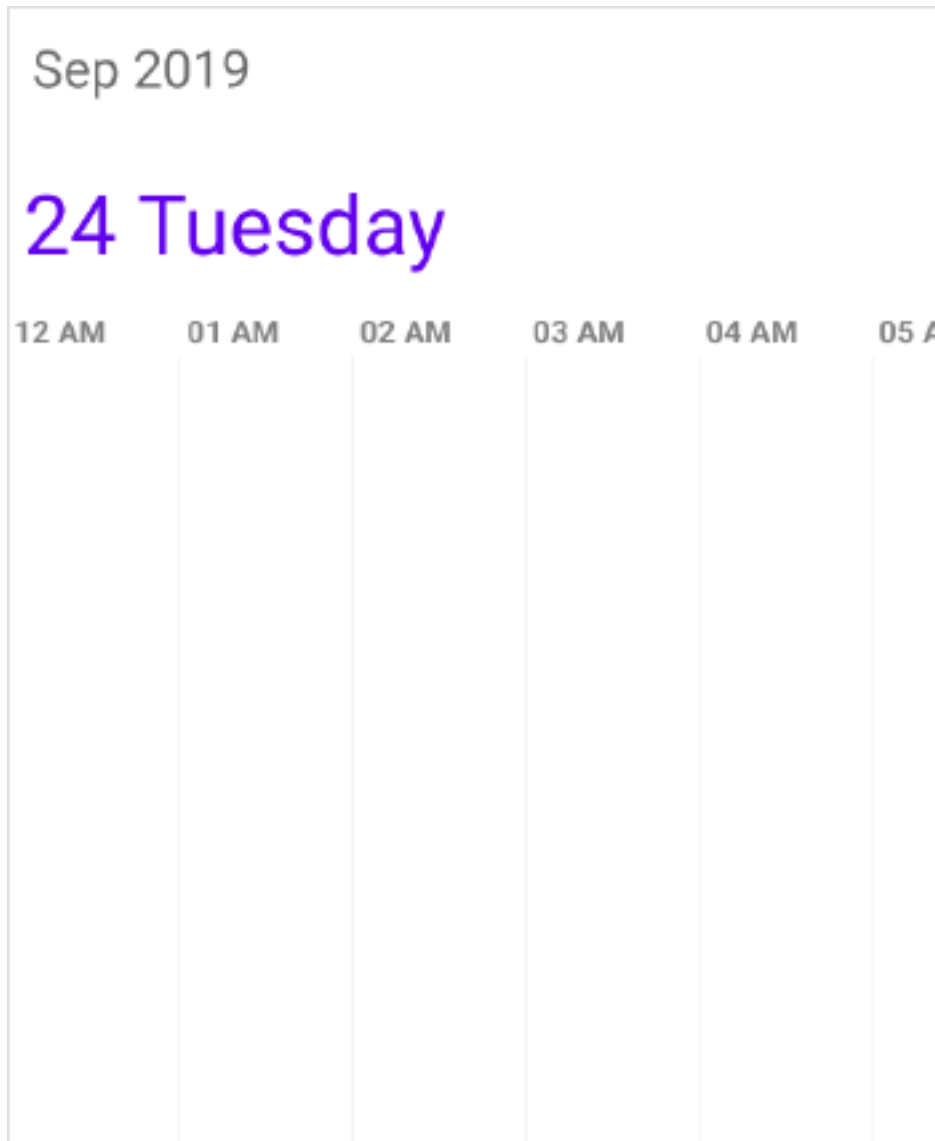
```
<syncfusion:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView" />
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
```







#### *Month View*

**MonthView** in Schedule control is to view entire dates of a particular month. Appointments can be viewed in inline by setting [ShowAppointmentsInline](#) property of **SfSchedule** as true.

#### **XML**

```
<syncfusion:SfSchedule
x:Name="schedule"
ScheduleView="MonthView"
ShowAppointmentsInline="True">
</syncfusion:SfSchedule>
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.MonthView;
//view appointments in inline
schedule.ShowAppointmentsInline = true;
```

September 2019						
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	1	2	3	4	5
6	7	8	9	10	11	12

### Binding data to SfSchedule control

Schedule control has a built-in capability to handle the appointment arrangement internally based on the `ScheduleAppointment` collections. You need to assign the created collection to the [DataSource](#) property of `SfSchedule`.

### Adding Appointments

[ScheduleAppointment](#) is a class, which holds the details about the appointment to be rendered in schedule. It has some basic properties such as [StartTime](#), [EndTime](#), [Subject](#) and some additional information about the appointment can be added using [Color](#), [Notes](#), [Location](#), [IsAllDay](#), [IsRecursive](#) properties.

### 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.DataSource = appointmentCollection;

```

You can download the source code for rendering of schedule appointment using `ScheduleAppointmentCollection` for Xamarin.Forms from

here [ScheduleAppointment](#)

#### *Adding Custom Appointments*

You can also map custom appointments data to our schedule.

---

#### **NOTE**

CustomAppointment class should contain two DateTime fields and a string field as mandatory.

Here steps to render `MeetingRoomScheduler` using SfSchedule control with respective custom data properties created in a class Meeting.

- [Creating custom class for appointments](#)
- [Creating view model](#)
- [Mapping custom class](#)
- [Setting data source for SfSchedule](#)

#### *Creating custom class for appointments*

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.

### Creating view model

You can schedule meetings for a particular day by setting **From** and **To** of **Meeting** class. Also you can change subject and color of appointment using **EventName** and **color** of **Meeting** class. In a separate **ViewModel** class you can describe the collection of custom appointments.

### C#

```
/// <summary>
/// Represents collection of appointments.
/// </summary>
public class ViewModel
{
    public ObservableCollection<Meeting> Meetings { get; set; }
    List<string> eventNameCollection;
    List<Color> 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<Color>();
colorCollection.Add(Color.FromHex("#FF339933"));
colorCollection.Add(Color.FromHex("#FF00ABA9"));
colorCollection.Add(Color.FromHex("#FFE671B8"));
colorCollection.Add(Color.FromHex("#FF1BA1E2"));
colorCollection.Add(Color.FromHex("#FFD80073"));
colorCollection.Add(Color.FromHex("#FFA2C139"));
colorCollection.Add(Color.FromHex("#FFA2C139"));
colorCollection.Add(Color.FromHex("#FFD80073"));
colorCollection.Add(Color.FromHex("#FF339933"));
colorCollection.Add(Color.FromHex("#FFE671B8"));
colorCollection.Add(Color.FromHex("#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 custom class

You can map those properties of `Meeting` class with our schedule control by using [ScheduleAppointmentMapping](#).

#### XML

```
<syncfusion:SfSchedule x:Name="schedule">
  <syncfusion:SfSchedule.AppointmentMapping>
  <syncfusion:ScheduleAppointmentMapping
    ColorMapping="color"
    EndTimeMapping="To"
    StartTimeMapping="From"
    SubjectMapping="EventName"
    IsAllDayMapping="AllDay"/>
  </syncfusion:SfSchedule.AppointmentMapping>
</syncfusion:SfSchedule>
```

#### C#

```
ScheduleAppointmentMapping dataMapping = new ScheduleAppointmentMapping();
dataMapping.ColorMapping = "color";
dataMapping.EndTimeMapping = "To";
dataMapping.StartTimeMapping = "From";
dataMapping.SubjectMapping = "EventName";
dataMapping.IsAllDayMapping = "AllDay";
schedule.AppointmentMapping = dataMapping;
```

### Setting data source for SfSchedule

Create meetings of type `ObservableCollection<Meeting>` and assign those appointments collection `Meetings` to the `DataSource` property of `SfSchedule`.

#### XML

```
<syncfusion:SfSchedule x:Name="schedule"
  DataSource="{Binding Meetings}"
  ScheduleView="WeekView">
  <syncfusion:SfSchedule.BindingContext>
  <local:ViewModel/>
  </syncfusion:SfSchedule.BindingContext>
</syncfusion:SfSchedule>
```

#### C#

```
ViewModel viewModel = new ViewModel();
schedule.DataSource = viewModel.Meetings;
```

You can download the entire source code of this demo for `Xamarin.Forms` from here [ScheduleGettingStarted](#)

## Header

You can customize the header of the Schedule using [HeaderStyle](#) and [HeaderHeight](#) property in schedule.

## Header Height

You can customize the height for the Header in Schedule using [HeaderHeight](#) in schedule.

### XML

```
<syncfusion:SfSchedule x:Name="schedule" HeaderHeight="50" />
```

### C#

```
schedule.HeaderHeight = 50;
```

## Appearance

You can change the header format and style using [HeaderStyle](#) property in schedule.

You can change the background color, font family, font attributes and font size using properties such as [BackgroundColor](#), [FontFamily](#), [FontAttributes](#), [FontSize](#), [TextColor](#), of Header using [HeaderStyle](#) property in schedule.

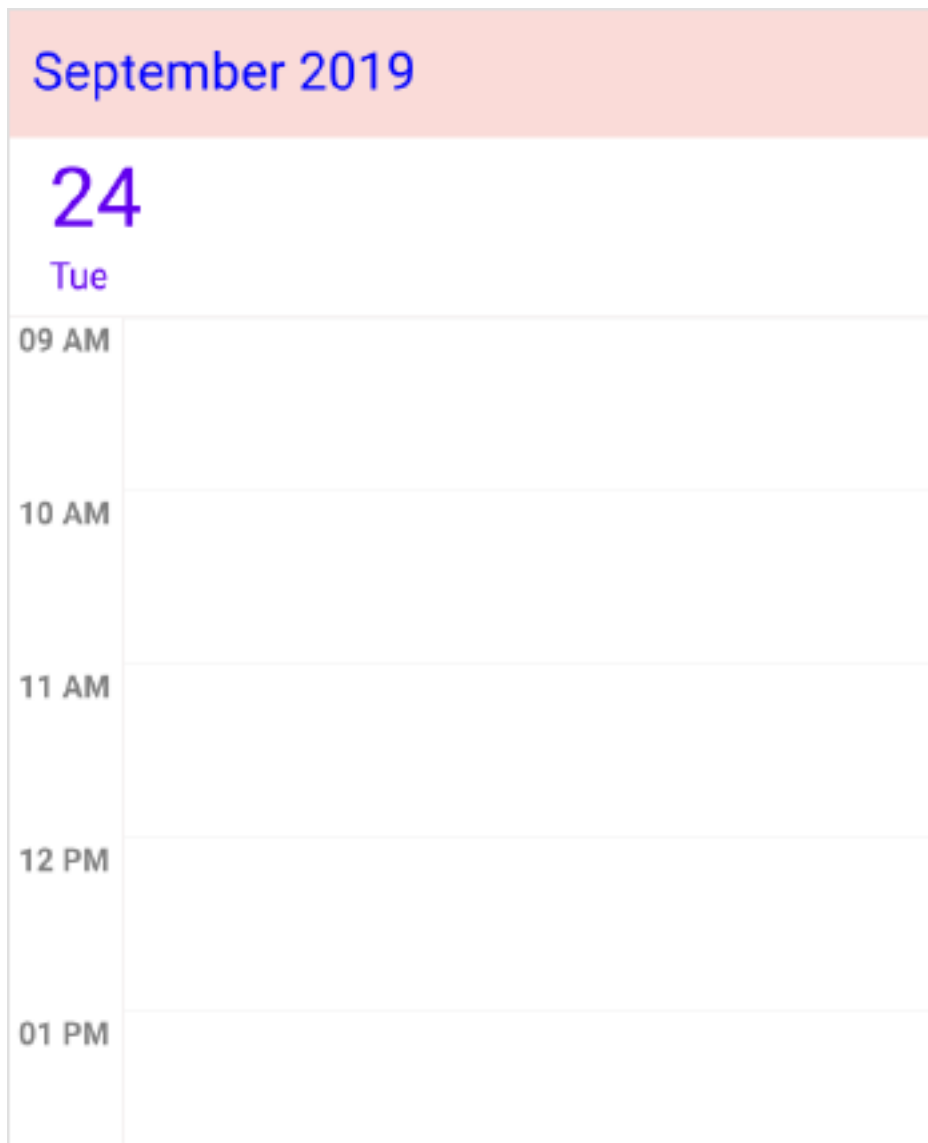
### XML

```
<syncfusion:SfSchedule x:Name="schedule" >  
  <syncfusion:SfSchedule.HeaderStyle>  
    <syncfusion:HeaderStyle  
      BackgroundColor="#FADBD8"  
      TextColor="Blue"  
      FontFamily="Arial"/>  
    </syncfusion:SfSchedule.HeaderStyle>  
  </syncfusion:SfSchedule>
```

### C#

```
HeaderStyle headerStyle = new HeaderStyle();  
headerStyle.BackgroundColor = Color.FromRgb(250, 219, 216);  
headerStyle.FontFamily = "Arial";  
headerStyle.TextColor=Color.Blue;  
schedule.HeaderStyle = headerStyle;
```



**NOTE**

FontAttributes and FontFamily are native to the platform. Custom font and the font which are not available in the specified platform will not be applied.

*Customize Font Appearance*

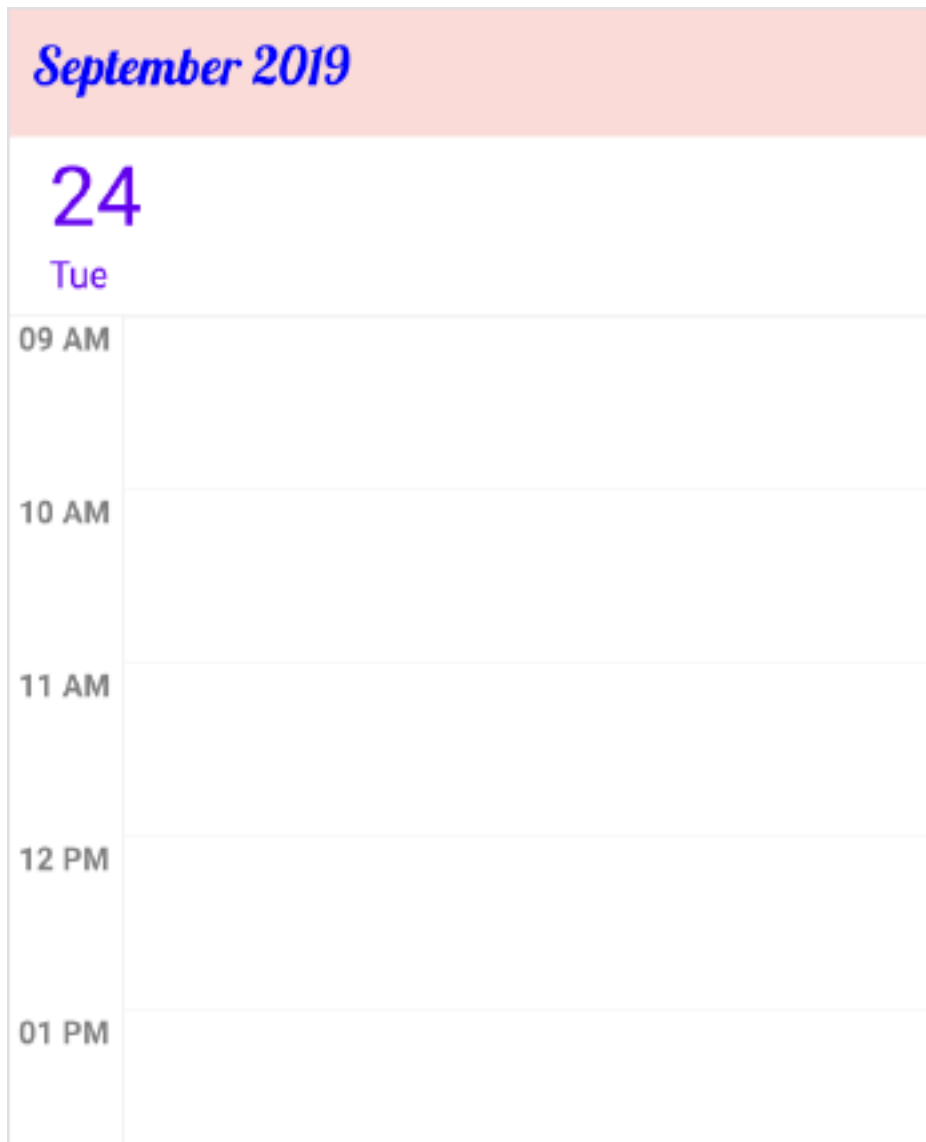
You can change the appearance of Font by setting the [FontFamily](#) property of [HeaderStyle](#) property in Schedule.

**XML**

```
<schedule:HeaderStyle.FontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:HeaderStyle.FontFamily>
```

**C#**

```
headerStyle.FontFamily = Device.OnPlatform("Lobster-Regular", "Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
```



Refer [this](#) to configure the custom fonts in Xamarin.Forms.

### Loading Custom Headers

You can collapse the default header of schedule by setting `HeaderHeight` property of `SfSchedule` as 0. Instead you can use your own custom header for it. While navigating views in schedule, text labels available in the header will be changed based on it visible dates, so while using custom header, respective text value can be obtained from the `VisibleDatesChanged` event of `SfSchedule`.

### C#

```
//triggering the visible dates changed event.  
schedule.VisibleDatesChangedEvent += Schedule_VisibleDatesChangedEvent;  
...
```

```
private void Schedule_VisibleDatesChangedEvent(object sender,
VisibleDatesChangedEventArgs args)
{
    List<DateTime> dateList = new List<DateTime>();
    dateList = (args.visibleDates);
    var headerString = String.Empty;
    var item = dateList[0];
    if (Schedule.ScheduleView == ScheduleView.DayView)
    {
        item = dateList[0];
        headerString = item.Date.ToString("MMMM, yyyy");
    }
    else
    {
        item = dateList[dateList.Count / 2];
        headerString = item.Date.ToString("MMMM, yyyy");
    }
}
```

You can get the complete sample for customizing the Header of Schedule [here](#)

#### Header Date Format

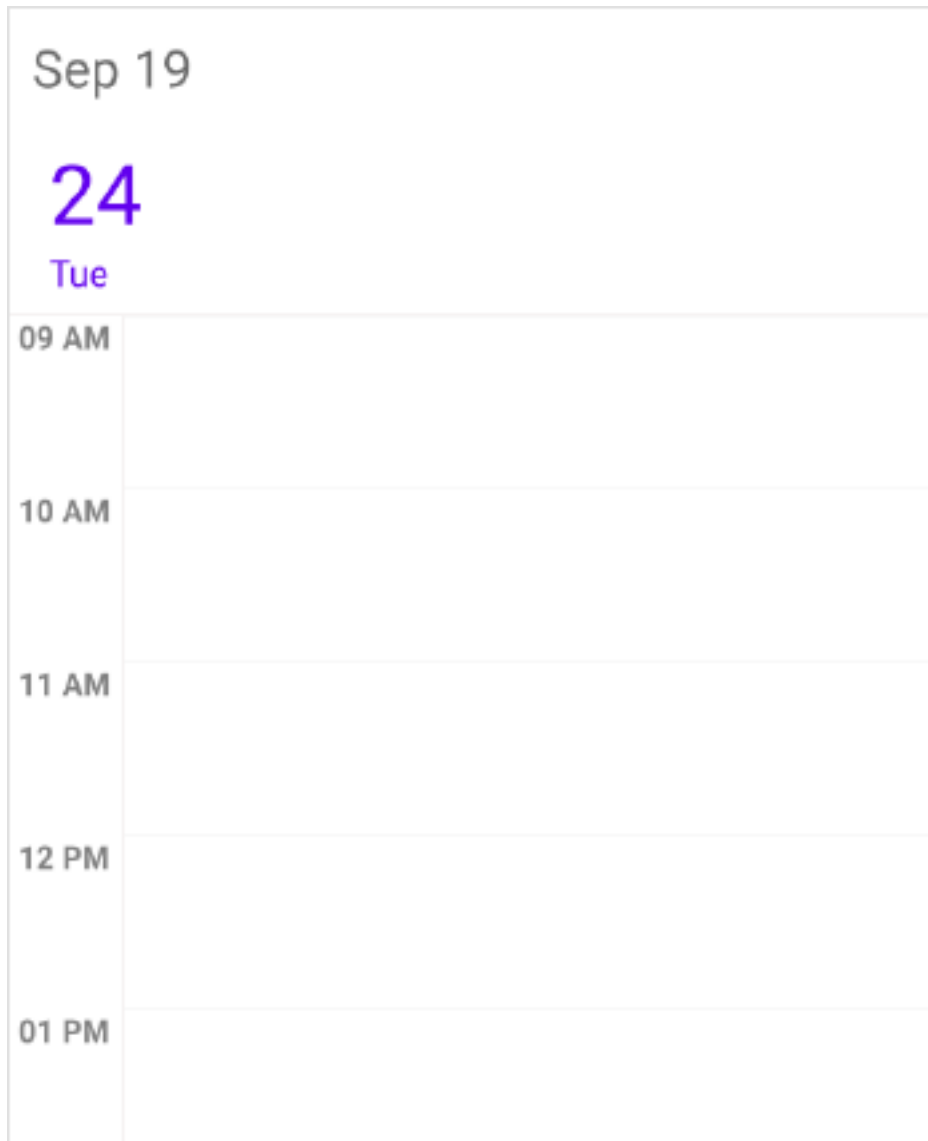
You can customize the date format of SfSchedule Header by using [ScheduleHeaderDateFormat](#) property of SfSchedule.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleHeaderDateFormat = "LLL yy" />
```

#### C#

```
//Creating instance of Schedule
SfSchedule schedule = new SfSchedule();
//Customizing date format
schedule.ScheduleHeaderDateFormat = "LLL yy";
```



### Header Tapped Event

You can handle single tap action of Header by using [HeaderTapped](#) event of [SfSchedule](#). This event will be triggered when the Header is Tapped. This event contains [HeaderTappedEventArgs](#) argument which holds [DateTime](#) details in it.

### XML

```
<schedule:SfSchedule x:Name="schedule" HeaderTapped="Handle_HeaderTapped" />
```

### C#

```
//Creating new instance of Schedule  
SfSchedule schedule = new SfSchedule();  
schedule.HeaderTapped += Handle_HeaderTapped;
```

### C#

```
private void Handle_HeaderTapped(object sender, HeaderTappedEventArgs e)
{
    var dateTime = e.DateTime;
}
```

## DayView

DayView is used to display a single day, current day will be visible by default. Appointments on a specific day will be arranged in respective timeslots based on its duration.

### ViewHeader Appearance

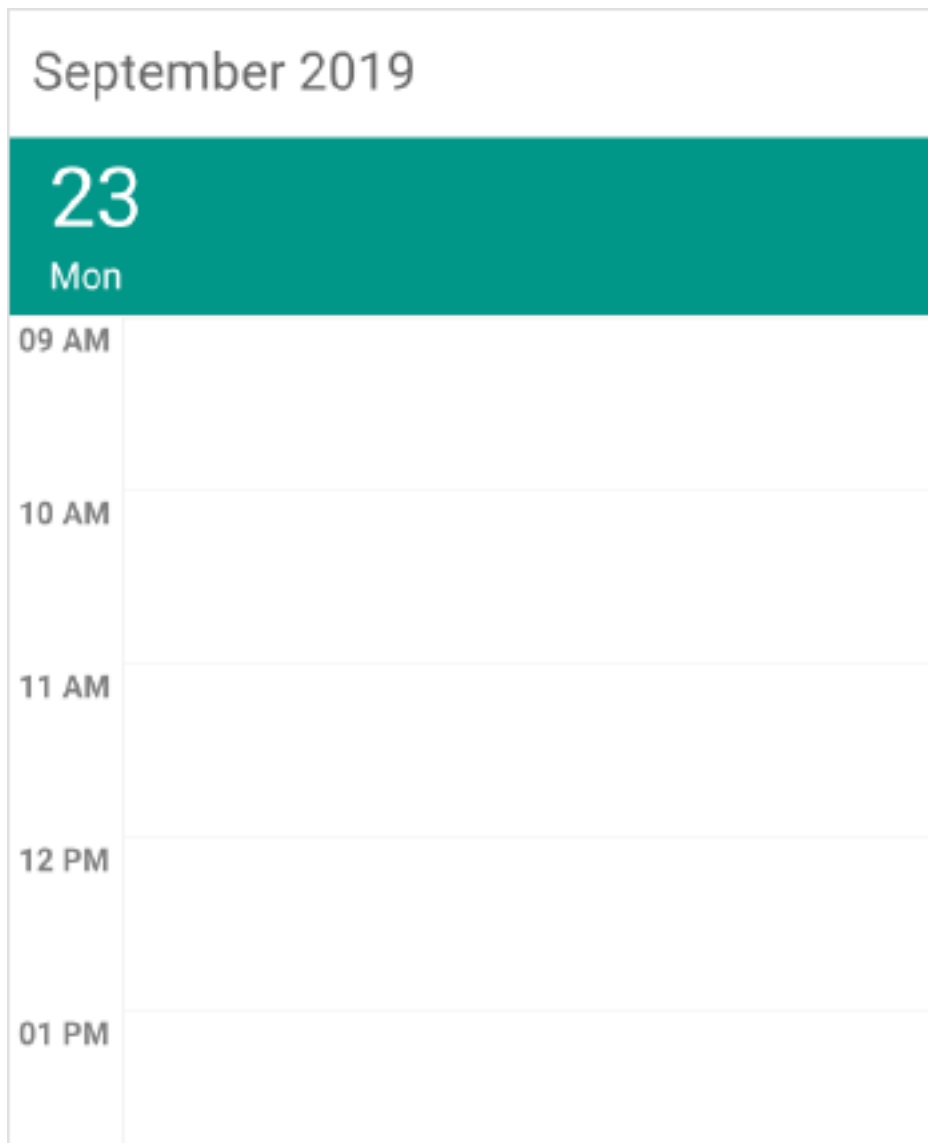
You can customize the default appearance of view header in [DayView](#) by setting [BackgroundColor](#), [DayTextColor](#), [DateTextColor](#), [DayFontFamily](#), [DateFontFamily](#), [CurrentDateTextColor](#), [CurrentDayTextColor](#), [DayFontAttributes](#), [DateFontAttributes](#), [DayFontSize](#) and [DateFontSize](#) properties of [ViewHeaderStyle](#) property in [SfSchedule](#).

### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView = "DayView">
<schedule:SfSchedule.ViewHeaderStyle>
<schedule:ViewHeaderStyle
BackgroundColor="#009688"
DayTextColor="#FFFFFF"
DateTextColor="#FFFFFF"
DayFontFamily="Arial"
DateFontFamily="Arial">
</schedule:ViewHeaderStyle>
</schedule:SfSchedule.ViewHeaderStyle>
</schedule:SfSchedule>
```

### C#

```
//Create new instance of Schedule
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.DayView;
//Customize the schedule view header
ViewHeaderStyle viewHeaderStyle = new ViewHeaderStyle();
viewHeaderStyle.BackgroundColor = Color.FromHex("#009688");
viewHeaderStyle.DayTextColor = Color.FromHex("#FFFFFF");
viewHeaderStyle.DateTextColor = Color.FromHex("#FFFFFF");
viewHeaderStyle.DayFontFamily = "Arial";
viewHeaderStyle.DateFontFamily = "Arial";
schedule.ViewHeaderStyle = viewHeaderStyle;
```

**NOTE**

FontAttributes and FontFamily are native to the platform. Custom font and the font which are not available in the specified platform will not be applied.

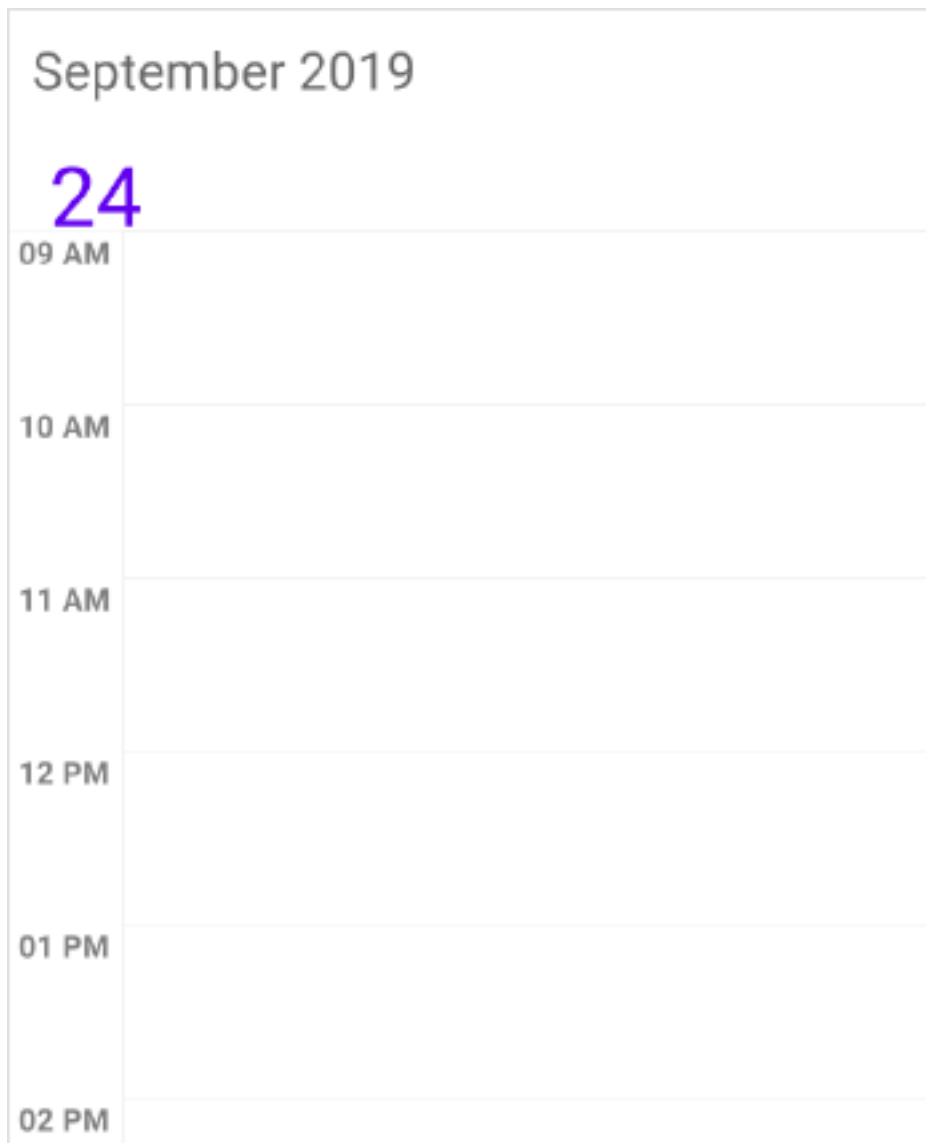
You can customize the height of the ViewHeader in **DayView** by setting [ViewHeaderHeight](#) property of SfSchedule.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView = "DayView"
ViewHeaderHeight="50" >
```

**C#**

```
schedule.ScheduleView = ScheduleView.DayView;
schedule.ViewHeaderHeight = 50;
```



#### *Customize Font Appearance*

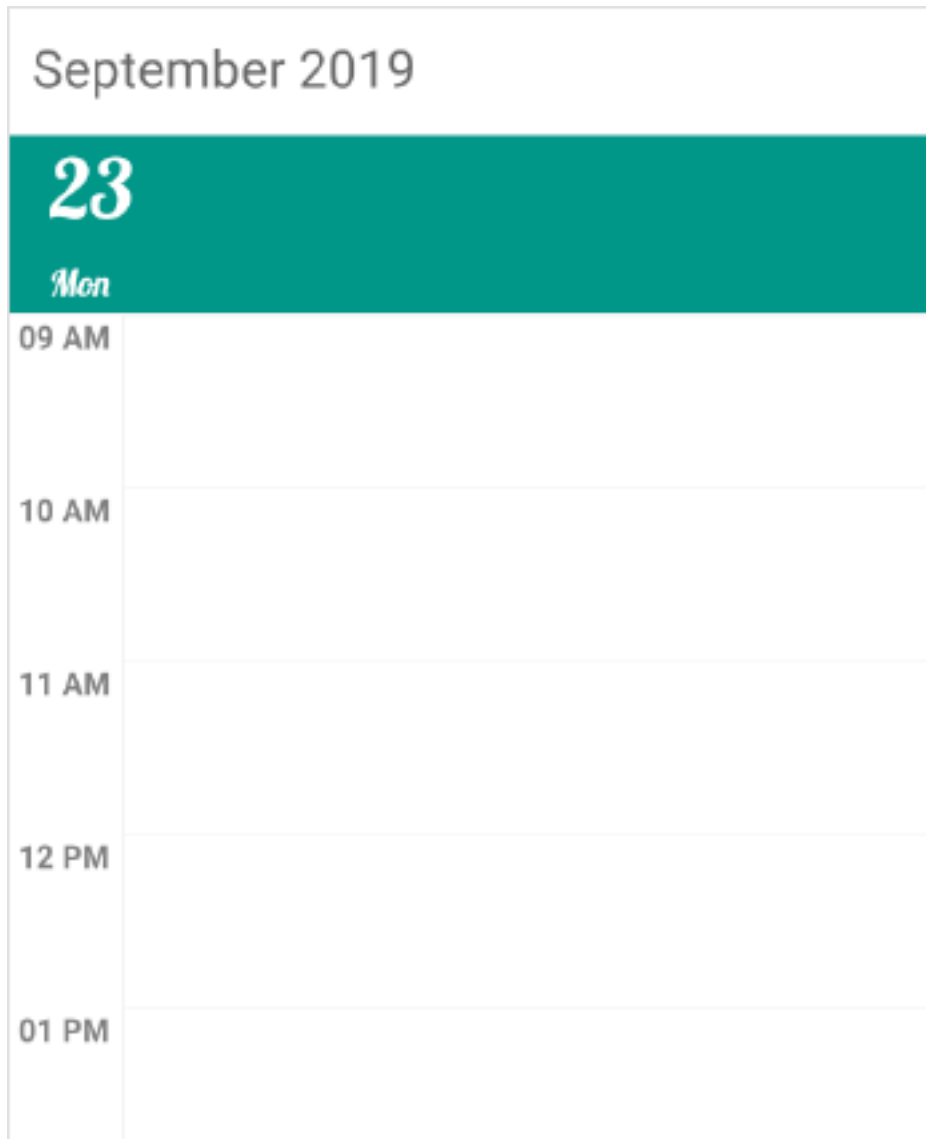
you can change the appearance of Font by setting the [DayFontFamily](#) and [DateFontFamily](#) property of [ViewHeaderStyle](#) property in Schedule.

#### **XML**

```
<schedule:ViewHeaderStyle.DayFontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:ViewHeaderStyle.DayFontFamily>
<schedule:ViewHeaderStyle.DateFontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:ViewHeaderStyle.DateFontFamily>
```

#### **C#**

```
viewHeaderStyle.DayFontFamily = Device.OnPlatform("Lobster-Regular",
"Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
viewHeaderStyle.DateFontFamily = Device.OnPlatform("Lobster-Regular",
"Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
```



Refer [this](#) to configure the custom fonts in Xamarin.Forms.

#### *ViewHeader Date Format*

You can customize the date and day format of SfSchedule ViewHeader by using [DateFormat](#) and [DayFormat](#) properties of [DayLabelSettings](#).

#### **XML**

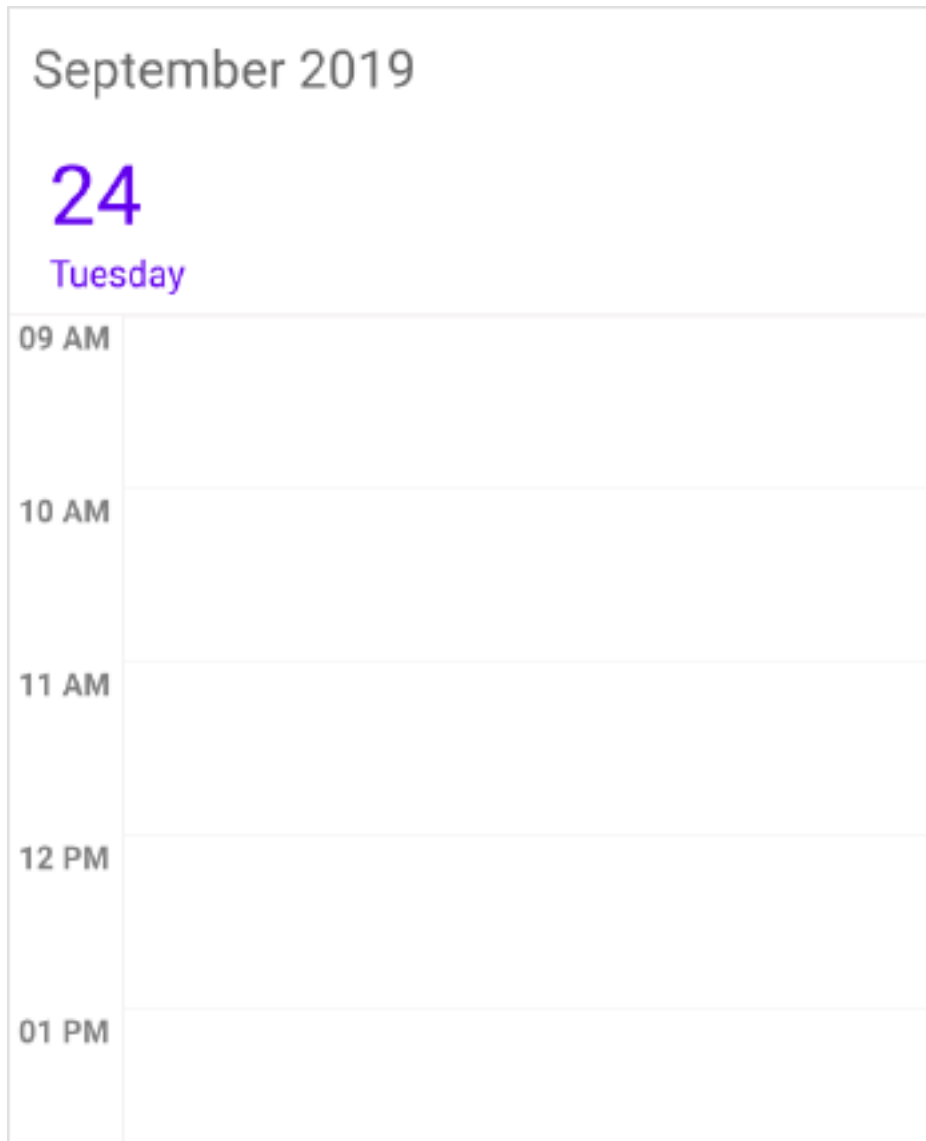
```
<schedule:SfSchedule>
  <schedule:SfSchedule.DayViewSettings>
    <schedule:DayViewSettings>
      <schedule:DayViewSettings.DayLabelSettings>
        <schedule:DayLabelSettings DateFormat="dd">
```



```
<schedule:DayLabelSettings.DayFormat>
<OnPlatform x:TypeArguments="x:String" iOS="EEE d MMMM YY" Android="EEEE"
WinPhone="dddd" />
</schedule:DayLabelSettings.DayFormat>
</schedule:DayLabelSettings>
</schedule:DayViewSettings.DayLabelSettings>
</schedule:DayViewSettings>
</schedule:SfSchedule.DayViewSettings>
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.DayView;
//Creating new instance of DayViewSettings
DayViewSettings dayViewSettings = new DayViewSettings();
//Creating new instance of DayLabelSettings
DayLabelSettings dayLabelSettings = new DayLabelSettings();
//Customizing date format
dayLabelSettings.DateFormat = "dd";
dayLabelSettings.DayFormat = Device.OnPlatform("EEE d MMMM YY", "EEEE",
"dddd");
dayViewSettings.DayLabelSettings = dayLabelSettings;
schedule.DayViewSettings = dayViewSettings;
```



#### *ViewHeader Tapped Event*

You can handle single tap action of ViewHeader by using [ViewHeaderTapped](#) event of **SfSchedule**. This event will be triggered when the ViewHeader is Tapped. This event contains [ViewHeaderTappedEventArgs](#) argument which holds [DateTime](#) details in it.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule"
ScheduleView="DayView"
ViewHeaderTapped="Handle_ViewHeaderTapped" >
</schedule:SfSchedule>
```

#### **C#**

```
//Creating new instance of Schedule
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.DayView;
schedule.ViewHeaderTapped += Handle_ViewHeaderTapped;
```

**C#**

```
private void Handle_ViewHeaderTapped(object sender,
ViewHeaderTappedEventArgs e)
{
    var dateTime = e.DateTime;
}
```

**Change Time Interval**

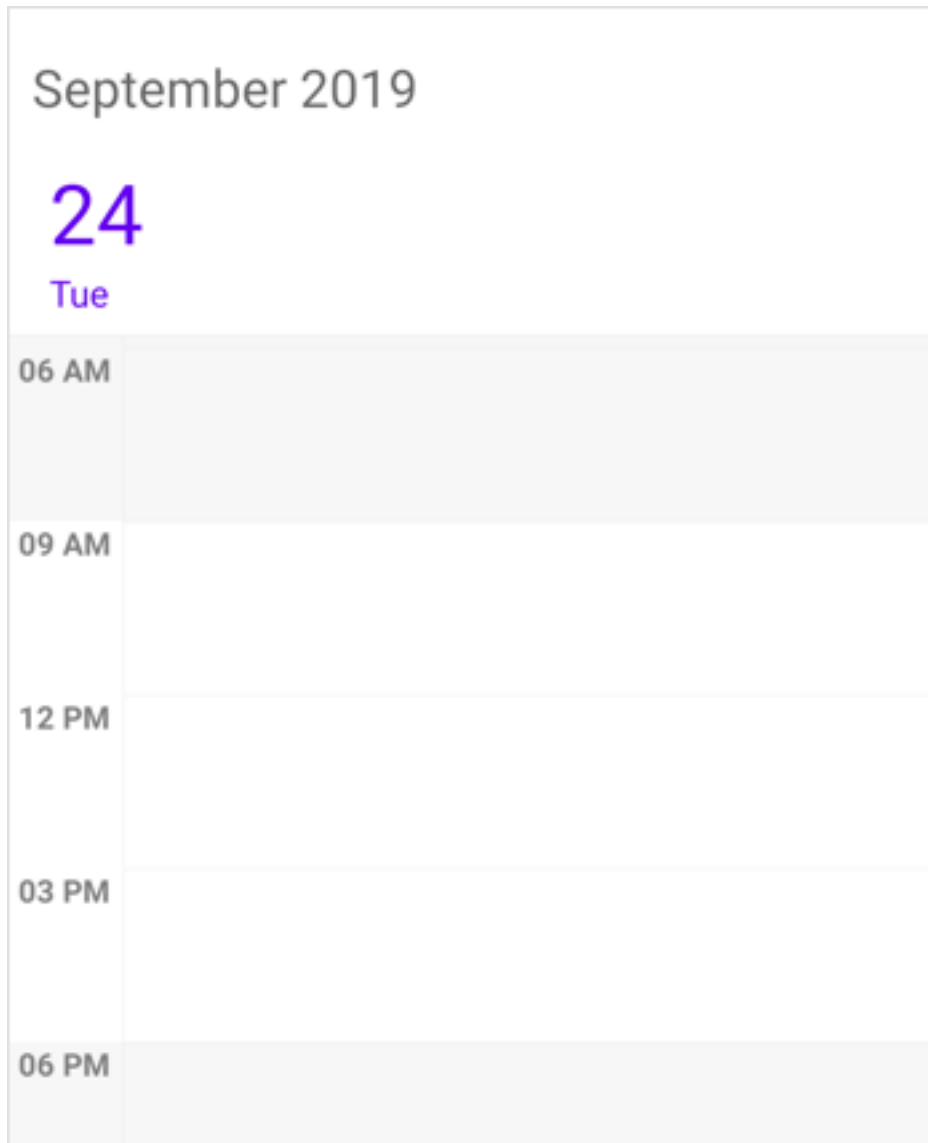
You can customize the interval of timeslots in **DayView** by setting [TimeInterval](#) property of **SfSchedule**.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView"
TimeInterval="180"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.DayView;
schedule.TimeInterval = 180;
```

**NOTE**

If you modify the `TimeInterval` value (in minutes), you need to change the time labels format by setting the `TimeFormat` value as "hh:mm". By default, `TimeFormat` value is "hh a". You can refer [here](#) for changing `TimeFormat` value.

**Change Time Interval Height**

You can customize the interval height of timeslots in `DayView` by setting [TimeIntervalHeight](#) property of `SfSchedule`.

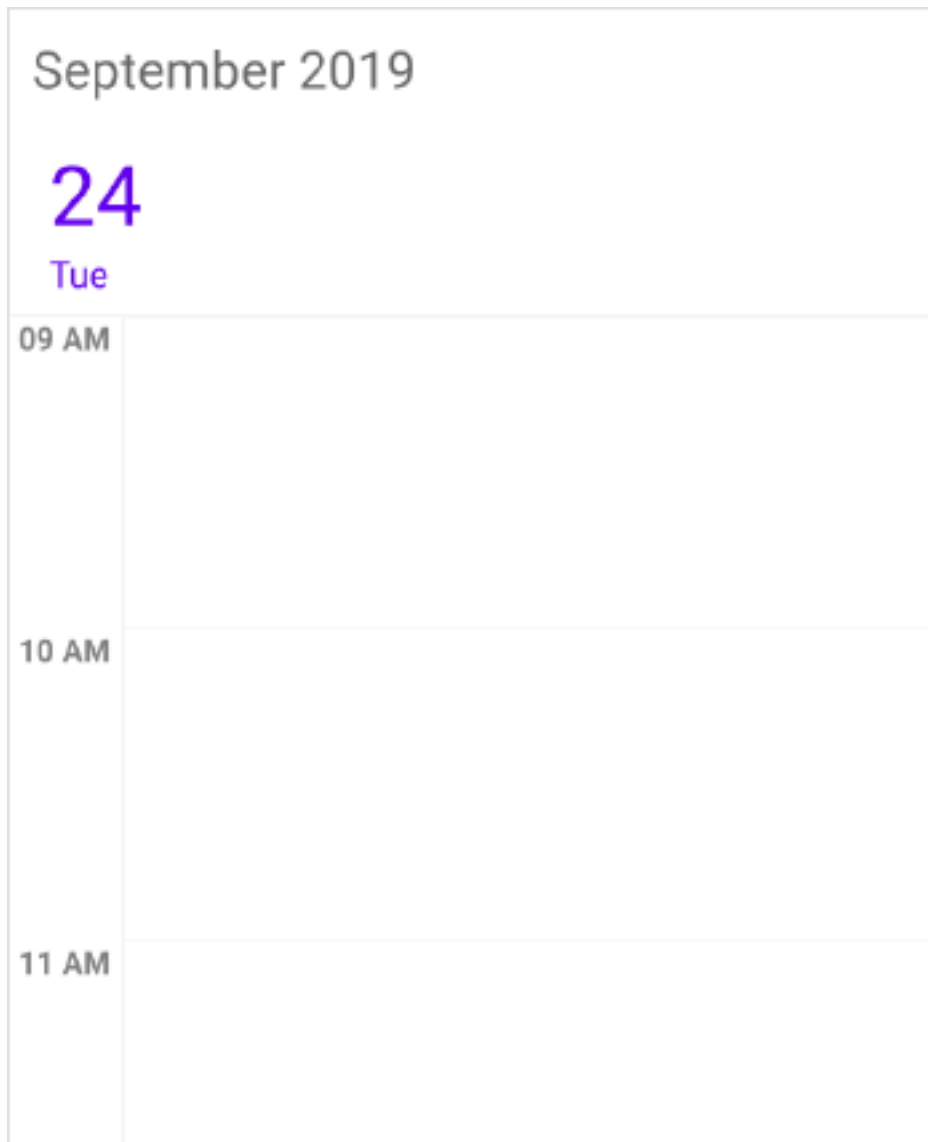
**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView"
    TimeIntervalHeight="180"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.DayView;
```

```
schedule.TimeIntervalHeight = 180;
```



#### *Full screen scheduler*

Schedule time interval height can be adjusted based on screen height by changing the value of `TimeIntervalHeight` property to -1. It will auto-fit to the screen height and width.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView"
TimeIntervalHeight="-1"/>
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.DayView;
schedule.TimeIntervalHeight = -1;
```

### Change Working hours

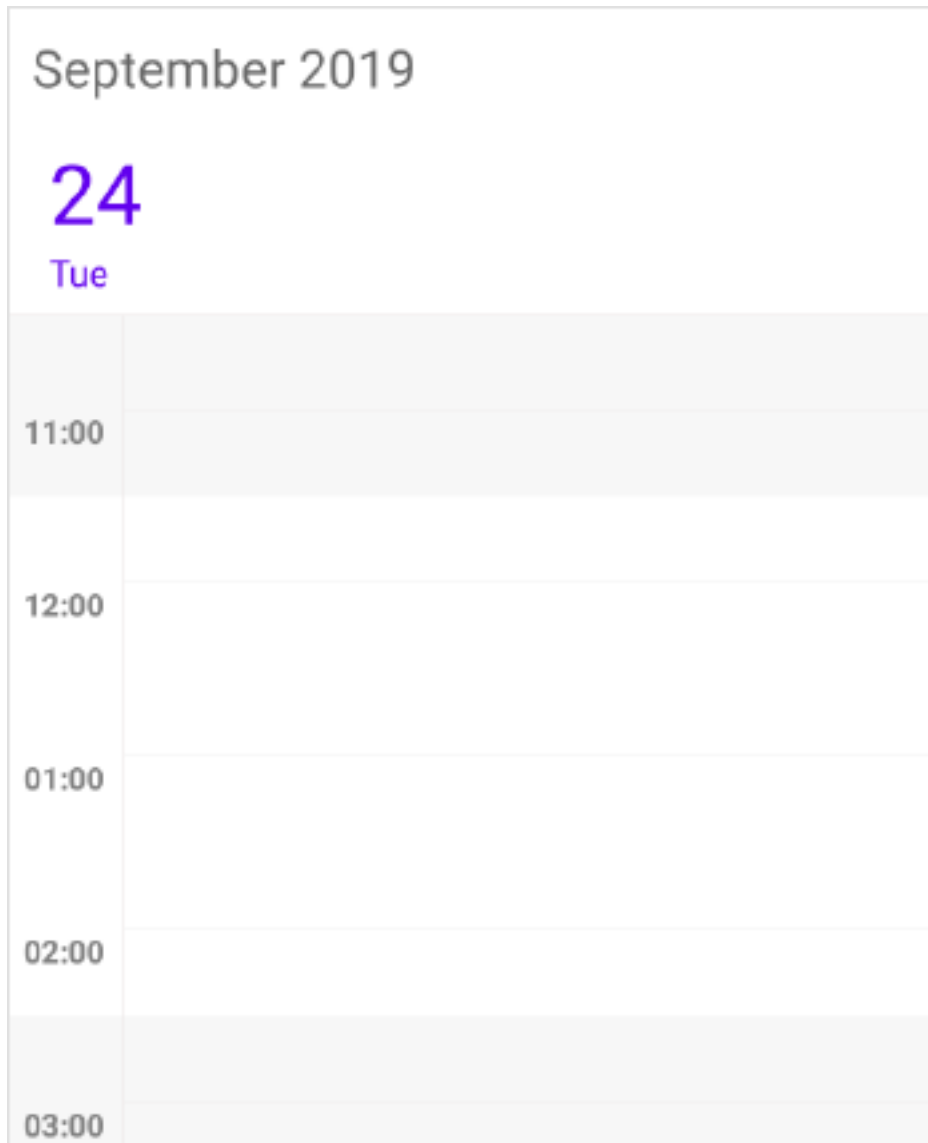
Working hours in **DayView** of Schedule control will be differentiated with non-working hours by separate color. By default, working hours will be between 09 to 18. You can customize the working hours by setting [WorkStartHour](#) and [WorkEndHour](#) properties of [DayViewSettings](#). You can also customize the working hours along with minutes by setting double value which will be converted to time.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView">
  <schedule:SfSchedule.DayViewSettings>
    <!--setting working hours properties -->
    <schedule:DayViewSettings
      WorkStartHour="11.5"
      WorkEndHour="17.5">
      <schedule:DayViewSettings.DayLabelSettings>
        <schedule:DayLabelSettings TimeFormat="hh:mm"/>
      </schedule:DayViewSettings.DayLabelSettings>
    </schedule:DayViewSettings>
  </schedule:SfSchedule.DayViewSettings>
</schedule:SfSchedule>
```

#### C#

```
schedule.ScheduleView = ScheduleView.DayView;
//Create new instance of DayViewSettings
DayViewSettings dayViewSettings = new DayViewSettings();
DayLabelSettings dayLabelSettings = new DayLabelSettings();
dayLabelSettings.TimeFormat = "hh:mm";
dayViewSettings.WorkStartHour = 11.5;
dayViewSettings.WorkEndHour = 14.5;
dayViewSettings.DayLabelSettings = dayLabelSettings;
schedule.DayViewSettings = dayViewSettings;
```

**NOTE**

No need to specify the decimal point values for `WorkStartHour` and `WorkEndHour`, if you don't want to set the minutes.

**Changing StartHour and EndHour**

Default value for `StartHour` and `EndHour` value is 0 to 24 to show all the time slots in `DayView`. You need to set `StartHour` and `EndHour` property of `DayView`, to show only the required time duration for end users. You can also set `StartHour` and `EndHour` in double value which will be converted to time to show required time duration in minutes.

**XML**

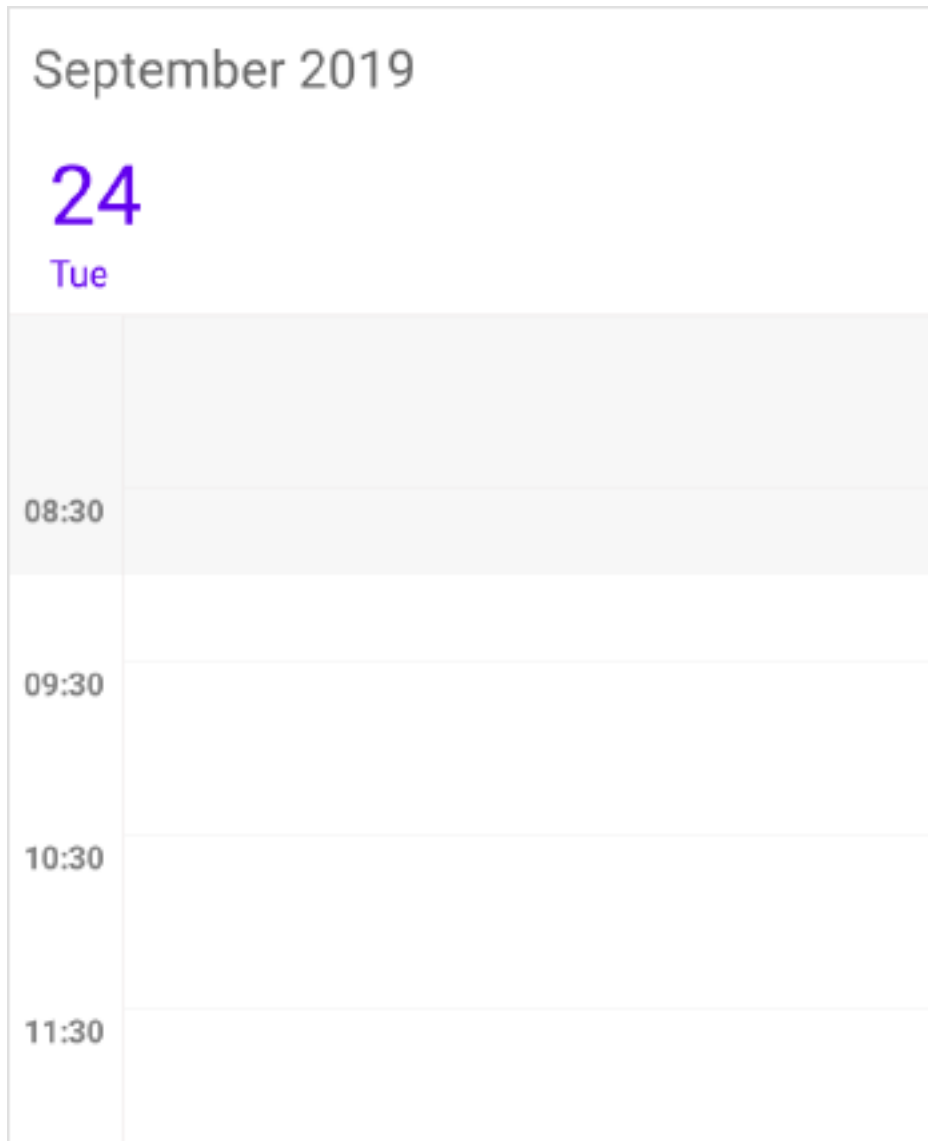
```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView">
  <schedule:SfSchedule.DayViewSettings>
    <!--setting working hours properties -->
    <schedule:DayViewSettings
      StartHour="7.5"
```

```
EndHour="18.5">
<schedule:DayViewSettings.DayLabelSettings>
<schedule:DayLabelSettings TimeFormat="hh:mm" />
</schedule:DayViewSettings.DayLabelSettings>
</schedule:DayViewSettings>
</schedule:SfSchedule.DayViewSettings>
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.DayView;
//Create new instance of DayViewSettings
DayViewSettings dayViewSettings = new DayViewSettings();
DayLabelSettings dayLabelSettings = new DayLabelSettings();
dayLabelSettings.TimeFormat = "hh:mm";
dayViewSettings.StartHour = 7.5;
dayViewSettings.EndHour = 18.5;
dayViewSettings.DayLabelSettings = dayLabelSettings;
schedule.DayViewSettings = dayViewSettings;
```



**NOTE**

- **StartHour** must be greater than or equal to 0 and **EndHour** must be lesser than or equal to 24, otherwise **InvalidDataException** will be thrown.
- **EndHour** value must be greater than **StartHour**, otherwise **InvalidDataException** will be thrown.
- Schedule UI such as Appointments and NonAccessibleBlocks 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 given **TimeInterval** should result in integer value (total minutes % **TimeInterval** = 0), otherwise next immediate time interval that result in integer value when divide total minutes of a day will be

considered. For example, if `TimeInterval="135"` (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 given `StartHour` and `EndHour` should result in integer value, otherwise next immediate time interval 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" ( $555 \% 37 = 0$ ) by considering (total minutes % `TimeInterval` = 0); it will return integer value for time slots rendering.

### Timeslot Appearance

You can customize the appearance of timeslots in `DayView`.

- [Timeslot customization in Work hours](#)
- [Timeslot customization in Non Working hours](#)

#### *Timeslot customization in Work hours*

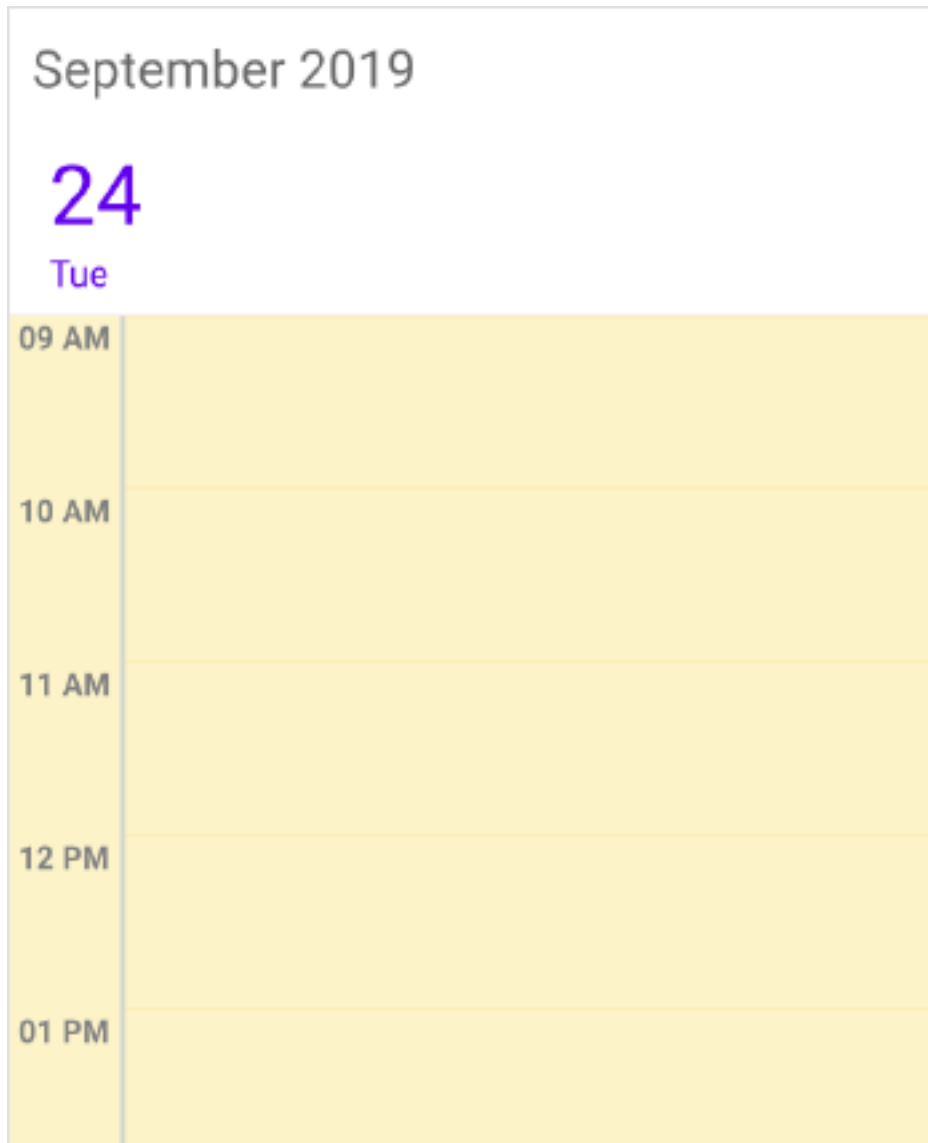
You can customize the appearance of the `WorkingHourTimeslot` by its color using [TimeSlotColor](#), [TimeSlotBorderColor](#), [TimeSlotBorderStrokeWidth](#), [VerticalLineColor](#) and [VerticalLineStrokeWidth](#) properties of `DayViewSettings`.

### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView">
  <schedule:SfSchedule.DayViewSettings>
    <!--setting DayView settings properties -->
    <schedule:DayViewSettings
      TimeSlotColor="#fcf3c9"
      TimeSlotBorderColor="#fceb9f"
      TimeSlotBorderStrokeWidth="5"
      VerticalLineColor="LightGray"
      VerticalLineStrokeWidth="5">
    </schedule:DayViewSettings>
  </schedule:SfSchedule.DayViewSettings>
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.DayView;
//Create new instance of DayViewSettings
DayViewSettings dayViewSettings = new DayViewSettings();
dayViewSettings.TimeSlotBorderColor = Color.FromHex("#fceb9f");
dayViewSettings.TimeSlotColor = Color.FromHex("#fcf3c9");
dayViewSettings.TimeSlotBorderStrokeWidth = 5;
dayViewSettings.VerticalLineColor = Color.LightGray;
dayViewSettings.VerticalLineStrokeWidth = 5;
schedule.DayViewSettings = dayViewSettings;
```



#### *Timeslot customization in Non Working hours*

You can customize the appearance of the Non-workingHourTimeslots by its color using [NonWorkingHoursTimeSlotBorderColor](#), [NonWorkingHoursTimeSlotColor](#), [VerticalLineColor](#) and [VerticalLineStrokeWidth](#) properties of `DayViewSettings`.

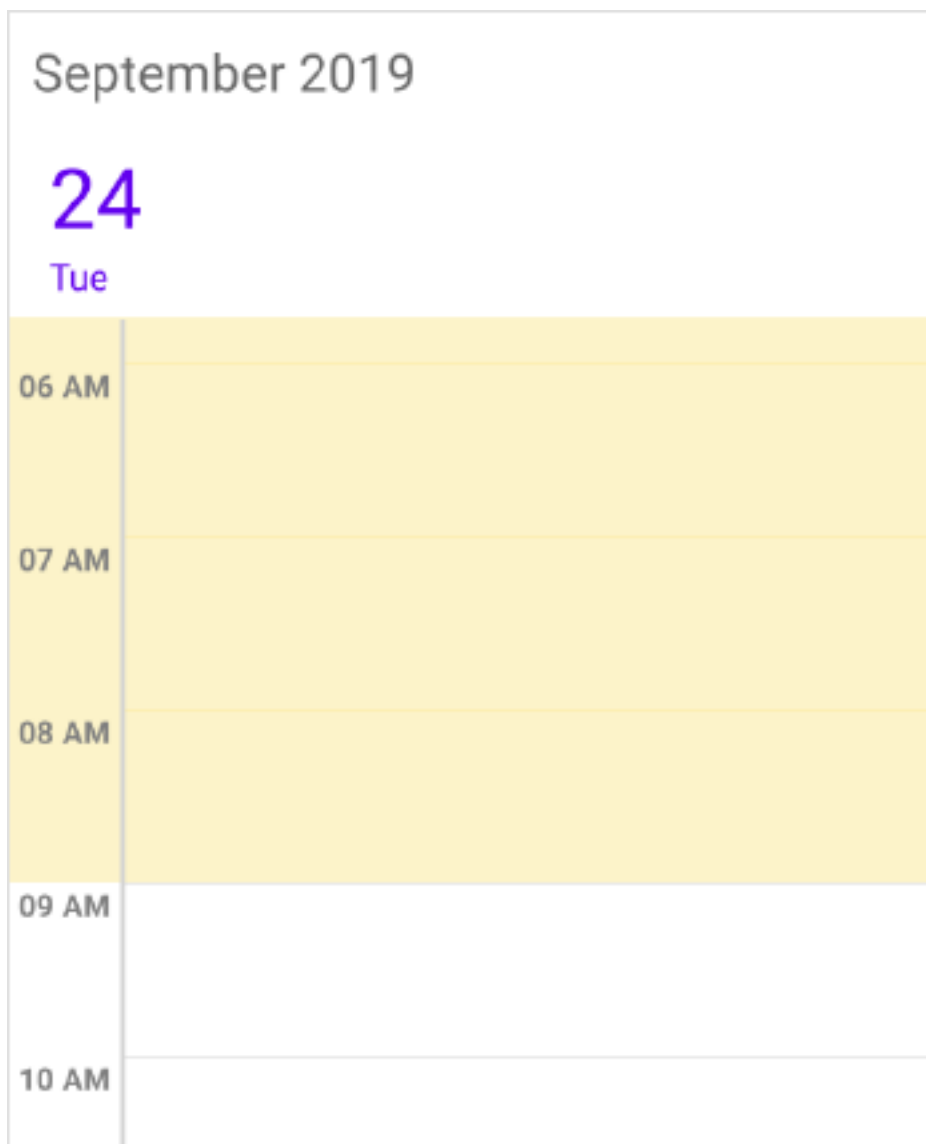
#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView">
  <schedule:SfSchedule.DayViewSettings>
    <!--setting Day view settings properties -->
    <schedule:DayViewSettings
      NonWorkingHoursTimeSlotColor="#fcf3c9"
      NonWorkingHoursTimeSlotBorderColor="#fceb9f"
      TimeSlotBorderStrokeWidth="5"
      VerticalLineColor="LightGray"
      VerticalLineStrokeWidth="5">
    </schedule:DayViewSettings>
  </schedule:SfSchedule.DayViewSettings>
```

```
</schedule:SfSchedule>
```

**C#**

```
schedule.ScheduleView = ScheduleView.DayView;  
//Create new instance of DayViewSettings  
DayViewSettings dayViewSettings = new DayViewSettings();  
dayViewSettings.NonWorkingHoursTimeSlotBorderColor =  
Color.FromHex("#fceb9f");  
dayViewSettings.NonWorkingHoursTimeSlotColor = Color.FromHex("#fcf3c9");  
dayViewSettings.TimeSlotBorderStrokeWidth = 5;  
dayViewSettings.VerticalLineColor = Color.LightGray;  
dayViewSettings.VerticalLineStrokeWidth = 5;  
schedule.DayViewSettings = dayViewSettings;
```

**NOTE**

TimeSlotBorderStrokeWidth, VerticalLineColor and VerticalLineStrokeWidth properties common for both Working hours and Non-Working hour time slot customization.

VerticalLineColor and VerticalLineStrokeWidth properties applicable for Android only in DayView.

### Non-Accessible timeslots

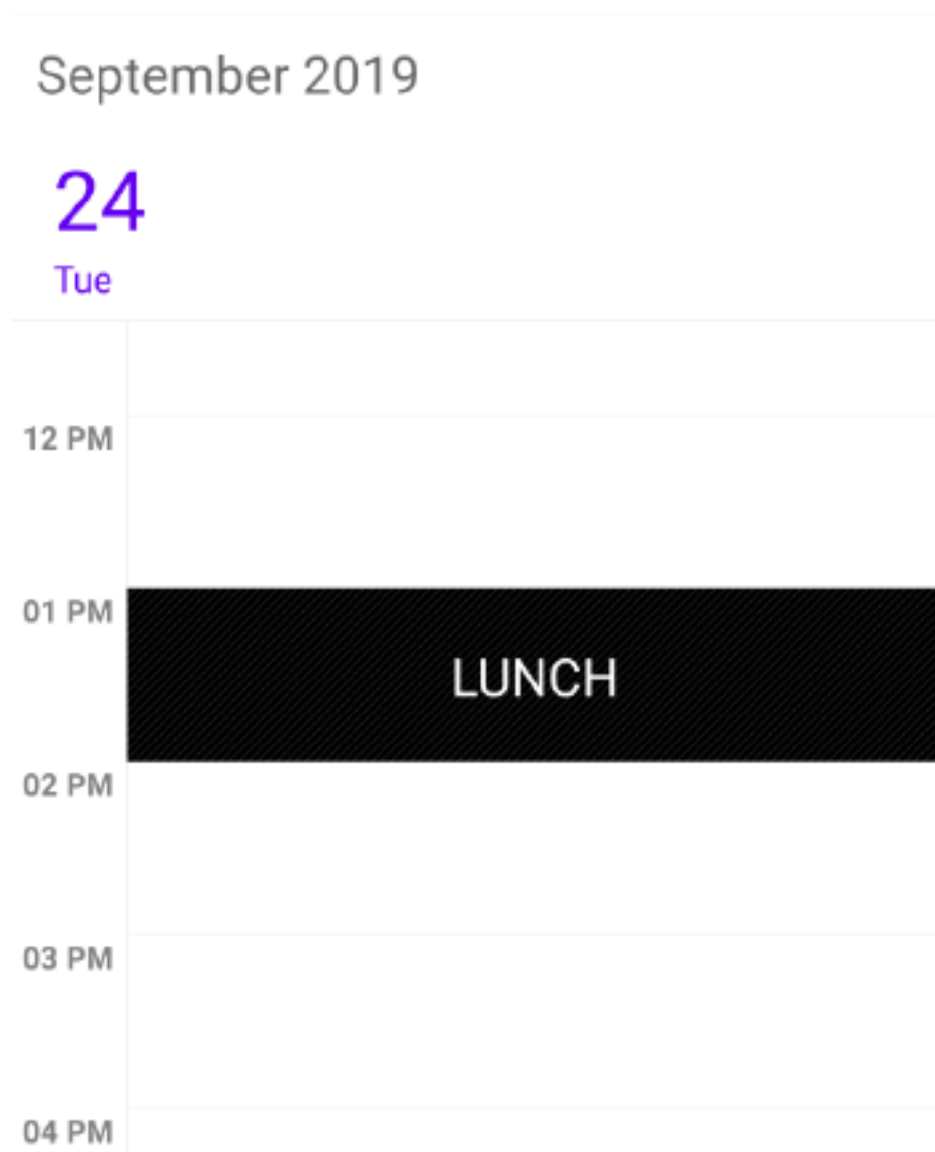
You can restrict or allocate certain timeslot as non-accessible blocks by using [NonAccessibleBlocks](#) of DayViewSettings, so that you can allocate those timeslots for predefined events/activities like Lunch hour using [StartTime](#), [EndTime](#), [Text](#) and [Color](#) properties of [NonAccessibleBlocks](#).

### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView">
  <!--setting non-accessing blocks-->
  <schedule:SfSchedule.DayViewSettings>
    <schedule:DayViewSettings>
      <schedule:DayViewSettings.NonAccessibleBlocks>
        <schedule:NonAccessibleBlock
          StartTime="13"
          EndTime="14"
          Text="LUNCH"
          Color="Black" />
      </schedule:DayViewSettings.NonAccessibleBlocks>
    </schedule:DayViewSettings>
  </schedule:SfSchedule.DayViewSettings>
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.DayView;
//Create new instance of NonAccessibleBlock
NonAccessibleBlock nonAccessibleBlock = new NonAccessibleBlock();
//Create new instance of NonAccessibleBlocksCollection
NonAccessibleBlocksCollection nonAccessibleBlocksCollection = new
NonAccessibleBlocksCollection();
DayViewSettings dayViewSettings = new DayViewSettings();
nonAccessibleBlock.StartTime = 13;
nonAccessibleBlock.EndTime = 14;
nonAccessibleBlock.Text = "LUNCH";
nonAccessibleBlock.Color = Color.Black;
nonAccessibleBlocksCollection.Add(nonAccessibleBlock);
dayViewSettings.NonAccessibleBlocks = nonAccessibleBlocksCollection;
schedule.DayViewSettings = dayViewSettings;
```

**NOTE**

Selection and related events will not be working in this blocks.

[Change first day of week](#)

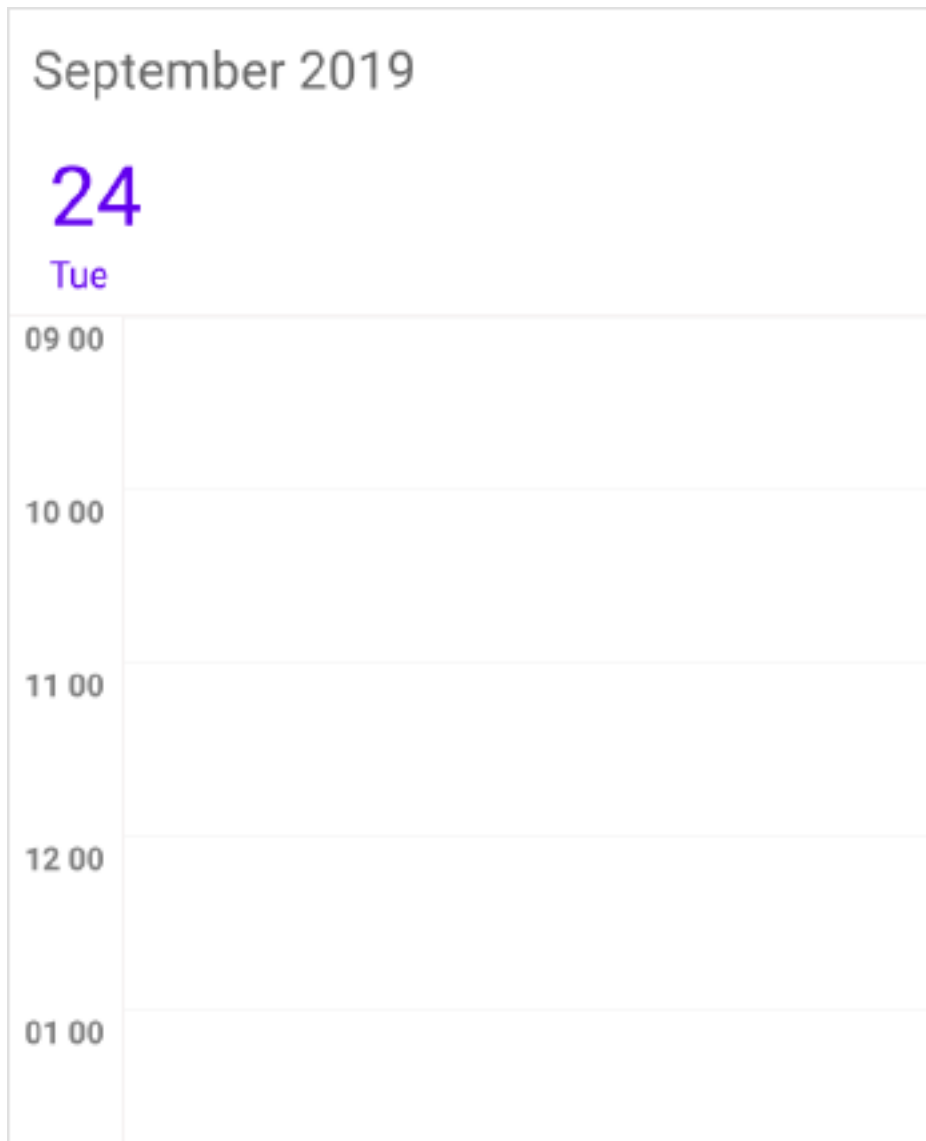
[FirstDayOfWeek](#) of `SfSchedule` is not applicable for `DayView` as it displays only one day.

[Time Label Formatting](#)

You can customize the format for the labels which are mentioning the time, by setting [TimeFormat](#) property of [DayLabelSettings](#) in `DayViewSettings`.

**C#**

```
schedule.ScheduleView = ScheduleView.DayView;  
DayViewSettings dayViewSettings = new DayViewSettings();  
DayLabelSettings dayLabelSettings = new DayLabelSettings();  
dayLabelSettings.TimeFormat = "hh mm";  
dayViewSettings.DayLabelSettings = dayLabelSettings;  
schedule.DayViewSettings = dayViewSettings;
```

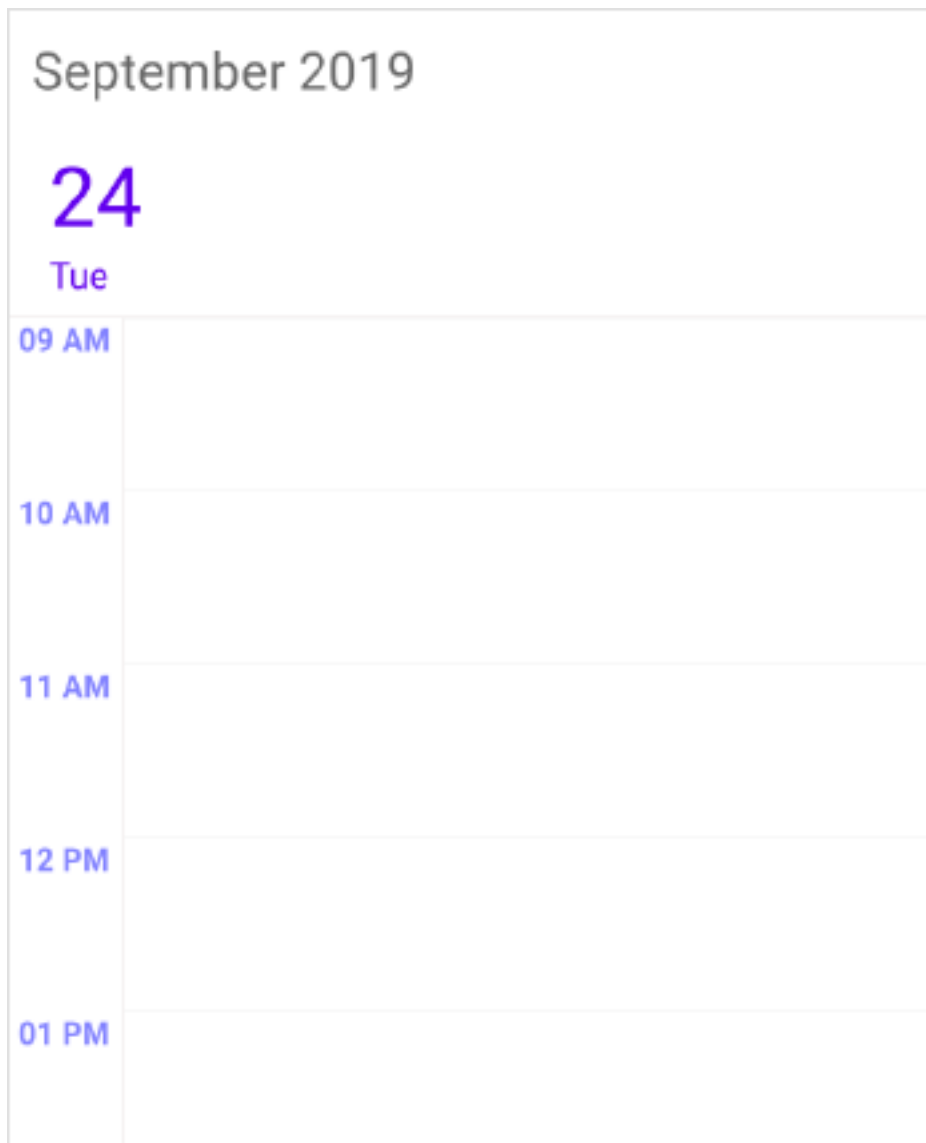


### Time Label Appearance

You can customize the color for the labels which are mentioning the time, by setting [TimeLabelColor](#) property of `DayLabelSettings` in `DayViewSettings`.

### C#

```
schedule.ScheduleView = ScheduleView.DayView;  
//Create new instance of DayViewSettings  
DayViewSettings dayViewSettings = new DayViewSettings();  
//Create new instance of DayLabelSettings  
DayLabelSettings dayLabelSettings = new DayLabelSettings();  
dayLabelSettings.TimeLabelColor = Color.FromHex("#8282ff");  
dayViewSettings.DayLabelSettings = dayLabelSettings;  
schedule.DayViewSettings = dayViewSettings;
```



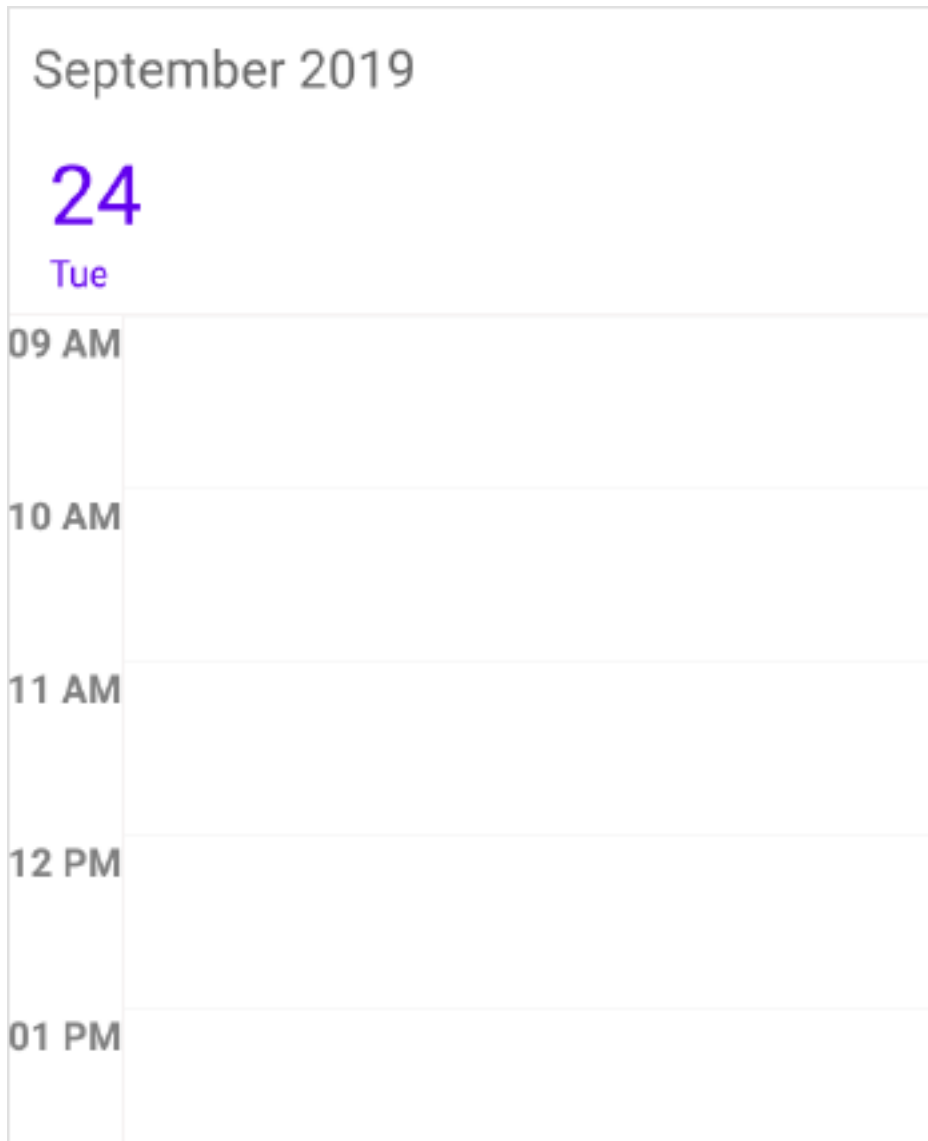
### Time Label Size

You can customize the size of the labels which are mentioning the time, by setting [TimeLabelSize](#) property of `DayLabelSettings` in `DayViewSettings`.

### C#

```
schedule.ScheduleView = ScheduleView.DayView;  
//Create new instance of DayViewSettings  
DayViewSettings dayViewSettings = new DayViewSettings();  
//Create new instance of DayLabelSettings  
DayLabelSettings dayLabelSettings = new DayLabelSettings();  
//Customizing the size of the time label  
dayLabelSettings.TimeLabelSize = 15;  
dayViewSettings.DayLabelSettings = dayLabelSettings;  
schedule.DayViewSettings = dayViewSettings;
```





### Selection

You can customize the default appearance of selection UI in the timeslots.

- [Selection customization using style](#)
- [Selection customization using custom View](#)
- [Programmatic selection](#)

### *Selection customization using style*

You can customize the timeslot selection by by setting the [BackgroundColor](#), [BorderColor](#), [BorderThickness](#), [BorderCornerRadius](#) properties to the [SelectionStyle](#) property of **SfSchedule**.

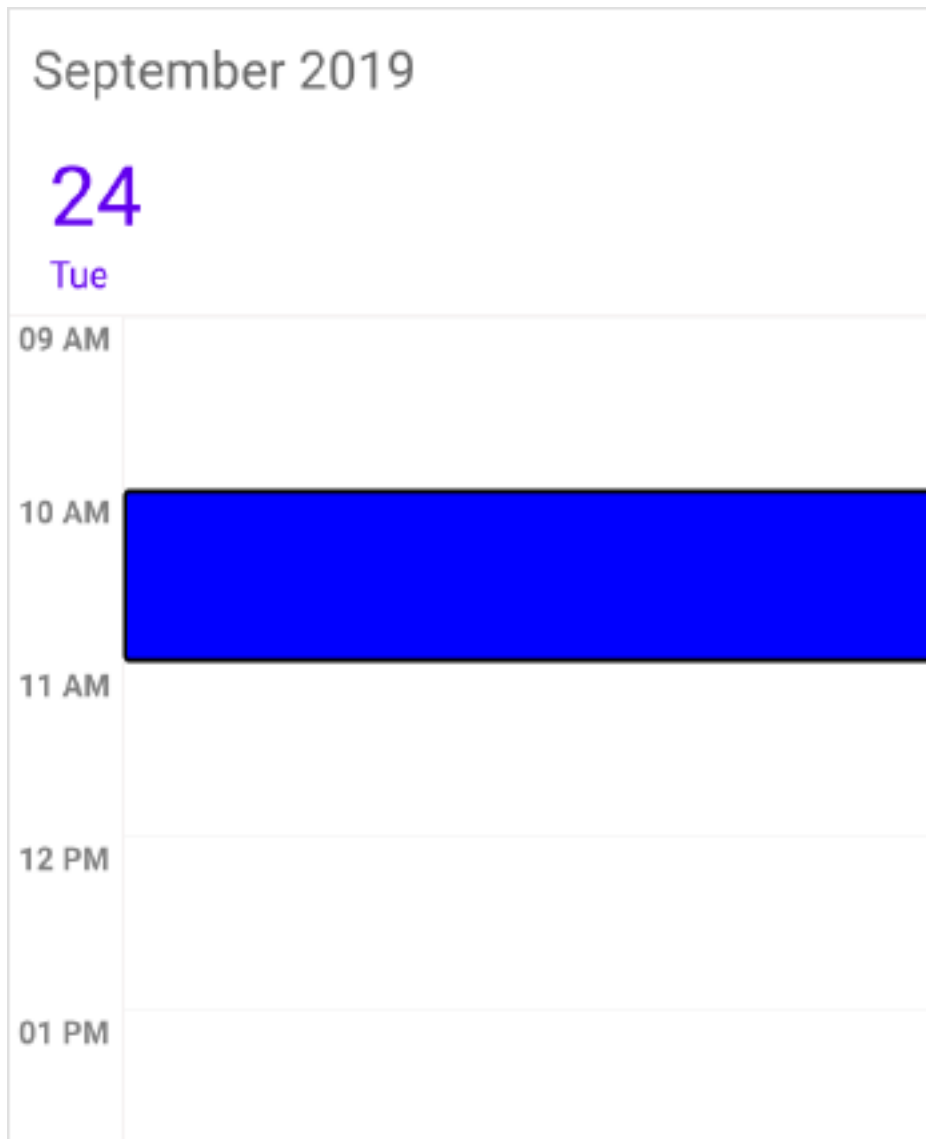
### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView">
  <schedule:SfSchedule.SelectionStyle>
    <schedule:SelectionStyle
      BackgroundColor="Blue"
    </schedule:SelectionStyle>
  </schedule:SfSchedule.SelectionStyle>
</schedule:SfSchedule>
```

```
BorderColor="Black"  
BorderThickness="5"  
BorderCornerRadius="5">  
</schedule:SelectionStyle>  
</schedule:SfSchedule.SelectionStyle>  
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.DayView;  
//Create new instance of SelectionStyle  
SelectionStyle selectionStyle = new SelectionStyle();  
selectionStyle.BackgroundColor = Color.Blue;  
selectionStyle.BorderColor = Color.Black;  
selectionStyle.BorderThickness = 5;  
selectionStyle.BorderCornerRadius = 5;  
schedule.SelectionStyle = selectionStyle;
```



*Selection customization using custom View*

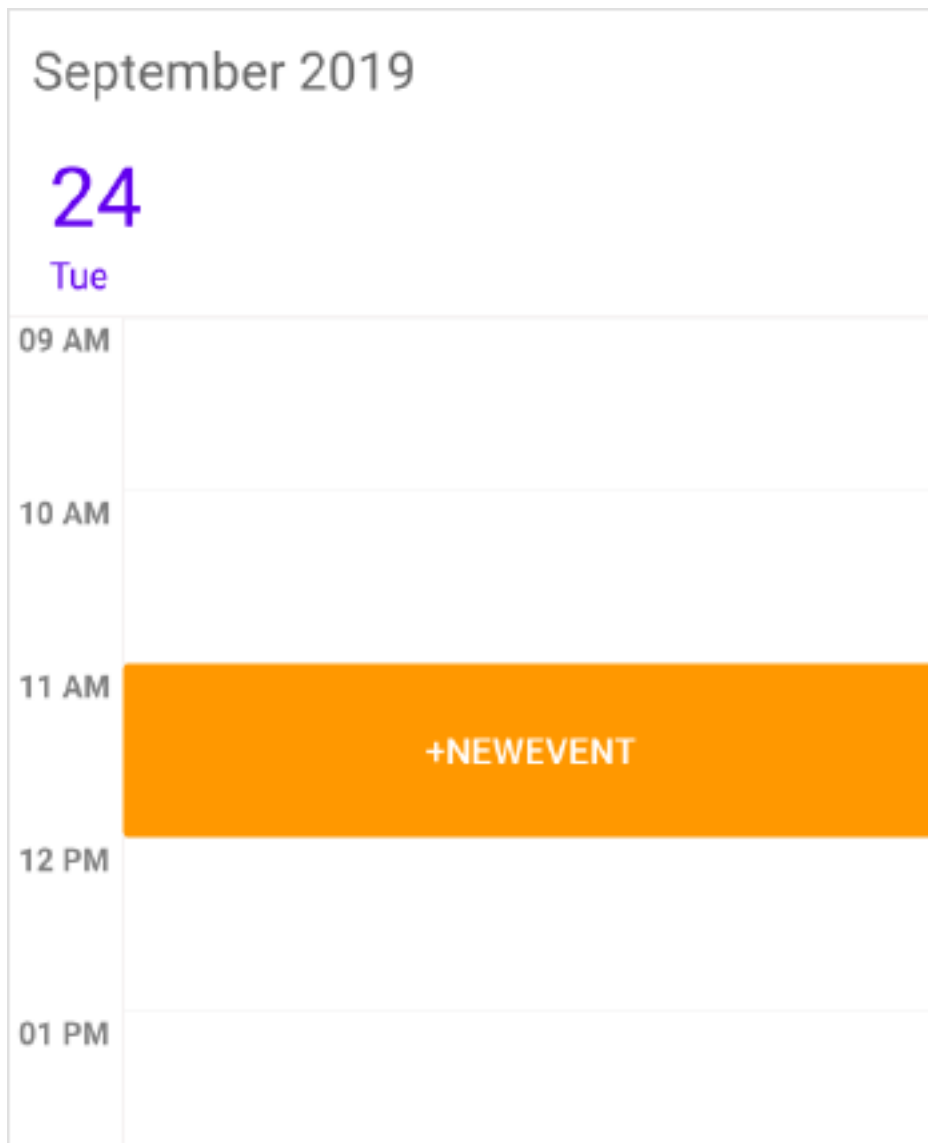
You can replace the default selection UI with your custom view by setting [SelectionView](#) property of SfSchedule.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="DayView">
  <schedule:SfSchedule.SelectionView>
    <Button
      BackgroundColor="#FF9800"
      Text="+NewEvent"
      TextColor="White"/>
  </schedule:SfSchedule.SelectionView>
</schedule:SfSchedule>
```

**C#**

```
schedule.ScheduleView = ScheduleView.DayView;
//Add the CustomView
Button customView = new Button();
customView.Text = "+NewEvent";
customView.BackgroundColor = Color.FromHex("#FF9800");
customView.TextColor = Color.White;
schedule.SelectionView = customView;
```



#### *Programmatic selection*

You can programmatically select the specific timeslot by setting corresponding date and time value to [SelectedDate](#) property of `SfSchedule`. By default, it is null.

#### **C#**

```
// Setting a date and time to select  
schedule.SelectedDate = new DateTime(2017, 10, 04, 10, 0, 0);
```

You can clear the selection by setting [SelectedDate](#) as null.

#### **C#**

```
// Setting null value to deselect  
schedule.SelectedDate = null;
```

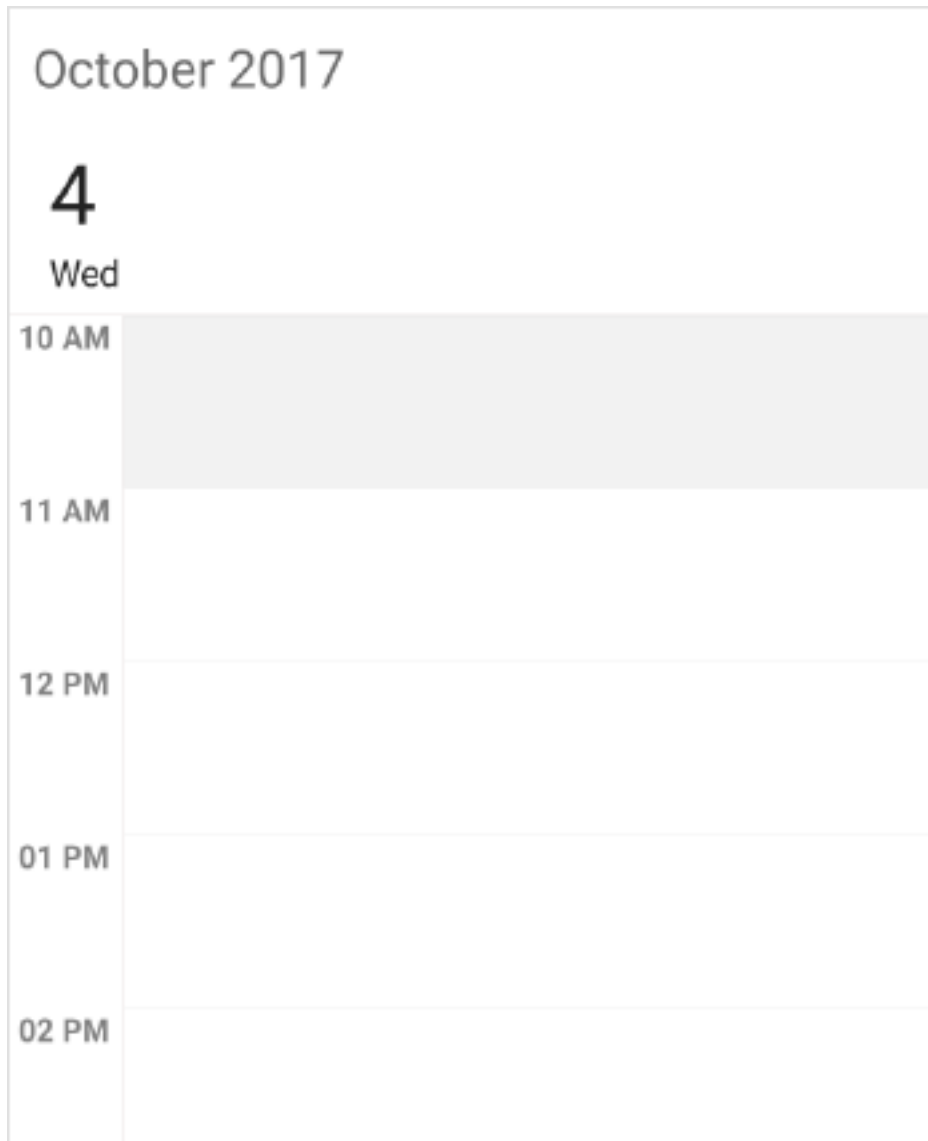
You can download the entire source code of this demo for Xamarin.Forms from here [Date Selection](#)

---

**NOTE**

---

- SfSchedule does not support multiple selection.
- SfSchedule supports two-way binding of SelectedDate property.



### WeekView

WeekView is to view all days of a particular week. Appointments will be arranged based on the dates on the week in respective timeslots.

### ViewHeader Appearance

You can customize the default appearance of view header in [WeekView](#) by using [ViewHeaderStyle](#) property of [SfSchedule](#).

### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView ="WeekView">
<schedule:SfSchedule.ViewHeaderStyle>
```

```
<schedule:ViewHeaderStyle
BackgroundColor="#009688"
DayTextColor="#FFFFFF"
DateTextColor="#FFFFFF"
DayFontFamily="Arial"
DateFontFamily="Arial">
</schedule:ViewHeaderStyle>
</schedule:SfSchedule.ViewHeaderStyle>
</schedule:SfSchedule>
```

## C#

```
//Create new instance of Schedule
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.WeekView;
//Customize the schedule view header
ViewHeaderStyle viewHeaderStyle = new ViewHeaderStyle();
viewHeaderStyle.BackgroundColor = Color.FromHex("#009688");
viewHeaderStyle.DayTextColor = Color.FromHex("#FFFFFF");
viewHeaderStyle.DateTextColor = Color.FromHex("#FFFFFF");
viewHeaderStyle.DayFontFamily = "Arial";
viewHeaderStyle.DateFontFamily = "Arial";
schedule.ViewHeaderStyle = viewHeaderStyle;
```

September 2019							
15 16 17 18 19 20 21							
Sun	Mon	Tue	Wed	Thu	Fri	Sat	
09 AM							
10 AM							
11 AM							
12 PM							
01 PM							

---

**NOTE**

FontAttributes and FontFamily are native to the platform. Custom font and the font which are not available in the specified platform will not be applied.

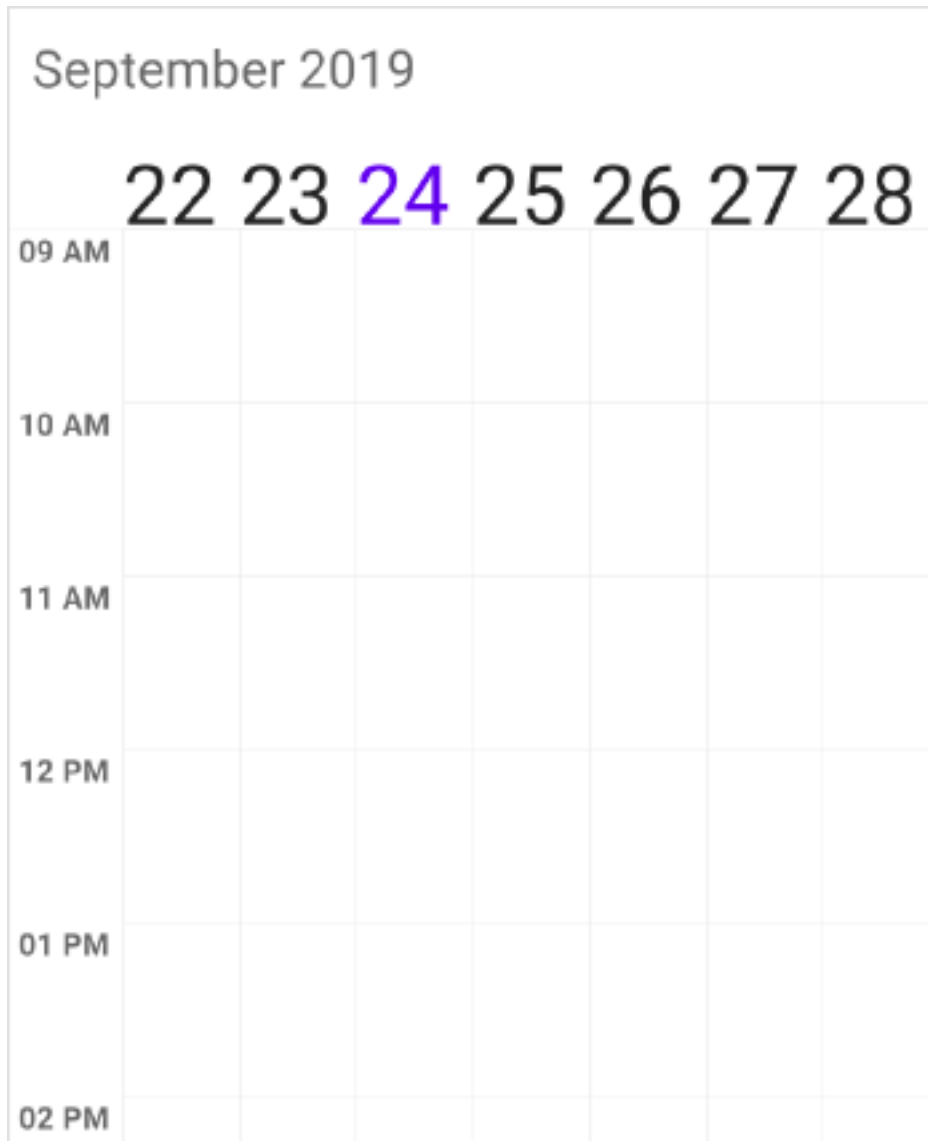
You can customize the height of the ViewHeader in **WeekView** by setting [ViewHeaderHeight](#) property of SfSchedule.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView ="WeekView"
ViewHeaderHeight="50" />
```

**C#**

```
schedule.ScheduleView = ScheduleView.WeekView;
schedule.ViewHeaderHeight = 50;
```



#### *Customize Font Appearance*

you can change the appearance of Font by setting the [DayFontFamily](#) and [DateFontFamily](#) property of [ViewHeaderStyle](#) property in Schedule.

#### **XML**

```
<schedule:ViewHeaderStyle.DayFontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:ViewHeaderStyle.DayFontFamily>
<schedule:ViewHeaderStyle.DateFontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:ViewHeaderStyle.DateFontFamily>
```

#### **C#**



```
viewHeaderStyle.DayFontFamily = Device.OnPlatform("Lobster-Regular",
"Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
viewHeaderStyle.DateFontFamily = Device.OnPlatform("Lobster-Regular",
"Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
```

September 2019						
22 23 24 25 26 27 28						
Sun Mon Tue Wed Thu Fri Sat						
09 AM						
10 AM						
11 AM						
12 PM						
01 PM						

Refer [this](#) to configure the custom fonts in Xamarin.Forms.

#### *ViewHeader Date Format*

You can customize the date and day format of SfSchedule ViewHeader by using [DateFormat](#) and [DayFormat](#) properties of [WeekLabelSettings](#)

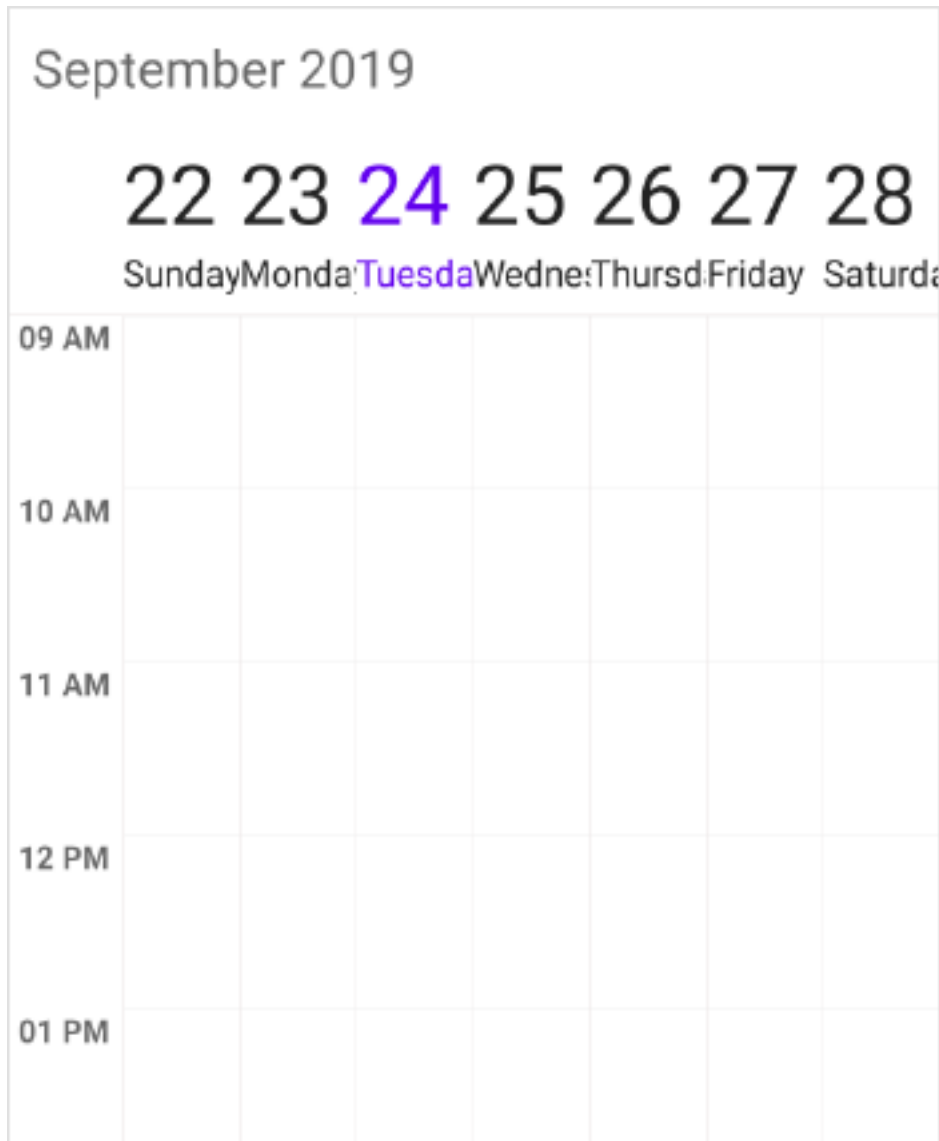
#### **XML**

```
<schedule:SfSchedule>
  <schedule:SfSchedule.WeekViewSettings>
    <schedule:WeekViewSettings>
      <schedule:WeekViewSettings.WeekLabelSettings>
        <schedule:WeekLabelSettings DateFormat="dd">
```

```
<schedule:WeekLabelSettings.DayFormat>
<OnPlatform x:TypeArguments="x:String" iOS="EEEE" Android="EEEE"
WinPhone="dddd" />
</schedule:WeekLabelSettings.DayFormat>
</schedule:WeekLabelSettings>
</schedule:WeekViewSettings.WeekLabelSettings>
</schedule:WeekViewSettings>
</schedule:SfSchedule.WeekViewSettings>
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.WeekView;
//Creating new instance of WeekViewSettings
WeekViewSettings weekViewSettings = new WeekViewSettings();
//Creating new instance of WeekLabelSettings
WeekLabelSettings weekLabelSettings = new WeekLabelSettings();
//Customizing date format
weekLabelSettings.DateFormat = "dd";
weekLabelSettings.DayFormat = Device.OnPlatform("EEEE", "EEEE", "dddd");
weekViewSettings.WeekLabelSettings = weekLabelSettings;
schedule.WeekViewSettings = weekViewSettings;
```



#### *ViewHeader Tapped Event*

You can handle single tap action of ViewHeader by using [ViewHeaderTapped](#) event of **SfSchedule**. This event will be triggered when the ViewHeader is Tapped. This event contains [ViewHeaderTappedEventArgs](#) argument which holds [DateTime](#) details in it.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule"
ScheduleView="WeekView"
ViewHeaderTapped="Handle_ViewHeaderTapped">
</schedule:SfSchedule>
```

#### **C#**

```
//Creating new instance of Schedule
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.WeekView;
schedule.ViewHeaderTapped += Handle_ViewHeaderTapped;
```

**C#**

```
private void Handle_ViewHeaderTapped(object sender,
ViewHeaderTappedEventArgs e)
{
    var dateTime = e.DateTime;
}
```

## Change Time Interval

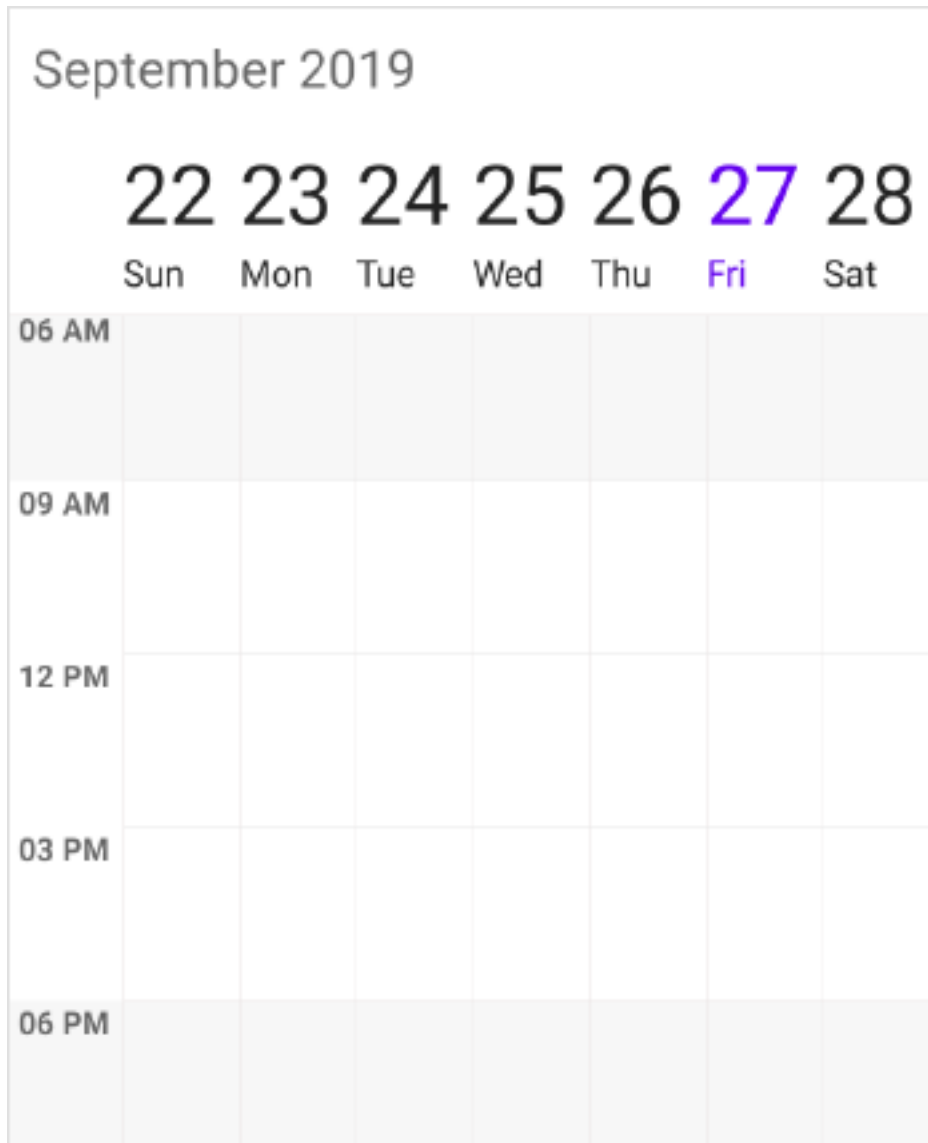
You can customize the interval of timeslots in **WeekView** by setting [TimeInterval](#) property of **SfSchedule**.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView"
TimeInterval="180"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.WeekView;
schedule.TimeInterval = 180;
```

**NOTE**

If you modify the `TimeInterval` value (in minutes), you need to change the time labels format by setting the `TimeFormat` value as "hh:mm". By default, `TimeFormat` value is "hh a". You can refer [here](#) for changing `TimeFormat` value.

**Change Time Interval Height**

You can customize the interval height of timeslots in `WeekView` by setting [TimeIntervalHeight](#) property of `SfSchedule`.

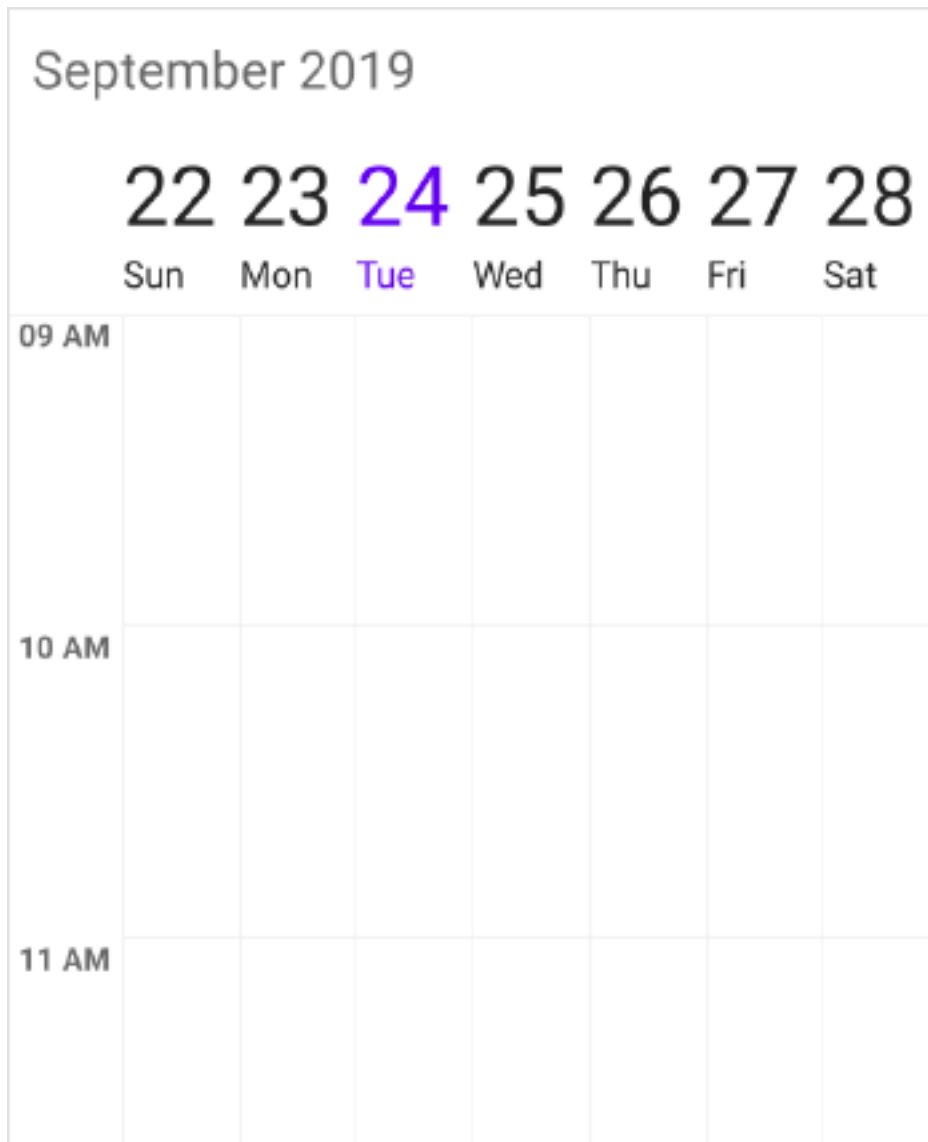
**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView"
    TimeIntervalHeight="180"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.WeekView;
```

```
schedule.TimeIntervalHeight = 180;
```



#### Full screen scheduler

Schedule time interval height can be adjusted based on screen height by changing the value of `TimeIntervalHeight` property to -1. It will auto-fit to the screen height and width.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView"
TimeIntervalHeight="-1"/>
```

#### C#

```
schedule.ScheduleView = ScheduleView.WeekView;
schedule.TimeIntervalHeight = -1;
```

### Change Working hours

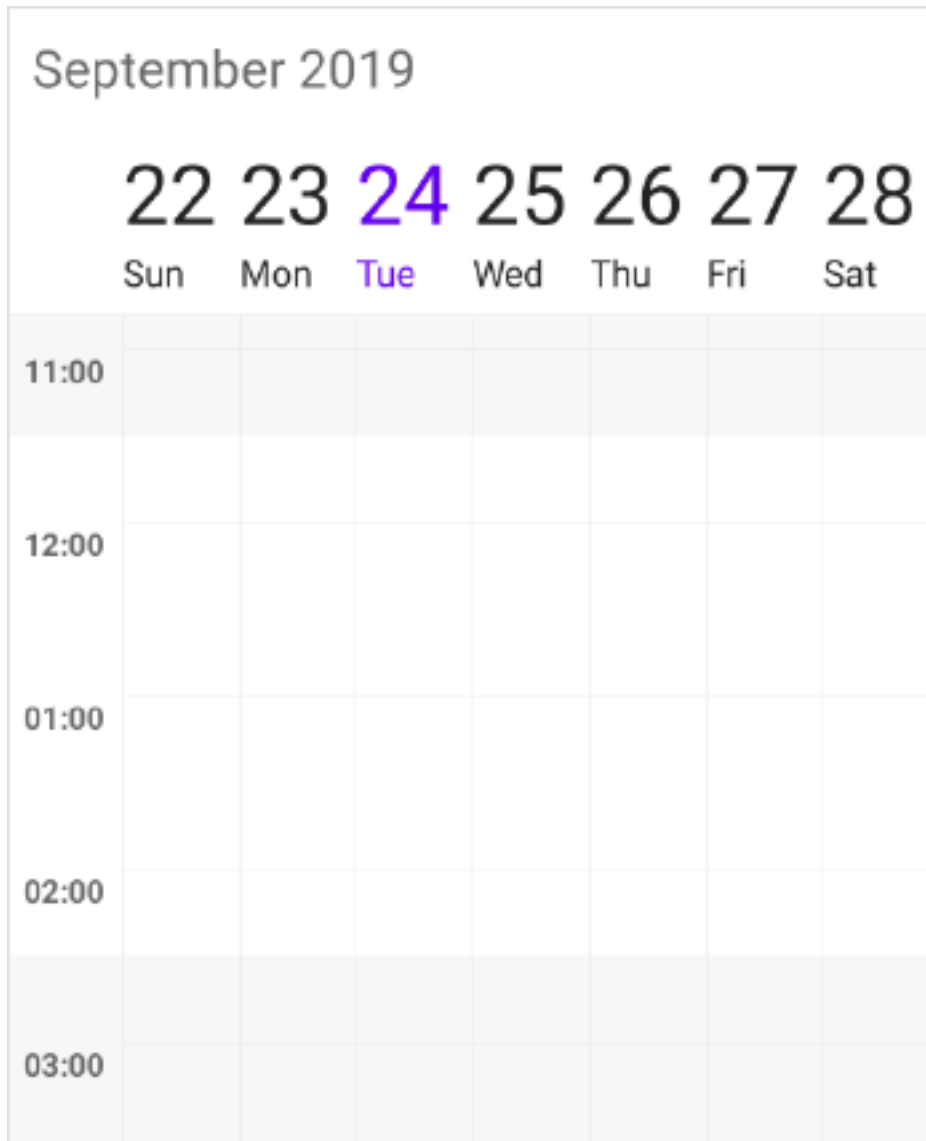
Working hours in **WeekView** of Schedule control will be differentiated with non-working hours by separate color. By default, working hours will be between 09 to 18. You can customize the working hours by setting [WorkStartHour](#) and [WorkEndHour](#) properties of [WeekViewSettings](#). You can also customize the working hours along with minutes by setting double value which will be converted to time.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView">
  <schedule:SfSchedule.WeekViewSettings>
    <!--setting working hours properties -->
    <schedule:WeekViewSettings
      WorkStartHour="11.5"
      WorkEndHour="17.5">
      <schedule:WeekViewSettings.WeekLabelSettings>
        <schedule:WeekLabelSettings TimeFormat="hh:mm" />
      </schedule:WeekViewSettings.WeekLabelSettings>
    </schedule:WeekViewSettings>
  </schedule:SfSchedule.WeekViewSettings>
</schedule:SfSchedule>
```

#### C#

```
schedule.ScheduleView = ScheduleView.WeekView;
//Create new instance of WeekViewSettings
WeekViewSettings weekViewSettings = new WeekViewSettings();
WeekLabelSettings weekLabelSettings = new WeekLabelSettings();
weekLabelSettings.TimeFormat = "hh:mm";
weekViewSettings.WorkStartHour = 11.5;
weekViewSettings.WorkEndHour = 14.5;
weekViewSettings.WeekLabelSettings = weekLabelSettings;
schedule.WeekViewSettings = weekViewSettings;
```

**NOTE**

No need to specify the decimal point values for `WorkStartHour` and `WorkEndHour`, if you don't want to set the minutes.

**Changing StartHour and EndHour**

Default value for `StartHour` and `EndHour` value is 0 to 24 to show all the time slots in `WeekView`. You need to set `StartHour` and `EndHour` property of `WeekView`, to show only the required time duration for end users. You can also set `StartHour` and `EndHour` in double value which will be converted to time to show required time duration in minutes.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView">
  <schedule:SfSchedule.WeekViewSettings>
    <!--setting working hours properties -->
    <schedule:WeekViewSettings
      StartHour="07.5"
```



```
EndHour="18.5">
<schedule:WeekViewSettings.WeekLabelSettings>
<schedule:WeekLabelSettings TimeFormat="hh:mm" />
</schedule:WeekViewSettings.WeekLabelSettings>
</schedule:WeekViewSettings>
</schedule:SfSchedule.WeekViewSettings>
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.WeekView;
//Create new instance of WeekViewSettings
WeekViewSettings weekViewSettings = new WeekViewSettings();
WeekLabelSettings weekLabelSettings = new
WeekLabelSettings(); weekLabelSettings.TimeFormat = "hh:mm";
weekViewSettings.StartHour = 07.5;
weekViewSettings.EndHour = 18.5;
weekViewSettings.WeekLabelSettings = weekLabelSettings;
schedule.WeekViewSettings = weekViewSettings;
```

September 2019						
22	23	24	25	26	27	28
Sun	Mon	Tue	Wed	Thu	Fri	Sat
08:30						
09:30						
10:30						
11:30						

**NOTE**

- **StartHour** must be greater than or equal to 0 and **EndHour** must be lesser than or equal to 24, otherwise **InvalidDataException** will be thrown.
- **EndHour** value must be greater than **StartHour**, otherwise **InvalidDataException** will be thrown.
- Schedule UI such as Appointments and NonAccessibleBlocks 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 given **TimeInterval** should result in integer value (total minutes % **TimeInterval** = 0), otherwise next immediate time interval that result in integer value when divide total minutes of a day will be

considered. For example, if `TimeInterval="135"` (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 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" ( $555 \% 37 = 0$ ) by considering (total minutes % `TimeInterval` = 0); it will return integer value for time slots rendering.

### Timeslot Appearance

You can customize the appearance of timeslots in `WeekView`.

- [Timeslot customization in Work hours](#)
- [Timeslot customization in Non Working hours](#)

#### *Timeslot customization in Work hours*

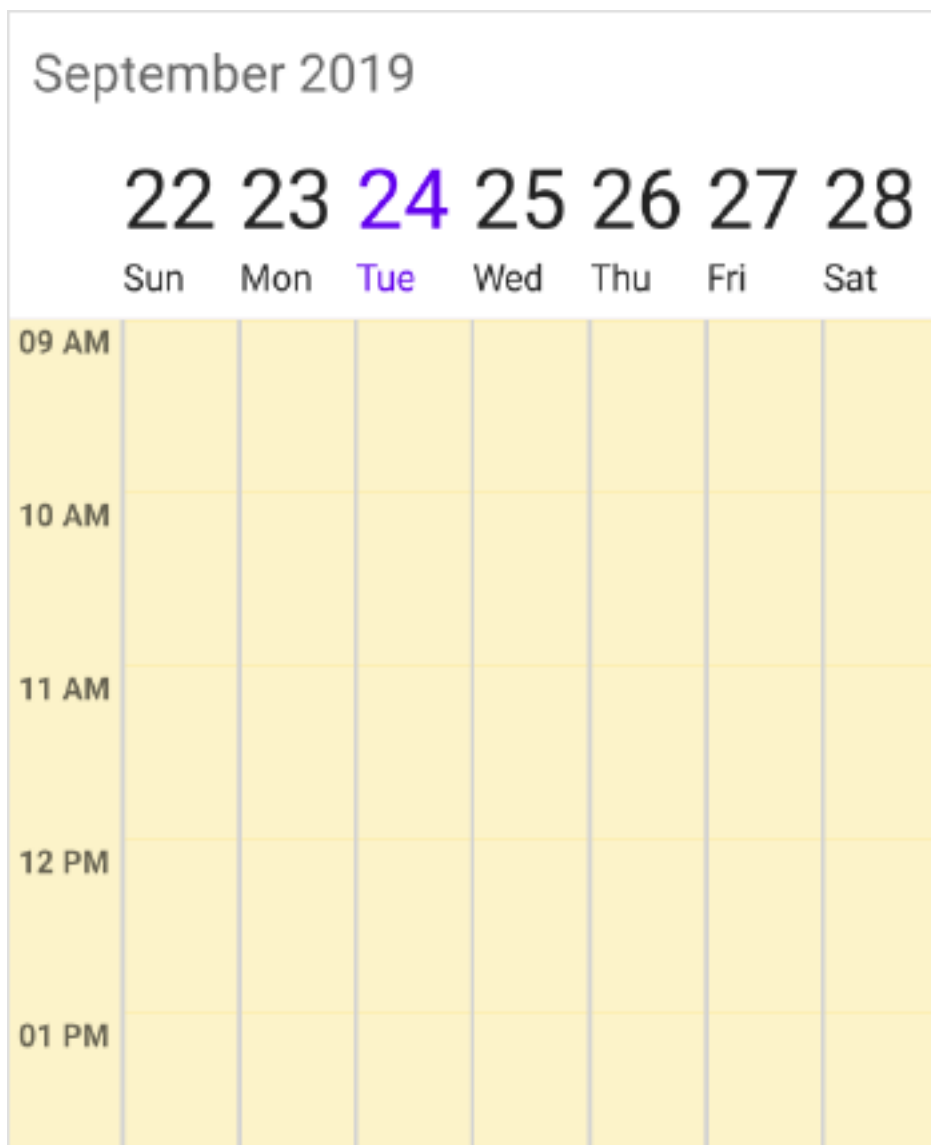
You can customize the appearance of the working hour timeslots by its color using [TimeSlotColor](#), [TimeSlotBorderColor](#), [VerticalLineStrokeWidth](#), [VerticalLineColor](#) and [TimeSlotBorderStrokeWidth](#) properties of `WeekViewSettings`.

### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView">
  <schedule:SfSchedule.WeekViewSettings>
    <!--setting week view settings properties -->
    <schedule:WeekViewSettings
      TimeSlotColor="#fcf3c9"
      TimeSlotBorderColor="#fceb9f"
      TimeSlotBorderStrokeWidth="5"
      VerticalLineStrokeWidth="5"
      VerticalLineColor="LightGray">
    </schedule:WeekViewSettings>
  </schedule:SfSchedule.WeekViewSettings>
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.WeekView;
//Create new instance of WeekViewSettings
WeekViewSettings weekViewSettings = new WeekViewSettings();
weekViewSettings.TimeSlotBorderColor = Color.FromHex("#fceb9f");
weekViewSettings.VerticalLineColor = Color.LightGray;
weekViewSettings.TimeSlotColor = Color.FromHex("#fcf3c9");
weekViewSettings.TimeSlotBorderStrokeWidth = 5;
weekViewSettings.VerticalLineStrokeWidth = 5;
schedule.WeekViewSettings = weekViewSettings;
```



#### *Timeslot customization in Non Working hours*

You can customize the appearance of the non-working hour timeslots by its color using [NonWorkingHoursTimeSlotBorderColor](#), [NonWorkingHoursTimeSlotColor](#), [VerticalLineStrokeWidth](#), [VerticalLineColor](#) and [TimeSlotBorderStrokeWidth](#) properties of [WeekViewSettings](#).

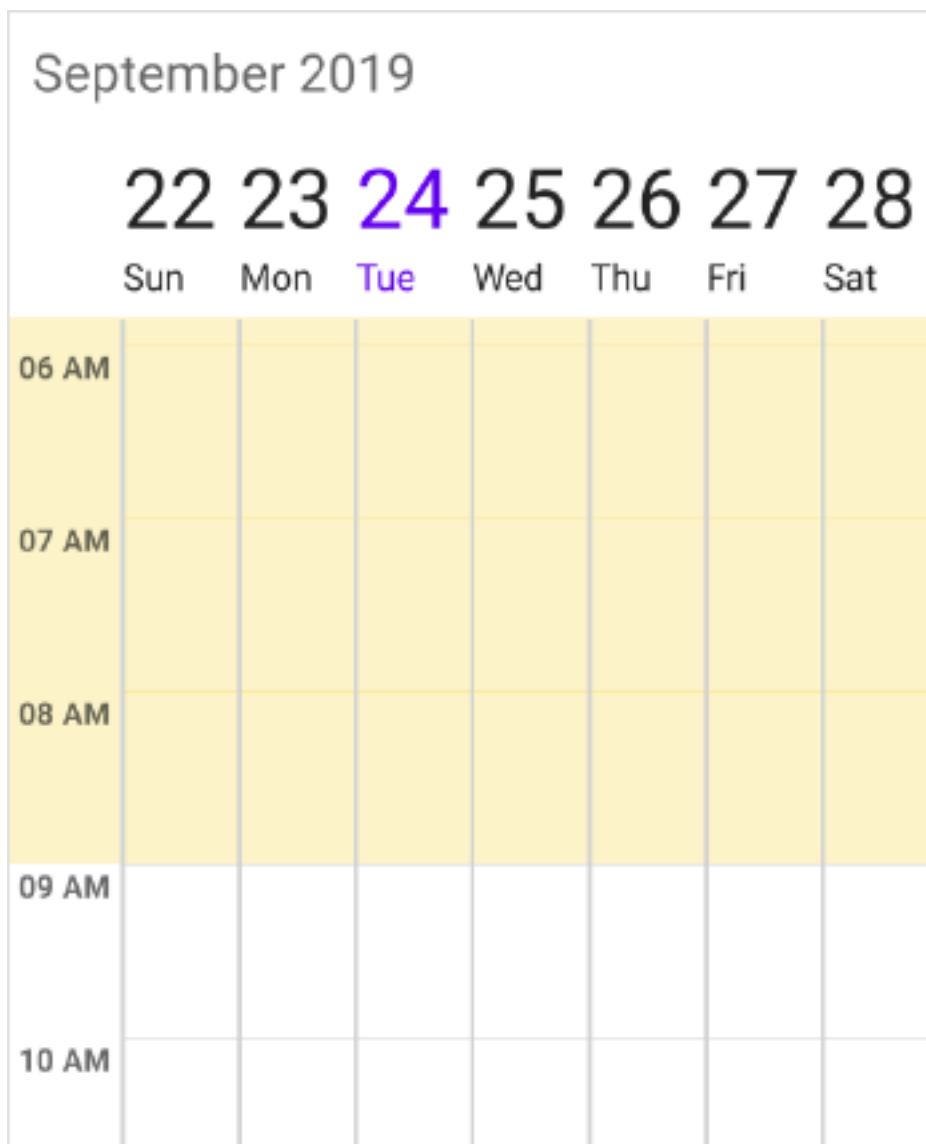
#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView">
  <schedule:SfSchedule.WeekViewSettings>
    <!--setting week view settings properties -->
    <schedule:WeekViewSettings
      NonWorkingHoursTimeSlotColor="#f3cf3c"
      NonWorkingHoursTimeSlotBorderColor="#fce94f"
      TimeSlotBorderStrokeWidth="5"
      VerticalLineStrokeWidth="5"
      VerticalLineColor="LightGray">
    </schedule:WeekViewSettings>
  </schedule:SfSchedule.WeekViewSettings>
```

```
</schedule:SfSchedule>
```

**C#**

```
schedule.ScheduleView = ScheduleView.WeekView;
//Create new instance of WeekViewSettings
WeekViewSettings weekViewSettings = new WeekViewSettings();
weekViewSettings.NonWorkingHoursTimeSlotBorderColor =
Color.FromHex("#fceb9f");
weekViewSettings.VerticalLineColor = Color.LightGray;
weekViewSettings.NonWorkingHoursTimeSlotColor = Color.FromHex("#fcf3c9");
weekViewSettings.TimeSlotBorderStrokeWidth = 5;
weekViewSettings.VerticalLineStrokeWidth = 5;
schedule.WeekViewSettings = weekViewSettings;
```

**NOTE**

TimeSlotBorderStrokeWidth and VerticalLineStrokeWidth properties are common to both Working hours and Non-Working hour time slot customization.

### Non-Accessible timeslots

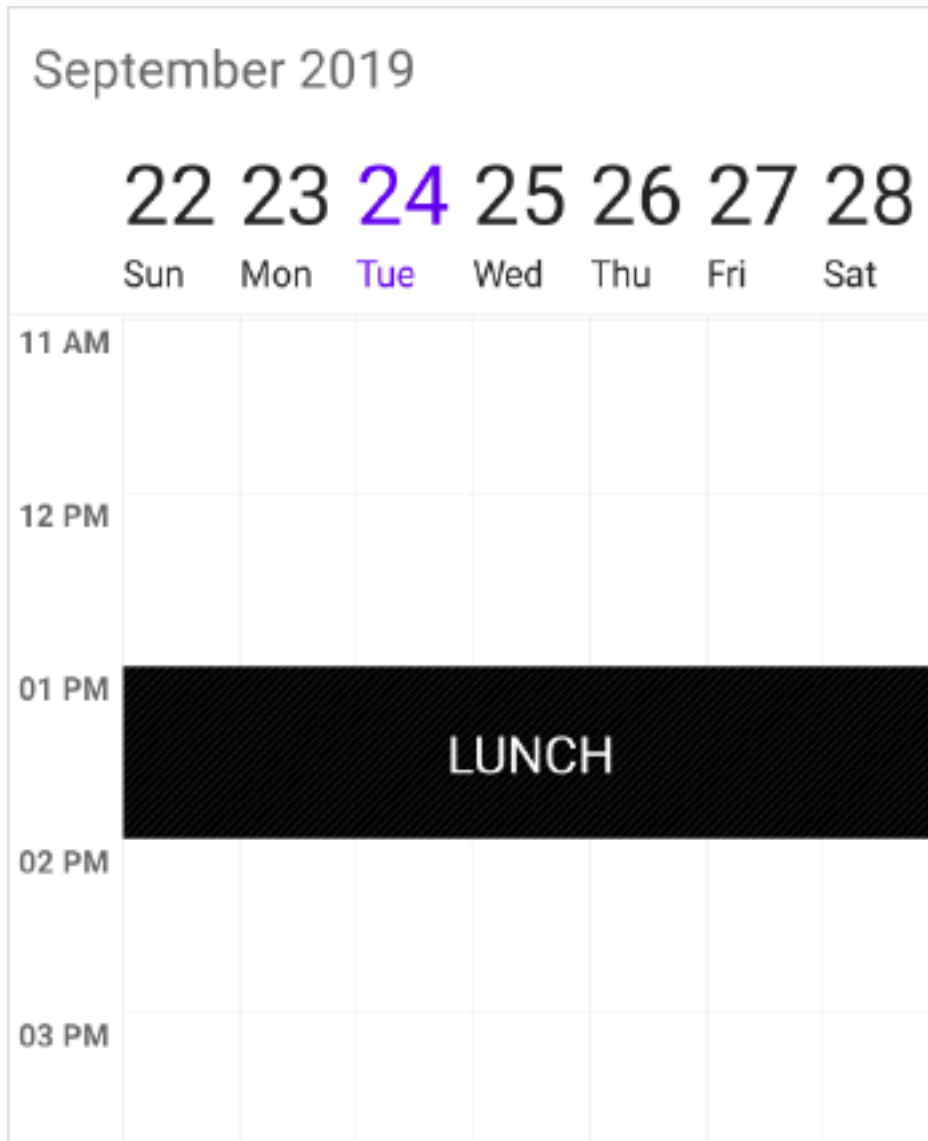
You can restrict or allocate certain timeslot as Non-accessible blocks by using [NonAccessibleBlocks](#) of [WeekViewSettings](#) so that you can allocate those timeslots for predefined events/activities like Lunch hour.

### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView">
  <!--setting non-accessing blocks-->
  <schedule:SfSchedule.WeekViewSettings>
    <schedule:WeekViewSettings>
      <schedule:WeekViewSettings.NonAccessibleBlocks>
        <schedule:NonAccessibleBlock
          StartTime="13"
          EndTime="14"
          Text="LUNCH"
          Color="Black" />
      </schedule:WeekViewSettings.NonAccessibleBlocks>
    </schedule:WeekViewSettings>
  </schedule:SfSchedule.WeekViewSettings>
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.WeekView;
//Create new instance of NonAccessibleBlock
NonAccessibleBlock nonAccessibleBlock = new NonAccessibleBlock();
//Create new instance of NonAccessibleBlocksCollection
NonAccessibleBlocksCollection nonAccessibleBlocksCollection = new
NonAccessibleBlocksCollection();
WeekViewSettings weekViewSettings = new WeekViewSettings();
nonAccessibleBlock.StartTime = 13;
nonAccessibleBlock.EndTime = 14;
nonAccessibleBlock.Text = "LUNCH";
nonAccessibleBlock.Color = Color.Black;
nonAccessibleBlocksCollection.Add(nonAccessibleBlock);
weekViewSettings.NonAccessibleBlocks = nonAccessibleBlocksCollection;
schedule.WeekViewSettings = weekViewSettings;
```

**NOTE**

Selection and related events will not be working in this blocks.

Change first day of week

By default, schedule control will be rendered with Sunday as the first day of the week, it can be customized to any day of the week by using [FirstDayOfWeek](#) property of [SfSchedule](#).

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView"
FirstDayOfWeek="3"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.WeekView;
schedule.FirstDayOfWeek = 3;
```

September 2019						
24	25	26	27	28	29	30
Tue	Wed	Thu	Fri	Sat	Sun	Mon
09 AM						
10 AM						
11 AM						
12 PM						
01 PM						

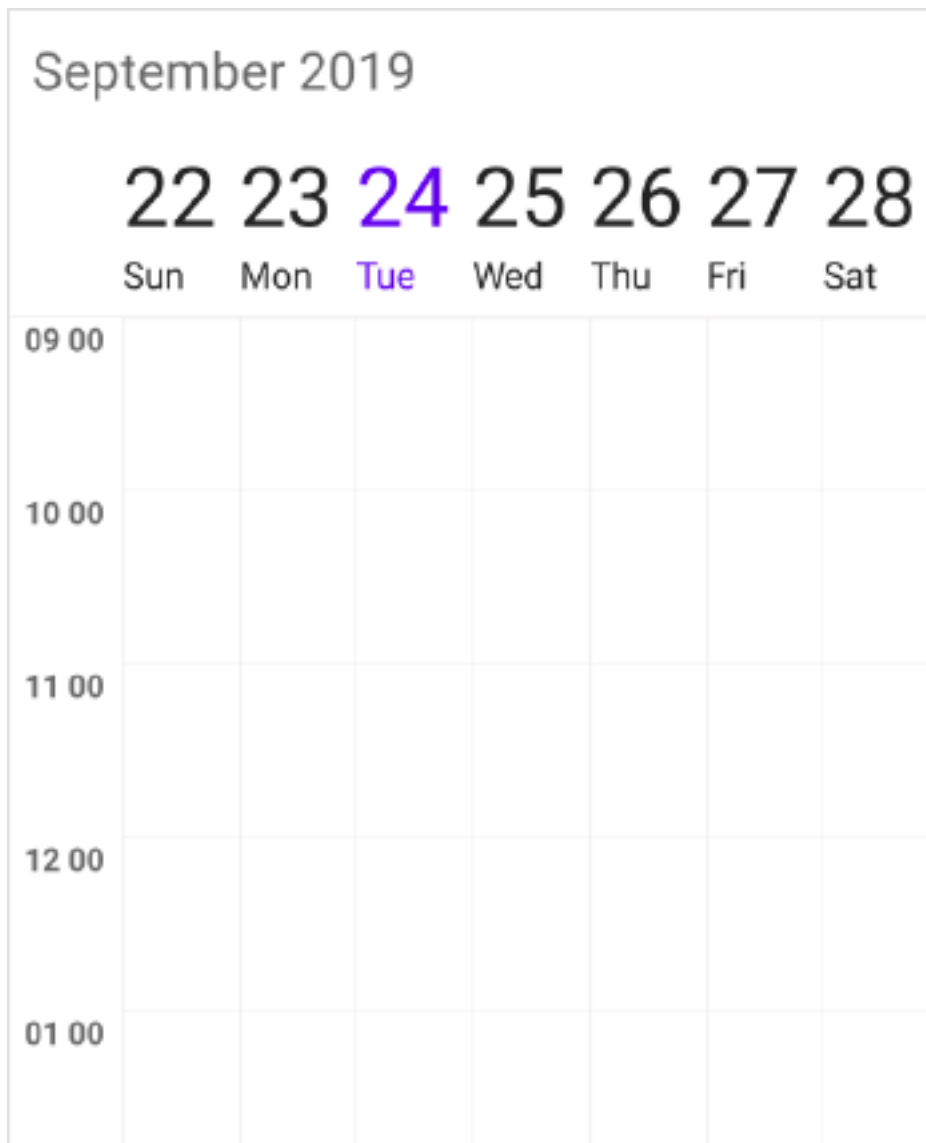
### Time Label Formatting

You can customize the format for the labels which are mentioning the time, by setting [TimeFormat](#) property of [WeekLabelSettings](#) in [WeekViewSettings](#).

### C#

```
schedule.ScheduleView = ScheduleView.WeekView;  
WeekViewSettings weekViewSettings = new WeekViewSettings();  
WeekLabelSettings weekLabelSettings = new WeekLabelSettings();  
weekLabelSettings.TimeFormat = "hh mm";  
weekViewSettings.WeekLabelSettings = weekLabelSettings;  
schedule.WeekViewSettings = weekViewSettings;
```





### Time Label Appearance

You can customize the color for the labels which are mentioning the time, by setting [TimeLabelColor](#) property of [WeekLabelSettings](#) in [WeekViewSettings](#).

### C#

```
schedule.ScheduleView = ScheduleView.WeekView;  
//Create new instance of WeekViewSettings  
WeekViewSettings weekViewSettings = new WeekViewSettings();  
//Create new instance of WeekLabelSettings  
WeekLabelSettings weekLabelSettings = new WeekLabelSettings();  
weekLabelSettings.TimeLabelColor = Color.FromHex("#8282ff");  
weekViewSettings.WeekLabelSettings = weekLabelSettings;  
schedule.WeekViewSettings = weekViewSettings;
```

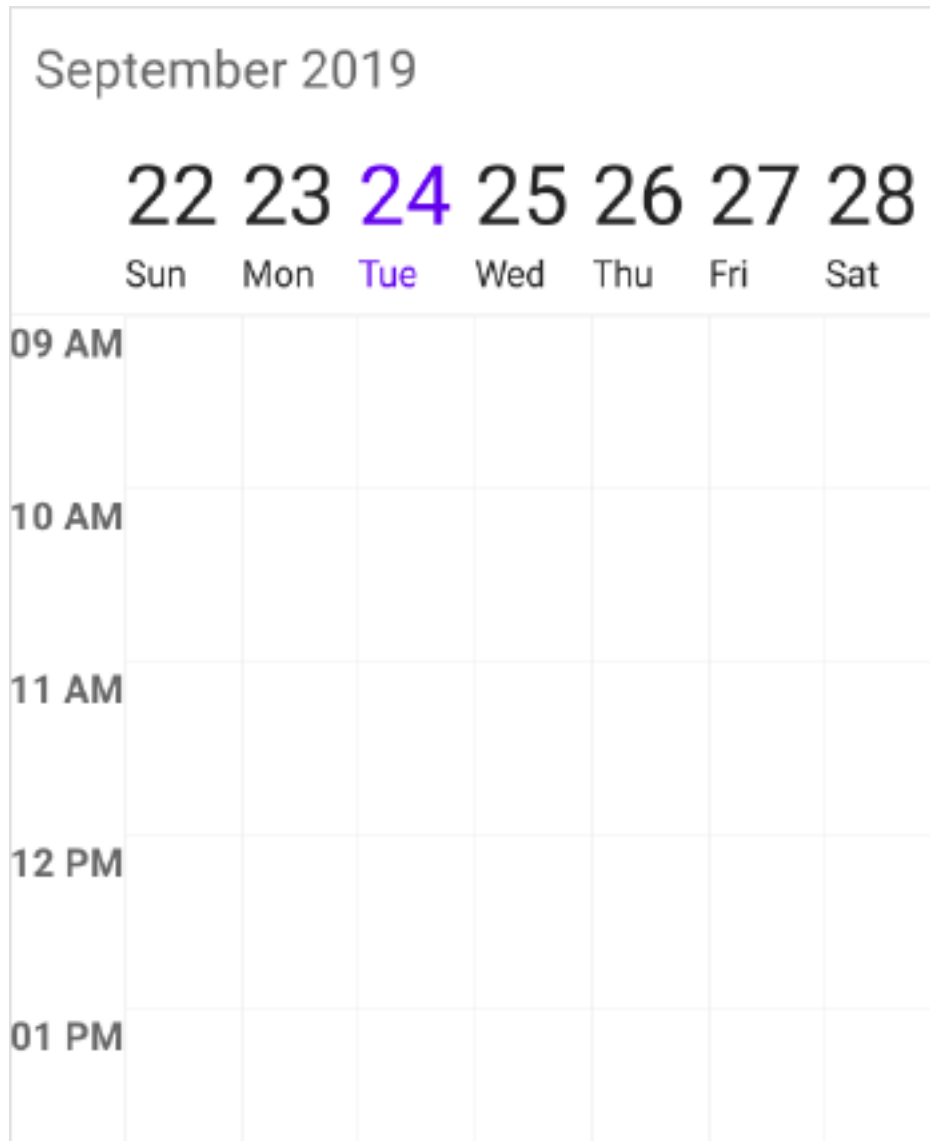
September 2019						
22	23	24	25	26	27	28
Sun	Mon	Tue	Wed	Thu	Fri	Sat
09 AM						
10 AM						
11 AM						
12 PM						
01 PM						

### Time Label Size

You can customize the size of the labels which are mentioning the time, by setting [TimeLabelSize](#) property of `WeekLabelSettings` in `WeekViewSettings`.

### C#

```
schedule.ScheduleView = ScheduleView.WeekView;  
//Create new instance of WeekViewSettings  
WeekViewSettings weekViewSettings = new WeekViewSettings();  
//Create new instance of WeekLabelSettings  
WeekLabelSettings weekLabelSettings = new WeekLabelSettings();  
//Customizing the size of the time label  
weekLabelSettings.TimeLabelSize = 15;  
weekViewSettings.WeekLabelSettings = weekLabelSettings;  
schedule.WeekViewSettings = weekViewSettings;
```



### Selection

You can customize the default appearance of selection UI in the timeslots.

- [Selection customization using style](#)
- [Selection customization using custom View](#)
- [Programmatic selection](#)

### *Selection customization using style*

You can customize the timeslot selection by using [SelectionStyle](#) property of `SfSchedule`.

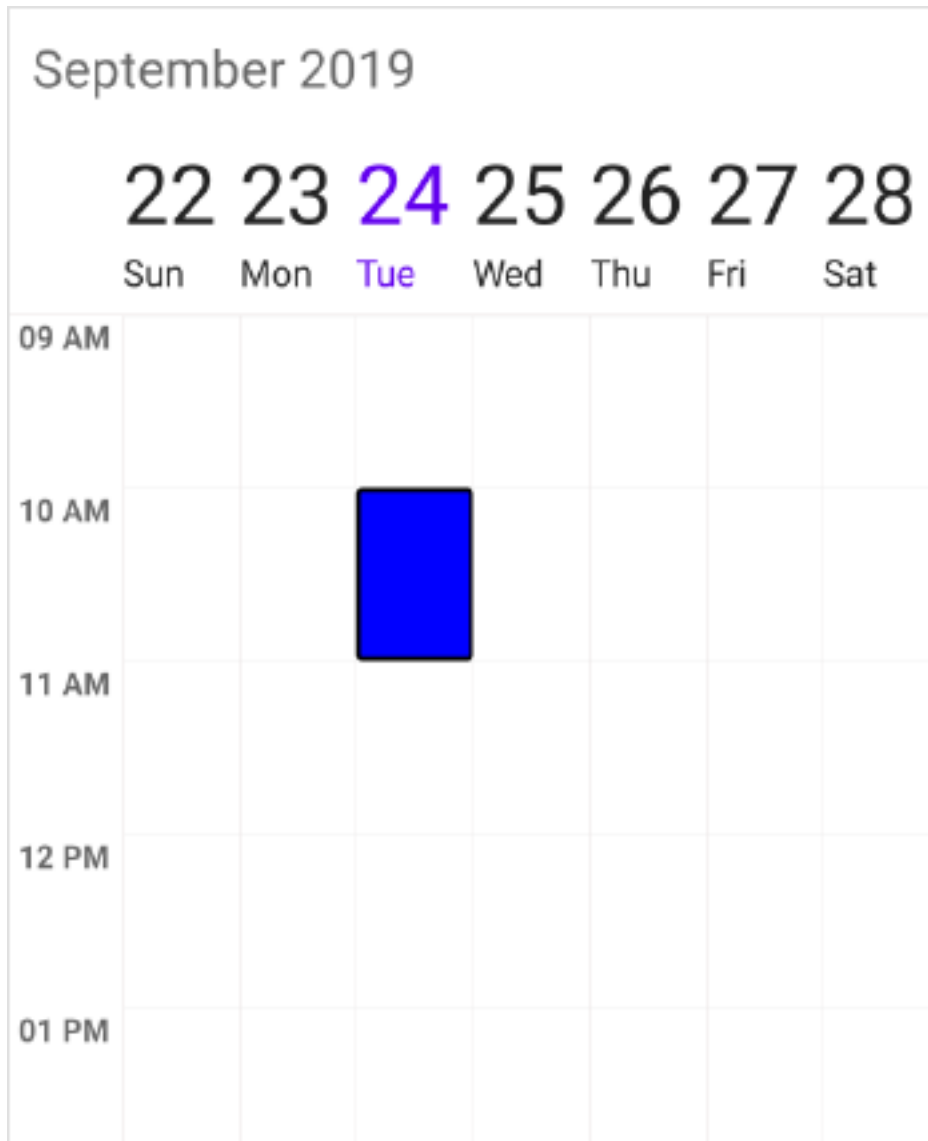
### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView">
  <schedule:SfSchedule.SelectionStyle>
    <schedule:SelectionStyle
      BackgroundColor="Blue"
      BorderColor="Black"
    />
  />
</schedule:SfSchedule>
```

```
BorderThickness="5"  
BorderCornerRadius="5">  
</schedule:SelectionStyle>  
</schedule:SfSchedule.SelectionStyle>  
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.WeekView;  
//Create new instance of SelectionStyle  
SelectionStyle selectionStyle = new SelectionStyle();  
selectionStyle.BackgroundColor = Color.Blue;  
selectionStyle.BorderColor = Color.Black;  
selectionStyle.BorderThickness = 5;  
selectionStyle.BorderCornerRadius = 5;  
schedule.SelectionStyle = selectionStyle;
```



### *Selection customization using custom View*

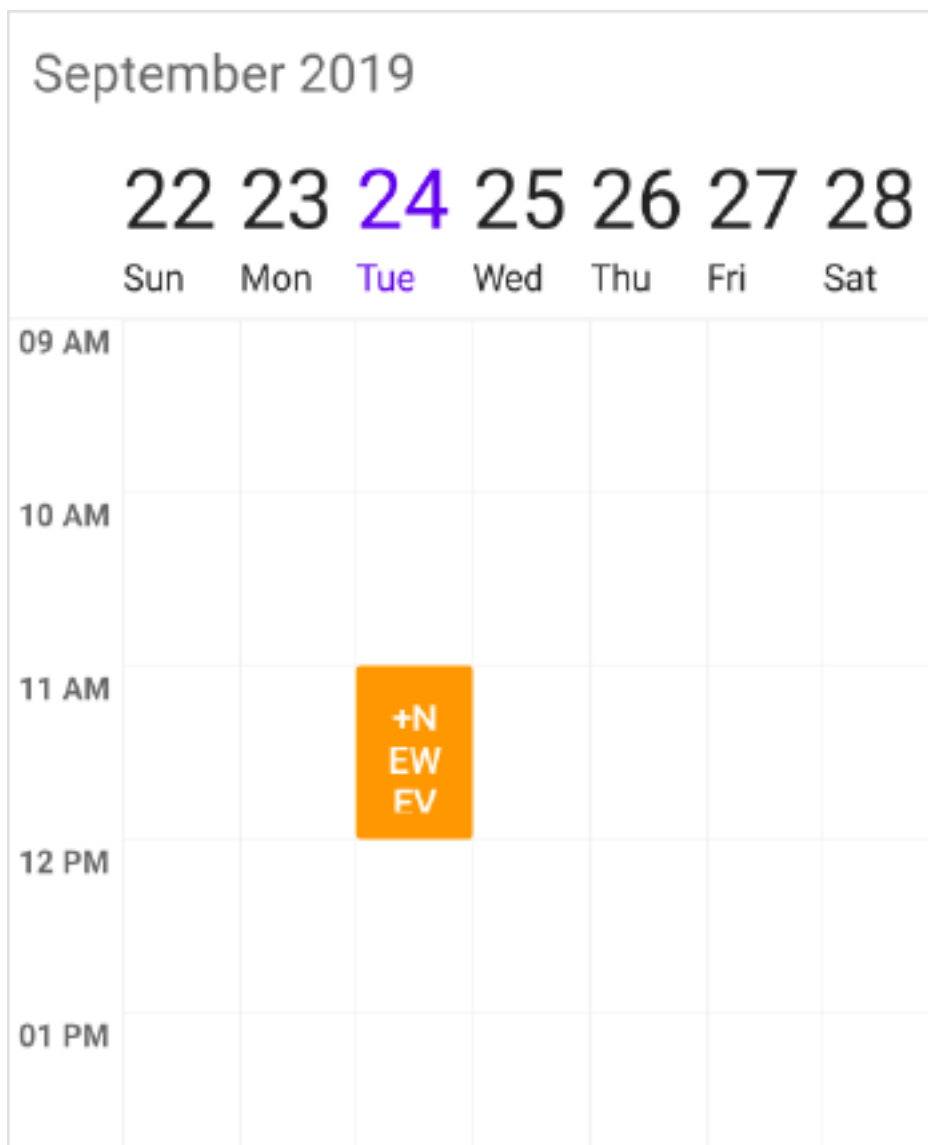
You can replace the default selection UI with your custom view by setting [SelectionView](#) property of SfSchedule.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView">
  <schedule:SfSchedule.SelectionView>
    <Button
      BackgroundColor="#FF9800"
      Text="+NewEvent"
      TextColor="White"/>
  </schedule:SfSchedule.SelectionView>
</schedule:SfSchedule>
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.WeekView;
//Add the CustomView
Button customView = new Button();
customView.Text = "+NewEvent";
customView.BackgroundColor = Color.FromHex("#FF9800");
customView.TextColor = Color.White;
schedule.SelectionView = customView;
```



#### Programmatic selection

You can programmatically select the specific timeslot by setting corresponding date and time value to [SelectedDate](#) property of [SfSchedule](#). By default, it is null.

#### C#

```
// Setting a date and time to select  
schedule.SelectedDate = new DateTime(2017, 10, 04, 10, 0, 0);
```

You can clear the selection by setting [SelectedDate](#) as null.

#### C#

```
// Setting null value to deselect  
schedule.SelectedDate = null;
```

You can download the entire source code of this demo for Xamarin.Forms from here [Date Selection](#)

**NOTE**

- SfSchedule does not support multiple selection.
- SfSchedule supports two-way binding of SelectedDate property.

October 2017						
	1	2	3	4	5	6
	Sun	Mon	Tue	Wed	Thu	Fri
10 AM						
11 AM						
12 PM						
01 PM						
02 PM						

[WorkWeekView](#)

WorkWeekView is to view only working days of a particular week. By default, Saturday and Sunday are the non-working days. You can be customize it with any days of a Week. Appointments arranged in timeslots based on its duration with respective day of the week.

[ViewHeader Appearance](#)

You can customize the default appearance of view header in [WorkWeekView](#) by using [ViewHeaderStyle](#) property of [SfSchedule](#).

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView ="WorkWeekView">
```

```
<schedule:SfSchedule.ViewHeaderStyle>
<schedule:ViewHeaderStyle
  BackgroundColor="#009688"
  DayTextColor="#FFFFFF"
  DateTextColor="#FFFFFF"
  DayFontFamily="Arial"
  DateFontFamily="Arial">
</schedule:ViewHeaderStyle>
</schedule:SfSchedule.ViewHeaderStyle>
</schedule:SfSchedule>
```

## **C#**

```
//Create new instance of Schedule
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.WorkWeekView;
//Customize the schedule view header
ViewHeaderStyle viewHeaderStyle = new ViewHeaderStyle();
viewHeaderStyle.BackgroundColor = Color.FromHex("#009688");
viewHeaderStyle.DayTextColor = Color.FromHex("#FFFFFF");
viewHeaderStyle.DateTextColor = Color.FromHex("#FFFFFF");
viewHeaderStyle.DayFontFamily = "Arial";
viewHeaderStyle.DateFontFamily = "Arial";
schedule.ViewHeaderStyle = viewHeaderStyle;
```



September 2019				
16	17	18	19	20
Mon	Tue	Wed	Thu	Fri
09 AM				
10 AM				
11 AM				
12 PM				
01 PM				

**NOTE**

FontAttributes and FontFamily are native to the platform. Custom font and the font which are not available in the specified platform will not be applied.

You can customize the height of the ViewHeader in **WeekView** by setting [ViewHeaderHeight](#) property of SfSchedule.

**C#**

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
schedule.ViewHeaderHeight = 50;
```

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView"
ViewHeaderHeight="50" />
```

September 2019					
	23	24	25	26	27
09 AM					
10 AM					
11 AM					
12 PM					
01 PM					
02 PM					

#### *Customize Font Appearance*

you can change the appearance of Font by setting the [DayFontFamily](#) and [DateFontFamily](#) property of [ViewHeaderStyle](#) property in Schedule.

#### **XML**

```
<schedule:ViewHeaderStyle.DayFontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:HeaderStyle.DayFontFamily>
<schedule:ViewHeaderStyle.DateFontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:HeaderStyle.DateFontFamily>
```

#### **C#**

```
viewHeaderStyle.DayFontFamily = Device.OnPlatform("Lobster-Regular",
"Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
viewHeaderStyle.DateFontFamily = Device.OnPlatform("Lobster-Regular",
"Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
```

September 2019					
	23	24	25	26	27
	Mon	Tue	Wed	Thu	Fri
09 AM					
10 AM					
11 AM					
12 PM					
01 PM					

Refer [this](#) to configure the custom fonts in Xamarin.Forms.

#### *ViewHeader Date Format*

You can customize the date and day format of SfSchedule ViewHeader by using [DateFormat](#) and [DayFormat](#) properties of [WorkWeekLabelSettings](#).

#### **XML**

```
<schedule:SfSchedule>
  <schedule:SfSchedule.WorkWeekViewSettings>
    <schedule:WorkWeekViewSettings>
      <schedule:WorkWeekViewSettings.WorkWeekLabelSettings>
        <schedule:WorkWeekLabelSettings DateFormat="dd">
```

```
<schedule:WorkWeekLabelSettings.DayFormat>
<OnPlatform x:TypeArguments="x:String" iOS="EEEE" Android="EEEE"
WinPhone="dddd" />
</schedule:WorkWeekLabelSettings.DayFormat>
</schedule:WorkWeekLabelSettings>
</schedule:WorkWeekViewSettings.WorkWeekLabelSettings>
</schedule:WorkWeekViewSettings>
</schedule:SfSchedule.WorkWeekViewSettings>
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
//Creating new instance of WorkWeekViewSettings
WorkWeekViewSettings workWeekViewSettings = new WorkWeekViewSettings();
//Creating new instance of WorkWeekLabelSettings
WorkWeekLabelSettings workWeekLabelSettings = new WorkWeekLabelSettings();
//Customizing date format
workWeekLabelSettings.DateFormat = "dd";
workWeekLabelSettings.DayFormat = Device.OnPlatform("EEEE", "EEEE", "dddd");
workWeekViewSettings.WorkWeekLabelSettings = workWeekLabelSettings;
schedule.WorkWeekViewSettings = workWeekViewSettings;
```

September 2019					
<div> <div>23</div> <div>24</div> <div>25</div> <div>26</div> <div>27</div> </div> <div> Monday Tuesday Wednesday Thursday Friday </div>					
09 AM					
10 AM					
11 AM					
12 PM					
01 PM					

#### *ViewHeader Tapped Event*

You can handle single tap action of ViewHeader by using [ViewHeaderTapped](#) event of `SfSchedule`. This event will be triggered when the ViewHeader is Tapped. This event contains [ViewHeaderTappedEventArgs](#) argument which holds [DateTime](#) details in it.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule"
ScheduleView="WorkWeekView"
ViewHeaderTapped="Handle_ViewHeaderTapped" >
</schedule:SfSchedule>
```

#### **C#**

```
//Creating new instance of Schedule
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.WorkWeekView;
schedule.ViewHeaderTapped += Handle_ViewHeaderTapped;
```

**C#**

```
private void Handle_ViewHeaderTapped(object sender,
ViewHeaderTappedEventArgs e)
{
    var dateTime = e.DateTime;
}
```

**Change Time Interval**

You can customize the interval of timeslots in **WorkWeekView** by setting [TimeInterval](#) property of **SfSchedule**.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView"
TimeInterval="180"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
schedule.TimeInterval = 180;
```

September 2019					
	23	24	25	26	27
	Mon	Tue	Wed	Thu	Fri
09 AM					
12 PM					
03 PM					
06 PM					
09 PM					

**NOTE**

If you modify the `TimeInterval` value (in minutes), you need to change the time labels format by setting the `TimeFormat` value as "hh:mm". By default, `TimeFormat` value is "hh a". You can refer [here](#) for changing `TimeFormat` value.

**Change Time Interval Height**

You can customize the interval height of timeslots in `WorkWeekView` by setting `TimeIntervalHeight` property of `SfSchedule`.

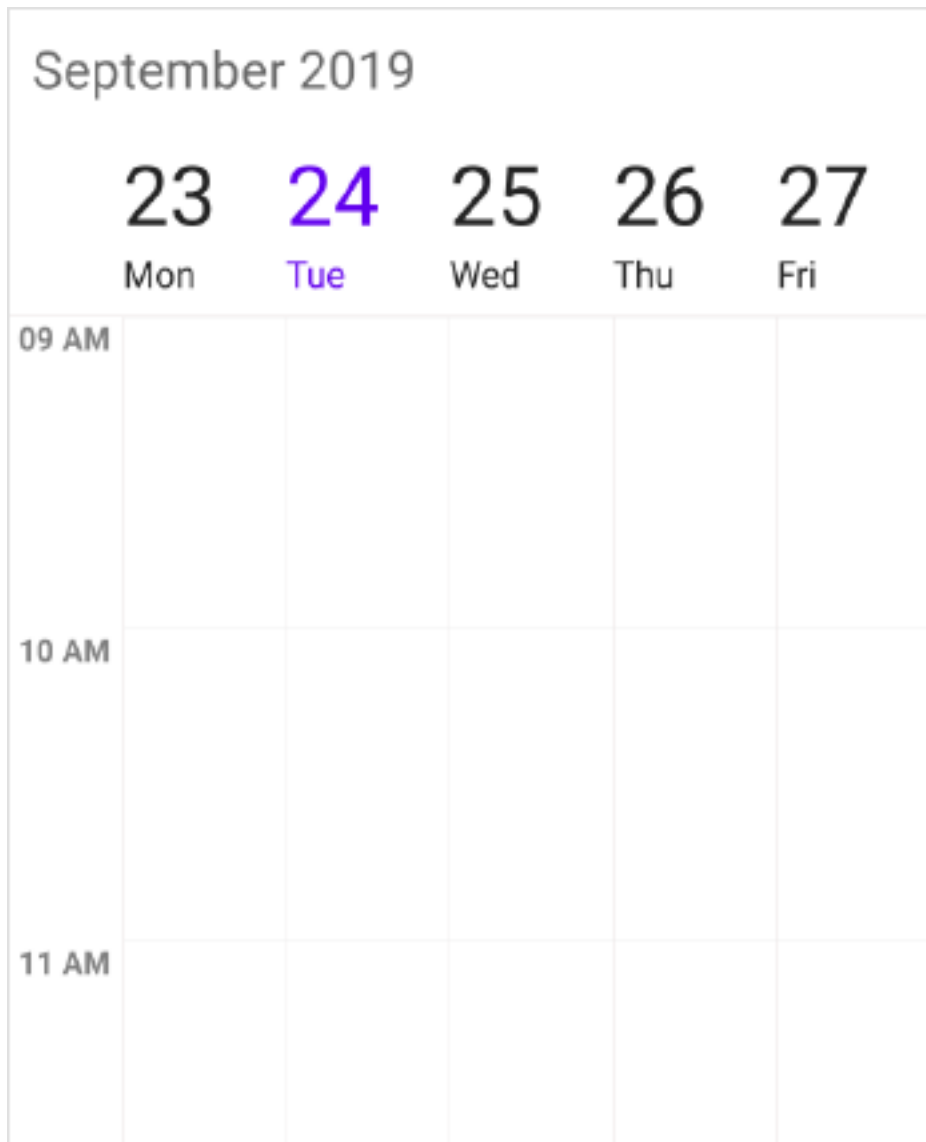
**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView"
    TimeIntervalHeight="180"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
```

```
schedule.TimeIntervalHeight = 180;
```



#### *Full screen scheduler*

Schedule time interval height can be adjusted based on screen height by changing the value of `TimeIntervalHeight` property to -1. It will auto-fit to the screen height and width.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView"
TimeIntervalHeight="-1"/>
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
schedule.TimeIntervalHeight = -1;
```



### Change Working hours

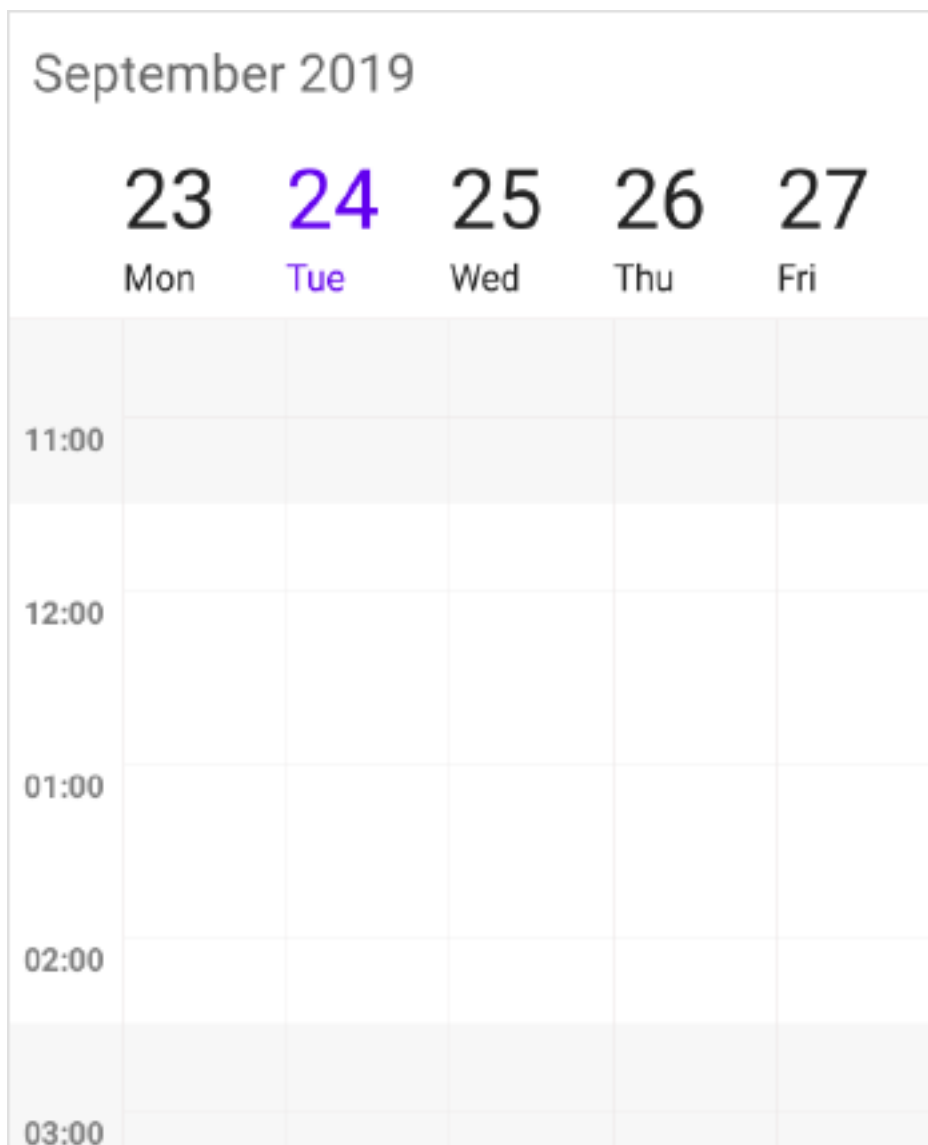
Working hours in **WorkWeekView** of Schedule control will be differentiated with non-working hours by separate color. By default, working hours will be between 09 to 18. You can customize the working hours by setting [WorkStartHour](#) and [WorkEndHour](#) properties of [WorkWeekViewSettings](#). You can also customize the working hours along with minutes by setting double value which will be converted to time.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView">
  <schedule:SfSchedule.WorkWeekViewSettings>
    <!--setting working hours properties -->
    <schedule:WorkWeekViewSettings
      WorkStartHour="11.5"
      WorkEndHour="17.5">
      <schedule:WorkWeekViewSettings.WorkWeekLabelSettings>
        <schedule:WorkWeekLabelSettings TimeFormat="hh:mm" />
      </schedule:WorkWeekViewSettings.WorkWeekLabelSettings>
    </schedule:WorkWeekViewSettings>
  </schedule:SfSchedule.WorkWeekViewSettings>
</schedule:SfSchedule>
```

#### C#

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
//Create new instance of WorkWeekViewSettings
WorkWeekViewSettings workWeekViewSettings = new WorkWeekViewSettings();
WorkWeekLabelSettings workWeekLabelSettings = new WorkWeekLabelSettings();
workWeekLabelSettings.TimeFormat = "hh:mm";
workWeekViewSettings.WorkStartHour = 11.5;
workWeekViewSettings.WorkEndHour = 14.5;
workWeekViewSettings.WorkWeekLabelSettings = workWeekLabelSettings;
schedule.WorkWeekViewSettings = workWeekViewSettings;
```

**NOTE**

No need to specify the decimal point values for `WorkStartHour` and `WorkEndHour`, if you don't want to set the minutes.

**Changing StartHour and EndHour**

Default value for `StartHour` and `EndHour` value is 0 to 24 to show all the time slots in `WorkWeekView`. You need to set `StartHour` and `EndHour` property of 'WorkWeekView', to show only the required time duration for end users. You can also set `StartHour` and `EndHour` in double value which will be converted to time to show required time duration in minutes.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView">
  <schedule:SfSchedule.WorkWeekViewSettings>
    <!--setting working hours properties -->
    <schedule:WorkWeekViewSettings
      StartHour ="7.5"
```

```
EndHour ="18.5">
</schedule:WorkWeekViewSettings>
<schedule:WorkWeekViewSettings.WorkWeekLabelSettings>
<schedule:WorkWeekLabelSettings TimeFormat="hh:mm" />
</schedule:WorkWeekViewSettings.WorkWeekLabelSettings>
</schedule:SfSchedule.WorkWeekViewSettings>
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
//Create new instance of WorkWeekViewSettings
WorkWeekViewSettings workWeekViewSettings = new WorkWeekViewSettings();
WorkWeekLabelSettings workWeekLabelSettings = new WorkWeekLabelSettings();
workWeekLabelSettings.TimeFormat = "hh:mm";
workWeekViewSettings.StartHour = 7.5;
workWeekViewSettings.EndHour = 18.5;
workWeekViewSettings.WorkWeekLabelSettings = workWeekLabelSettings;
schedule.WorkWeekViewSettings = workWeekViewSettings;
```

September 2019					
	23	24	25	26	27
	Mon	Tue	Wed	Thu	Fri
08:30					
09:30					
10:30					
11:30					

**NOTE**

- **StartHour** must be greater than or equal to 0 and **EndHour** must be lesser than or equal to 24, otherwise **InvalidDataException** will be thrown.
- **EndHour** value must be greater than **StartHour**, otherwise **InvalidDataException** will be thrown.
- Schedule UI such as **Appointments** and **NonAccessibleBlocks** 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 given **TimeInterval** should result in integer value (total minutes % **TimeInterval** = 0), otherwise next immediate time interval that result in integer value when divide total minutes of a day will be

considered. For example, if `TimeInterval="135"` (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 given `StartHour` and `EndHour` should result in integer value, otherwise next immediate time interval 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" ( $555 \% 37 = 0$ ) by considering (total minutes % `TimeInterval` = 0); it will return integer value for time slots rendering.

### Changing NonWorking Days

You can customize the Non-Working days of a week by using [NonWorkingsDays](#) property of `WorkWeekViewSettings`.

#### C#

```
var nonWorkingDays = new ObservableCollection<DayOfWeek>();
nonWorkingDays.Add(DayOfWeek.Monday);
nonWorkingDays.Add(DayOfWeek.Friday);
var workWeekViewSettings = new WorkWeekViewSettings();
workWeekViewSettings.NonWorkingsDays = nonWorkingDays;
schedule.WorkWeekViewSettings = workWeekViewSettings;
```

### Timeslot Appearance

You can customize the appearance of timeslots in `WeekView`.

- [Timeslot customization in Work hours](#)
- [Timeslot customization in Non Working hours](#)

#### *Timeslot customization in Work hours*

You can customize the appearance of the working hour timeslots by its color using [TimeSlotColor](#), [TimeSlotBorderColor](#), [VerticalLineStrokeWidth](#), [VerticalLineColor](#) and [TimeSlotBorderStrokeWidth](#) properties of `WorkWeekViewSettings`.

#### XML

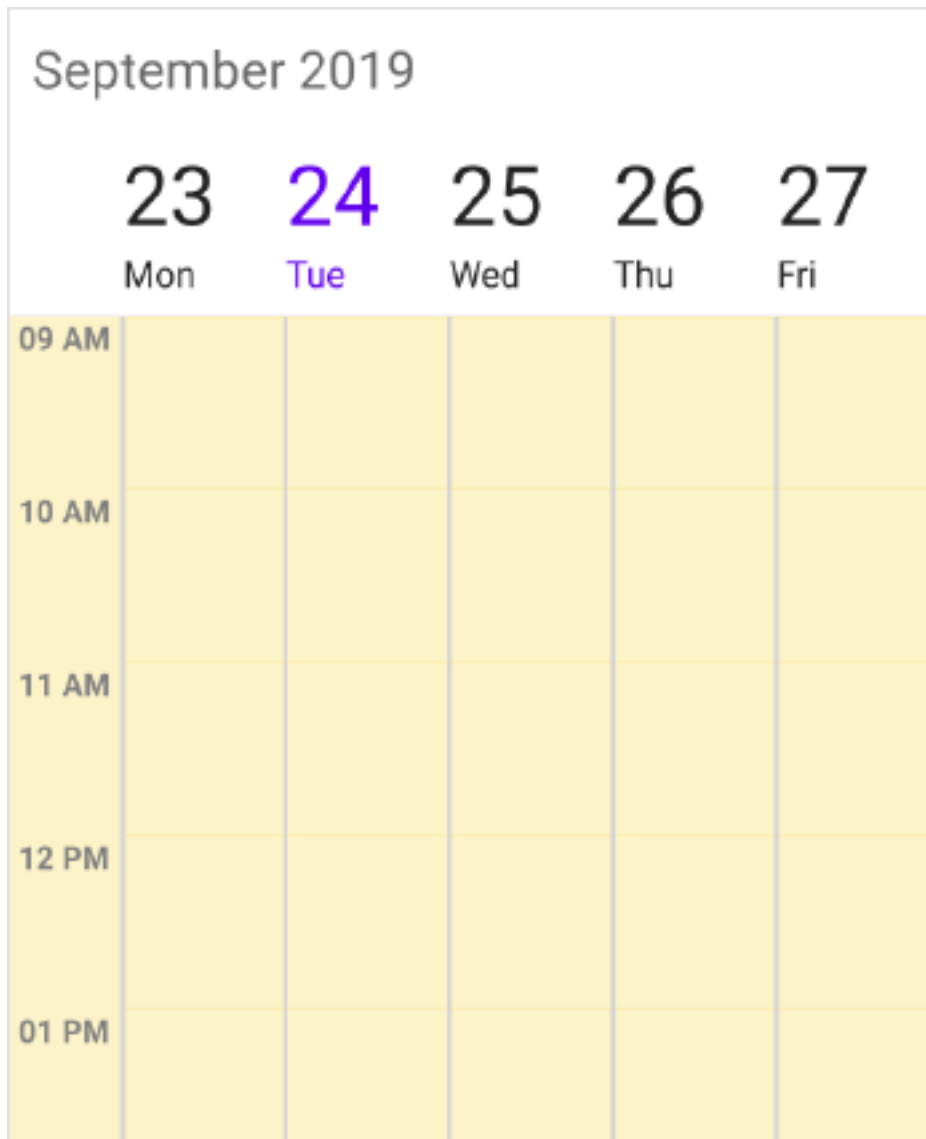
```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView">
  <schedule:SfSchedule.WorkWeekViewSettings>
    <!--setting work week view settings properties -->
    <schedule:WorkWeekViewSettings
      TimeSlotColor="#fcf3c9"
      TimeSlotBorderColor="#fceb9f"
      TimeSlotBorderStrokeWidth="5"
      VerticalLineStrokeWidth="5"
      VerticalLineColor="LightGray">
    </schedule:WorkWeekViewSettings>
  </schedule:SfSchedule.WorkWeekViewSettings>
</schedule:SfSchedule>
```

**C#**

```

schedule.ScheduleView = ScheduleView.WorkWeekView;
//Create new instance of WorkWeekViewSettings
WorkWeekViewSettings workWeekViewSettings = new WorkWeekViewSettings();
workWeekViewSettings.TimeSlotBorderColor = Color.FromHex("#fceb9f") ;
workWeekViewSettings.VerticalLineColor = Color.LightGray;
workWeekViewSettings.TimeSlotColor = Color.FromHex("#fcf3c9");
workWeekViewSettings.TimeSlotBorderStrokeWidth = 5;
workWeekViewSettings.VerticalLineStrokeWidth = 5;
schedule.WorkWeekViewSettings = workWeekViewSettings;

```

*Timeslot customization in Non Working hours*

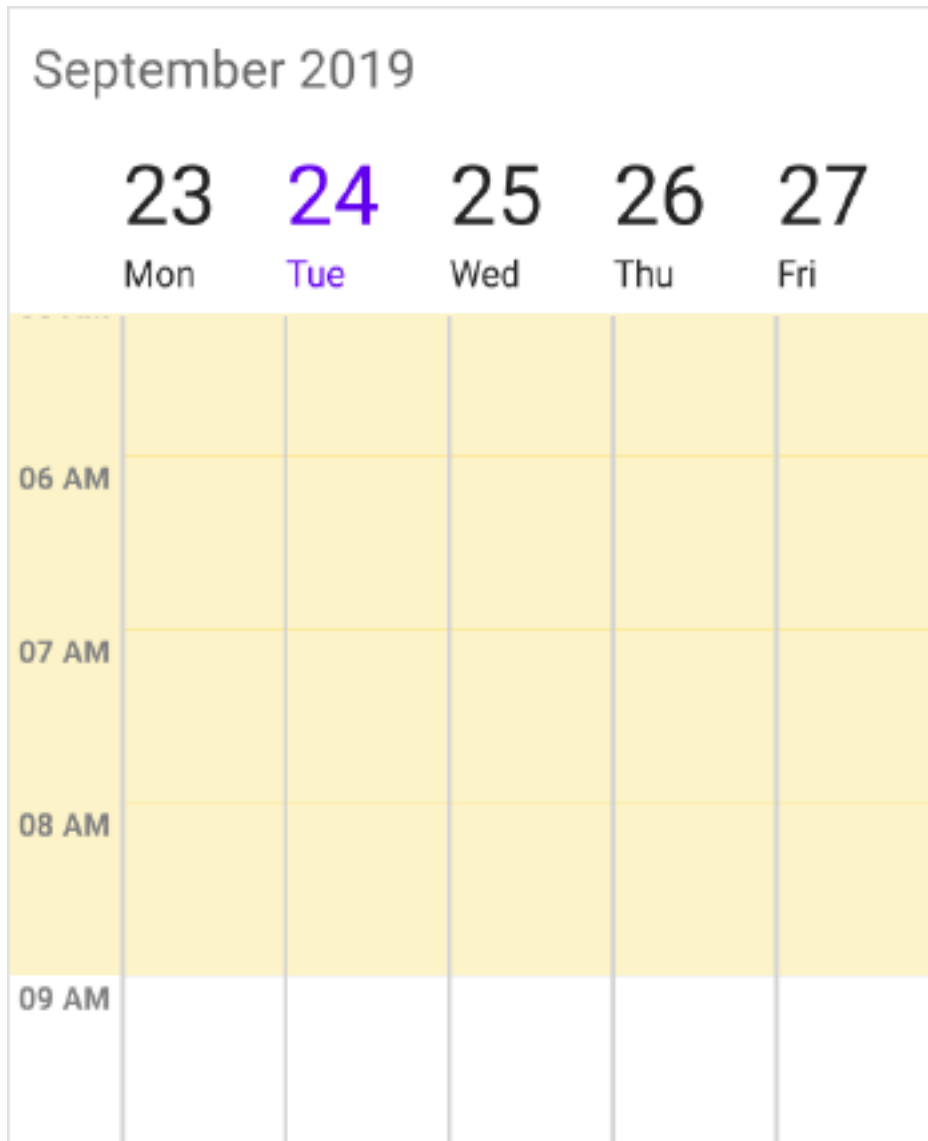
You can customize the appearance of the non-working hour timeslots by its color using [NonWorkingHoursTimeSlotBorderColor](#), [NonWorkingHoursTimeSlotColor](#), [VerticalLineStrokeWidth](#), [VerticalLineColor](#) and [TimeSlotBorderStrokeWidth](#) properties of [WorkWeekViewSettings](#).

## XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView">
<schedule:SfSchedule.WorkWeekViewSettings>
  <!--setting work week view settings properties -->
<schedule:WorkWeekViewSettings
NonWorkingHoursTimeSlotColor="#fcf3c9"
NonWorkingHoursTimeSlotBorderColor="#fceb9f"
TimeSlotBorderStrokeWidth="5"
VerticalLineStrokeWidth="5"
VerticalLineColor="LightGray">
</schedule:WorkWeekViewSettings>
</schedule:SfSchedule.WorkWeekViewSettings>
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
//Create new instance of WorkWeekViewSettings
WorkWeekViewSettings workWeekViewSettings = new WorkWeekViewSettings();
workWeekViewSettings.NonWorkingHoursTimeSlotBorderColor =
Color.FromHex("#fceb9f") ;
workWeekViewSettings.VerticalLineColor = Color.LightGray;
workWeekViewSettings.NonWorkingHoursTimeSlotColor =
Color.FromHex("#fcf3c9");
workWeekViewSettings.TimeSlotBorderStrokeWidth = 5;
workWeekViewSettings.VerticalLineStrokeWidth = 5;
schedule.WorkWeekViewSettings = workWeekViewSettings;
```

**NOTE**

TimeSlotBorderStrokeWidth and VerticalLineStrokeWidth properties are common to both Working hours and Non-Working hour time slot customization.

**Non-Accessible timeslots**

You can restrict or allocate certain timeslot as Non-accessible blocks by using [NonAccessibleBlocks](#) of `WorkWeekViewSettings` so that you can allocate those timeslots for predefined events/activities like Lunch hour.

**XML**

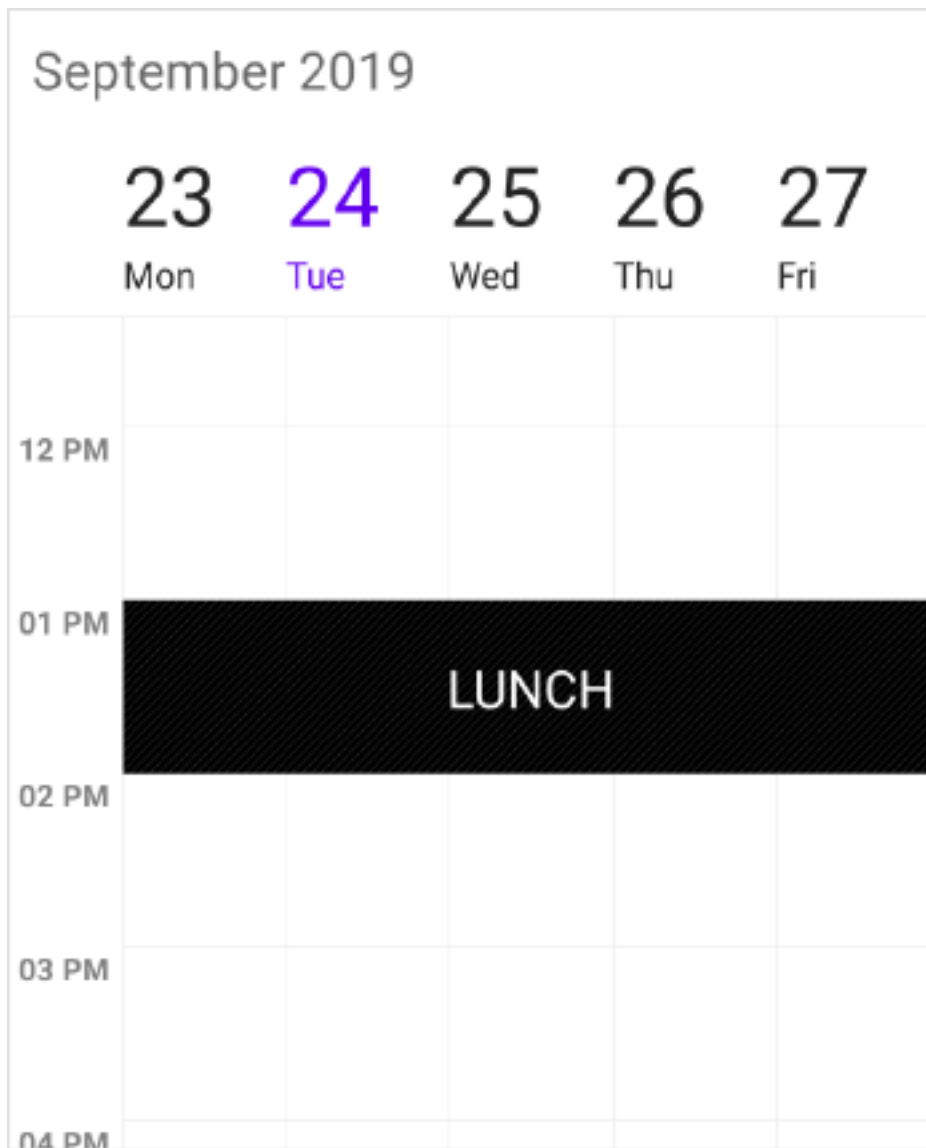
```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView">
  <!--setting non-accessing blocks-->
  <schedule:SfSchedule.WorkWeekViewSettings>
    <schedule:WorkWeekViewSettings>
      <schedule:WorkWeekViewSettings.NonAccessibleBlocks>
        <schedule:NonAccessibleBlock
```



```
StartTime="13"  
EndTime="14"  
Text="LUNCH"  
Color="Black" />  
</schedule:WorkWeekViewSettings.NonAccessibleBlocks>  
</schedule:WorkWeekViewSettings>  
</schedule:SfSchedule.WorkWeekViewSettings>  
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.WorkWeekView;  
//Create new instance of NonAccessibleBlock  
NonAccessibleBlock nonAccessibleBlock = new NonAccessibleBlock();  
//Create new instance of NonAccessibleBlocksCollection  
NonAccessibleBlocksCollection nonAccessibleBlocksCollection = new  
NonAccessibleBlocksCollection();  
WorkWeekViewSettings workWeekViewSettings = new WorkWeekViewSettings();  
nonAccessibleBlock.StartTime = 13;  
nonAccessibleBlock.EndTime = 14;  
nonAccessibleBlock.Text = "LUNCH";  
nonAccessibleBlock.Color = Color.Black;  
nonAccessibleBlocksCollection.Add(nonAccessibleBlock);  
workWeekViewSettings.NonAccessibleBlocks = nonAccessibleBlocksCollection;  
schedule.WorkWeekViewSettings = workWeekViewSettings;
```

**NOTE**

Selection and related events will not be working in this blocks.

Change first day of week

By default, schedule control will be rendered with Sunday as the first day of the week, it can be customized to any day of the week by using [FirstDayOfWeek](#) property of [SfSchedule](#).

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView"
FirstDayOfWeek="3"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
schedule.FirstDayOfWeek = 3;
```

September 2019					
	24 Tue	25 Wed	26 Thu	27 Fri	30 Mon
09 AM					
10 AM					
11 AM					
12 PM					
01 PM					

### Time Label Formatting

You can customize the format for the labels which are mentioning the time, by setting [TimeFormat](#) property of [WorkWeekLabelSettings](#) in [WorkWeekViewSettings](#).

### C#

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
WorkWeekViewSettings workweekViewSettings = new WorkWeekViewSettings();
WorkWeekLabelSettings workWeekLabelSettings = new WorkWeekLabelSettings();
workWeekLabelSettings.TimeFormat = "hh mm";
workweekViewSettings.WorkWeekLabelSettings = workWeekLabelSettings;
schedule.WorkWeekViewSettings = workweekViewSettings;
```

September 2019					
	23	24	25	26	27
	Mon	Tue	Wed	Thu	Fri
09 00					
10 00					
11 00					
12 00					
01 00					

### Time Label Appearance

You can customize the color for the labels which are mentioning the time, by setting [TimeLabelColor](#) property of `WorkWeekLabelSettings` in `WorkWeekViewSettings`.

### C#

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
//Create new instance of WorkWeekViewSettings
WorkWeekViewSettings workWeekViewSettings = new WorkWeekViewSettings();
//Create new instance of WorkWeekLabelSettings
WorkWeekLabelSettings workWeekLabelSettings = new WorkWeekLabelSettings();
workWeekLabelSettings.TimeLabelColor = Color.FromHex("#8282ff");
workWeekViewSettings.WorkWeekLabelSettings = workWeekLabelSettings;
schedule.WorkWeekViewSettings = workWeekViewSettings;
```

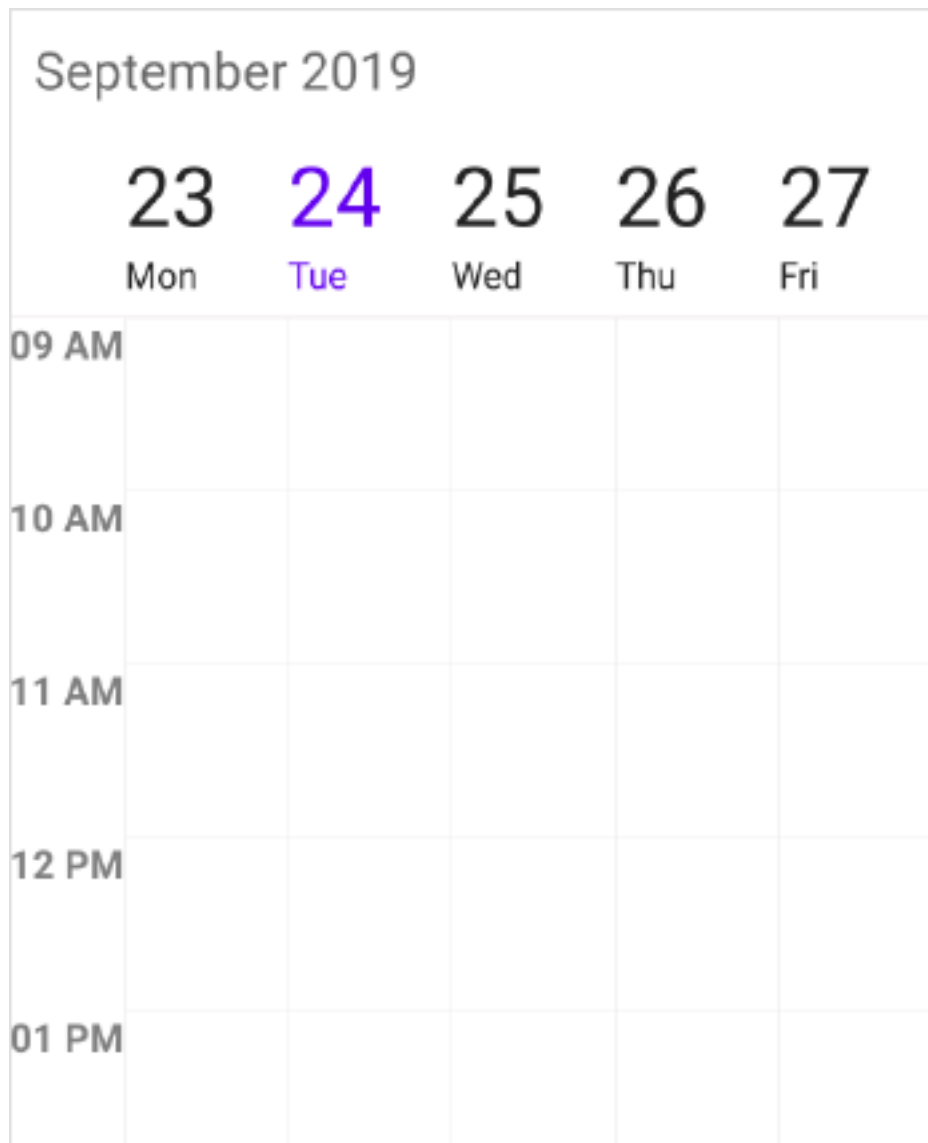
September 2019					
	23	24	25	26	27
	Mon	Tue	Wed	Thu	Fri
09 AM					
10 AM					
11 AM					
12 PM					
01 PM					

### Time Label Size

You can customize the size of the labels which are mentioning the time, by setting [TimeLabelSize](#) property of `WorkWeekLabelSettings` in `WorkWeekViewSettings`.

### C#

```
schedule.ScheduleView = ScheduleView.WorkWeekView;  
//Create new instance of WorkWeekViewSettings  
WorkWeekViewSettings workWeekViewSettings = new WorkWeekViewSettings();  
//Create new instance of WorkWeekLabelSettings  
WorkWeekLabelSettings workWeekLabelSettings = new WorkWeekLabelSettings();  
//Customizing the size of the time label  
workWeekLabelSettings.TimeLabelSize = 15;  
workWeekViewSettings.WorkWeekLabelSettings = workWeekLabelSettings;  
schedule.WorkWeekViewSettings = workWeekViewSettings;
```



### Selection

You can customize the default appearance of selection UI in the timeslots.

- [Selection customization using style](#)
- [Selection customization using custom View](#)
- [Programmatic selection](#)

### *Selection customization using style*

You can customize the timeslot selection by using [SelectionStyle](#) property of `SfSchedule`.

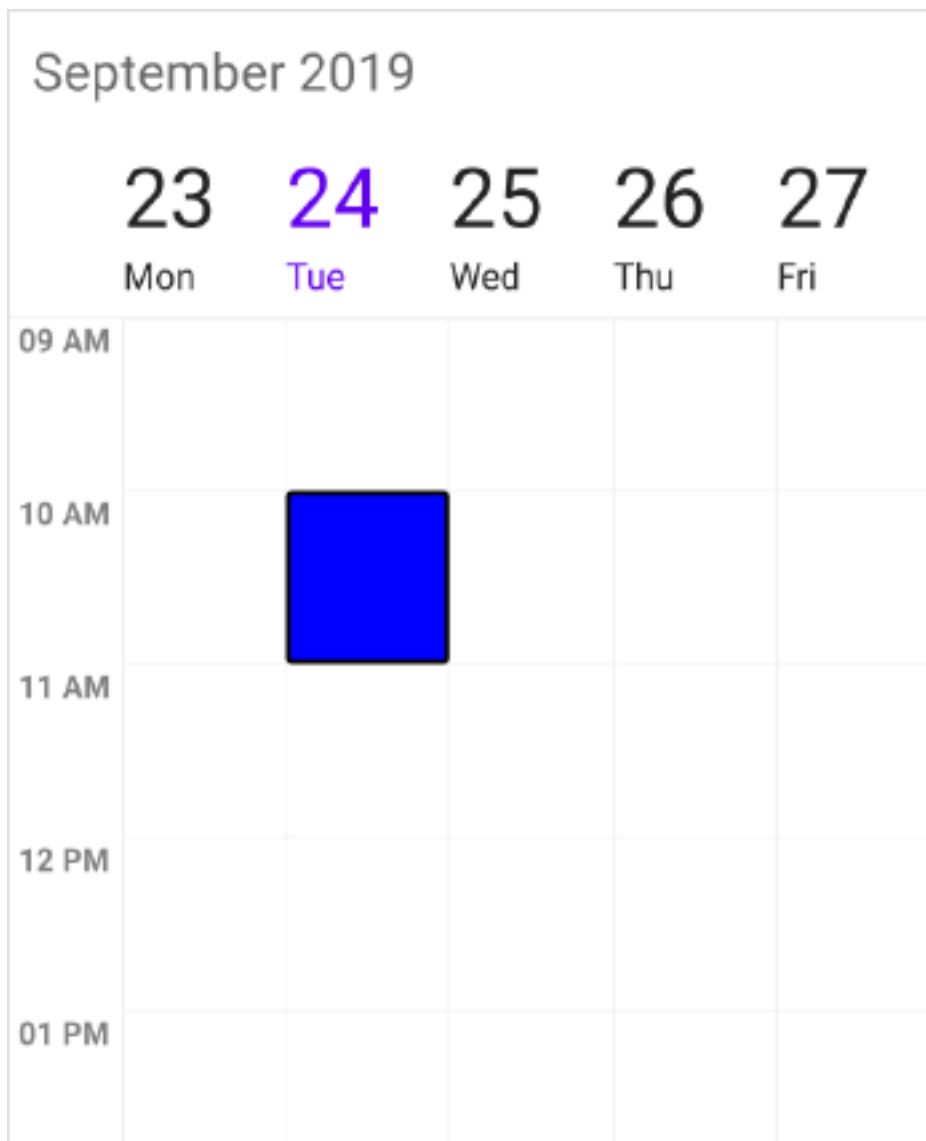
### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView">
  <schedule:SfSchedule.SelectionStyle>
    <schedule:SelectionStyle
      BackgroundColor="Blue"
      BorderColor="Black"
    />
  />
</schedule:SfSchedule>
```

```
BorderThickness="5"  
BorderCornerRadius="5">  
</schedule:SelectionStyle>  
</schedule:SfSchedule.SelectionStyle>  
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.WorkWeekView;  
//Create new instance of SelectionStyle  
SelectionStyle selectionStyle = new SelectionStyle();  
selectionStyle.BackgroundColor = Color.Blue;  
selectionStyle.BorderColor = Color.Black;  
selectionStyle.BorderThickness = 5;  
selectionStyle.BorderCornerRadius = 5;  
schedule.SelectionStyle = selectionStyle;
```



*Selection customization using custom View*

You can replace the default selection UI with your custom view by setting [SelectionView](#) property of SfSchedule.

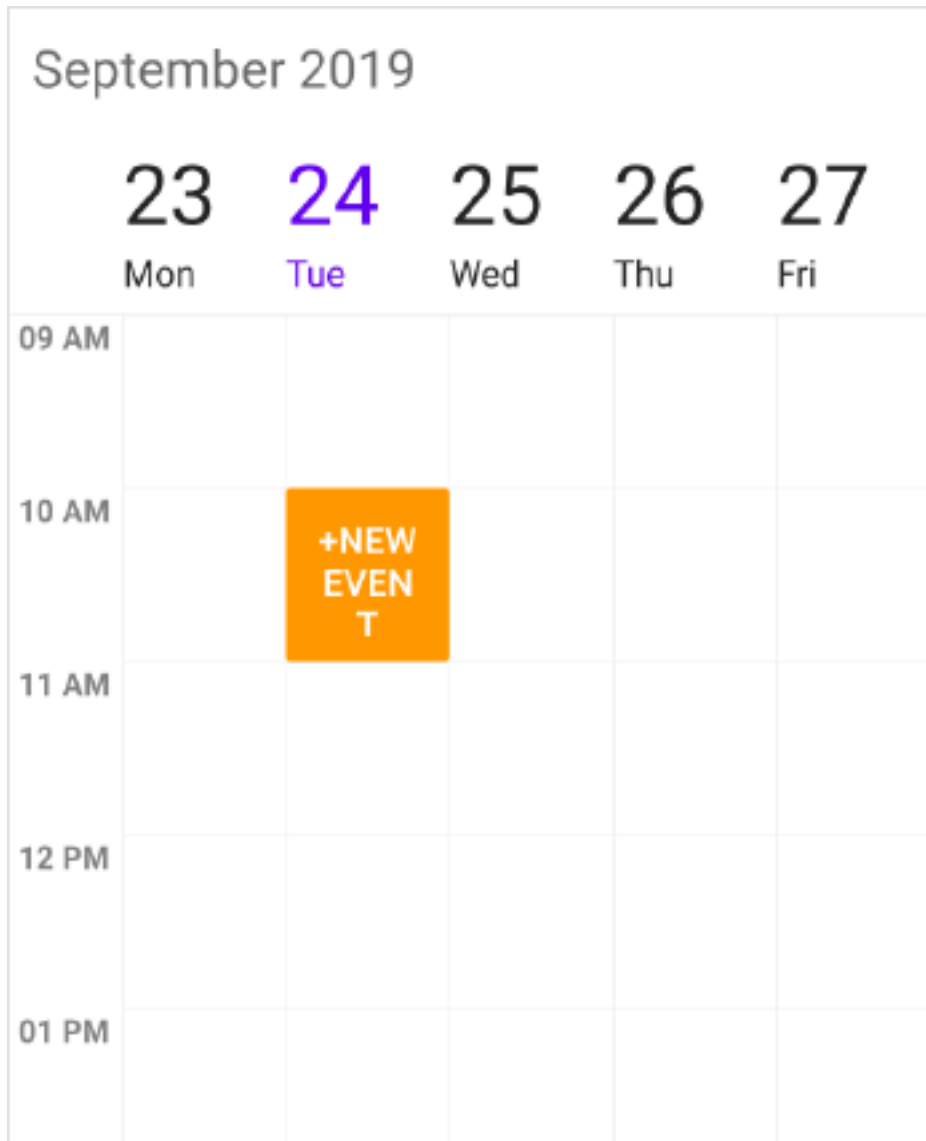
**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WorkWeekView">
<schedule:SfSchedule.SelectionView>
<Button
BackgroundColor="#FF9800"
Text="+NewEvent"
TextColor="White"/>
</schedule:SfSchedule.SelectionView>
</schedule:SfSchedule>
```

**C#**

```
schedule.ScheduleView = ScheduleView.WorkWeekView;
//Add the CustomView
Button customView = new Button();
customView.Text = "+NewEvent";
customView.BackgroundColor = Color.FromHex("#FF9800");
customView.TextColor = Color.White;
schedule.SelectionView = customView;
```





#### *Programmatic selection*

You can programmatically select the specific timeslot by setting corresponding date and time value to [SelectedDate](#) property of `SfSchedule`. By default, it is null.

#### **C#**

```
// Setting a date and time to select  
schedule.SelectedDate = new DateTime(2017, 10, 04, 10, 0, 0);
```

You can clear the selection by setting [SelectedDate](#) as null.

#### **C#**

```
// Setting null value to deselect  
schedule.SelectedDate = null;
```

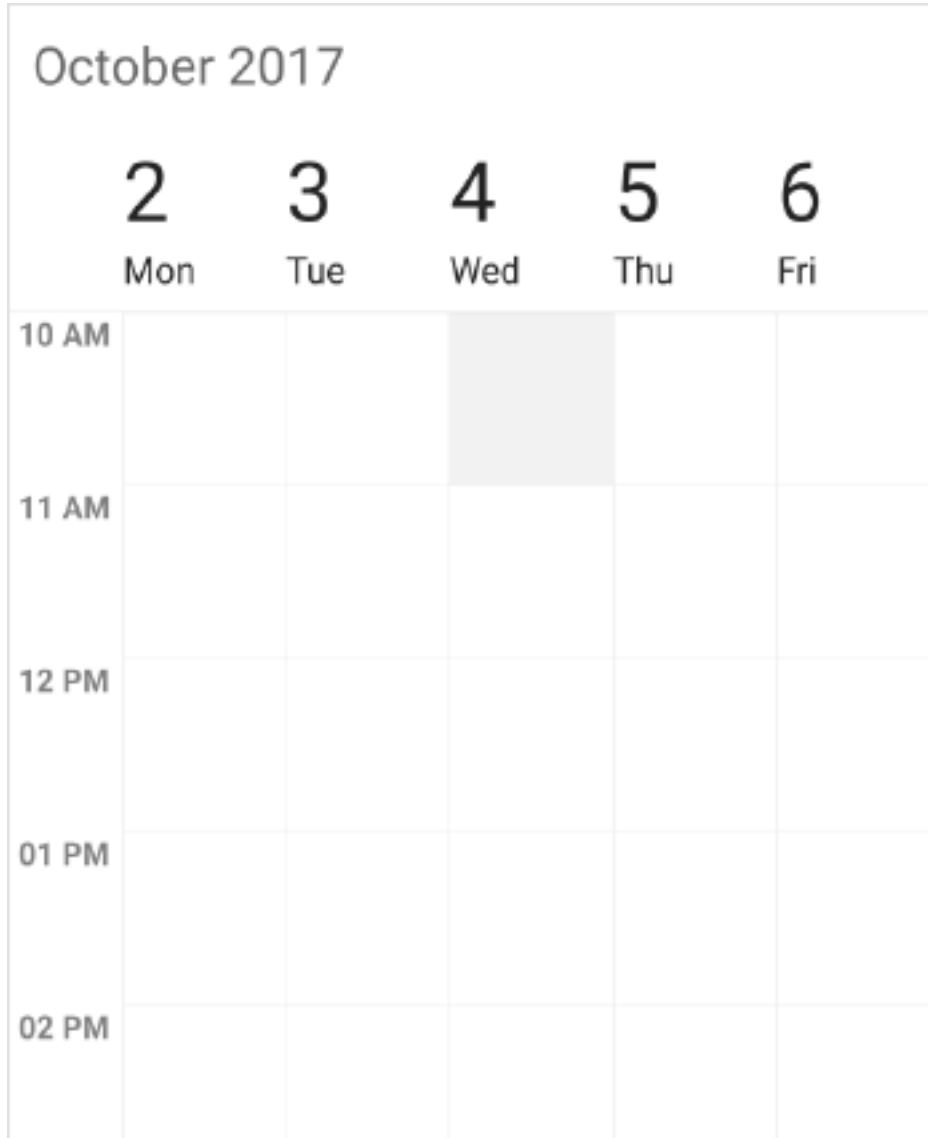
You can download the entire source code of this demo for Xamarin.Forms from here [Date Selection](#)

---

**NOTE**

---

- SfSchedule does not support multiple selection.
- SfSchedule supports two-way binding of SelectedDate property.



### Timeline view

**TimelineView** displays the dates in horizontal time axis with the desired day's count. You can see the past or future dates by scrolling to the right or left. Each view displays events accurately across the time slots with an intuitive drag-and-drop feature. It provides support to highlight the selected region of time slots and handle interaction.

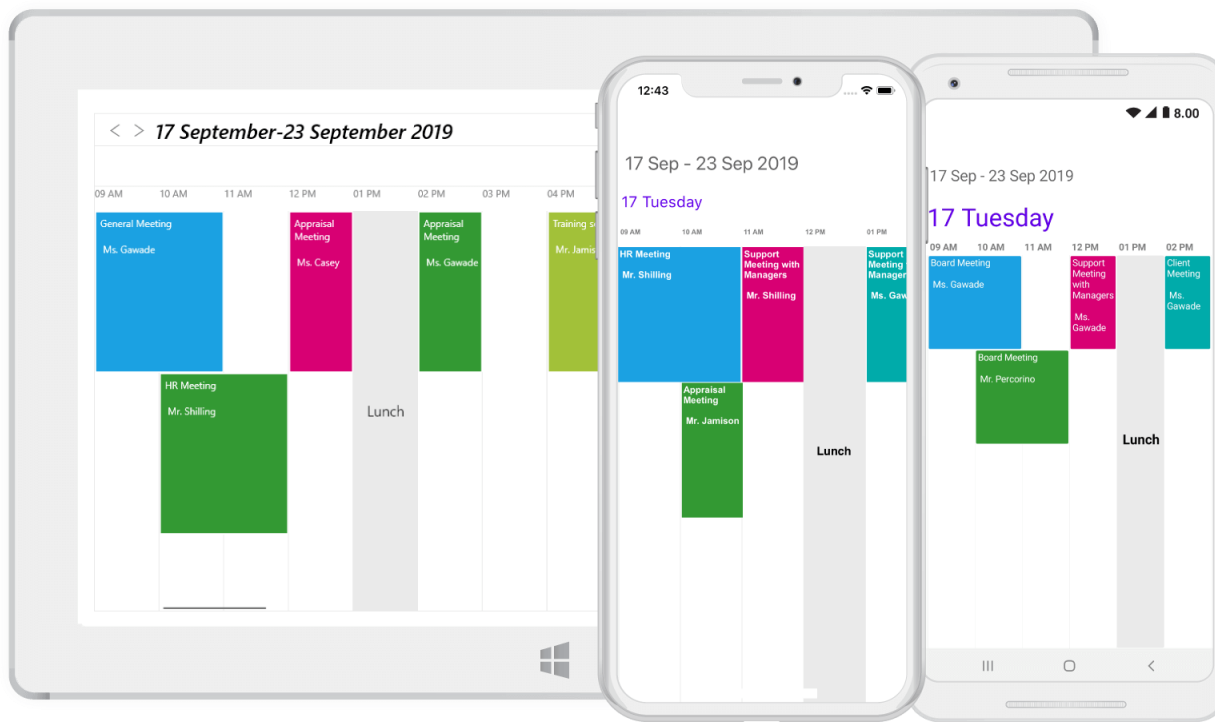
### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView"/>
```

### C#

```
schedule.ScheduleView = ScheduleView.TimelineView;
```

You can download the entire source code of this demo for Xamarin.Forms from [here](#).



### Timeline view days count

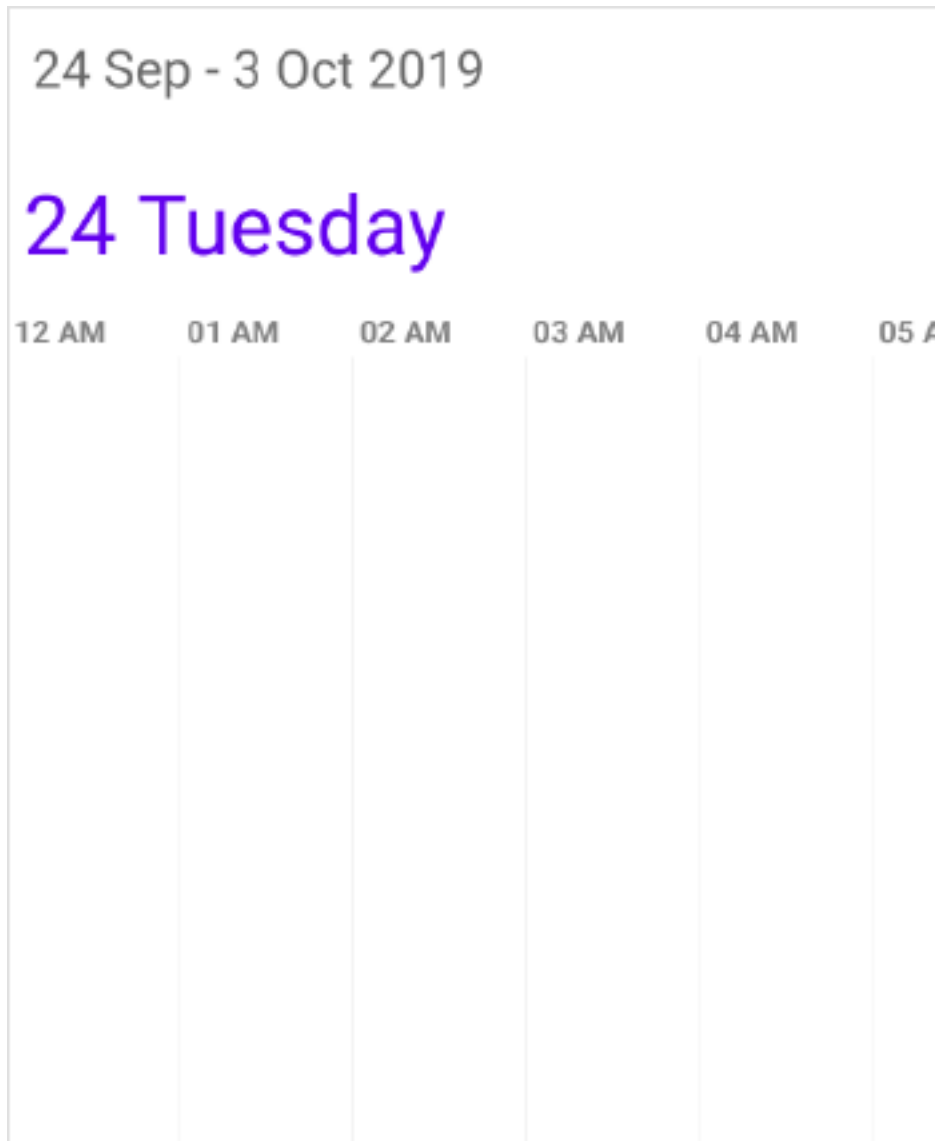
You can customize the number of days in `TimelineView` using the `DaysCount` property of `TimelineViewSettings`. By default, value of the timeline days count is -1, and single day will be visible.

### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">
  <syncfusion:SfSchedule.TimelineViewSettings>
    <!--setting days count property-->
    <syncfusion:TimelineViewSettings
      DaysCount="10" />
  </syncfusion:SfSchedule.TimelineViewSettings>
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.TimelineView;
//Creating new instance of TimelineViewSettings
TimelineViewSettings timelineViewSettings = new TimelineViewSettings();
//Customizing days count
timelineViewSettings.DaysCount = 10;
schedule.TimelineViewSettings = timelineViewSettings;
```



Timeline view based on day, week, work week, and month.

You can achieve timeline day, timeline week, timeline work week, and timeline month view with the default value of `DaysCount` by dynamically switching between day, week, work week, and month view to timeline view with respective visible dates.

---

**NOTE**

For other value of `DaysCount`, only timeline view visible dates will be displayed as mentioned days count on dynamic view switching.

Customized working hours

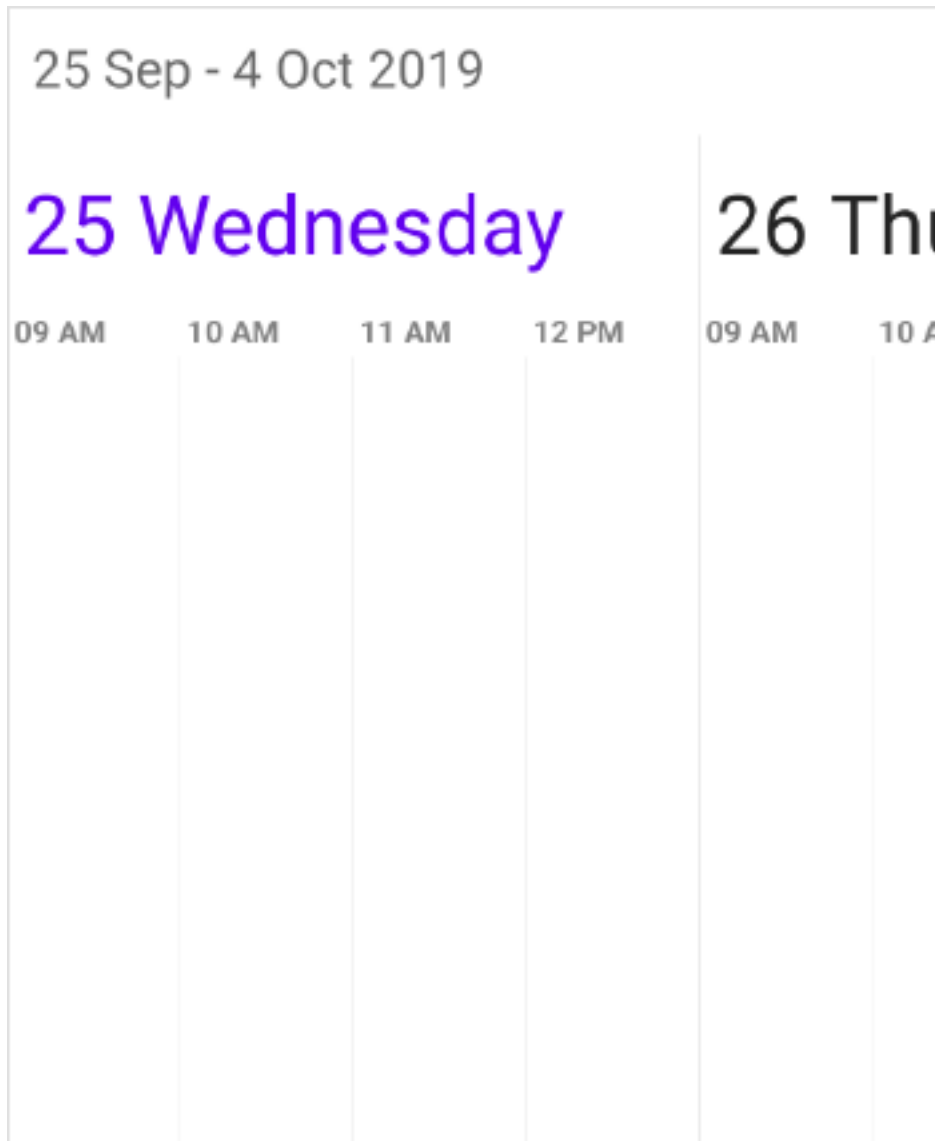
You can customize the `StartHour` and `EndHour` properties of `TimelineView` to show only the required time duration for end users. You can also set `StartHour` and `EndHour` in double value, which will be converted to time to show the required time duration in minutes. The default value for `StartHour` and `EndHour` value is 0 to 24 to show all the time slots in `TimelineView`.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">
  <schedule:SfSchedule.TimelineViewSettings>
    <!--setting visible hours properties-->
    <schedule:TimelineViewSettings
      StartHour="09"
      EndHour="13">
    </schedule:TimelineViewSettings>
  </schedule:SfSchedule.TimelineViewSettings>
</schedule:SfSchedule>
```

**C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;
//Creating new instance of TimelineViewSettings
TimelineViewSettings timelineViewSettings = new TimelineViewSettings();
timelineViewSettings.StartHour = 09;
timelineViewSettings.EndHour = 13;
schedule.TimelineViewSettings = timelineViewSettings;
```



---

**NOTE**

- **StartHour** must be greater than or equal to 0 and **EndHour** must be lesser than or equal to 24, otherwise **InvalidDataException** will be thrown.
- **EndHour** value must be greater than **StartHour**, otherwise **InvalidDataException** will be thrown.
- Schedule UI such as **Appointments** and **NonAccessibleBlocks** 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 given **TimeInterval** should result in integer value (total minutes % **TimeInterval** = 0), otherwise next immediate time interval that result in integer value when divide total minutes of a day will be

considered. For example, if `TimeInterval="135"` (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 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" ( $555\%37=0$ ) by considering (total minutes % `TimeInterval` = 0); it will return integer value for time slots rendering.

### Special time regions

You can restrict user interaction such as selection and highlight specific region of time in `TimelineView` by adding `SpecialTimeRegion` in the `SpecialTimeRegions` property of `SfSchedule`. You need to set the `StartHour` and `EndHour` properties of `TimeRegionSettings` to create `SpecialTimeRegion`. You can also set `StartHour` and `EndHour` in double value, which will be converted to time to show the required time duration in minutes. By default, the values of `StartHour` and `EndHour` are 0.

#### Special time region appearance

You can customize the appearance of `SpecialTimeRegion` using the `Color` and `TextColor` properties of `TimeRegionSettings`.

#### Selection restriction in time slots

You can enable/disable the touch interaction of `SpecialTimeRegion` using `CanEdit` property of `TimeRegionSettings`. By default, its value is true.

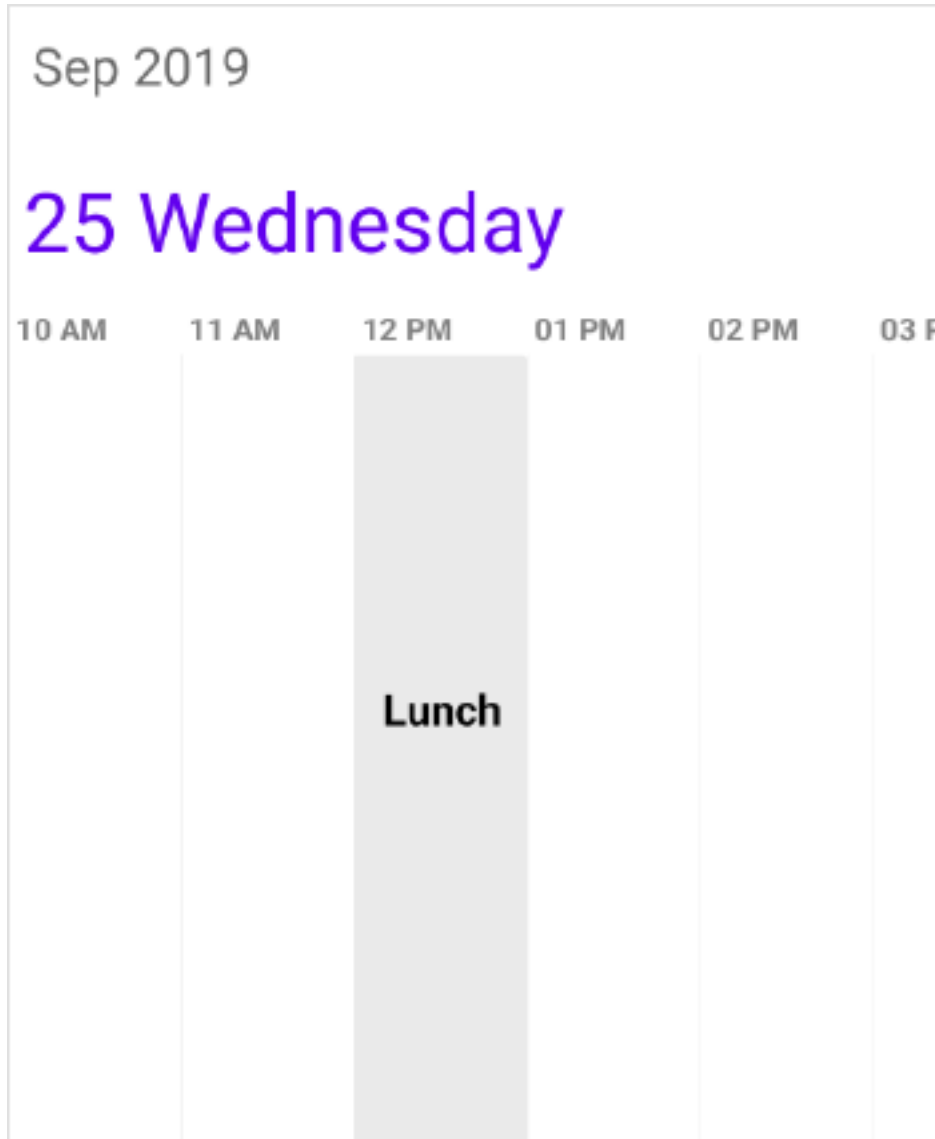
### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">
  <!--setting Special time regions property-->
  <schedule:SfSchedule.SpecialTimeRegions>
    <schedule:TimeRegionSettings
      StartHour="12"
      EndHour="13"
      Text="Lunch"
      CanEdit="False"
      Color="#EAEAEA"
      TextColor="Black"/>
    </schedule:SfSchedule.SpecialTimeRegions>
  </schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.TimelineView;
ObservableCollection<TimeRegionSettings> specialTimeRegions = new
ObservableCollection<TimeRegionSettings>();
//Setting Special time regions property
TimeRegionSettings timeRegionSettings = new TimeRegionSettings();
timeRegionSettings.StartHour = 12;
timeRegionSettings.EndHour = 13;
timeRegionSettings.Text = "Lunch";
timeRegionSettings.Color = Color.FromHex("#EAEAEA");
```

```
timeRegionSettings.TextColor = Color.Black;  
timeRegionSettings.CanEdit = false;  
specialTimeRegions.Add(timeRegionSettings);  
schedule.SpecialTimeRegions = specialTimeRegions;
```



#### Time interval

You can customize the interval of time slots in `TimelineView` by setting [TimeInterval](#) property of `SfSchedule`.

#### XML

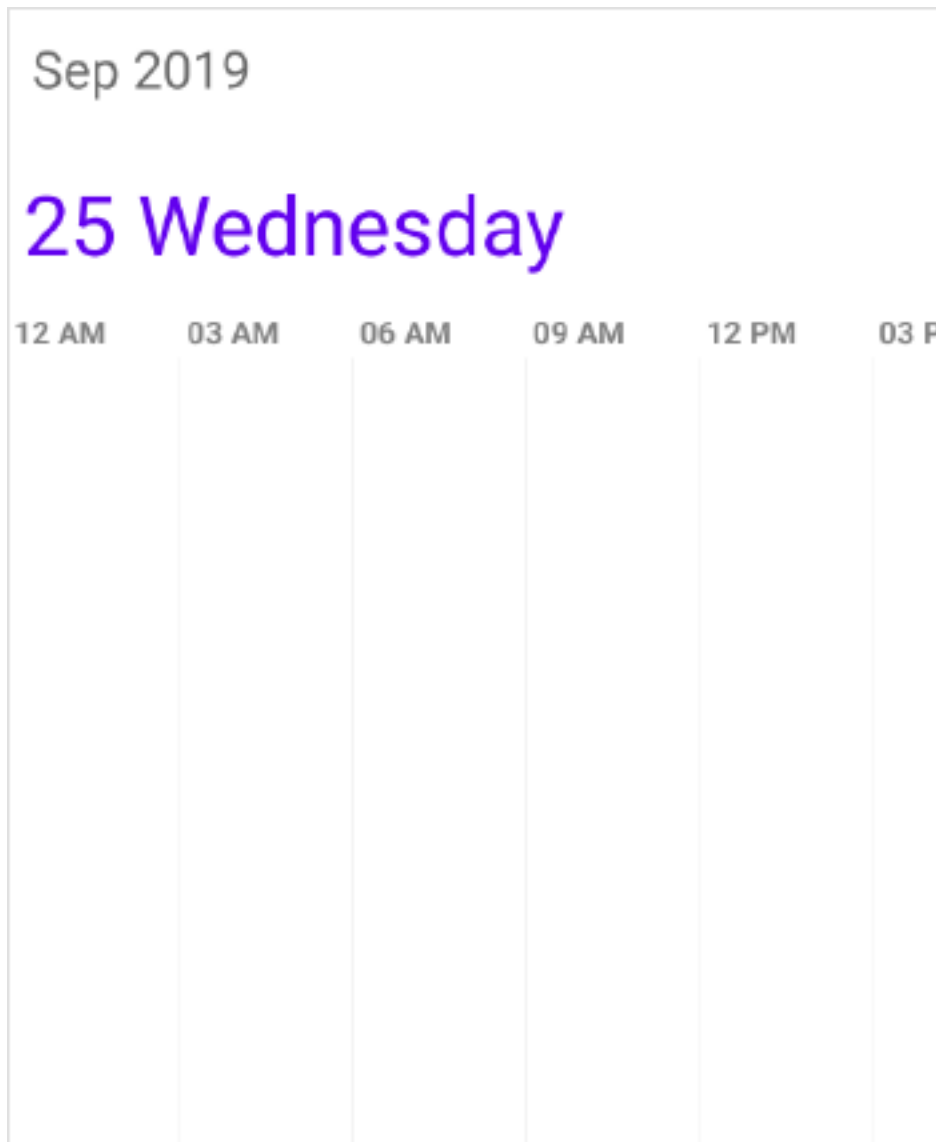
```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView"  
TimeInterval="180"/>
```

#### C#

```
schedule.ScheduleView = ScheduleView.TimelineView;
```



```
schedule.TimeInterval = 180;
```



#### NOTE

If you modify the `TimeInterval` value (in minutes), you need to change the time labels format by setting the `TimeFormat` as "hh:mm". By default, `TimeFormat` is "hh a". Refer to this [documentation](#) for changing `TimeFormat` value.

#### Time interval height

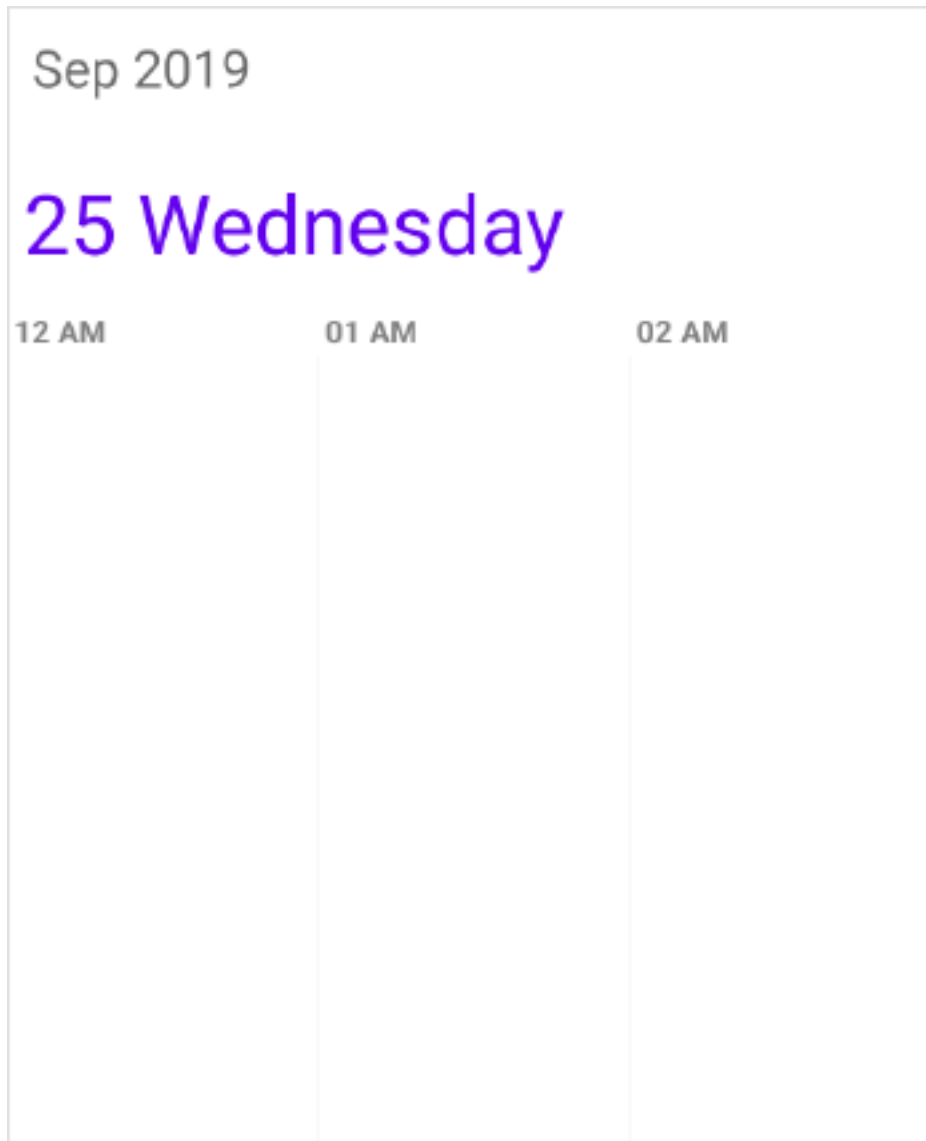
You can customize the interval height of time slots in `TimelineView` by setting the [TimeIntervalHeight](#) property of `SfSchedule`.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView"
    TimeIntervalHeight="180"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;  
schedule.TimeIntervalHeight = 180;
```

*Full screen scheduler*

Schedule time interval height can be adjusted based on screen height by changing the value of `TimeIntervalHeight` property to -1. It will auto-fit to the screen height and width.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView"  
TimeIntervalHeight="-1"/>
```

**C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;
```

```
schedule.TimeIntervalHeight = -1;
```

## NOTE

In Timeline view, If the time slot duration such as `DaysCount`, `StartHour`, or `EndHour` is within the width of the screen, the `TimeIntervalHeight` will automatically be adjusted to the screen, otherwise the `TimeIntervalHeight` will be adjusted to 100 offset value.

### Nonworking days

You can add the non-working days in `TimelineView` using `NonWorkingsDays` property of `TimelineViewSettings`. By default, there is no non-working day in `TimelineView`.

### C#

```
var nonWorkingDays = new ObservableCollection<DayOfWeek>();
nonWorkingDays.Add(DayOfWeek.Monday);
nonWorkingDays.Add(DayOfWeek.Friday);
var timelineViewSettings = new TimelineViewSettings();
timelineViewSettings.NonWorkingsDays = nonWorkingDays;
schedule.TimelineViewSettings = timelineViewSettings;
```

### First day of week

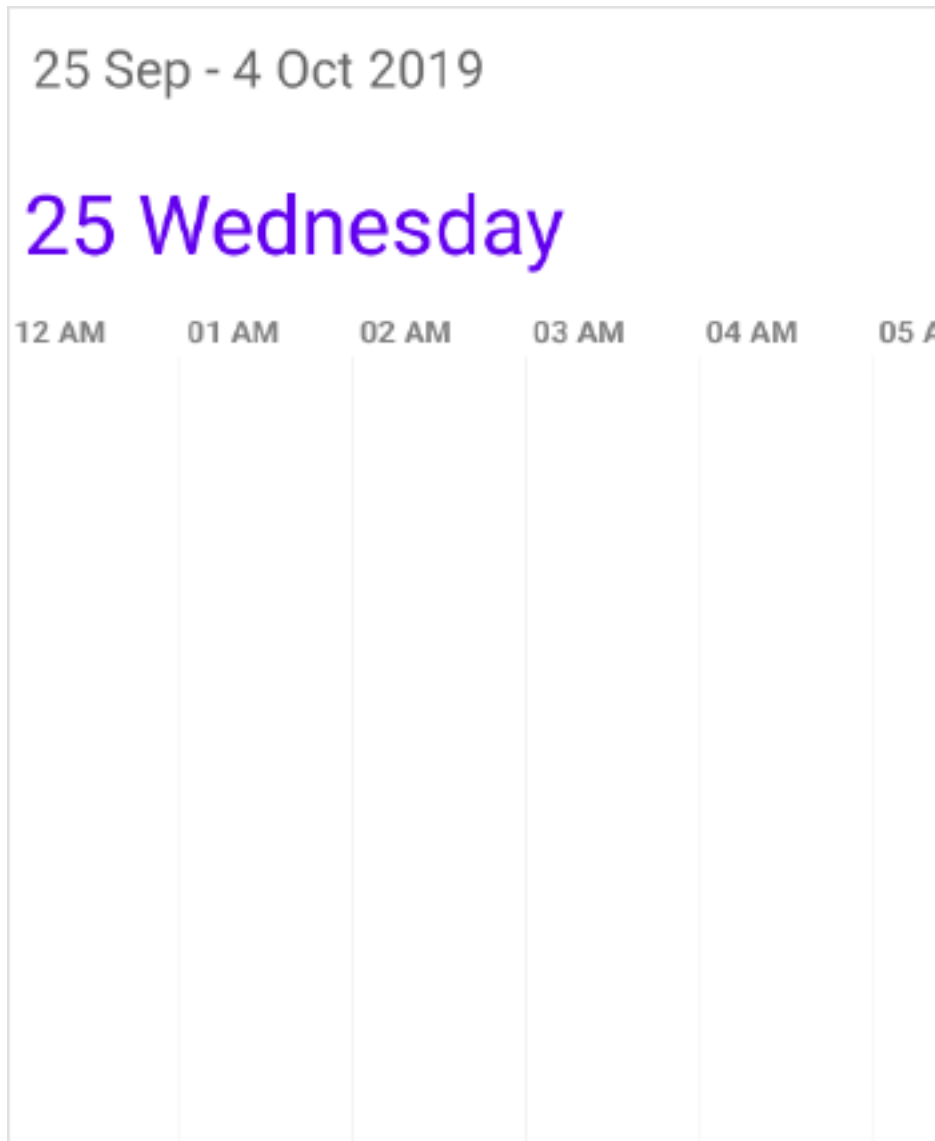
By default, schedule control will be rendered with Sunday as the first day of the week. It can be customized to any day of the week using the `FirstDayOfWeek` property of `SfSchedule`.

### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView"
FirstDayOfWeek="3"/>
```

### C#

```
schedule.ScheduleView = ScheduleView.TimelineView;
schedule.FirstDayOfWeek = 3;
```



#### NOTE

In Timeline view, `FirstDayOfWeek` will be applied only when `DayCounts` property of `TimelineViewSettings` is 7.

#### Appointment height

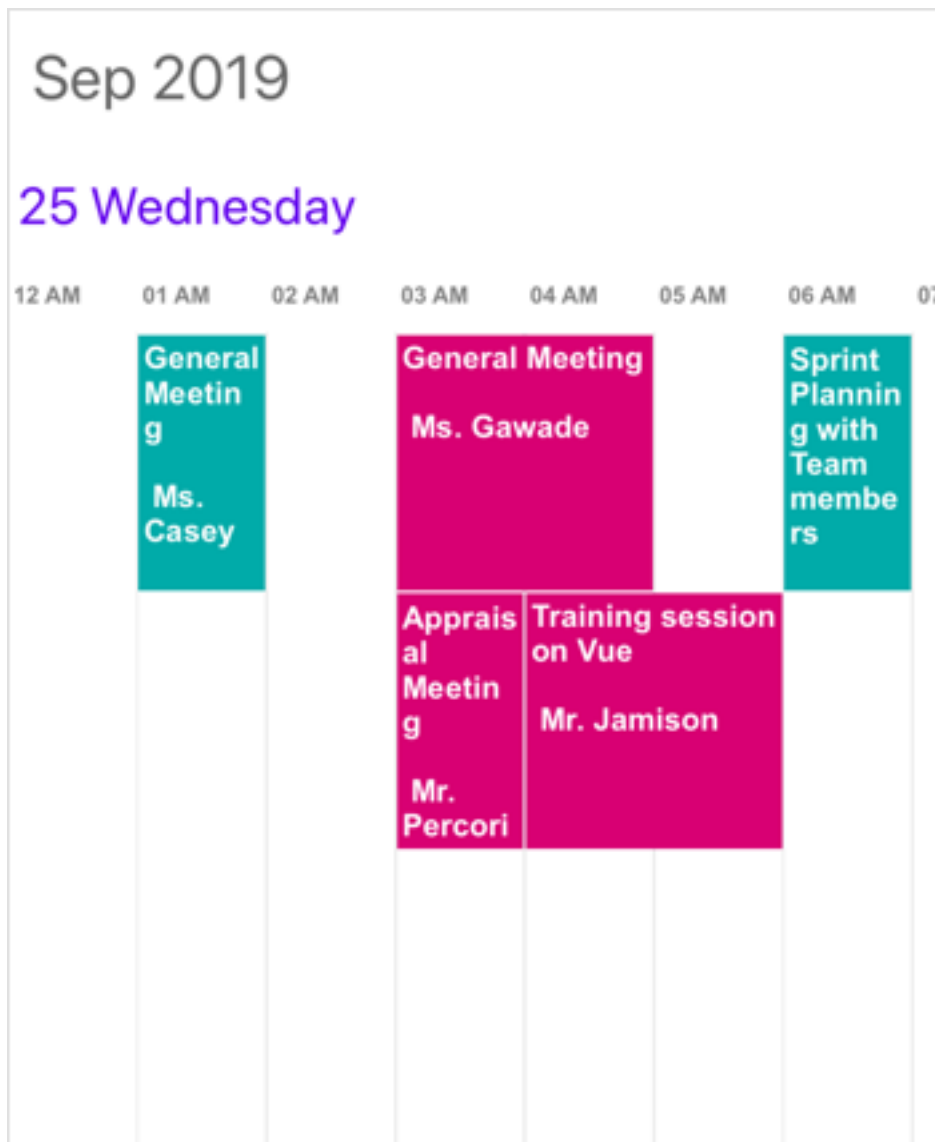
You can customize the height of the appointment in `TimelineView` using the [AppointmentHeight](#) property of `TimelineViewSettings`. By default, its value is 50.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">
  <schedule:SfSchedule.TimelineViewSettings>
    <!--setting appointment height property-->
    <schedule:TimelineViewSettings
      AppointmentHeight="100" />
  </schedule:SfSchedule.TimelineViewSettings>
</schedule:SfSchedule>
```

**C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;  
//Create new instance of TimelineViewSettings  
TimelineViewSettings timelineViewSettings = new TimelineViewSettings();  
timelineViewSettings.AppointmentHeight = 100;  
schedule.TimelineViewSettings = timelineViewSettings;
```

**NOTE**

When a greater number of appointments is added in the same time slot, appointment height will be calculated automatically without considering the `AppointmentHeight` property to display all the appointment in the view.

### View header tapped event

You can handle single tap action of ViewHeader using the `ViewHeaderTapped` event of `SfSchedule`. This event occurs when the `ViewHeader` is tapped. This event contains `ViewHeaderTappedEventArgs` argument, which holds the details of `DateTime` in it.

#### XML

```
<schedule:SfSchedule x:Name="schedule"
ScheduleView="TimelineView "
ViewHeaderTapped="Handle_ViewHeaderTapped">
</schedule:SfSchedule>
```

#### C#

```
//Creating new instance of Schedule
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.TimelineView;
schedule.ViewHeaderTapped += OnViewHeaderTapped;
```

#### C#

```
private void OnViewHeaderTapped(object sender, ViewHeaderTappedEventArgs e)
{
    var dateTime = e.DateTime;
}
```

### View header customization

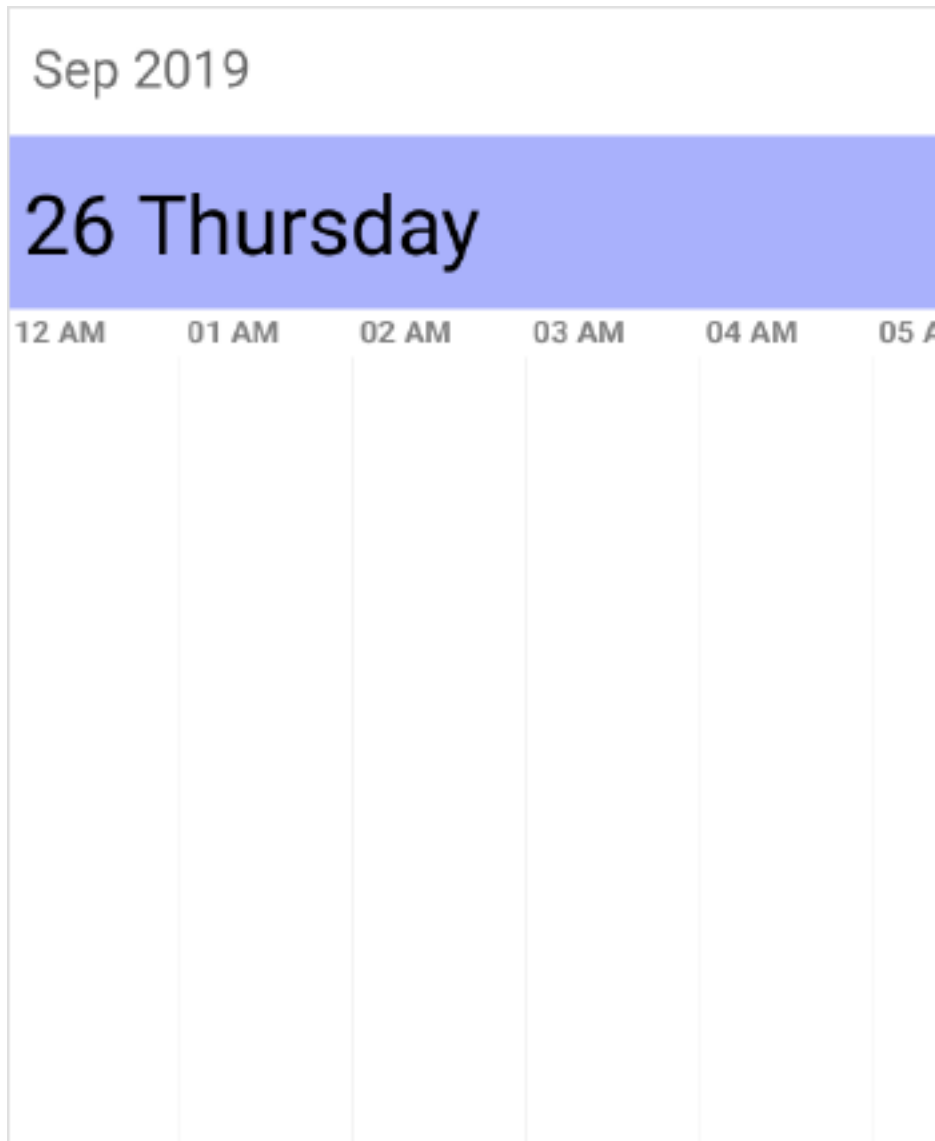
You can customize the default appearance of view header in `TimelineView` by using `ViewHeaderStyle` property of `SfSchedule`.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView =" TimelineView">
<schedule:SfSchedule.ViewHeaderStyle>
<schedule:ViewHeaderStyle
BackgroundColor="#a9b1fc"
DateTextColor="Black"
DateFontFamily="Arial">
</schedule:ViewHeaderStyle>
</schedule:SfSchedule.ViewHeaderStyle>
</schedule:SfSchedule>
```

#### C#

```
//Create new instance of Schedule
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.TimelineView;
//Customize the schedule view header
ViewHeaderStyle viewHeaderStyle = new ViewHeaderStyle();
viewHeaderStyle.BackgroundColor = Color.FromHex("#a9b1fc");
viewHeaderStyle.DateTextColor = Color.Black;
viewHeaderStyle.DateFontFamily = "Arial";
schedule.ViewHeaderStyle = viewHeaderStyle;
```

**NOTE**

`FontAttributes` and `FontFamily` are native to the platform. Custom font and the font that are not available in the specified platform will not be applied.

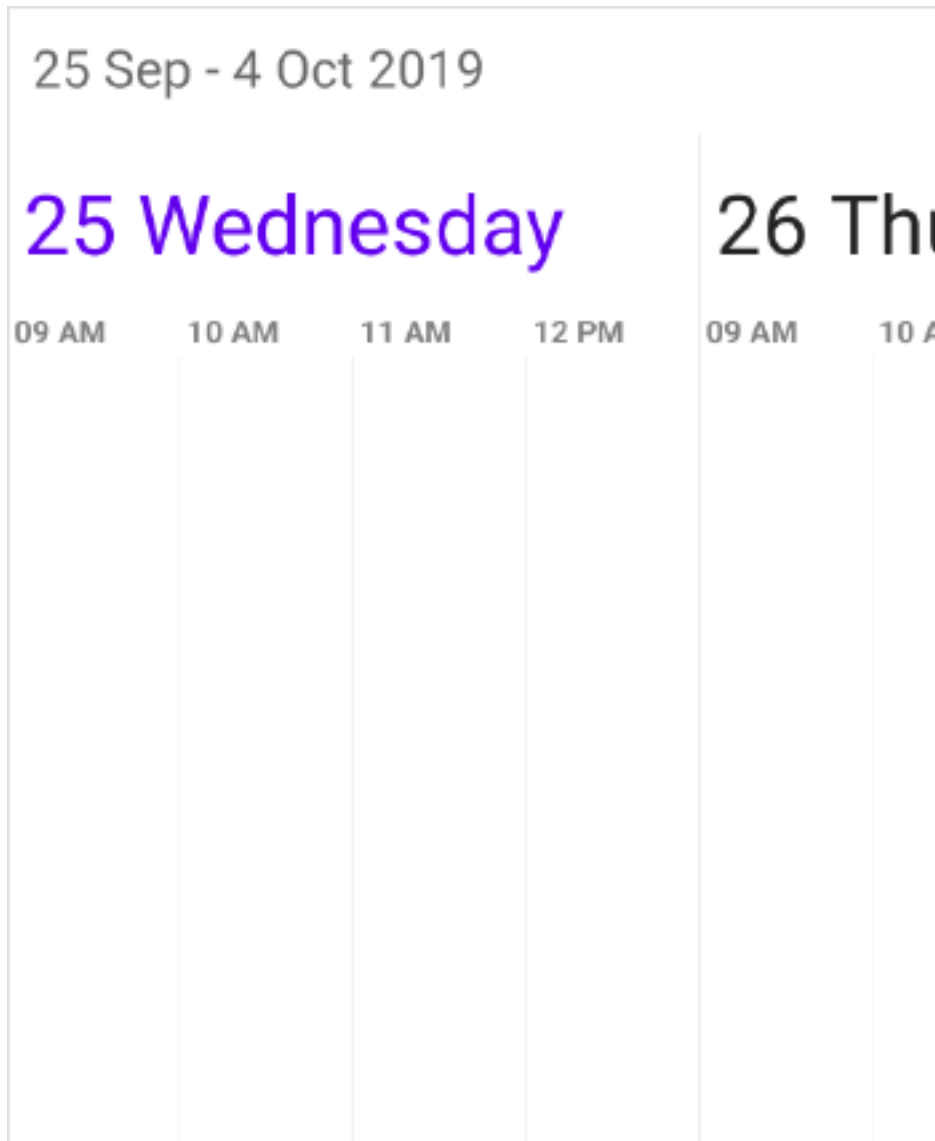
You can customize the height of `ViewHeader` in `TimelineView` by setting the [ViewHeaderHeight](#) property of `SfSchedule`.

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView ="TimelineView "  
ViewHeaderHeight="50" />
```

**C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;  
schedule.ViewHeaderHeight = 50;
```



#### *View header date format*

You can customize the date format of `ViewHeader` in `TimelineView` using the [DateFormat](#) property of `TimelineLabelSettings`.

#### **XML**

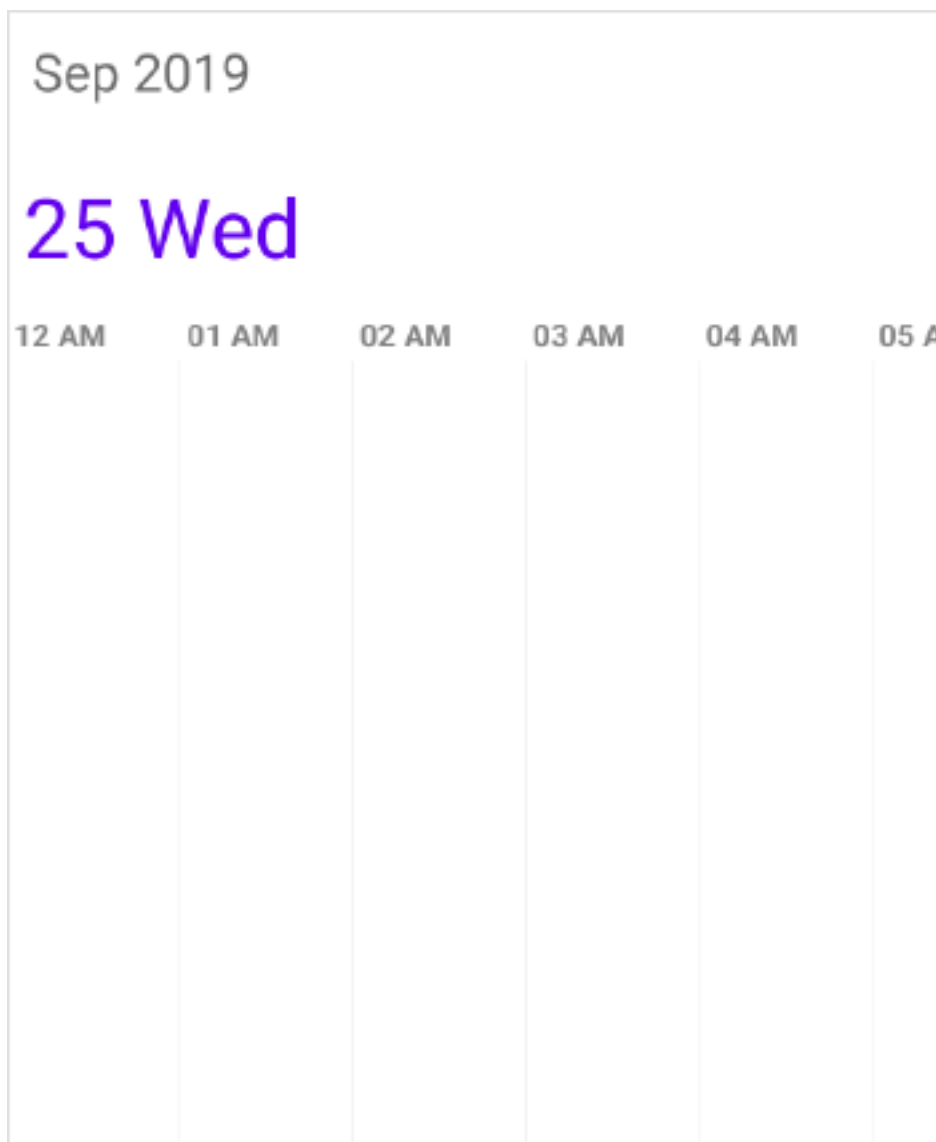
```
<schedule:SfSchedule>
  <schedule:SfSchedule.TimelineViewSettings>
    <schedule:TimelineViewSettings>
      <schedule:TimelineViewSettings.LabelSettings>
        <schedule:TimelineLabelSettings>
          <schedule:TimelineLabelSettings.DateFormat>
            <OnPlatform x:TypeArguments="x:String" iOS="d EEE" Android="d EEE"
WinPhone="%d ddd" />
          </schedule:TimelineLabelSettings.DateFormat>
        </schedule:TimelineLabelSettings>
      </schedule:TimelineViewSettings.LabelSettings>
    </schedule:TimelineViewSettings>
  </schedule:SfSchedule.TimelineViewSettings>
</schedule:SfSchedule>
```



```
</schedule:SfSchedule.TimelineViewSettings>  
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.TimelineView;  
//Creating new instance of TimelineViewSettings  
TimelineViewSettings timelineViewSettings = new TimelineViewSettings();  
//Creating new instance of TimelineLabelSettings  
TimelineLabelSettings labelSettings = new TimelineLabelSettings();  
//Customizing date format  
labelSettings.DateFormat = Device.OnPlatform("d EEE", "d EEE", "%d ddd");  
timelineViewSettings.LabelSettings = labelSettings;  
schedule.TimelineViewSettings = timelineViewSettings;
```



### Timeslot customization

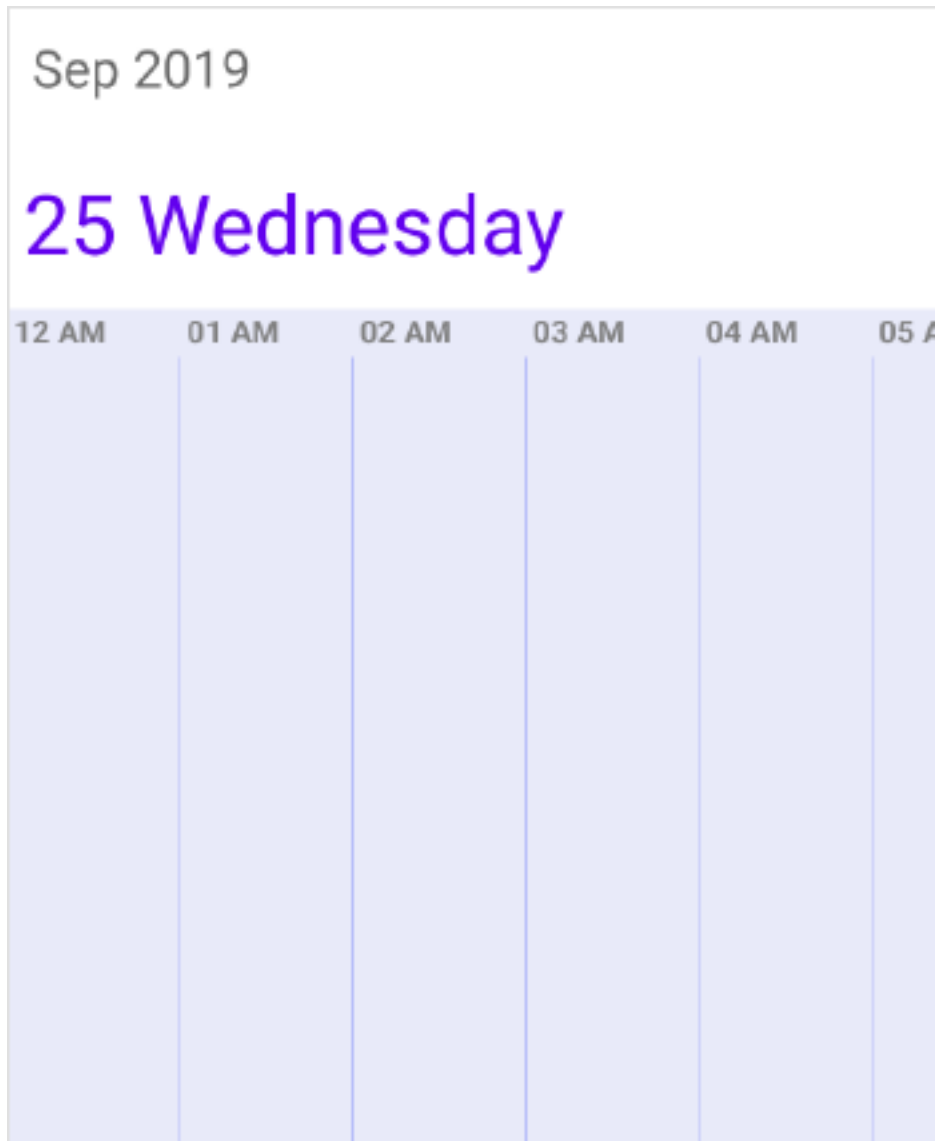
You can customize the appearance of time slots using the [Color](#), [BorderColor](#) and [BorderWidth](#) properties of `TimelineViewSettings`.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">
  <schedule:SfSchedule.TimelineViewSettings>
    <!--setting timeline view settings properties -->
    <schedule:TimelineViewSettings
      Color="#e8eaf9"
      BorderColor="#8490f9"
      BorderWidth="2" />
    </schedule:SfSchedule.TimelineViewSettings>
  </schedule:SfSchedule>
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;
//Create new instance of TimelineViewSettings
TimelineViewSettings timelineViewSettings = new TimelineViewSettings();
timelineViewSettings.BorderColor = Color.FromHex("#8490f9");
timelineViewSettings.Color = Color.FromHex("#e8eaf9");
timelineViewSettings.BorderWidth = 2;
schedule.TimelineViewSettings = timelineViewSettings;
```



#### Time label customization

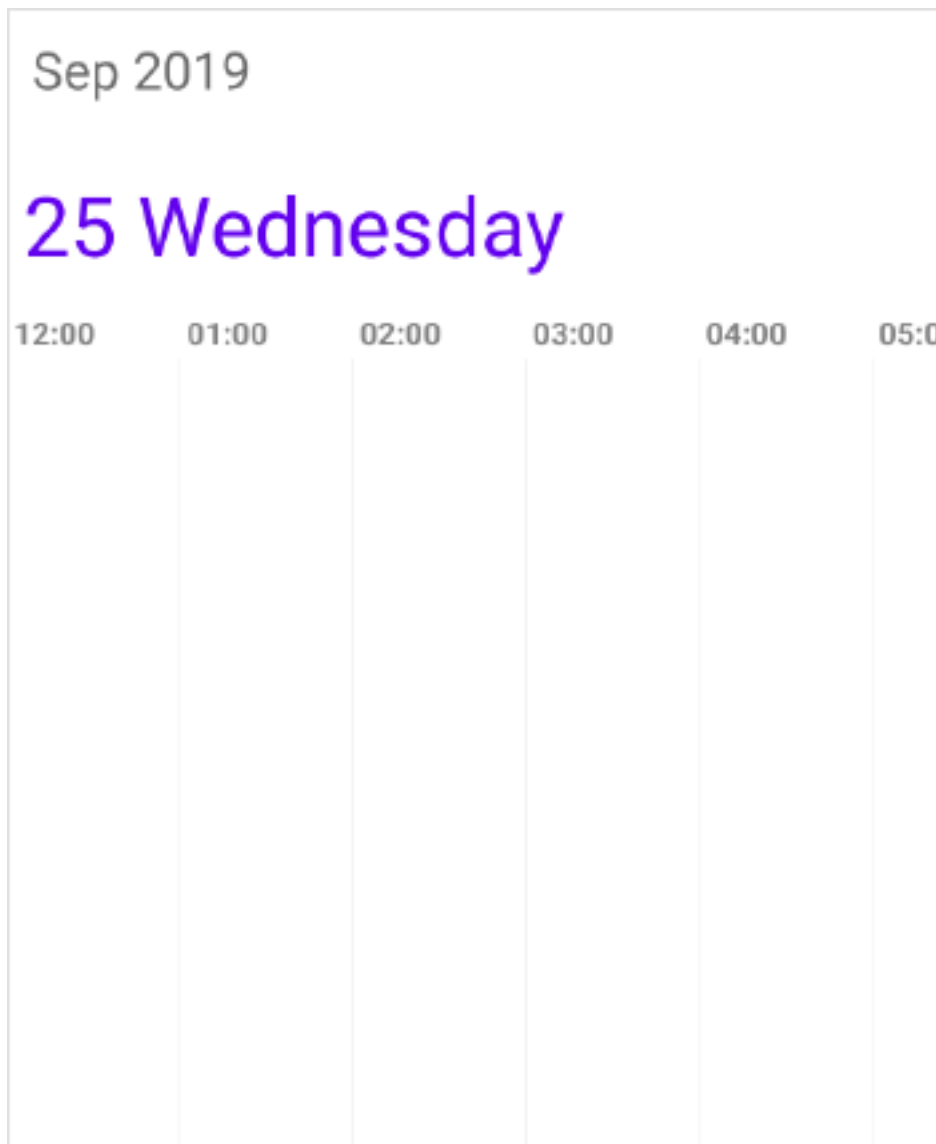
You can customize the format for the labels that mention the time by setting the [TimeFormat](#) property of [LabelSettings](#) in `TimelineViewSettings`.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">
  <schedule:SfSchedule.TimelineViewSettings>
    <!--setting time format property-->
    <schedule:TimelineViewSettings>
      <schedule:TimelineViewSettings.LabelSettings>
        <schedule:TimelineLabelSettings TimeFormat="hh:mm" />
      </schedule:TimelineViewSettings.LabelSettings>
    </schedule:TimelineViewSettings>
  </schedule:SfSchedule.TimelineViewSettings>
</schedule:SfSchedule>
```

**C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;  
//Creating new instance of TimelineViewSettings  
TimelineViewSettings timelineViewSettings = new TimelineViewSettings();  
TimelineLabelSettings labelSettings = new TimelineLabelSettings();  
labelSettings.TimeFormat = "hh:mm";  
timelineViewSettings.LabelSettings = labelSettings;  
schedule.TimelineViewSettings = timelineViewSettings;
```

*Time label appearance*

You can customize the color for the labels that mention the time by setting the [TimeLabelColor](#) property of [LabelSettings](#) in [TimelineViewSettings](#).

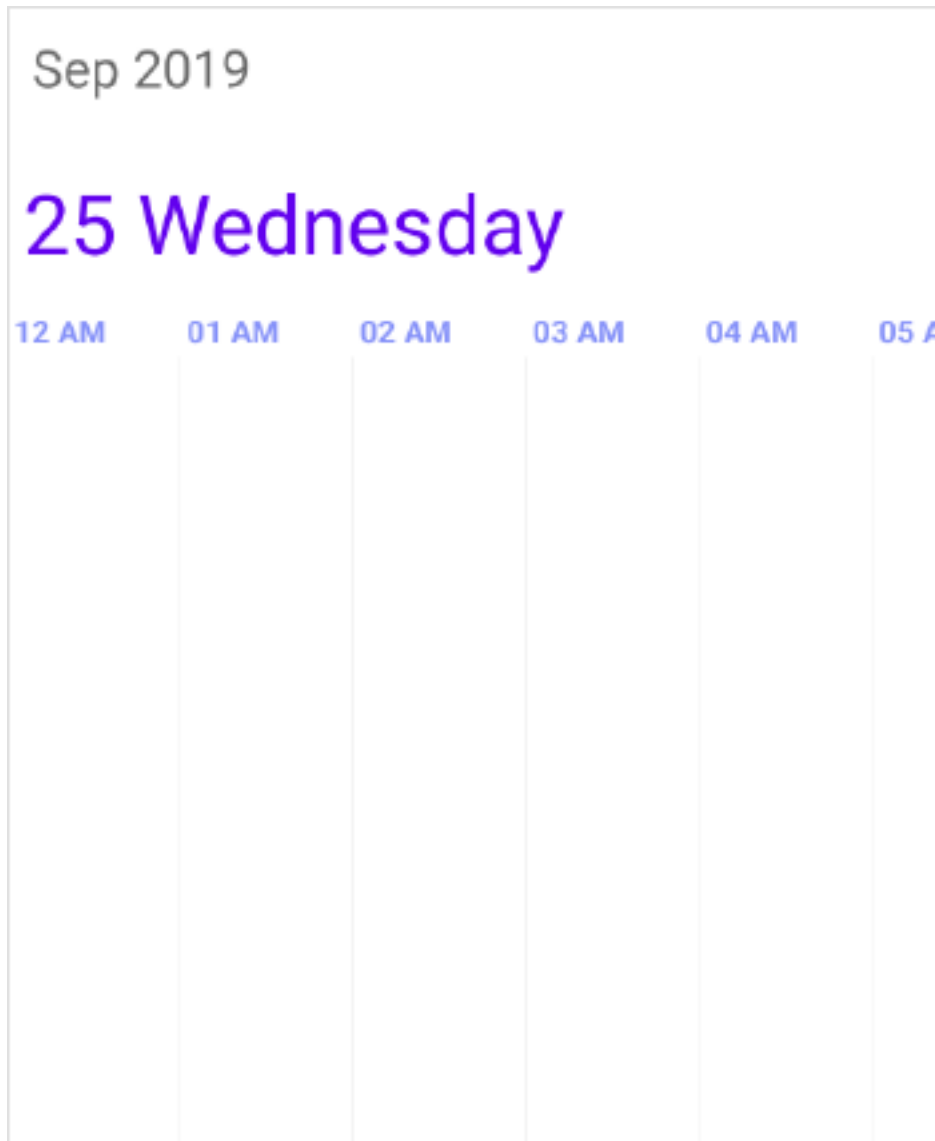
**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">  
  <schedule:SfSchedule.TimelineViewSettings>
```

```
<!--setting time format property-->
<schedule:TimelineViewSettings>
<schedule:TimelineViewSettings.LabelSettings>
<schedule:TimelineLabelSettings TimeLabelColor="#8490f9" />
</schedule:TimelineViewSettings.LabelSettings>
</schedule:TimelineViewSettings>
</schedule:SfSchedule.TimelineViewSettings>
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.TimelineView;
//Creating new instance of TimelineViewSettings
TimelineViewSettings timelineViewSettings = new TimelineViewSettings();
TimelineLabelSettings labelSettings = new TimelineLabelSettings();
labelSettings.TimeLabelColor = Color.FromHex("#8490f9");
timelineViewSettings.LabelSettings = labelSettings;
schedule.TimelineViewSettings = timelineViewSettings;
```



#### Time Label size

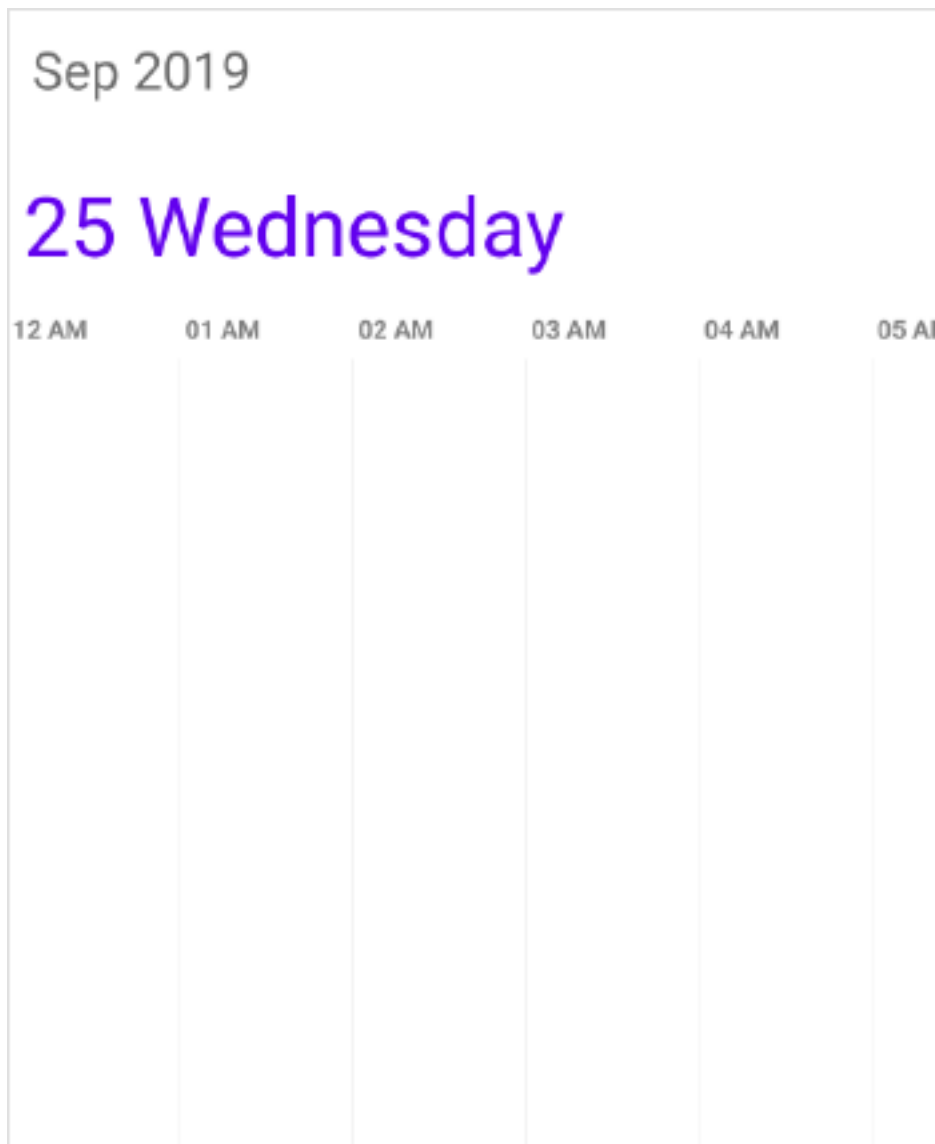
You can customize the size of the labels that mention the time by setting the [TimeLabelSize](#) property of `LabelSettings` in `TimelineViewSettings`.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">
  <schedule:SfSchedule.TimelineViewSettings>
    <!--setting time format property-->
    <schedule:TimelineViewSettings>
      <schedule:TimelineViewSettings.LabelSettings>
        <schedule:TimelineLabelSettings TimeLabelSize="15" />
      </schedule:TimelineViewSettings.LabelSettings>
    </schedule:TimelineViewSettings>
  </schedule:SfSchedule.TimelineViewSettings>
</schedule:SfSchedule>
```

**C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;  
//Creating new instance of TimelineViewSettings  
TimelineViewSettings timelineViewSettings = new TimelineViewSettings();  
TimelineLabelSettings labelSettings = new TimelineLabelSettings();  
labelSettings.TimeLabelSize = 10;  
timelineViewSettings.LabelSettings = labelSettings;  
schedule.TimelineViewSettings = timelineViewSettings;
```

**Selection**

You can customize the default appearance of selection UI in the timeslots.

- [Selection customization using style](#)
- [Selection customization using custom View](#)
- [Programmatic selection](#)

### *Selection customization using style*

You can customize the timeslot selection by using [SelectionMode](#) property of SfSchedule.

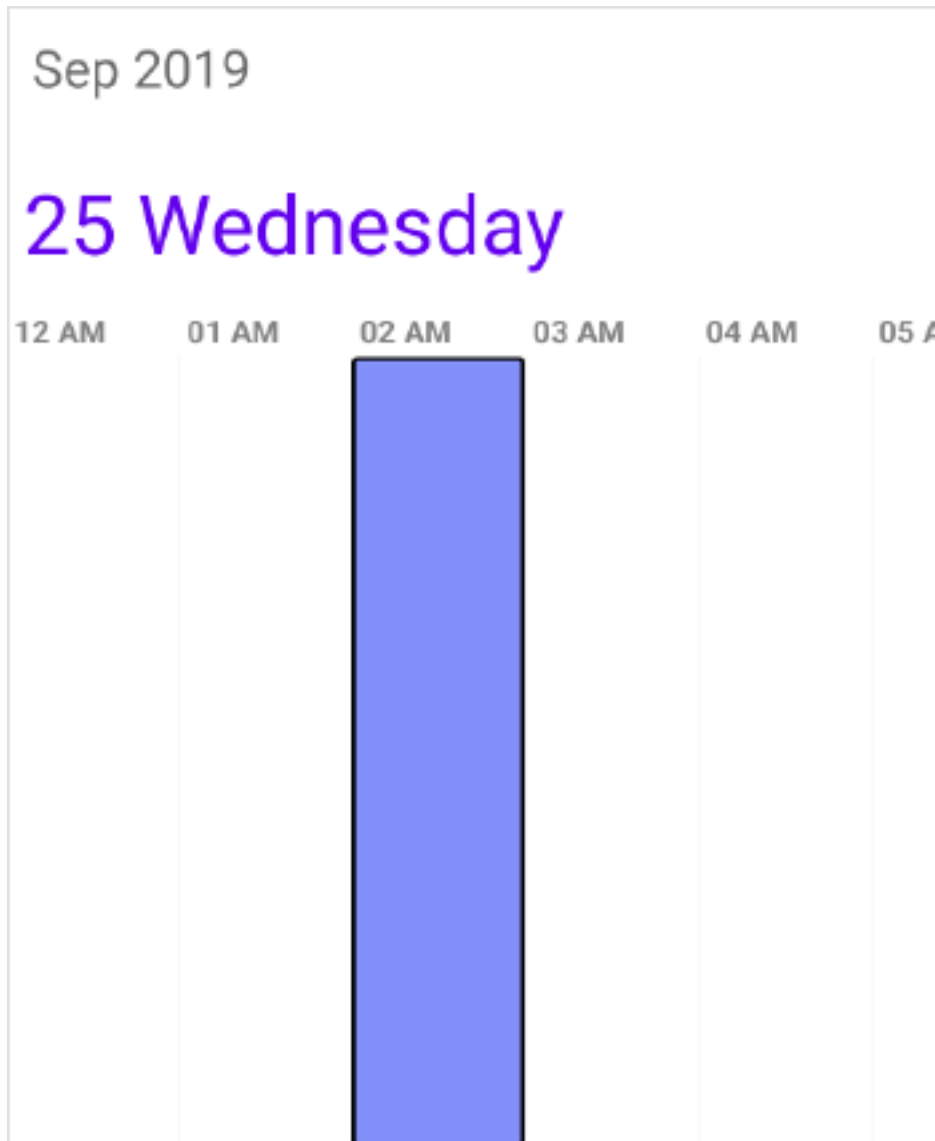
#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">
  <schedule:SfSchedule.SelectionStyle>
    <schedule:SelectionMode
      BackgroundColor="#8490f9"
      BorderColor="Black"
      BorderThickness="5"
      BorderCornerRadius="5">
    </schedule:SelectionMode>
  </schedule:SfSchedule.SelectionStyle>
</schedule:SfSchedule>
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;
//Create new instance of SelectionStyle
SelectionMode selectionStyle = new SelectionStyle();
selectionStyle.BackgroundColor = Color.FromHex("#8490f9");
selectionStyle.BorderColor = Color.Black;
selectionStyle.BorderThickness = 5;
selectionStyle.BorderCornerRadius = 5;
schedule.SelectionStyle = selectionStyle;
```





#### *Selection customization using custom View*

You can replace the default selection UI with your custom view by setting [SelectionView](#) property of SfSchedule.

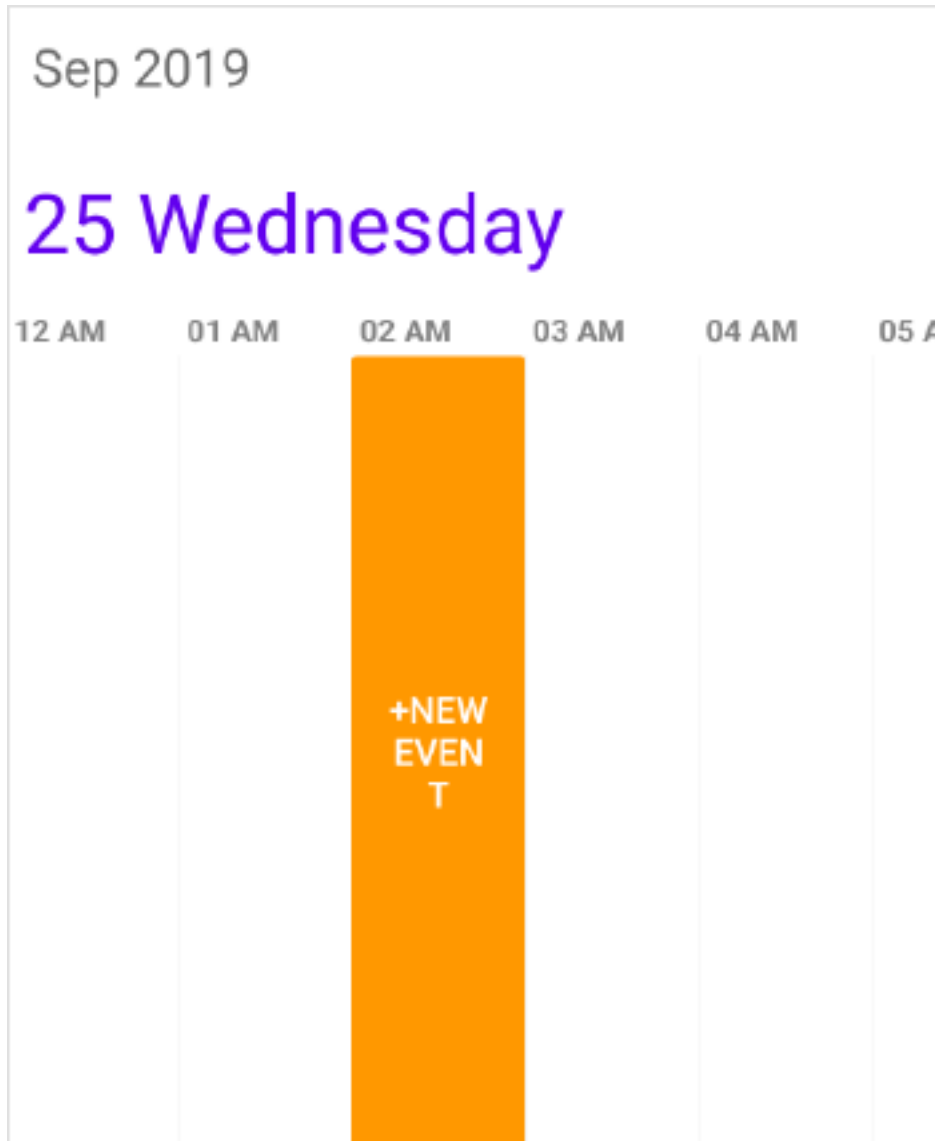
#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="TimelineView">
  <schedule:SfSchedule.SelectionView>
    <Button
      BackgroundColor="#FF9800"
      Text="+NewEvent"
      TextColor="White"/>
  </schedule:SfSchedule.SelectionView>
</schedule:SfSchedule>
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.TimelineView;
```

```
//Add the CustomView
Button customView = new Button();
customView.Text = "+NewEvent";
customView.BackgroundColor = Color.FromHex("#FF9800");
customView.TextColor = Color.White;
schedule.SelectionView = customView;
```



#### *Programmatic selection*

You can programmatically select the specific timeslot by setting corresponding date and time value to [SelectedDate](#) property of [SfSchedule](#). By default, it is null.

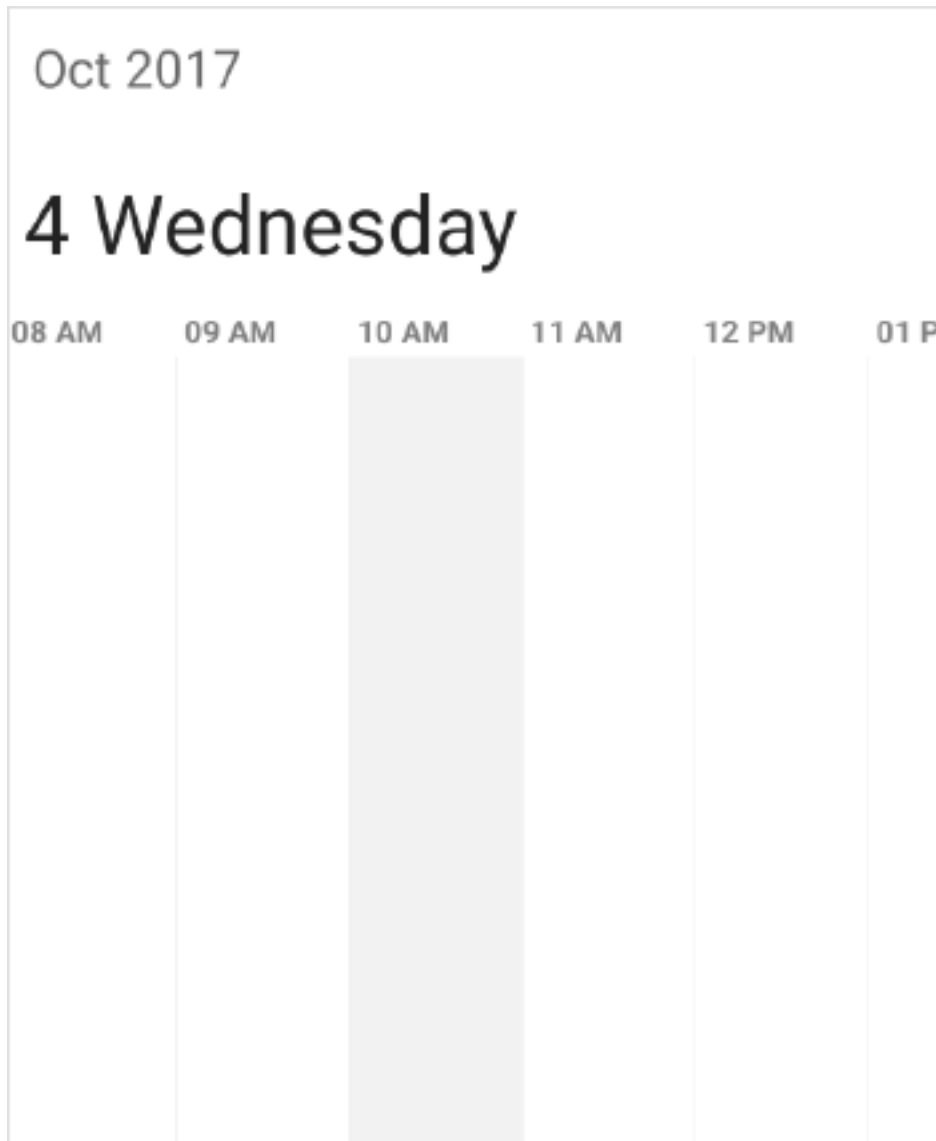
#### **C#**

```
// Setting a date and time to select
schedule.SelectedDate = new DateTime(2017, 10, 04, 10, 0, 0);
```

You can clear the selection by setting [SelectedDate](#) as null.

**C#**

```
// Setting null value to deselect  
schedule.SelectedDate = null;
```

**NOTE**

- SfSchedule does not support multiple selection.

**Month View**

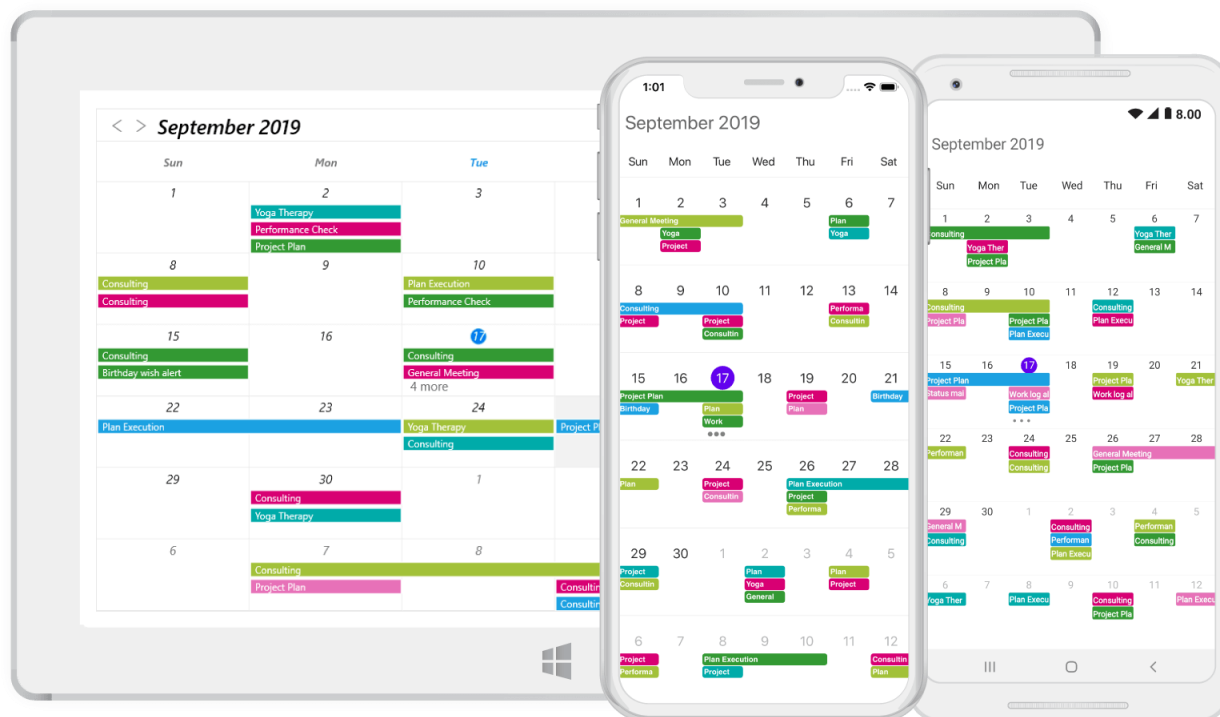
MonthView of SfSchedule used to display entire dates of the specific month, current month will be displayed by default initially. Current date color is differentiated with other dates of the current month, also the color differentiation for dates will be applicable for previous and next month dates.

**XML**

```
<schedule:SfSchedule x:Name="schedule"
ScheduleView="MonthView">
</schedule:SfSchedule>
```

## C#

```
//setting schedule view
schedule.ScheduleView = ScheduleView.MonthView;
```



### Month Appointment indicator

In **MonthView**, appointments are not viewed in the month cell instead appointment indicators are drawn. You can customize the number of appointment indicators displayed in month cell using [AppointmentIndicatorCount](#) property of [MonthViewSettings](#) in **SfSchedule**, by default Appointment indicator count is 3.

## XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
<schedule:SfSchedule.MonthViewSettings>
<schedule:MonthViewSettings
AppointmentIndicatorCount = "2" >
</schedule:MonthViewSettings>
</schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>
```

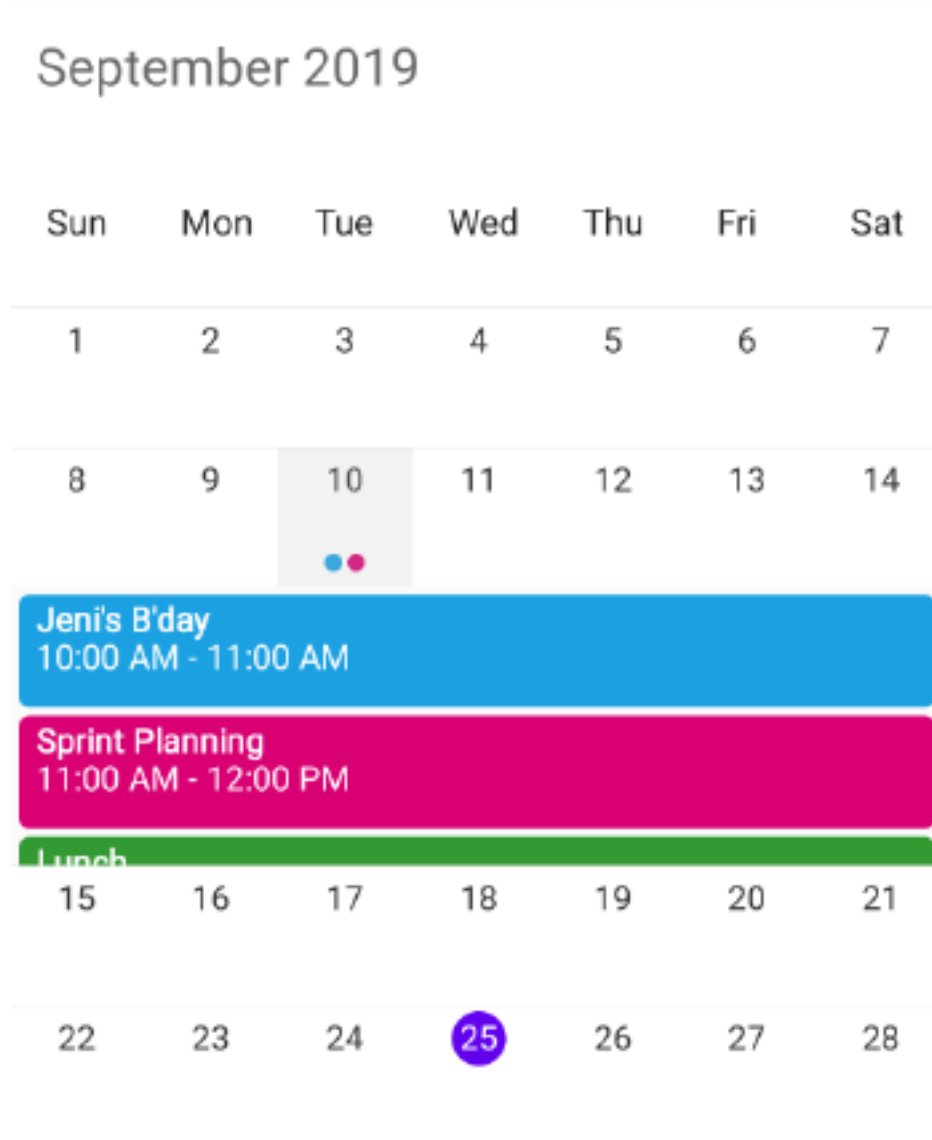
## C#

```
//creating new instance for MonthViewSettings
MonthViewSettings monthViewSettings = new MonthViewSettings();
```

```
monthViewSettings.AppointmentIndicatorCount = 2;
schedule.MonthViewSettings = monthViewSettings;
```

**NOTE**

If appointments count are lesser than the AppointmentIndicatorCount value in the particular day, then according to number of appointments available, indicator will be displayed in the month cell. Maximum number of appointment indicators drawn in the month cell is 6 in android and ios platforms.

**Month Appointment display mode**

You can handle the schedule month view appointment display by using [AppointmentDisplayMode](#) property of `MonthViewSettings`. By default `AppointmentDisplayMode` is set as `Indicator`, using the `AppointmentDisplayMode` you can set the month view appointments display as follows.

**Indicator** - appointment will be denoted as the circle.

**Appointment** - appointment subject will be displayed in month cell.

**None** - appointment will not be displayed.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
  <schedule:SfSchedule.MonthViewSettings>
    <schedule:MonthViewSettings
      AppointmentDisplayMode="Appointment" >
    </schedule:MonthViewSettings>
  </schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>
```

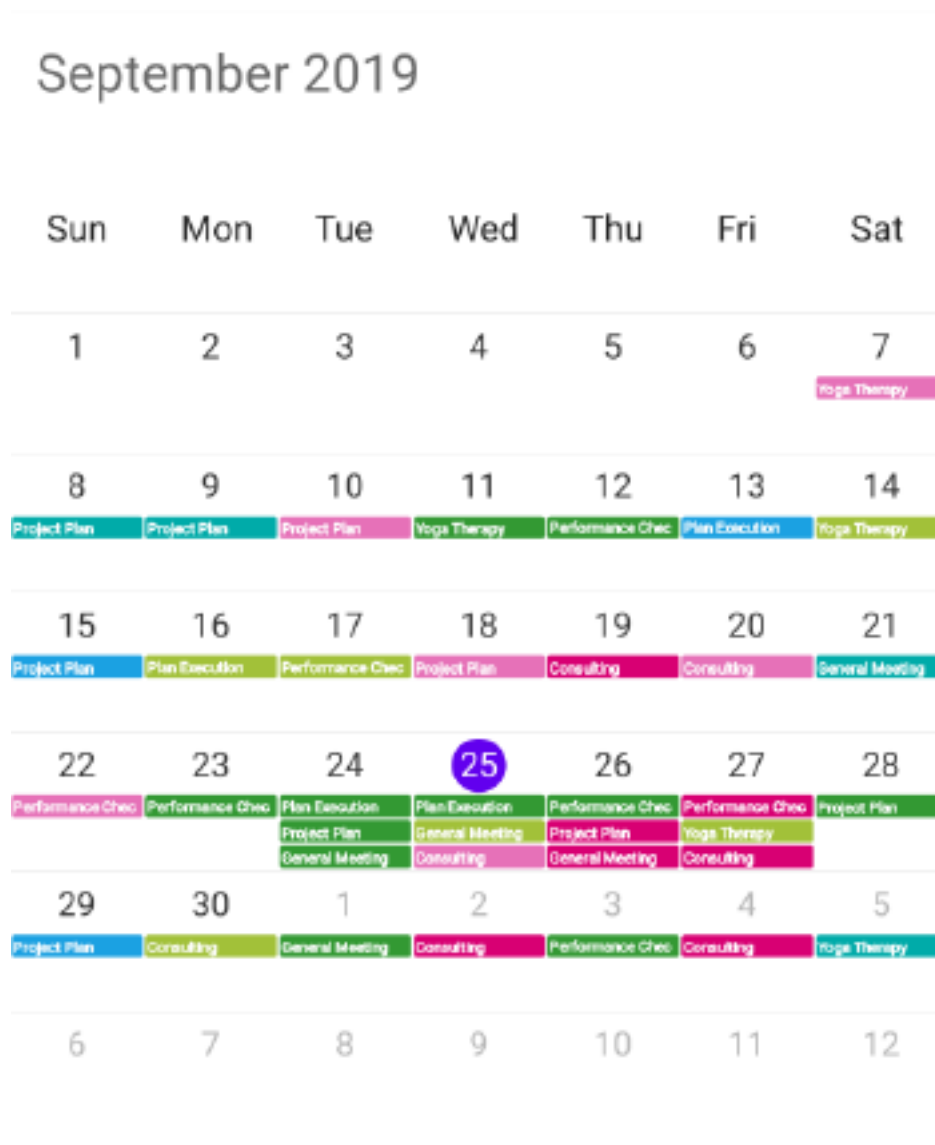
#### **C#**

```
MonthViewSettings monthViewSettings = new MonthViewSettings();
monthViewSettings.AppointmentDisplayMode =
AppointmentDisplayMode.Appointment;
schedule.MonthViewSettings = monthViewSettings;
```

---

#### **NOTE**

**AppointmentDisplayMode.None** is applicable only for XForms UWP and WPF platforms.

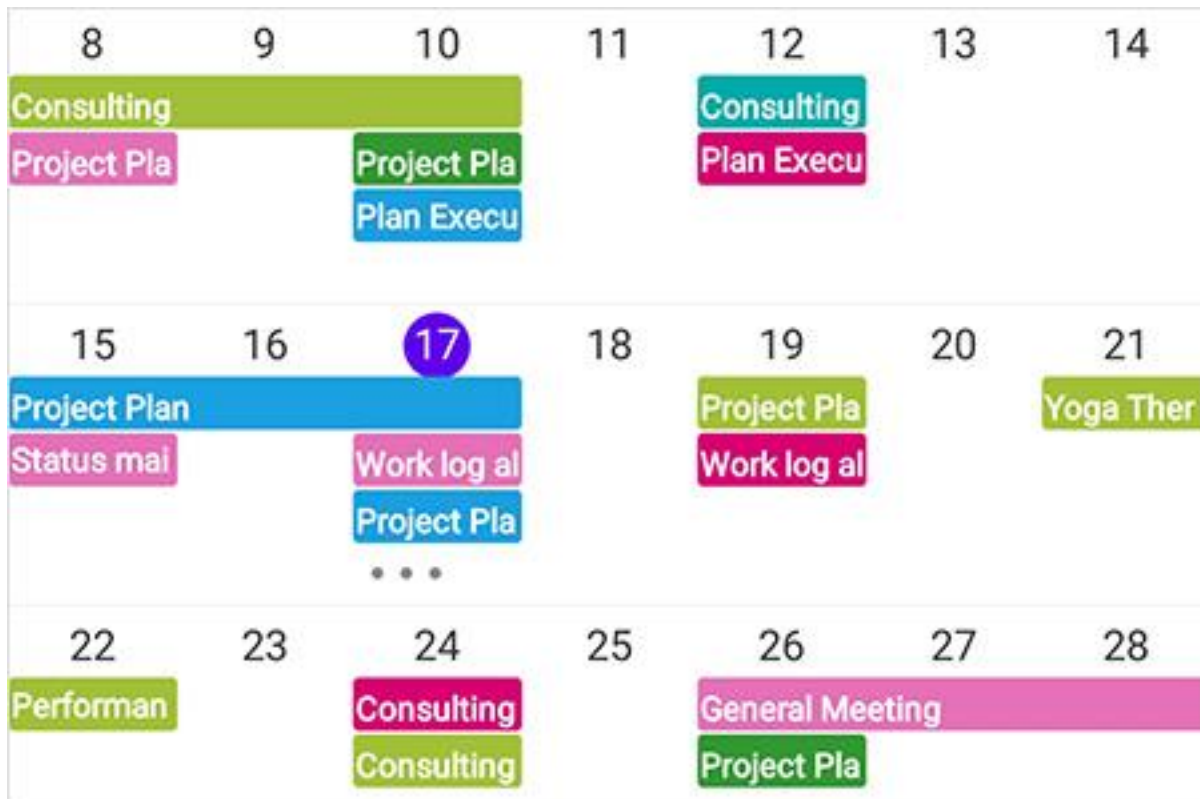


### Month Appointment display count

[AppointmentDisplayCount](#) or [AppointmentIndicatorCount](#) is used to define the maximum number of appointments to be displayed in a month cell in month view. If [AppointmentDisplayCount](#) or [AppointmentIndicatorCount](#) value is 1 and the month cell have more than 1 appointments, single appointment will be displayed and remaining appointments in month cell will be displayed as more appointments.

### NOTE

- The [AppointmentIndicatorCount](#) support is applicable for XForms Android and iOS platforms.
- The [AppointmentDisplayCount](#) support is applicable for XForms UWP and WPF platforms.
- By clicking more option, schedule navigates to the day view by default in XForms UWP and WPF platforms.



#### Disable navigation to DayView

You can disable the navigation to day view by triggering the `CellTappedEvent` and set the argument `CancelNavigation` value as `true`. `IsMoreAppointments` argument is used to determine whether the month cell more appointments count element has been tapped or not while the appointment display mode as Appointment.

#### C#

```
monthViewSettings.AppointmentDisplayCount = 4;
monthViewSettings.AppointmentDisplayMode =
AppointmentDisplayMode.Appointment;
schedule.CellTapped += Schedule_CellTapped;
private void Schedule_CellTapped(object sender, CellTappedEventArgs e)
{
    IsMoreElementTapped = e.IsMoreAppointments;
    e.CancelNavigation = true;
}
```

#### NOTE

`CancelNavigation` and `IsMoreAppointments` arguments applicable only for XForms UWP and WPF platform.

#### Month InlineView

You can use `ShowAppointmentsInline` bool property in `SfSchedule` to enable / disable the month inline view, by setting `ShowAppointmentsInline` property as `true` you can view the Appointments in the specific date.



**XML**

```
<schedule:SfSchedule
x:Name="schedule"
ScheduleView="MonthView"
ShowAppointmentsInline="true">
</schedule:SfSchedule>
```

**C#**

```
schedule.ShowAppointmentsInline = true;
```

## September 2019

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
<div>Jeni's B'day 10:00 AM - 11:00 AM</div> <div>Meeting 11:00 AM - 12:00 PM</div>						
15	16	17	18	19	20	21
22	23	24	25	26	27	28

**NOTE**

If appointments not there in the selected day, Inline view displays the text as "No Events"

### Agenda View

The Schedule month view displays a divided agenda view which is used to show the selected date's appointments below the month. You can show agenda view by setting [ShowAgendaView](#) property as true.

#### **XML**

```
<schedule:SfSchedule ScheduleView="MonthView">
  <schedule:SfSchedule.MonthViewSettings>
    <schedule:MonthViewSettings ShowAgendaView="true" />
  </schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>
```

#### **C#**

```
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.MonthView;
MonthViewSettings monthViewSettings = new MonthViewSettings();
monthViewSettings.ShowAgendaView = true;
schedule.MonthViewSettings = monthViewSettings;
```

## September 2019

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

10  
Tue

Jeni's B'day  
10:00:AM - 11:00:AM

Meeting  
11:00:AM - 12:00:PM

### NOTE

- An agenda view displays text as “No Selected Date” until no date is selected.
- If there is no appointment in a selected day, agenda view displays the text as “No Events”.
- If you enable ShowAgendaView and ShowAppointmentsInline properties together, both of the views (Agenda View and Appointment Inline View) will be displayed in schedule month view.

### Agenda View Appearance

You can customize the Agenda view appointment and Selected Date Text by setting [AgendaViewStyle](#) property of MonthViewSettings. Agenda view [DateFontColor](#), [HeaderHeight](#), [DateFormat](#), [DateFontAttributes](#), [DateFontSize](#), [DateFontFamily](#), [TimeFontColor](#), [TimeFontSize](#), [TimeFontAttributes](#), [TimeFontFamily](#), [TimeFormat](#), [SubjectFontColor](#), [SubjectFontSize](#), [SubjectFontFamily](#), [SubjectFontAttributes](#), [BackgroundColor](#) can be customized using AgendaViewStyle properties.

### XML

```

<schedule:SfSchedule ScheduleView="MonthView" x:Name="schedule">
  <schedule:SfSchedule.MonthViewSettings>
    <schedule:MonthViewSettings ShowAgendaView="true">
      <schedule:MonthViewSettings.AgendaViewStyle>
        <schedule:AgendaViewStyle DateFontColor="Purple" HeaderHeight="40"
DateFormat="dd MMMM, yyyy" DateFontAttributes="Bold" DateFontSize="15"
DateFontFamily="Arial" TimeFontColor="Red" TimeFontSize="13"
TimeFontFamily="Arial" TimeFormat="hh a"
TimeFontAttributes="Bold" BackgroundColor="#DEF0DE" SubjectFontColor="Blue"
SubjectFontSize="13" SubjectFontFamily="Arial" SubjectFontAttributes="Bold"
/>
      </schedule:MonthViewSettings.AgendaViewStyle>
    </schedule:MonthViewSettings>
  </schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>

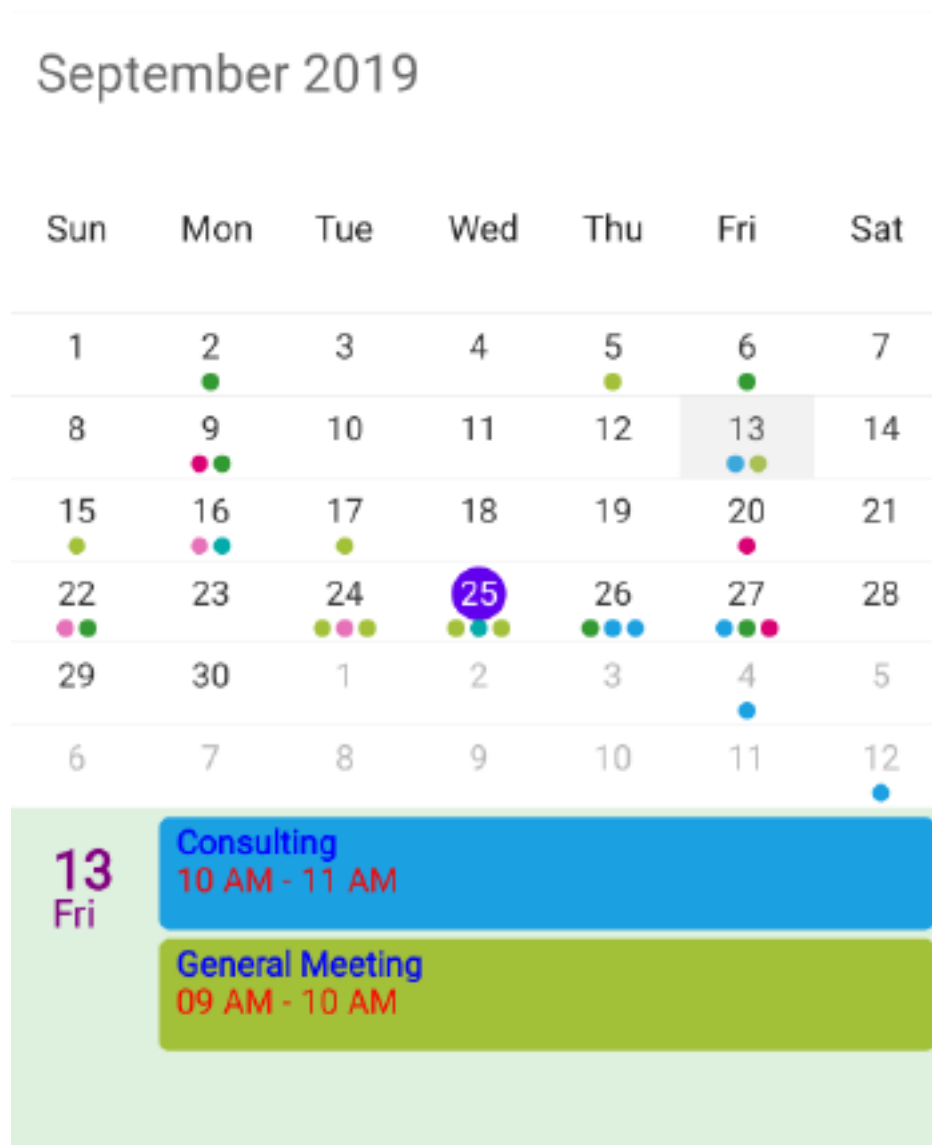
```

**C#**

```

SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.MonthView;
MonthViewSettings monthViewSettings = new MonthViewSettings();
monthViewSettings.ShowAgendaView = true;
schedule.MonthViewSettings = monthViewSettings;
AgendaViewStyle agendaViewStyle = new AgendaViewStyle();
// Customize selected Date Text
agendaViewStyle.DateFontColor = Color.Purple;
agendaViewStyle.HeaderHeight = 40;
agendaViewStyle.DateFormat = "dd MMMM, yyyy";
agendaViewStyle.DateFontAttributes = FontAttributes.Bold;
agendaViewStyle.DateFontSize = 15;
agendaViewStyle.DateFontFamily = "Arial";
// Customize appointment
agendaViewStyle.TimeFontColor = Color.Red;
agendaViewStyle.TimeFontSize = 13;
agendaViewStyle.TimeFontAttributes = FontAttributes.None;
agendaViewStyle.TimeFontFamily = "Arial";
agendaViewStyle.TimeFormat = "hh a";
agendaViewStyle.SubjectFontColor = Color.Blue;
agendaViewStyle.SubjectFontSize = 13;
agendaViewStyle.SubjectFontFamily = "Arial";
agendaViewStyle.SubjectFontAttributes = FontAttributes.None;
agendaViewStyle.BackgroundColor = Color.FromRgb(222, 240, 222);
schedule.MonthViewSettings.AgendaViewStyle = agendaViewStyle;

```

**NOTE**

Agenda View Appearance customization is not applicable for UWP platform.

*Agenda Item Template*

The default appearance of the Appointment can be customized by using the [AgendaItemTemplate](#) property of the `MonthViewSettings`.

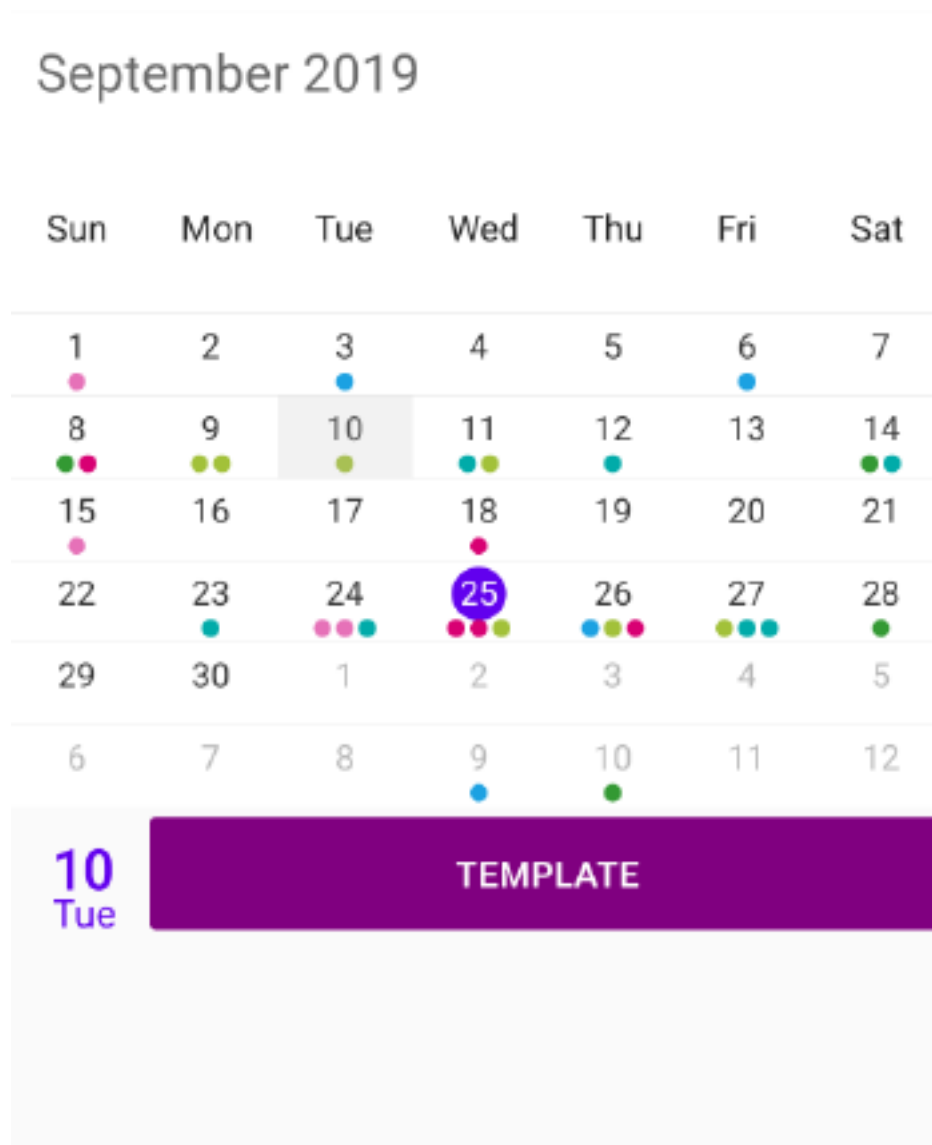
**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
  <schedule:SfSchedule.MonthViewSettings>
    <schedule:MonthViewSettings ShowAgendaView="True">
      <schedule:MonthViewSettings.AgendaItemTemplate>
        <DataTemplate>
          <Button BackgroundColor="Purple" Text="Template" TextColor="White" />
        </DataTemplate>
      </schedule:MonthViewSettings.AgendaItemTemplate>
    </schedule:MonthViewSettings>
  </schedule:SfSchedule>
```

```
</schedule:SfSchedule.MonthViewSettings>  
</schedule:SfSchedule>
```

### **C#**

```
SfSchedule schedule = new SfSchedule();  
schedule.ScheduleView = ScheduleView.MonthView;  
MonthViewSettings monthViewSettings = new MonthViewSettings();  
monthViewSettings.ShowAgendaView = true;  
monthViewSettings.AgendaItemTemplate = new DataTemplate(() =>  
{  
    return new Button  
    {  
        Text = "Template",  
        TextColor = Color.White,  
        BackgroundColor = Color.Purple  
    };  
});  
schedule.MonthViewSettings = monthViewSettings;
```

**NOTE**

Agenda item template support is not applicable for UWP platform.

### [Agenda View Using Template Selector](#)

AgendaTemplateSelector can be used to choose a DataTemplate at runtime based on the value of a data-bound to agenda appointment property through AgendaiItemTemplate. It lets you choose a different data template for each appointment, customizing the appearance of a particular inline appointment based on certain conditions. DataTemplateSelector for inline appointment includes ScheduleAppointment or custom appointment as object and Schedule as bindable object.

**XML**

```
<ContentPage.Resources>
  <ResourceDictionary>
    <local:AgendaTemplateSelector x:Key="AgendaTemplateSelector" />
  </ResourceDictionary>
</ContentPage.Resources>
```

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
  <schedule:SfSchedule.MonthViewSettings>
    <schedule:MonthViewSettings ShowAgendaView="True"
    AgendaItemTemplate="{StaticResource AgendaTemplateSelector}">
  </schedule:MonthViewSettings>
</schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>
```

**C#**

```
public class AgendaTemplateSelector : DataTemplateSelector
{
    public DataTemplate DayAppointmentTemplate { get; set; }
    public DataTemplate AllDayAppointmentTemplate { get; set; }
    public AgendaTemplateSelector()
    {
        DayAppointmentTemplate = new DataTemplate(typeof(DayTemplate));
        AllDayAppointmentTemplate = new DataTemplate(typeof(AllDayTemplate));
    }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject container)
    {
        var schedule = (container as SfSchedule);
        if (schedule == null) return null;
        if ((item as ScheduleAppointment).IsAllDay)
            return AllDayAppointmentTemplate;
        else
            return DayAppointmentTemplate;
    }
}
```

Used button to display day appointment and all day appointment.

**XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<Button
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="SchPreview.DayTemplate"
  HorizontalOptions="FillAndExpand"
  VerticalOptions="FillAndExpand"
  BackgroundColor="{Binding Color}"
  Text="{Binding Subject}"
  FontAttributes="Bold"
  FontSize="15"
  TextColor="White">
</Button>
```

**XML**

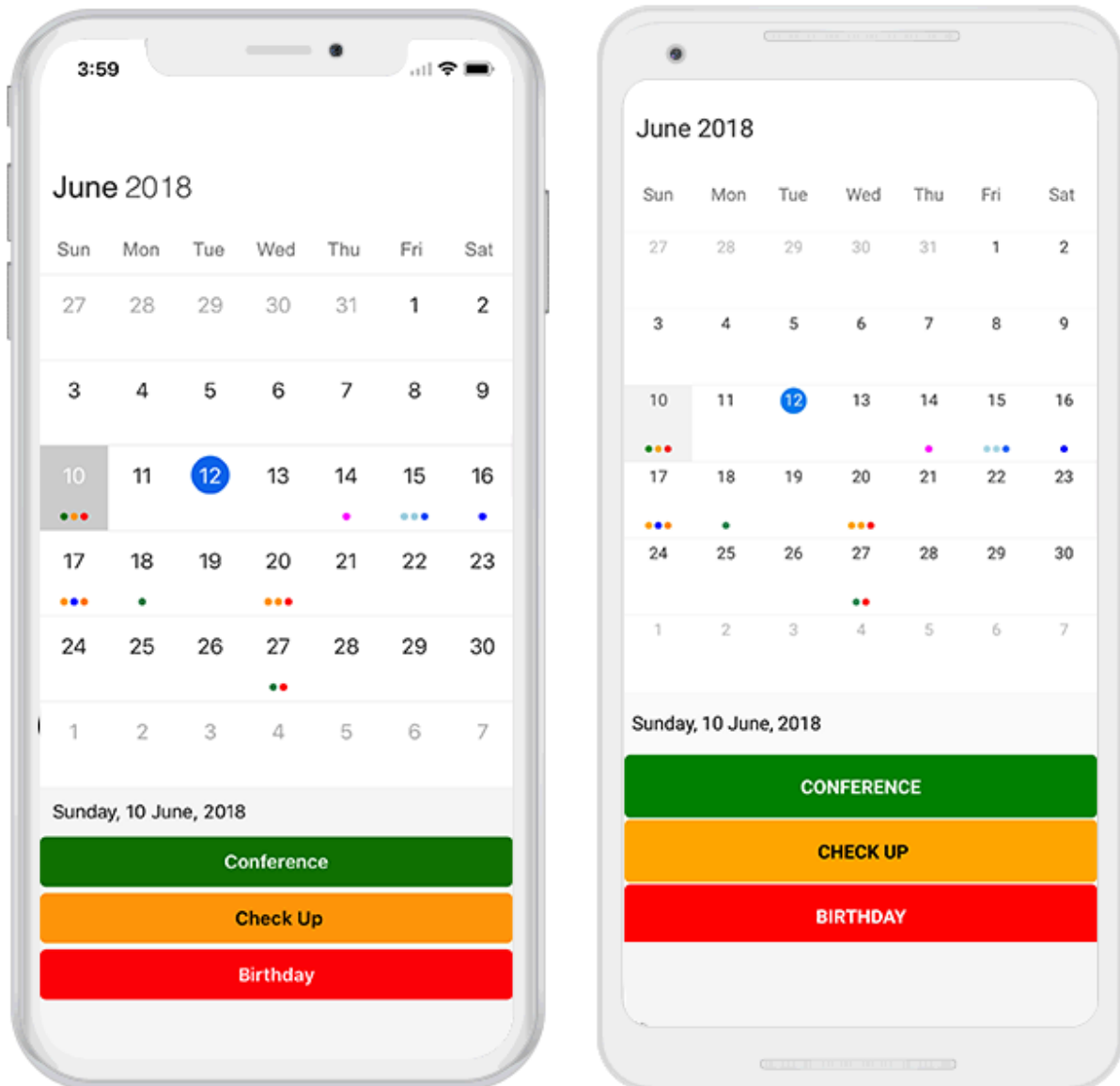
```
<?xml version="1.0" encoding="UTF-8"?>
<Button
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```



```

x:Class="SchPreview.AllDayTemplate"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand"
BackgroundColor="{Binding Color}"
Text="{Binding Subject}"
FontAttributes="Bold"
FontSize="15"
TextColor="Black">
</Button>

```



## NOTE

Agenda item template selector support is not applicable for UWP platform.

### Month Navigation direction

MonthView of Schedule can be navigated in both horizontal and vertical direction. You can change the direction of navigation through [MonthNavigationDirection](#) property of [MonthViewSettings](#) in SfSchedule, by default Month navigation direction is [Horizontal](#).

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
  <schedule:SfSchedule.MonthViewSettings>
    <schedule:MonthViewSettings
      MonthNavigationDirection="Vertical">
    </schedule:MonthViewSettings>
  </schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>
```

#### C#

```
//creating new instance for MonthViewSettings
MonthViewSettings monthViewSettings = new MonthViewSettings();
//To navigate vertically
monthViewSettings.MonthNavigationDirection =
MonthNavigationDirections.Vertical;
schedule.MonthViewSettings = monthViewSettings;
```

### Restricted days in Month

You can disable the interaction for certain date in Month view by using [BlackoutDates](#) of [MonthViewSettings](#), using this you can allocate / restrict the specific date for predefined events.

#### C#

```
//creating new instance for MonthViewSettings
MonthViewSettings monthViewSettings = new MonthViewSettings();
ObservableCollection<DateTime> blackoutDateCollection = new
ObservableCollection<DateTime>();
DateTime blockedDate1 = DateTime.Now.Date.AddDays(1);
DateTime blockedDate2 = DateTime.Now.Date.AddDays(2);
DateTime blockedDate3 = DateTime.Now.Date.AddDays(3);
DateTime blockedDate4 = DateTime.Now.Date.AddDays(4);
DateTime blockedDate5 = DateTime.Now.Date.AddDays(5);
blackoutDateCollection.Add(blockedDate1);
blackoutDateCollection.Add(blockedDate2);
blackoutDateCollection.Add(blockedDate3);
blackoutDateCollection.Add(blockedDate4);
blackoutDateCollection.Add(blockedDate5);
monthViewSettings.BlackoutDates = blackoutDateCollection;
schedule.MonthViewSettings = monthViewSettings;
```

## September 2019

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

### First day of Week in Month

You can set First day of week using [FirstDayOfWeek](#) property of `SfSchedule`, by default schedule control will rendered with **Sunday** as the first day of the week.

#### XML

```
<schedule:SfSchedule
x:Name="schedule"
FirstDayOfWeek="3"
ScheduleView="MonthView">
</schedule:SfSchedule>
```

#### C#

```
//setting FirstDayOfWeek
schedule.FirstDayOfWeek = 3; // Tuesday
```

## September 2019

Tue	Wed	Thu	Fri	Sat	Sun	Mon
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

### Week Number of the Year in Month

You can display the Week Number of the year in `MonthView` by setting `ShowWeekNumber` property of `MonthViewSettings` as `true`, by default it is `false`.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
  <schedule:SfSchedule.MonthViewSettings>
    <schedule:MonthViewSettings
      ShowWeekNumber="true">
    </schedule:MonthViewSettings>
  </schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>
```

#### C#

```
monthViewSettings.ShowWeekNumber = true;
```

## September 2019

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
36	1	2	3	4	5	6	7
37	8	9	10	11	12	13	14
38	15	16	17	18	19	20	21
39	22	23	24	25	26	27	28
40	29	30	1	2	3	4	5
41	6	7	8	9	10	11	12

### Week Number Appearance

You can customize the Week Number appearance by using [WeekNumberStyle](#) property of [MonthViewSettings](#). Week number [BackgroundColor](#), [TextColor](#), [FontFamily](#), [FontSize](#), [FontAttributes](#) can be customized using [WeekNumberStyle](#) properties.

### C#

```
//creating new instance for WeekNumberStyle
WeekNumberStyle weekNumberStyle = new WeekNumberStyle();
weekNumberStyle.FontFamily = "Arial";
weekNumberStyle.FontSize = 15;
weekNumberStyle.FontAttributes = FontAttributes.None;
weekNumberStyle.BackgroundColor = Color.Blue;
weekNumberStyle.TextColor = Color.White;
monthViewSettings.WeekNumberStyle = weekNumberStyle;
```

## September 2019

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
36	1	2	3	4	5	6	7
37	8	9	10	11	12	13	14
38	15	16	17	18	19	20	21
39	22	23	24	25	26	27	28
40	29	30	1	2	3	4	5
41	6	7	8	9	10	11	12

### View Header Appearance

You can customize the View Header appearance by using [ViewHeaderStyle](#) property in SfSchedule. View Header [BackgroundColor](#), [DayTextColor](#) and [DayFontFamily](#) can be customized using [ViewHeaderStyle](#) properties.

### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
  <schedule:SfSchedule.ViewHeaderStyle>
    <schedule:ViewHeaderStyle
      BackgroundColor="Blue"
      DayTextColor="White"
      DayFontFamily="Arial">
    </schedule:ViewHeaderStyle>
  </schedule:SfSchedule.ViewHeaderStyle>
</schedule:SfSchedule>
```

**C#**

```
//creating new instance for viewHeaderStyle
ViewHeaderStyle viewHeaderStyle = new ViewHeaderStyle();
viewHeaderStyle.BackgroundColor = Color.Blue;
viewHeaderStyle.DayTextColor = Color.White;
viewHeaderStyle.DayFontFamily = "Arial";
schedule.ViewHeaderStyle = viewHeaderStyle;
```

## September 2019

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

**NOTE**

FontAttributes and FontFamily are native to the platform. Custom font and the font which are not available in the specified platform will not be applied.

### *ViewHeader Date Format*

You can customize the date and day format of SfSchedule ViewHeader by using [DateFormat](#) and [DayFormat](#) properties of [MonthLabelSettings](#).

## XML

```
<schedule:SfSchedule>
<schedule:SfSchedule.MonthViewSettings>
<schedule:MonthViewSettings>
<schedule:MonthViewSettings.MonthLabelSettings>
<schedule:MonthLabelSettings DateFormat="dd">
<schedule:MonthLabelSettings.DayFormat>
<OnPlatform x:TypeArguments="x:String"
iOS="EEEE"
Android="EEEE"
WinPhone="dddd" />
</schedule:MonthLabelSettings.DayFormat>
</schedule:MonthLabelSettings>
</schedule:MonthViewSettings.MonthLabelSettings>
</schedule:MonthViewSettings>
</schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>
```

## C#

```
schedule.ScheduleView = ScheduleView.MonthView;
//Creating new instance of MonthViewSettings
MonthViewSettings monthViewSettings = new MonthViewSettings();
//Creating new instance of MonthLabelSettings
MonthLabelSettings monthLabelSettings = new MonthLabelSettings();
//Customizing date format
monthLabelSettings.DateFormat = "dd";
monthLabelSettings.DayFormat = Device.OnPlatform("EEEE", "EEEE", "dddd");
monthViewSettings.MonthLabelSettings = monthLabelSettings;
schedule.MonthViewSettings = monthViewSettings;
```



## September 2019

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
06	07	08	09	10	11	12

### *ViewHeader Tapped Event*

You can handle single tap action of ViewHeader by using [ViewHeaderTapped](#) event of **SfSchedule**. This event will be triggered when the ViewHeader is Tapped. This event contains [ViewHeaderTappedEventArgs](#) argument which holds [DateTime](#) details in it.

### **XML**

```
<schedule:SfSchedule x:Name="schedule"
ScheduleView="MonthView"
ViewHeaderTapped="Handle_ViewHeaderTapped" >
</schedule:SfSchedule>
```

### **C#**

```
//Creating new instance of Schedule
SfSchedule schedule = new SfSchedule();
schedule.ScheduleView = ScheduleView.MonthView;
schedule.ViewHeaderTapped += Handle_ViewHeaderTapped;
```

## C#

```
private void Handle_ViewHeaderTapped(object sender,
ViewHeaderTappedEventArgs e)
{
    var dateTime = e.DateTime;
}
```

### MonthCell Appearance

You can customize the Month view cell in three ways,

- [Customize month cell using style](#)
- [Customize month cell using event](#)
- [Customize month cell with custom UI](#)
- [Customize month cell using DataTemplate](#)
- [Customize month cell using DataTemplateSelector](#)

#### *Customize month cell using style*

By using [MonthCellStyle](#) of [SfSchedule](#) you can customize the month cell properties such as [BackgroundColor](#), [NextMonthBackgroundColor](#), [NextMonthTextColor](#), [PreviousMonthBackgroundColor](#), [PreviousMonthTextColor](#), [TextColor](#), [FontSize](#), [FontAttributes](#), [FontFamily](#), [TodayBackgroundColor](#), [TodayTextColor](#).

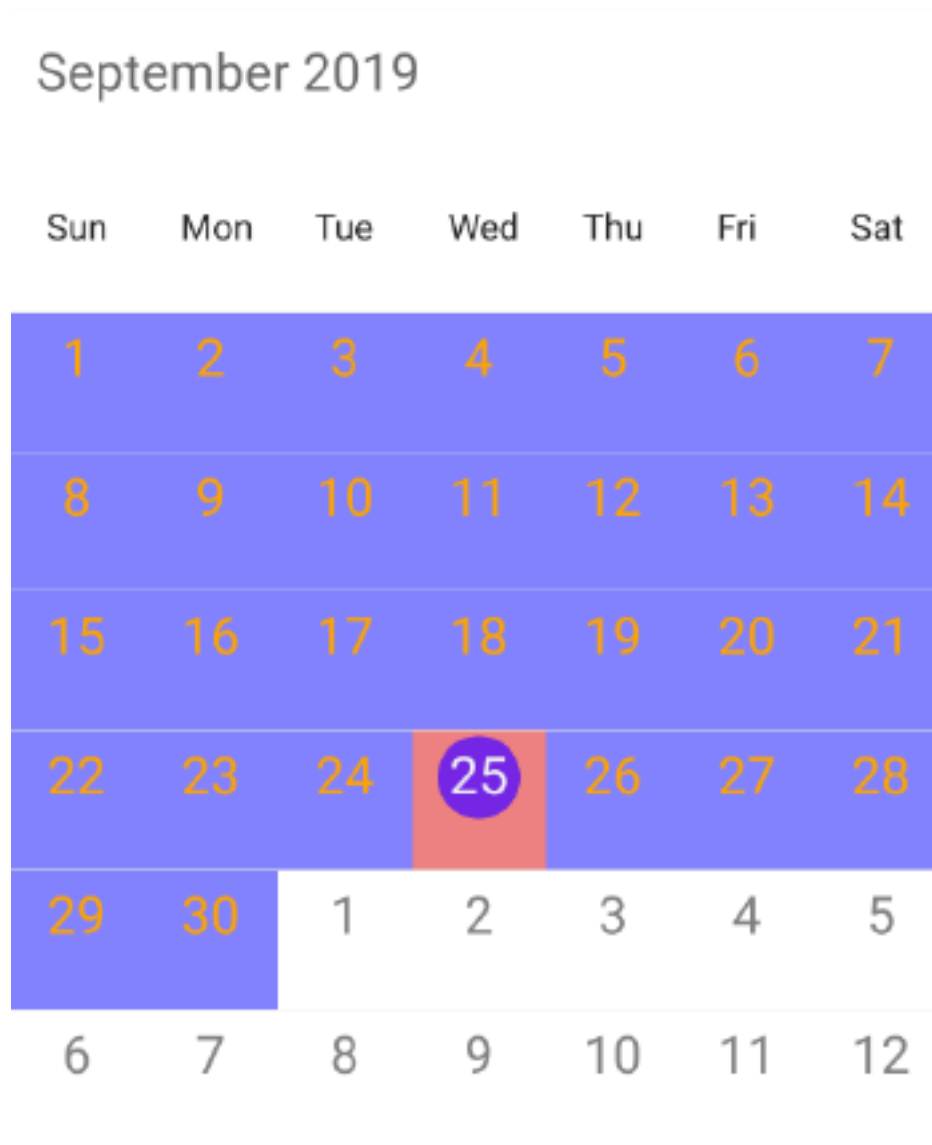
## XML

```
<schedule:SfSchedule.MonthCellStyle>
<schedule:MonthViewCellStyle
    BackgroundColor="#8282ff"
    TextColor="Orange"
    FontSize="20"
    FontFamily = "Arial"
    FontAttributes = "None"
    NextMonthBackgroundColor="Gray"
    NextMonthTextColor="Gray"
    PreviousMonthBackgroundColor="White"
    PreviousMonthTextColor="Gray"
    TodayBackgroundColor="#f97272"
    TodayTextColor="White">
</schedule:MonthViewCellStyle>
</schedule:SfSchedule.MonthCellStyle>
```

## C#

```
//MonthCell Appearance
MonthViewCellStyle monthCellStyle = new MonthViewCellStyle();
monthCellStyle.BackgroundColor = Color.FromHex("#8282ff");
monthCellStyle.NextMonthBackgroundColor = Color.White;
monthCellStyle.NextMonthTextColor = Color.Gray;
monthCellStyle.PreviousMonthBackgroundColor = Color.White;
monthCellStyle.PreviousMonthTextColor = Color.Gray;
monthCellStyle.TextColor = Color.Orange;
monthCellStyle.FontFamily = "Arial";
```

```
monthCellStyle.FontSize = 20;
monthCellStyle.FontAttributes = FontAttributes.None;
monthCellStyle.TodayBackgroundColor = Color.FromHex("#f97272");
monthCellStyle.TodayTextColor = Color.White;
schedule.MonthCellStyle = monthCellStyle;
```



#### Customize month cell using event

By using [OnMonthCellLoadedEvent](#) in `SfSchedule`, you can customize the month cell properties in the run time. In `OnMonthCellLoadedEvent`, arguments such as [cellStyle](#), [appointments](#), [date](#), [view](#) and boolean properties such as [isToday](#), [isNextMonthDate](#), [isPreviousMonthDate](#) and [isBlackOutDate](#) are in the [MonthCellLoadedEventArgs](#).

You can customize the month cell appearance in run time by setting [TextColor](#), [FontSize](#), [FontFamily](#), [FontAttributes](#), and [BackgroundColor](#) properties of `cellStyle` argument in `MonthCellLoadedEventArgs`.

**C#**

```
schedule.OnMonthCellLoadedEvent += Schedule_OnMonthCellLoadedEvent;
...
private void Schedule_OnMonthCellLoadedEvent(object sender,
MonthCellLoadedEventArgs args)
{
    args.cellStyle = new CellStyle();
    if (args.isToday)
    {
        args.cellStyle.BackgroundColor = Color.FromHex("#f97272");
        args.cellStyle.TextColor = Color.White;
        args.cellStyle.FontFamily = "Arial";
        args.cellStyle.FontSize = 25;
    }
    else if (args.isNextMonthDate)
    {
        args.cellStyle.BackgroundColor = Color.White;
        args.cellStyle.TextColor = Color.Gray;
        args.cellStyle.FontFamily = "Arial";
        args.cellStyle.FontSize = 10;
    }
    else if (args.isPreviousMonthDate)
    {
        args.cellStyle.BackgroundColor = Color.White;
        args.cellStyle.TextColor = Color.Gray;
        args.cellStyle.FontFamily = "Arial";
        args.cellStyle.FontSize = 10;
    }
    else
    {
        args.cellStyle.BackgroundColor = Color.FromHex("#8282ff");
        args.cellStyle.TextColor = Color.White;
        args.cellStyle.FontFamily = "Arial";
        args.cellStyle.FontSize = 20;
    }
}
```



## NOTE

FontAttributes and FontFamily are native to the platform. Custom font and the font which are not available in the specified platform will not be applied.

[Customize month cell with custom UI](#)

You can set the Custom UI for the month cell using [view](#) property of `MonthCellLoadedEventArgs` in the `OnMonthCellLoadedEvent`.

## C#

```
schedule.OnMonthCellLoadedEvent += Schedule_OnMonthCellLoadedEvent;
...
private void Schedule_OnMonthCellLoadedEvent(object sender,
MonthCellLoadedEventArgs args)
{
    Button button = new Button();
    button.Text = args.date.Day.ToString();
    button.BackgroundColor = Color.Blue;
}
```

```
button.TextColor = Color.White;
args.view = button;
if (args.isToday)
{
    button.BackgroundColor = Color.Red;
    button.TextColor = Color.White;
}
}
```

## September 2019

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

### Customize month cell using DataTemplate

You can customize the default appearance of the month cell by using the [MonthCellTemplate](#) property of `MonthViewSettings`.

### XML

```
<schedule:SfSchedule
    x:Name="schedule" ScheduleView="MonthView">
    <schedule:SfSchedule.MonthViewSettings>
```

```

<schedule:MonthViewSettings>
<schedule:MonthViewSettings.MonthCellTemplate>
<DataTemplate>
<Label BackgroundColor = "Purple" TextColor="White" Text="{Binding Date,
StringFormat='{0:dd}'}"/>
</DataTemplate>
</schedule:MonthViewSettings.MonthCellTemplate>
</schedule:MonthViewSettings>
</schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>

```

September 2019						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
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
06	07	08	09	10	11	12

#### *Customize month cell using DataTemplateSelector*

You can use `DataTemplateSelector` to choose a `DataTemplate` at runtime based on the value of a data-bound to Schedule month cell through `MonthCellTemplate`. It lets you choose a different data template for each month cell, customizing the appearance of a particular month cell based on certain conditions. `DataTemplateSelector` for month cell includes [MonthCellItem](#) as object item and `Schedule`

as bindable object. **MonthCellItem** consists of following properties, [Date](#), [Appointments](#), [IsLeadingDay](#), [IsTrailingDay](#), [IsBlockOutDay](#).

### XML

```
<ContentPage.Resources>
  <ResourceDictionary>
    <samplelocal:MonthCellDataTemplateSelector
      x:Key="monthCellDataTemplateSelector" />
  </ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
  <schedule:SfSchedule ScheduleView="MonthView" >
    <schedule:SfSchedule.MonthViewSettings>
      <schedule:MonthViewSettings x:Name="monthViewSettings"
        MonthCellTemplate="{StaticResource monthCellDataTemplateSelector}" />
    </schedule:SfSchedule.MonthViewSettings>
  </schedule:SfSchedule>
</ContentPage.Content>
```

### Creating a DataTemplateSelector

#### C#

```
public class MonthCellDataTemplateSelector : DataTemplateSelector
{
    public DataTemplate MonthAppointmentTemplate { get; set; }
    public DataTemplate MonthCellDatesTemplate { get; set; }
    public MonthCellDataTemplateSelector()
    {
        MonthAppointmentTemplate = new
        DataTemplate(typeof(MonthAppointmentTemplate));
        MonthCellDatesTemplate = new DataTemplate(typeof(MonthCellDatesTemplate));
    }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject
    container)
    {
        var sfSchedule = (container as Syncfusion.SfSchedule.XForms.SfSchedule);
        if (sfSchedule == null) return null;
        if (sfSchedule != null)
        {
            var appointments = (IList)(item as MonthCellItem).Appointments;
            foreach (var appointment in appointments)
            {
                CustomizationViewModel.ScheduleAppointment = appointment as
                ScheduleAppointment;
                return MonthAppointmentTemplate;
            }
            CustomizationViewModel.MonthCellItem = item as MonthCellItem;
            return MonthCellDatesTemplate;
        }
        else
            return null;
    }
}
```



Used Label to display current ,next and previous month cell dates and StackLayout with label and image to denote the month cell with appointments.

### XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Label as Template to display month dates-->
<Label xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="MonthCellTemplate_Forms.CurrentViewDatesTemplate"
xmlns:samplelocal="clr-
namespace:MonthCellTemplate_Forms;assembly=MonthCellTemplate_Forms"
BackgroundColor ="Transparent" FontSize="13" TextColor="Black"
Text = "{Binding Date, StringFormat='{0:dd}'}">
<Label.Behaviors>
<samplelocal:MonthCellDateBehavior />
</Label.Behaviors>
</Label>
.....
<?xml version="1.0" encoding="UTF-8"?>
<!--StackLayout with Label and Image as Template to display month cell with
appointment-->
<StackLayout xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="MonthCellTemplate_Forms.MonthAppointmentTemplate"
BackgroundColor="Transparent" Orientation="Vertical"
xmlns:samplelocal="clr-
namespace:MonthCellTemplate_Forms;assembly=MonthCellTemplate_Forms"
VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand">
<Label x:Name="label" Text="{Binding Date, StringFormat='{0:dd}'}"
VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand">
<Label.FontSize>
<OnPlatform x:TypeArguments="x:Double" iOS="15" Android="13" WinPhone="13"
/>
</Label.FontSize>
</Label>
<Image x:Name="Image" BackgroundColor="Transparent" Margin="15,0,0,0"
HorizontalOptions="Start" VerticalOptions="Start" HeightRequest="40"
WidthRequest="20" >
</Image>
<StackLayout.Behaviors>
<samplelocal:MonthCellCustomViewBehavior />
</StackLayout.Behaviors>
</StackLayout>
```





### C#

```
public class MonthCellDateBehavior : Behavior<Label>
{
    protected override void OnAttachedTo(Label bindable)
    {
        base.OnAttachedTo(bindable);
        if (CustomizationViewModel.MonthCellItem == null)
            return;
        if (CustomizationViewModel.MonthCellItem.IsLeadingDay ||
            CustomizationViewModel.MonthCellItem.IsTrailingDay)
```

```

{
    bindable.TextColor = Color.Gray;
}
else
{
    bindable.TextColor = Color.Black;
}
}
}

```

September 2019						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
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

#### Getting Inline Appointment details

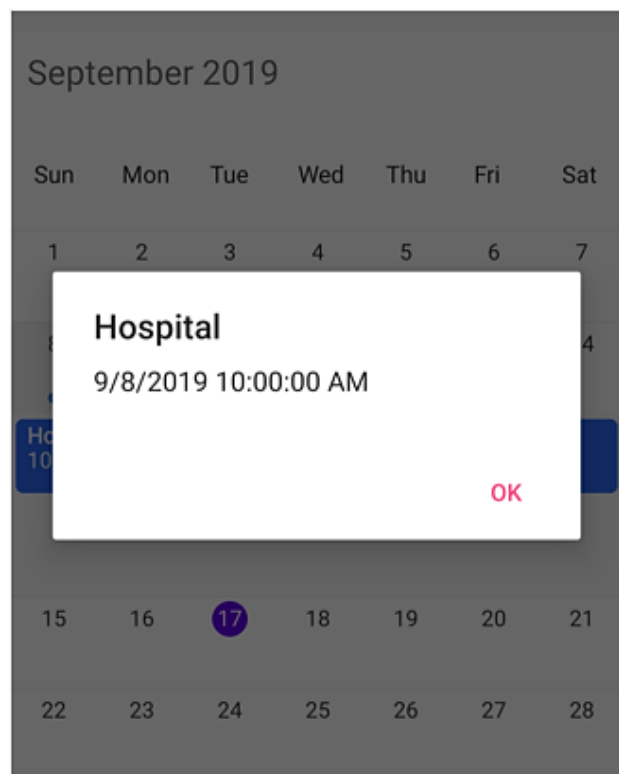
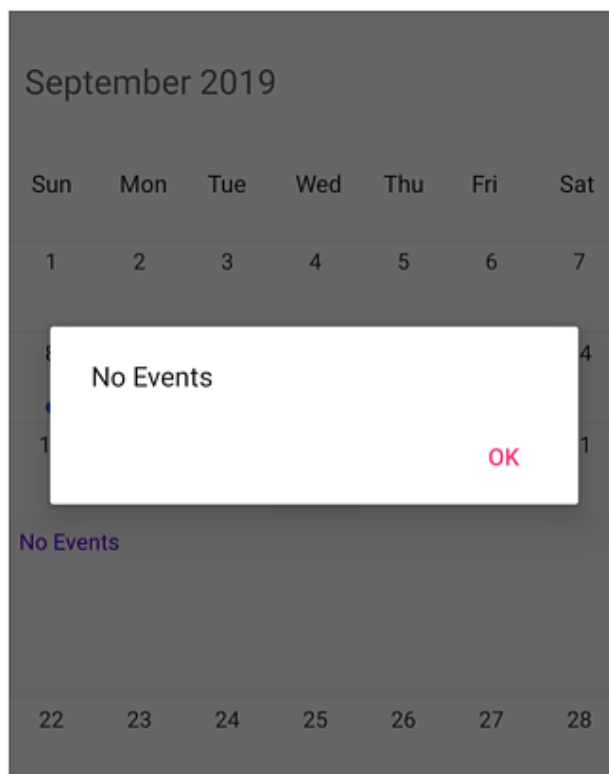
Using [Appointment](#) argument in the [MonthInlineAppointmentTappedEventArgs](#) of [MonthInlineAppointmentTapped](#) event, you can get the Month Inline Appointments details while tapping the specific appointment, and you can get the selected date by using the [selectedDate](#) property. [MonthInlineAppointmentTapped](#) also trigger while tapping the [No Events](#) view in inline. You can do the required functions while tapping the inline appointment using this event.

**C#**

```

schedule.MonthInlineAppointmentTapped +=
Schedule_MonthInlineAppointmentTapped;
...
private void Schedule_MonthInlineAppointmentTapped(object sender,
MonthInlineAppointmentTappedEventArgs args)
{
    if (args.Appointment != null)
    {
        var appointment = (args.Appointment as ScheduleAppointment);
        DisplayAlert(appointment.Subject, appointment.StartTime.ToString(), "ok");
    }
    else
    {
        DisplayAlert("", "No Events", "ok");
    }
}

```

**Note:**

The `MonthInlineAppointmentTapped` is also applicable for `AgendaView`.

**InlineView Appearance**

You can customize the inline view with two ways,

- Using style
- Using custom view

### Using style

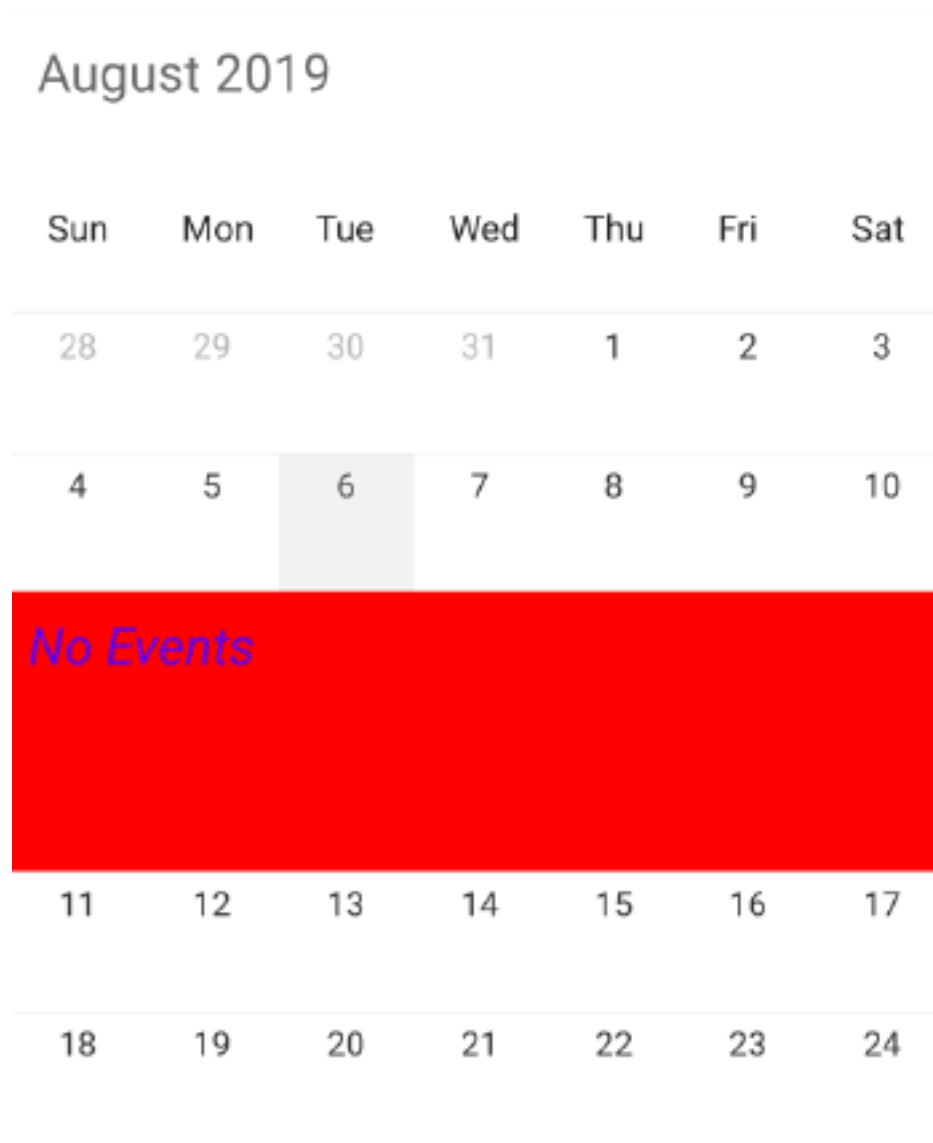
You can customize the month inline view by setting the [MonthInlineViewStyle](#) properties such as [Background](#), [TextColor](#), [FontSize](#), [FontAttributes](#), [FontFamily](#), [TimeTextColor](#), [TimeTextSize](#), and [TimeTextFormat](#) at runtime using [OnMonthInlineLoadedEvent](#) in [SfSchedule](#). By using [OnMonthInlineLoadedEvent](#) you can get arguments such as [monthInlineViewStyle](#), [appointments](#), [selectedDate](#) in the [MonthInlineLoadedEventArgs](#).

### C#

```
schedule.OnMonthInlineLoadedEvent += Schedule_OnMonthInlineLoadedEvent;
...
private void Schedule_OnMonthInlineLoadedEvent(object sender,
MonthInlineLoadedEventArgs args)
{
    var appointments = e.appointments.Cast<ScheduleAppointment>().ToList();
    MonthInlineViewStyle monthInlineViewStyle = new MonthInlineViewStyle();
    if (appointments != null && appointments.Count > 0)
    {
        monthInlineViewStyle.BackgroundColor = Color.AliceBlue;
        monthInlineViewStyle.TextColor = Color.White;
        monthInlineViewStyle.FontSize = 10;
        monthInlineViewStyle.FontAttributes = FontAttributes.None;
        monthInlineViewStyle.FontFamily = "Times New Roman";
        monthInlineViewStyle.TimeTextColor = Color.Yellow;
        monthInlineViewStyle.TimeTextSize = 15;
        monthInlineViewStyle.TimeTextFormat = "hh a";
    }
    else
    {
        // Style to customize the No Events label
        monthInlineViewStyle.BackgroundColor = Color.Red;
        monthInlineViewStyle.FontAttributes = FontAttributes.Italic;
        monthInlineViewStyle.FontFamily = "Times New Roman";
        monthInlineViewStyle.TimeTextColor = Color.White;
        monthInlineViewStyle.TimeTextSize = 20;
    }
    e.monthInlineViewStyle = monthInlineViewStyle;
}
```

## September 2019

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
<div>Jeni's B'day 10 AM - 11 AM</div> <div>Meeting 11 AM - 12 PM</div> <div>Lunch</div>						
15	16	17	18	19	20	21
22	23	24	25	26	27	28



Get the complete sample for this [here](#).

### NOTE

FontAttributes and FontFamily are native to the platform. Custom font and the font which are not available in the specified platform will not be applied.

*Using custom view*

You can set custom view for the inline view by using [InlineView](#) property of `Schedule`

### C#

```
SfSchedule schedule = new SfSchedule();
Button button = new Button();
button.BackgroundColor = Color.Red;
button.Text = "No Event";
button.TextColor = Color.White;
schedule.InlineView = button;
```

**XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView"
ShowAppointmentsInline="True">
<schedule:SfSchedule.InlineView>
<Button BackgroundColor="Red" Text="No Events" TextColor="White"/>
</schedule:SfSchedule.InlineView>
</schedule:SfSchedule>
```

## August 2019

Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
NO EVENT						
11	12	13	14	15	16	17
18	19	20	21	22	23	24

**InlineAppointment Appearance**

You can customize the Month inline view Appointment by using [OnMonthInlineAppointmentLoadedEvent](#) in [SfSchedule](#), using [view](#) of [MonthInlineAppointmentLoadedEventArgs](#) argument. You can get the details of Appointment in the [appointment](#) argument.

**C#**

```

schedule.OnMonthInlineAppointmentLoadedEvent +=
Schedule_OnMonthInlineAppointmentLoadedEvent;
...
private void Schedule_OnMonthInlineAppointmentLoadedEvent(object sender,
MonthInlineAppointmentLoadedEventArgs args)
{
    var appointment = (args.appointment as ScheduleAppointment);
    Button button = new Button();
    button.Text = appointment.Subject;
    button.BackgroundColor = Color.Blue;
    button.TextColor = Color.White;
    args.view = button;
}

```

## October 2019

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	1	2	3	4	5
6	7	8	9	10	11	12
PROJECT PLAN						
GENERAL MEETING						
13	14	15	16	17	18	19
20	21	22	23	24	25	26

### NOTE



Inline view customization - There is no support for inline view appointments customization using custom view, TimeTextSize, TimeTextFormat and TimeTextColor properties in XForms UWP.

## Selection

You can customize the default appearance of selection UI in the month cells.

- [Selection text color customization](#)
- [Selection indicator color customization](#)
- [Selection customization using style](#)
- [Selection customization using custom View](#)
- [Programmatic selection](#)

### *Selection text color customization*

Month cell Selection Text Color can be customized using [SelectionTextColor](#) property of `MonthViewSettings`.

## XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
  <schedule:SfSchedule.MonthViewSettings>
    <schedule:MonthViewSettings
      SelectionTextColor="Red" >
    </schedule:MonthViewSettings>
  </schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>
```

## C#

```
//creating new instance for MonthViewSettings
MonthViewSettings monthViewSettings = new MonthViewSettings();
monthViewSettings.SelectionTextColor = Color.Red;
schedule.MonthViewSettings = monthViewSettings;
```

September 2019						
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	1	2	3	4	5
6	7	8	9	10	11	12

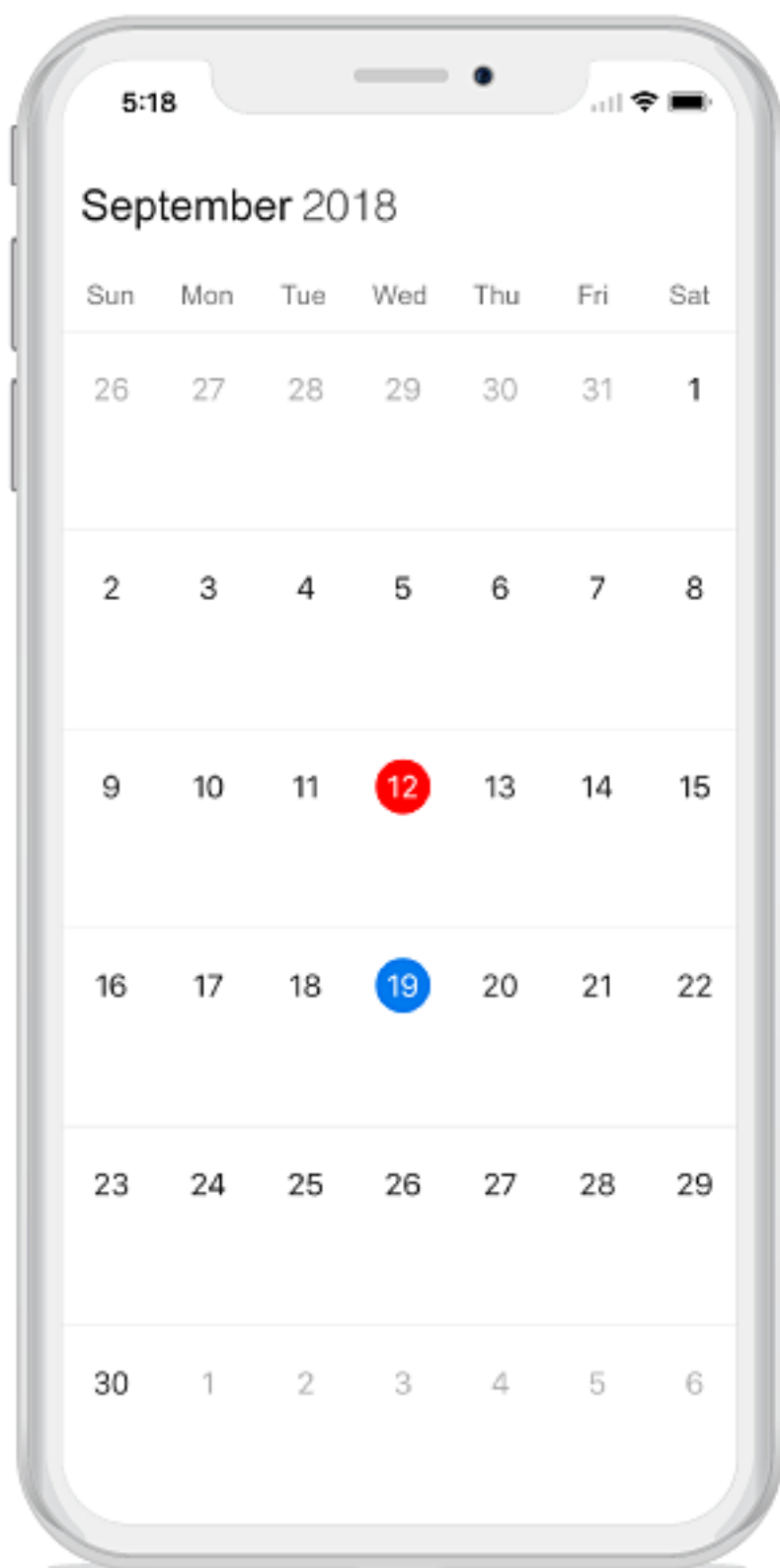
#### Selection indicator color customization

Month cell selection indicator color can be customized using [SelectionIndicatorColor](#) property of `MonthViewSettings` in Xamarin.iOS (Native). You can achieve the same in Xamarin.Forms by setting the required color to `SelectionIndicatorColor` property of the Xamarin.Forms (iOS) project using reflection.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    LoadApplication(new App());
    SfScheduleRenderer.Init();
    // Here ScheduleSamplePage is the Schedule rendered page in PCL
    // FormsSfSchedule is initialized public static SfSchedule in the
    ScheduleSamplePage
}
```

```
ScheduleSamplePage.FormsSfSchedule.SelectionStyle.BackgroundColor =
Color.Transparent;
ScheduleSamplePage.FormsSfSchedule.CellTapped += FormsSchedule_CellTapped;
return base.FinishedLaunching(app, options);
}
private void FormsSchedule_CellTapped(object sender,
Syncfusion.SfSchedule.XForms.CellTappedEventArgs e)
{
    // Gets the field nativeObject from SfSchedule by using reflection
    // GetType() used to get the type of schedule instance
    var fieldInfo = GetField(ScheduleSamplePage.FormsSfSchedule.GetType(),
    "nativeObject");
    // Gets the value of nativeObject field by using reflection
    var fieldInfo = GetField(ScheduleSamplePage.FormsSfSchedule.GetType(),
    "nativeObject");
    var nativeSchedule =
    fieldInfo.GetValue(ScheduleSamplePage.FormsSfSchedule);
    var monthViewSettings = new Syncfusion.SfSchedule.iOS.MonthViewSettings();
    // Setting Month cell selection indicator color
    monthViewSettings.SelectionIndicatorColor = UIColor.Red;
    (nativeSchedule as SfSchedule).MonthViewSettings = monthViewSettings;
}
```



---

**NOTE**

---

This support won't apply for current day indicator, you can use `TodayBackgroundColor` property to customize it.

#### *Selection customization using style*

You can customize the month cell selection by using [SelectionStyle](#) property of `SfSchedule`.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
  <schedule:SfSchedule.SelectionStyle>
    <schedule:SelectionStyle
      BackgroundColor="Blue"
      BorderColor="Black"
      BorderThickness="5"
      BorderCornerRadius="5">
    </schedule:SelectionStyle>
  </schedule:SfSchedule.SelectionStyle>
</schedule:SfSchedule>
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.MonthView;
//Create new instance of SelectionStyle
SelectionMode selectionStyle = new SelectionStyle();
selectionStyle.BackgroundColor = Color.Blue;
selectionStyle.BorderColor = Color.Black;
selectionStyle.BorderThickness = 5;
selectionStyle.BorderCornerRadius = 5;
schedule.SelectionStyle = selectionStyle;
```

September 2019						
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	1	2	3	4	5
6	7	8	9	10	11	12

#### *Selection customization using custom View*

You can replace the default selection UI with your custom view by setting [SelectionView](#) property of SfSchedule.

#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="MonthView">
  <schedule:SfSchedule.SelectionView>
    <Button
      BackgroundColor="#FF9800"
      Text="+NewEvent"
      TextColor="White"/>
  </schedule:SfSchedule.SelectionView>
</schedule:SfSchedule>
```

#### **C#**

```
schedule.ScheduleView = ScheduleView.MonthView;
```

```
//Add the CustomView
Button customView = new Button();
customView.Text = "+NewEvent";
customView.BackgroundColor = Color.FromHex("#FF9800");
customView.TextColor = Color.White;
schedule.SelectionView = customView;
```

## September 2019

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	+NE WE	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5
6	7	8	9	10	11	12

### Programmatic selection

You can programmatically select the specific cell by setting corresponding date to [SelectedDate](#) property of SfSchedule. By default, it is null.

### C#

```
// Setting a date to select
schedule.SelectedDate = new DateTime(2017, 10, 04);
```

You can clear the selection by setting [SelectedDate](#) as null.

**C#**

```
// Setting null value to deselect
schedule.SelectedDate = null;
```

You can download the entire source code of this demo for Xamarin.Forms from here [Date Selection](#)

**NOTE**

- SfSchedule does not support multiple selection.
- SfSchedule supports two-way binding of SelectedDate property.

## October 2017

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	1	2	3	4
5	6	7	8	9	10	11

### Today Background Color

You can customize the current date background of SfSchedule by using [TodayBackground](#) property of [MonthViewSettings](#).



**XML**

```
<schedule:SfSchedule ScheduleView="MonthView">
</schedule:SfSchedule.MonthViewSettings>
<schedule:MonthViewSettings TodayBackground="Red" />
</schedule:SfSchedule.MonthViewSettings>
</schedule:SfSchedule>
```

**C#**

```
schedule.ScheduleView = ScheduleView.MonthView;
//Creating new instance of MonthViewSettings
MonthViewSettings monthViewSettings = new MonthViewSettings();
//Customizing background color
monthViewSettings.TodayBackground = Color.Red;
schedule.MonthViewSettings = monthViewSettings;
```

## September 2019

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

### Custom Font

You can change the appearance of Font by setting the `FontFamily` property to the following classes.

- [ViewHeaderStyle](#) - You can change the appearance of [ViewHeaderStyle](#) by setting the [DayFontFamily](#) and [DateFontFamily](#) properties of `Schedule ViewHeaderStyle`.
- [MonthCellStyle](#) - You can change the appearance of [MonthCellStyle](#) by setting the [FontFamily](#) property of `Schedule MonthCellStyle`.
- [MonthInlineViewStyle](#) - You can change the appearance of [MonthInlineViewStyle](#) by setting the [FontFamily](#) property of `Schedule MonthInlineViewStyle`.
- [WeekNumberStyle](#) - You can change the appearance of [WeekNumber](#) by setting the [FontFamily](#) property of `Schedule WeekNumberStyle`.

### XML

```
<schedule:ViewHeaderStyle.DayFontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:ViewHeaderStyle.DayFontFamily>
```

### C#

```
viewHeaderStyle.DayFontFamily = Device.OnPlatform("Lobster-Regular",
"Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
```

## September 2019

<i>Sun</i>	<i>Mon</i>	<i>Tue</i>	<i>Wed</i>	<i>Thu</i>	<i>Fri</i>	<i>Sat</i>
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5
6	7	8	9	10	11	12

### XML

```
<schedule:MonthCellStyle.FontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:ViewHeaderStyle.FontFamily>
```

### C#

```
monthCellStyle.FontFamily = Device.OnPlatform("Lobster-Regular", "Lobster-
Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
```

## September 2019

Sun	Mon	Tue	Wed	Thu	Fri	Sat
<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>8</i>	<i>9</i>	<i>10</i>	<i>11</i>	<i>12</i>	<i>13</i>	<i>14</i>
<i>15</i>	<i>16</i>	<i>17</i>	<i>18</i>	<i>19</i>	<i>20</i>	<i>21</i>
<i>22</i>	<i>23</i>	<i>24</i>	<b>25</b>	<i>26</i>	<i>27</i>	<i>28</i>
<i>29</i>	<i>30</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>	<i>11</i>	<i>12</i>

### C#

```

schedule.OnMonthInlineLoadedEvent += Schedule_OnMonthInlineLoadedEvent;
...
private void Schedule_OnMonthInlineLoadedEvent(object sender,
MonthInlineLoadedEventArgs args)
{
    MonthInlineViewStyle monthInlineViewStyle = new MonthInlineViewStyle();
    monthInlineViewStyle.FontFamily = Device.OnPlatform("Lobster-Regular",
    "Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
    args.monthInlineViewStyle = monthInlineViewStyle;
}

```

## September 2019

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
<b>Plan Execution</b> 10:00-11:00 AM - 11:00-12:00 PM						
<b>Plan Execution</b> 09:00-10:00 AM - 10:00-11:00 AM						
15	16	17	18	19	20	21
22	23	24	25	26	27	28

### C#

```
weekNumberStyle.FontFamily = Device.OnPlatform("Lobster=Regular", "Lobster-Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
```

## September 2019

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
36	1	2	3	4	5	6	7
37	8	9	10	11	12	13	14
38	15	16	17	18	19	20	21
39	22	23	24	25	26	27	28
40	29	30	1	2	3	4	5
41	6	7	8	9	10	11	12

Following steps will explain how to configure the custom fonts.

### *Custom Font Setting in Xamarin.Forms (Android)*

- Add your custom Font (e.g. Lobster-Regular.ttf) to the Assets folder in the Assets folder of the Xamarin.Forms (Android) project.
- Then, use the Custom Font name as FontFamily.

### *Custom Font Setting in Xamarin.Forms (ios)*

- Add your custom Font (e.g. Lobster-Regular.ttf) to the Resources folder of the Xamarin.Forms (iOS) project.
- Edit info.plist and add a key Fonts provided by application (value type should be Array). In item0 of the array enter the name of the FontFamily you added in the Resource folder. (Such as Lobster-Regular.ttf).
- Then, use the Custom Font name as FontFamily.

---

**NOTE**

---

No need to mention .ttf when set the Custom Font in iOS.

*Custom Font Setting in Xamarin.Forms (UWP)*

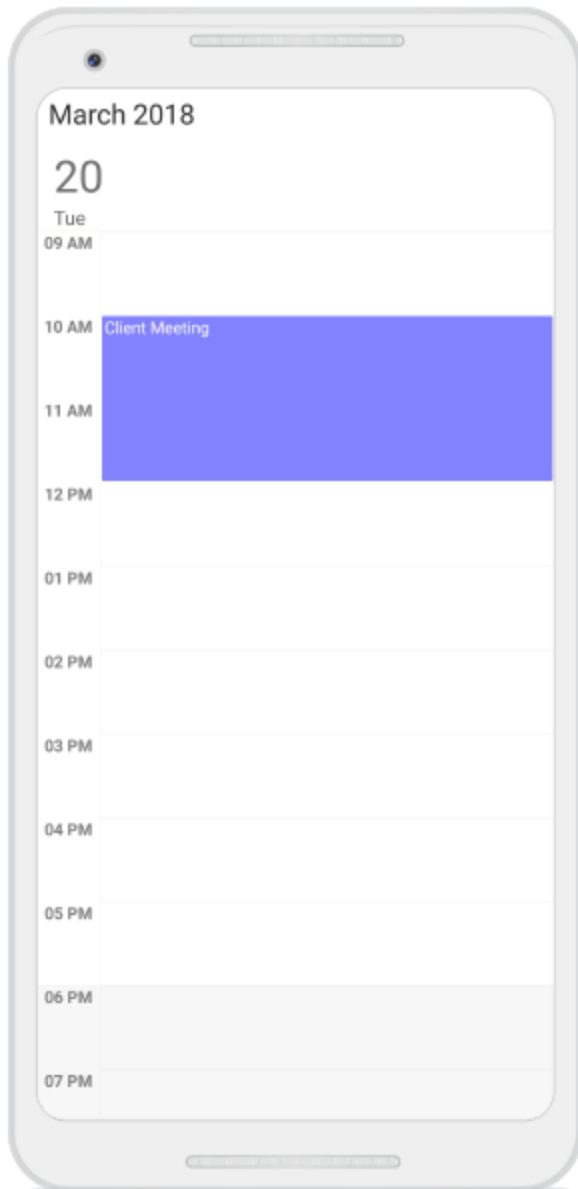
- Add your custom Font (e.g. Lobster-Regular.ttf) to the Assets folder of the Xamarin.Forms (UWP) project.
- Then, use the Custom Font name as FontFamily. When Setting custom font in UWP use the format (FontFamily = "Assets/Lobster-Regular.ttf#Lobster").

## Appointments

[SfSchedule](#) control has a built-in capability to handle the appointment arrangement internally based on the [ScheduleAppointmentCollection](#). [ScheduleAppointment](#) is a class, which holds the details about the appointment to be rendered in schedule.

### C#

```
// Creating an instance for schedule appointment collection
ScheduleAppointmentCollection scheduleAppointmentCollection = new ScheduleAp
pointmentCollection();
//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, 12, 0, 0),
    Subject = "Meeting",
    Location = "Hutchison road",
});
//Adding schedule appointment collection to DataSource of SfSchedule
schedule.DataSource=scheduleAppointmentCollection;
```



### Mapping

Schedule supports full data binding to any type of IEnumerable source. Specify the [ScheduleAppointmentMapping](#) attributes to map the properties in the underlying data source to the schedule appointments.

Property Name	Description
-----	-----
-----	-----
-----	-----

<a href="#">StartTimeMapping</a>   This property is to map the property name of custom class which is equivalent for StartTime of ScheduleAppointment.
--



| [StartTimeZoneMapping](#) | This property is to map the property name of custom class which is equivalent for Start time zone of ScheduleAppointment. |

| [EndTimeMapping](#) | This property is to map the property name of custom class which is equivalent for EndTime of ScheduleAppointment. |

| [EndTimeZoneMapping](#) | This property is to map the property name of custom class which is equivalent for End time zone of ScheduleAppointment. |

| [SubjectMapping](#) | This property is to map the property name of custom class which is equivalent for Subject of ScheduleAppointment. |

| [ColorMapping](#) | This property is to map the property name of custom class which is equivalent for Color of ScheduleAppointment. |

| [IsAllDayMapping](#) | This property is to map the property name of custom class which is equivalent for IsAllDay of ScheduleAppointment. |

| [RecurrenceRuleMapping](#) | This property is to map the property name of custom class which is equivalent for RecurrenceRule of ScheduleAppointment. |

| [NotesMapping](#) | This property is to map the property name of custom class which is equivalent for Notes of ScheduleAppointment. |

| [LocationMapping](#) | This property is to map the property name of custom class which is equivalent for Location of ScheduleAppointment. |

| [IsRecursiveMapping](#) | This property is to map the property name of custom class which is equivalent for IsRecursive of ScheduleAppointment. |

| [MinHeightMapping](#) | This property is to map the property name of custom class which is equivalent for MinHeight of ScheduleAppointment. |

| [ResourceIdsMapping](#) | This property is to map the property name of custom class which is equivalent for ResourceIds of ScheduleAppointment. |

---

**Note:** CustomAppointment class should contain two DateTime fields and a string field as mandatory.

---

#### *Creating custom Appointments*

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 DateTime From { get; set; }
    public DateTime To { get; set; }
    public Color Color { 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](#).

#### XML

```
<syncfusion:SfSchedule x:Name="schedule" ScheduleView="DayView" DataSource="{Binding Meetings}">
  <syncfusion:SfSchedule.AppointmentMapping>
    <syncfusion:ScheduleAppointmentMapping
      SubjectMapping="EventName"
      ColorMapping="Color"
      StartTimeMapping="From"
      EndTimeMapping="To">
    </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.ColorMapping = "Color";
schedule.AppointmentMapping = dataMapping;
```

You can schedule 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 `DataSource` property which is of `IEnumerable` type.

#### C#

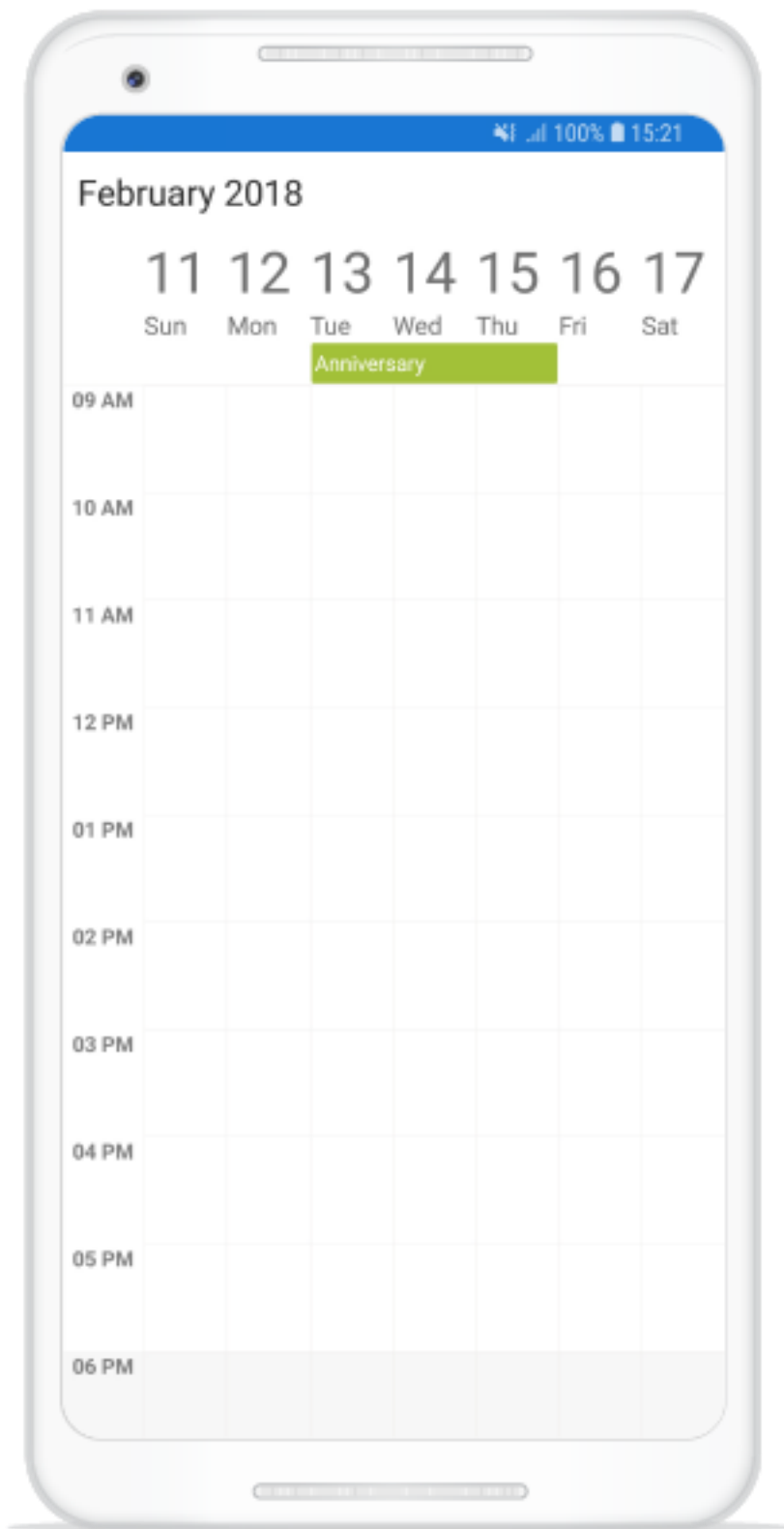
```
// Creating instance for custom appointment class
Meeting meeting = new Meeting();
// Setting start time of an event
meeting.From = new DateTime(2017, 05, 08, 10, 0, 0);
// Setting end time of an event
meeting.To = meeting.From.AddHours(1);
// Setting start time for an event
meeting.EventName = "Anniversary";
// Setting color for an event
meeting.Color = Color.Green;
// 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 DataSource of SfSchedule
schedule.DataSource = Meetings;
```

#### Spanned Appointments

Spanned Appointment is an appointment which lasts more than 24 hours. It doesn't block out time slots in `SfSchedule`, it will render in [All-Day appointment](#) panel exclusively.

**C#**

```
public ObservableCollection<Meeting> Meetings { get; set; }  
// Creating instance for collection of custom appointments  
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(2).AddHours(1);  
// Setting start time for an event  
meeting.EventName = "Anniversary";  
// Setting color for an event  
meeting.Color = Color.Green;  
// Adding a custom appointment in CustomAppointmentCollection  
Meetings.Add(meeting);  
//Adding schedule appointment collection to DataSource of SfSchedule  
schedule.DataSource= Meetings ;
```



### All Day Appointments

All-Day appointment is an appointment which is scheduled for a whole day. It can be set by using `IsAllDay` property in the `ScheduleAppointment`.

#### C#

```
// Creating an instance for schedule appointment collection
ScheduleAppointmentCollection scheduleAppointmentCollection = new ScheduleAp
pointmentCollection();
//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, 12, 0, 0),
    Subject = "Meeting",
    Location = "Hutchison road",
    IsAllDay = true
});
//Adding schedule appointment collection to DataSource of SfSchedule
schedule.DataSource=scheduleAppointmentCollection;
```

#### NOTE

Appointment which lasts through an entire day (exact 24 hours) will be considered as all day appointment without setting `IsAllDay` property. For example 06/09/2018 12:00AM to 06/10/2018 12:00AM.

#### All-Day Appointment Panel

All-day appointment and Spanned appointment doesn't block out entire time slot in SfSchedule, rather it will render in separate layout exclusively for all-day appointment. It can be enabled by setting `ShowAllDay` property of `DayViewSettings`, `WeekViewSettings` and `WorkWeekViewSettings` of `DayView`, `WeekView` and `WorkWeekView` respectively.

#### C#

```
schedule.ScheduleView = ScheduleView.WeekView;
WeekViewSettings weekViewSeetings = new WeekViewSettings();
weekViewSeetings.ShowAllDay = true;
schedule.WeekViewSettings = weekViewSeetings;
```

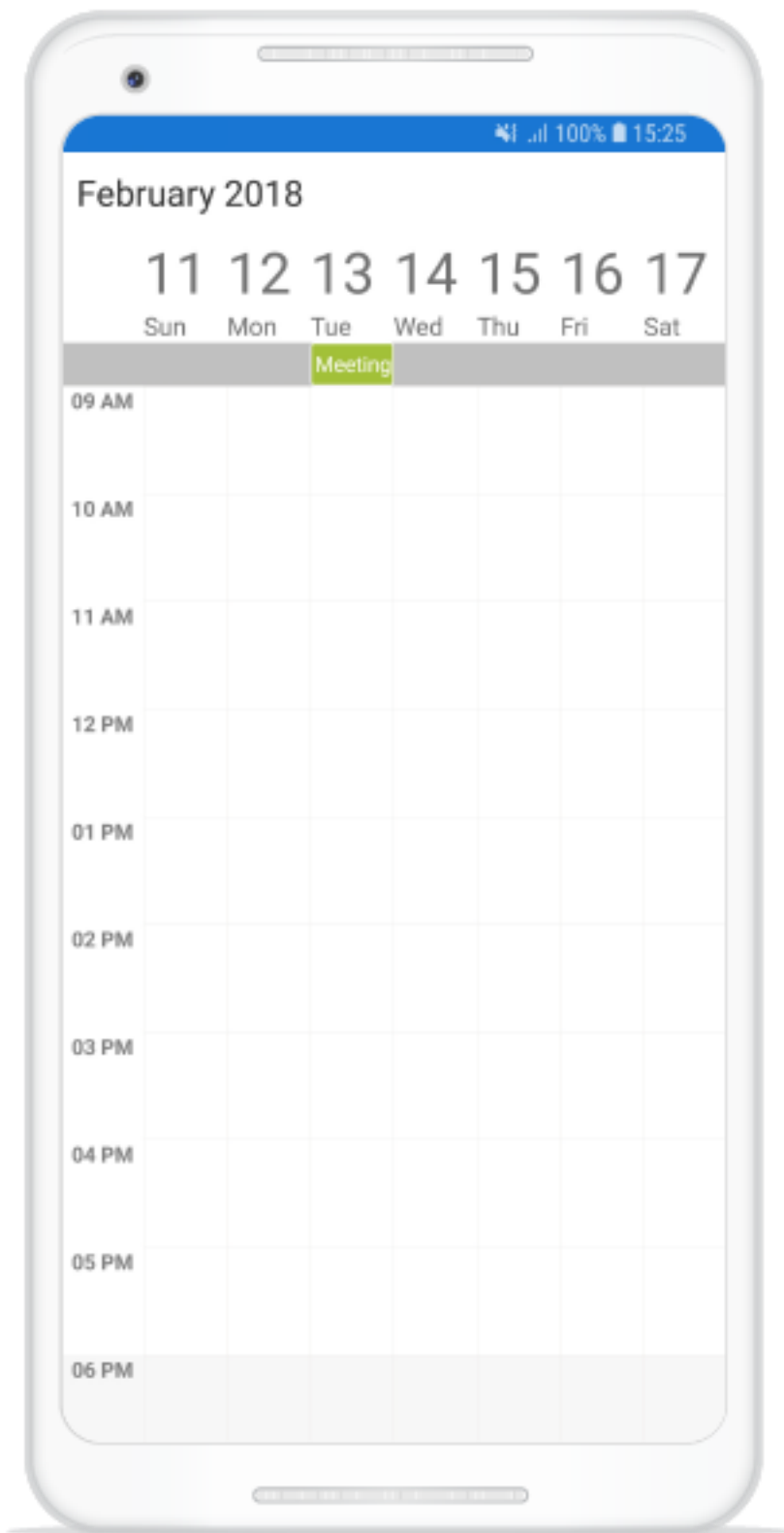
#### NOTE

Appointments which lasts less than 24 hours with different start date and end date will be rendered in time slot.

All-Day panel background can be customized by setting `AllDayAppointmentLayoutColor` of the respective view settings.

#### C#

```
weekViewSeetings.AllDayAppointmentLayoutColor = Color.Silver;
```



### Recurrence Appointment

Recurring appointment on a daily, weekly, monthly, or yearly interval. Recurring appointments can be created by setting **RecurrenceRule** property in Schedule appointments.

#### Recurrence Rule

The **RecurrenceRule** is a string value, that contains the details of the recurrence appointments like repeat type - daily/weekly/monthly/yearly, how many times it needs to be repeated, the interval duration and also the time period to render the appointment, etc.

**RecurrenceRule** has the following properties and based on this property value, the recurrence appointments are rendered in the SfSchedule control with its respective time period.

Property Name	Purpose
<b>FREQ</b>	Maintains the Repeat type value of the appointment. (Example: Daily, Weekly, Monthly, Yearly, Every week day) Example: FREQ=DAILY;INTERVAL=1
<b>INTERVAL</b>	Maintains the interval value of the 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
<b>COUNT</b>	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
<b>UNTIL</b>	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
<b>BYDAY</b>	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
<b>BYMONTHDAY</b>	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
<b>BYMONTH</b>	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*

Recurrence pattern used in the control are in iCal standard. Schedule control supports all four types of [recurrence patterns](#). You can set the recurrence pattern using [RecurrenceType](#) property of [RecurrenceRule](#).

RecurrenceType	RecurrenceProperties	Description
-----	-----	-----
Daily	<a href="#">Interval</a>	Gets or sets the day interval on which recurrence has to be set.
Weekly	<a href="#">Interval</a>	Gets or sets the day interval on which recurrence has to be set.
	<a href="#">DayOfWeek</a>	Gets or sets the day of week on which recurrence has to be set.
	<a href="#">WeekDays</a>	Gets or sets the <a href="#">day/days</a> in a week on which recurrence has to be set.
	<a href="#">Week</a>	Gets or sets the week of month on which recurrence has to be set.
Monthly	<a href="#">Interval</a>	Gets or sets the day interval on which recurrence has to be set.
	<a href="#">DayOfWeek</a>	Gets or sets the day of week on which recurrence has to be set.
	<a href="#">Week</a>	Gets or sets the week of month on which recurrence has to be set.
	<a href="#">DayOfMonth</a>	Gets or sets the day on which recurrence has to be set for every month.
Yearly	<a href="#">Interval</a>	Gets or sets the day interval on which recurrence has to be set.
	<a href="#">DayOffMonth</a>	Gets or sets the day on which recurrence has to be set for every month.
	<a href="#">DayOfWeek</a>	Gets or sets the day of week on which recurrence has to be set.
	<a href="#">Month</a>	Gets or sets the specific month of year on which recurrence has to be set.
	<a href="#">Week</a>	Gets or sets the week of month on which recurrence has to be set.
Common	<a href="#">RecurrenceRange</a>	Gets or sets the type of the <a href="#">recurrence range</a> for the time limit of recurrence appointment.



	<a href="#">RecurrenceCount</a>   Gets or sets the count for recurring appointment.	
	<a href="#">StartDate</a>   Gets or sets the date to start the recurrence appointment.	
	<a href="#">EndDate</a>   Gets or sets the date to end the recurrence appointment.	

Find the following **RecurrenceRule** possibilities available in the Schedule control while creating the recurrence appointment.

PossibilityType	Description	RecurrenceProperties	Examples
-----------------	-------------	----------------------	----------

-----	-----	-----
-----	-----	-----

Daily	Appointment is created with Ends Never	RecurrenceType = RecurrenceType.Daily, Interval = 1, RecurrenceRange = RecurrenceRange.NoEndDate	FREQ=DAILY; INTERVAL=1
-------	--	--	------------------------

	Appointment is created with Ends After	RecurrenceType = RecurrenceType.Daily, Interval = 1, RecurrenceRange = RecurrenceRange.Count, RecurrenceCount = 15	FREQ=DAILY; INTERVAL=1; COUNT=5
--	--	--	---------------------------------

	Appointment is created with Ends On	RecurrenceType = RecurrenceType.Daily, Interval = 1, RecurrenceRange = RecurrenceRange.EndDate, EndDate = new DateTime(2017, 06, 20)	FREQ=DAILY; INTERVAL=1; UNTIL=06/20/2017
--	-------------------------------------	--	--

	Appointment is created with Every (Interval)	RecurrenceType = RecurrenceType.Daily, Interval = 2, RecurrenceRange = RecurrenceRange.Count, RecurrenceCount = 10	FREQ=DAILY; INTERVAL=2; COUNT=10
--	--	--	----------------------------------

Weekly	Appointment is created with Ends Never	RecurrenceType = RecurrenceType.Weekly, Interval = 1, WeekDays = WeekDays.Monday	WeekDays.Wednesday   WeekDays.Friday, RecurrenceRange = RecurrenceRange.NoEndDate	FREQ=WEEKLY; INTERVAL=1; BYDAY=MO, WE, FR
--------	--	--	---	---

	Appointment is created with Ends After	RecurrenceType = RecurrenceType.Weekly, Interval = 1, WeekDays = WeekDays.Thursday, RecurrenceRange = RecurrenceRange.Count, RecurrenceCount = 10	FREQ=WEEKLY; INTERVAL=1; BYDAY=TH; COUNT=10
--	--	---	---

	Appointment is created with Ends On	RecurrenceType = RecurrenceType.Weekly, Interval = 1, WeekDays = WeekDays.Monday, RecurrenceRange = RecurrenceRange.EndDate, EndDate = new DateTime(2017, 07, 20)	FREQ=WEEKLY; INTERVAL=1; BYDAY=MO; UNTIL=07/20/2017
--	-------------------------------------	---	---

	Appointment is created with selecting multiple day	RecurrenceType = RecurrenceType.Weekly, Interval = 2, WeekDays = WeekDays.Monday   WeekDays.Wednesday   WeekDays.Friday, RecurrenceRange = RecurrenceRange.Count, RecurrenceCount = 10	FREQ=WEEKLY; INTERVAL=2; BYDAY=MO, WE, FR; COUNT=10
--	--	--	---

Every Day	Appointment is created with Ends Never	RecurrenceType = RecurrenceType.Weekly, Interval = 1, WeekDays = WeekDays.Monday   WeekDays.Tuesday   WeekDays.Wednesday   WeekDays.Thursday   WeekDays.Friday , RecurrenceRange = RecurrenceRange.NoEndDate	FREQ=WEEKLY; BYDAY=MO, TU, WE, TH, FR
-----------	--	--	---------------------------------------

	Appointment is created with Ends After	RecurrenceType = RecurrenceType.Weekly, Interval = 1, WeekDays =
--	--	--

WeekDays.Monday | WeekDays.Tuesday | WeekDays.Wednesday | WeekDays.Thursday | WeekDays.Friday  
, RecurrenceRange = RecurrenceRange.Count, RecurrenceCount = 10 | FREQ=WEEKLY; BYDAY=MO, TU,  
WE, TH, FR; COUNT=10 |

| | Appointment is created with Ends On | RecurrenceType = RecurrenceType.Weekly, Interval = 1,  
WeekDays =

WeekDays.Monday | WeekDays.Tuesday | WeekDays.Wednesday | WeekDays.Thursday | WeekDays.Friday  
, RecurrenceRange = RecurrenceRange.EndDate, EndDate = new DateTime(2017, 07, 15) |  
FREQ=WEEKLY; BYDAY=MO, TU, WE, TH, FR; UNTIL=07/15/2017 |

| Monthly | Appointment is created with selected date Ends Never | RecurrenceType =  
RecurrenceType.Monthly, Interval = 1, DayOfMonth = 15, RecurrenceRange =  
RecurrenceRange.NoEndDate | FREQ=MONTHLY; BYMONTHDAY=15; INTERVAL=1 |

| | Appointment is created with selected date and Ends After | RecurrenceType =  
RecurrenceType.Monthly, Interval = 1, DayOfMonth = 16, RecurrenceRange = RecurrenceRange.Count,  
RecurrenceCount = 10 | FREQ=MONTHLY; BYMONTHDAY=16; INTERVAL=1; COUNT=10 |

| | Appointment is created with selected date and Ends On | RecurrenceType =  
RecurrenceType.Monthly, Interval = 1, DayOfMonth = 16, RecurrenceRange =  
RecurrenceRange.EndDate, EndDate = new DateTime(2018, 06, 11) | FREQ=MONTHLY;  
BYMONTHDAY=17; INTERVAL=1; UNTIL=06/11/2018 |

| | Appointment is created with selected day Ends Never | RecurrenceType = RecurrenceType.Monthly,  
Interval = 1, Week = 2, DayOfWeek = 6, RecurrenceRange = RecurrenceRange.NoEndDate |  
FREQ=MONTHLY; BYDAY=FR; BYSETPOS=2; INTERVAL=1 |

| | Appointment is created with selected day and Ends After | RecurrenceType =  
RecurrenceType.Monthly, Interval = 1, Week = 4, DayOfWeek = 4, RecurrenceRange =  
RecurrenceRange.Count, RecurrenceCount = 10 | FREQ=MONTHLY; BYDAY=WE; BYSETPOS=4;  
INTERVAL=1; COUNT=10 |

| | Appointment is created with selected day and Ends On | RecurrenceType =  
RecurrenceType.Monthly, Interval = 1, Week = 4, DayOfWeek = 6, RecurrenceRange =  
RecurrenceRange.EndDate, EndDate = new DateTime(2018, 06, 11) | FREQ=MONTHLY; BYDAY=FR;  
BYSETPOS=4; INTERVAL=1; UNTIL=06/11/2018 |

| Yearly | Appointment is created with selected date and month Ends Never | RecurrenceType =  
RecurrenceType.Yearly, Interval = 1, Month = 12, DayOfMonth = 15, RecurrenceRange =  
RecurrenceRange.NoEndDate | FREQ=YEARLY; BYMONTHDAY=15; BYMONTH=12; INTERVAL=1 |

| | Appointment is created with selected date and month Ends After | RecurrenceType =  
RecurrenceType.Yearly, Interval = 1, Month = 12, DayOfMonth = 10, RecurrenceRange =  
RecurrenceRange.Count, RecurrenceCount = 10 | FREQ=YEARLY; BYMONTHDAY=10; BYMONTH=12;  
INTERVAL=1; COUNT=10 |

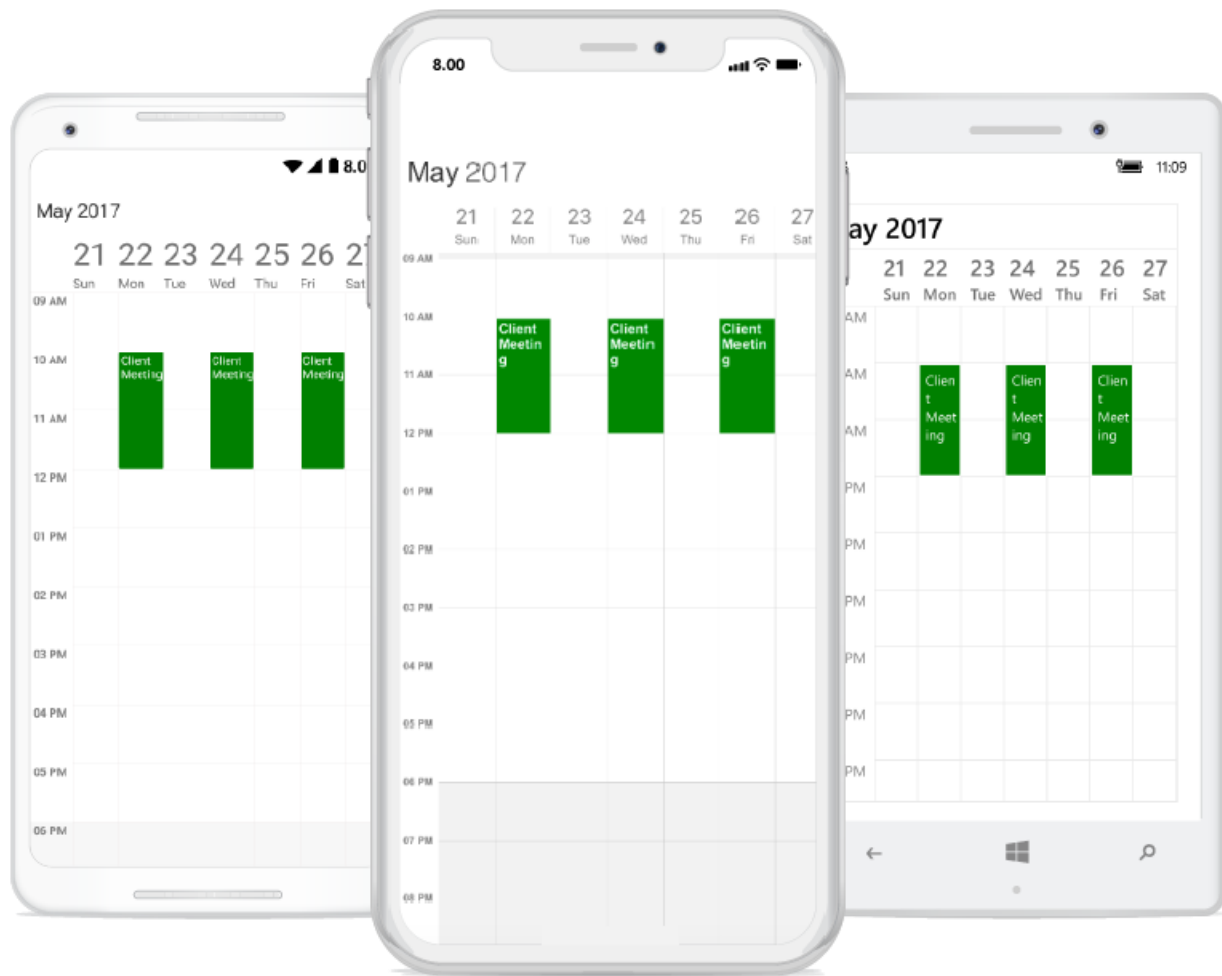
| | Appointment is created with selected date and month Ends On | RecurrenceType =  
RecurrenceType.Yearly, Interval = 1, Month = 12, DayOfMonth = 12, RecurrenceRange =  
RecurrenceRange.EndDate, EndDate = new DateTime(2018, 06, 11) | FREQ=YEARLY; BYMONTHDAY=12;  
BYMONTH=12; INTERVAL=1; UNTIL=06/11/2018 |

*Adding Recurrence Appointment using Recurrence Builder*

Schedule appointment [RecurrenceRule](#) is used to populate the required recurring appointment collection in a specific pattern. **RRULE** can be easily created through **RecurrenceBuilder** engine by using [RRuleGenerator](#) method in schedule.

**C#**

```
// Creating an instance for schedule appointment collection
ScheduleAppointmentCollection scheduleAppointmentCollection = new
ScheduleAppointmentCollection();
//Adding schedule appointment in schedule appointment collection
var scheduleAppointment = new ScheduleAppointment()
{
    StartTime = new DateTime(2017, 05, 08, 10, 0, 0),
    EndTime = new DateTime(2017, 05, 08, 12, 0, 0),
    Subject = "Occurs every alternate day",
};
//Adding schedule appointment in schedule appointment collection
scheduleAppointmentCollection.Add(scheduleAppointment);
//Adding schedule appointment collection to DataSource of SfSchedule
schedule.DataSource=scheduleAppointmentCollection;
// Creating recurrence rule
RecurrenceProperties recurrenceProperties = new RecurrenceProperties();
recurrenceProperties.RecurrenceType = RecurrenceType.Daily;
recurrenceProperties.RecurrenceRange = RecurrenceRange.Count;
recurrenceProperties.Interval = 2;
recurrenceProperties.RecurrenceCount = 10;
scheduleAppointment.RecurrenceRule =
schedule.RRuleGenerator(recurrenceProperties, scheduleAppointment.StartTime,
scheduleAppointment.EndTime);
```



### Creating Custom Recurrence Appointment using Recurrence Builder

For creating custom recurrence appointment you need to create a custom class `Meeting` with mandatory fields `From`, `To`, `EventName` 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 Color Color { get; set; }
    public string RecurrenceRule { 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" ScheduleView="DayView" DataSource="{Binding Meetings}">
  {Binding Meetings}>
</syncfusion:SfSchedule>
<syncfusion:SfSchedule.AppointmentMapping>
<syncfusion:ScheduleAppointmentMapping
  SubjectMapping="EventName"
  ColorMapping="Color"
  StartTimeMapping="From"
  EndTimeMapping="To"
  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.ColorMapping = "Color";
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 = Color.Green;
// Creating recurrence rule
RecurrenceProperties recurrenceProperties = new RecurrenceProperties();
recurrenceProperties.RecurrenceType = RecurrenceType.Weekly;
recurrenceProperties.RecurrenceRange = RecurrenceRange.Count;
recurrenceProperties.Interval = 1;
recurrenceProperties.WeekDays =
  WeekDays.Monday | WeekDays.Wednesday | WeekDays.Friday;
recurrenceProperties.RecurrenceCount = 10;
```

```

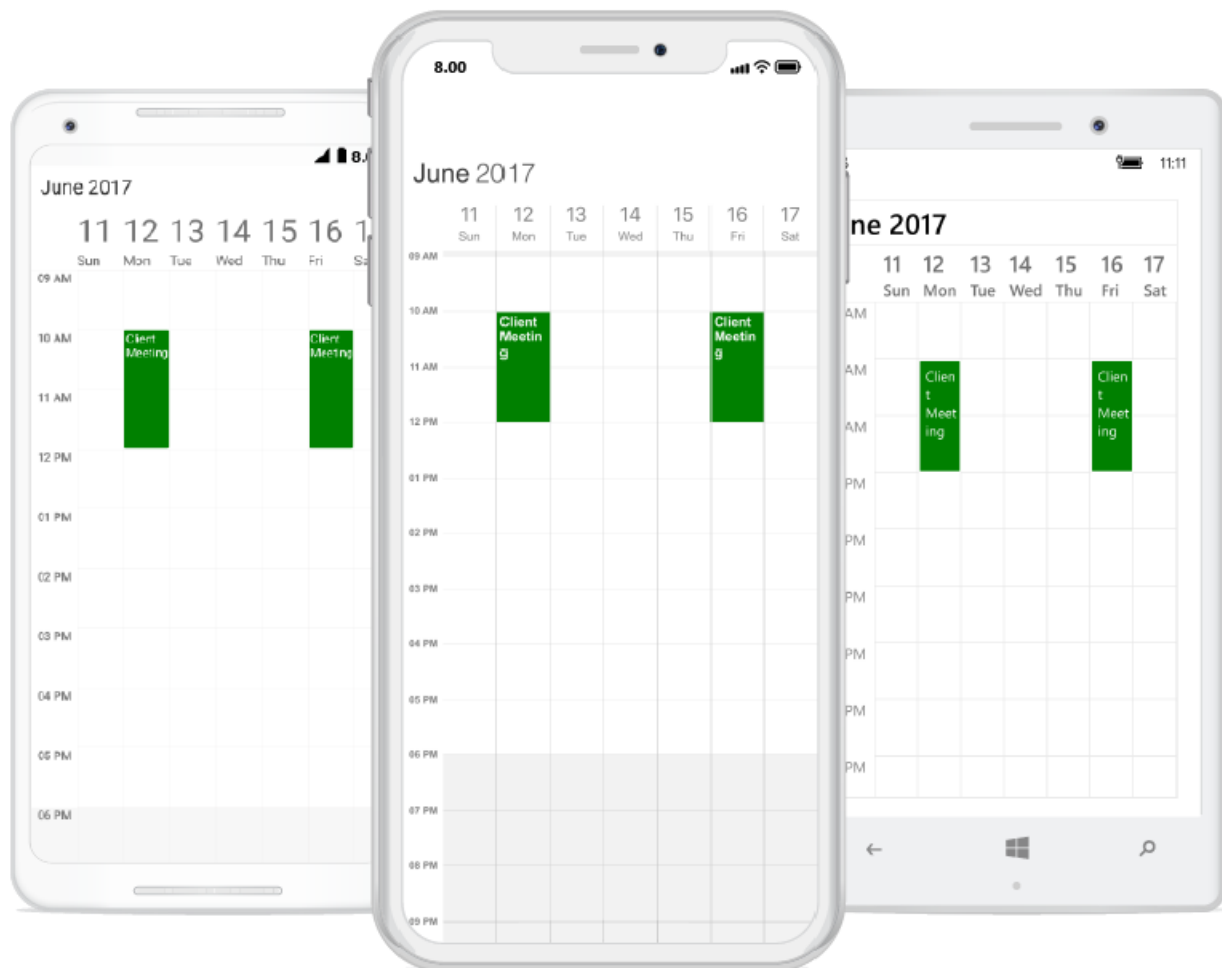
recurrenceProperties.RecurrenceRule =
schedule.RRuleGenerator(recurrenceProperties, meeting.From, Meeting.To);
// Setting recursive rule for an event
meeting.RecurrenceRule = recurrenceProperties.RecurrenceRule;
// Creating instance for collection of custom appointments
var Meetings = new ObservableCollection<Meeting>();
// Adding a custom appointment in CustomAppointmentCollection
Meetings.Add(meeting);
// Adding custom appointments in DataSource of SfSchedule
schedule.DataSource = Meetings;

```

You can download the entire source code of this demo for Xamarin.Forms from [here Recurrence Appointment](#).

## NOTE

In Schedule "Xamarin.Forms UWP", there is no need to set IsRecursive property for recurrence appointments. When a RecurrenceRule is set to schedule appointment, value of IsRecursive property will be set as true automatically for these appointments. So even if IsRecursive is set as false, there will be no effect on recurring appointments.



### *How to get the Recurrence editor field values from RRULE?*

You can get the Recurrence properties from **RRULE** using the [RRuleParser](#) method of **SfSchedule**.

#### **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 = RecurrenceType.Daily;
```

```
recurrenceProperties.Interval = 1;
```

```
recurrenceProperties.RecurrenceCount = 3;
```

```
recurrenceProperties.RecurrenceRange = RecurrenceRange.Count;
```

### *How to get the occurrences date time list of recurring appointment from RRULE?*

You can get the occurrences date time list of recurring appointment from **RRULE** using the [GetRecurrenceDateTimeCollection](#) method of **SfSchedule**.

#### **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;
```

### *Recurrence Pattern Exceptions*

You can delete or change any recurrence pattern appointment by handling exception dates and exception appointments to that recurring appointment.

#### *Recurrence Exception Dates*

You can delete any occurrence appointment which is exception from the recurrence pattern appointment by adding exception dates to the recurring appointment.

#### *Recurrence Exception appointment*

You can also change any occurrence appointment which is exception from recurrence pattern appointment by adding the recurrence exception appointment in the schedule **DataSource**.

#### *Create recurrence exceptions for schedule appointment*

You can add/remove the recurrence exception appointments and recurrence exception dates to **ScheduleAppointment** by using its property, [RecurrenceExceptionDates](#), [RecurrenceId](#), [ExceptionOccurrenceActualDate](#).

Delete occurrence from recurrence pattern appointment or adding exception dates to recurrence pattern appointment

You can delete any of occurrence which is exception from recurrence pattern appointment by using **RecurrenceExceptionDates** property of **ScheduleAppointment**. The deleted occurrence date will be considered as recurrence exception dates.

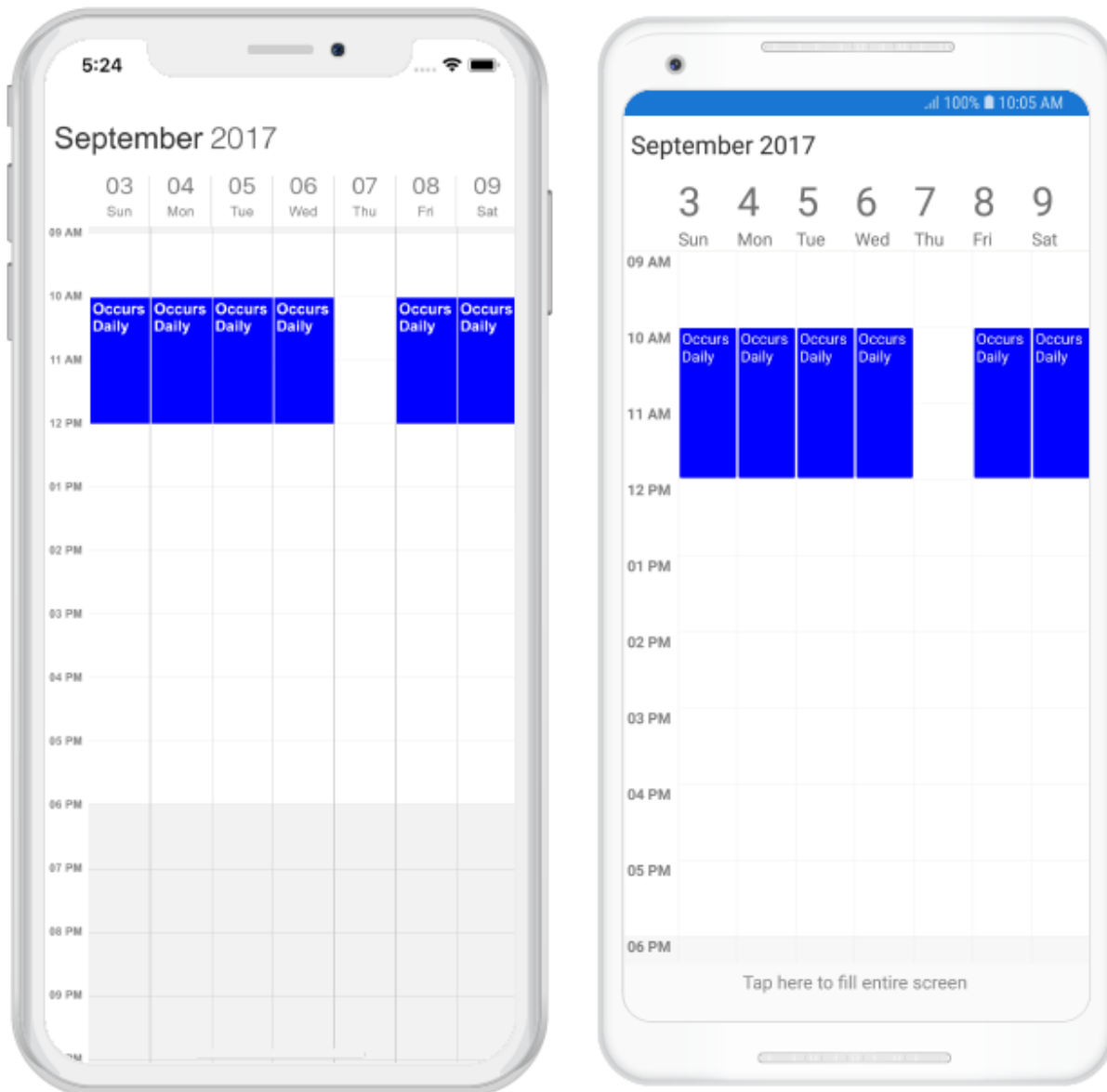
### C#

```
// 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 RecurrenceExceptionDates to appointment.
recurrenceAppointment.RecurrenceExceptionDates = new
ObservableCollection<DateTime>()
{
    exceptionDate
};
```

### NOTE

- Exception dates should be Universal Time Coordinates (UTC) time zone.
- You can also update the RecurrenceExceptionDates collection dynamically.





[Delete occurrence from recurrence pattern dynamically or add exception dates to recurrence pattern dynamically](#)

You can also delete any occurrence from the recurrence pattern appointment by adding exception date to the `RecurrenceExceptionDates` collection.

### C#

```
var recurrenceAppointment = scheduleAppointmentCollection[0];
recurrenceAppointment.RecurrenceExceptionDates.Add(new DateTime(2017, 09, 05));
```

[Add deleted occurrence to recurrence pattern dynamically or remove exception dates from recurrence pattern dynamically](#)

You can also add the deleted occurrence to the recurrence pattern appointment by removing exception date from the `RecurrenceExceptionDates` collection.

#### C#

```
var recurrenceAppointment = scheduleAppointmentCollection[0];
recurrenceAppointment.RecurrenceExceptionDates.RemoveAt(0);
```

#### NOTE

If you add the deleted occurrence to the recurrence pattern by removing exception date when any [exception appointment](#) has been created for the mentioned exception date, the respective exception appointment will be deleted by matching with `RecurrenceId` and `ExceptionOccurrenceActualDate` from Schedule `DataSource` and recurrence pattern appointment created for that exception date.

[Add all deleted occurrences to recurrence pattern dynamically or clear exception dates from recurrence pattern dynamically](#)

You can also add all deleted occurrences to the recurrence pattern appointment by clearing the exception dates from the `RecurrenceExceptionDates` collection.

#### C#

```
var recurrenceAppointment = scheduleAppointmentCollection[0];
recurrenceAppointment.RecurrenceExceptionDates.Clear();
```

[Add exception appointment to recurrence pattern](#)

You can change any occurrence appointment which is an exception from the recurrence pattern appointment by using the `RecurrenceId` property which is used to map the exception appointment with recurrence pattern appointment and `ExceptionOccurrenceActualDate` property which is used to mention the actual pattern occurrence date of exception appointment of `ScheduleAppointment`.

You should add the created exception recurrence appointment to the schedule [DataSource](#).

#### 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 RecurrenceExceptionDates to appointment.
recurrenceAppointment.RecurrenceExceptionDates = new
ObservableCollection<DateTime>()
{
    exceptionDate
};
```

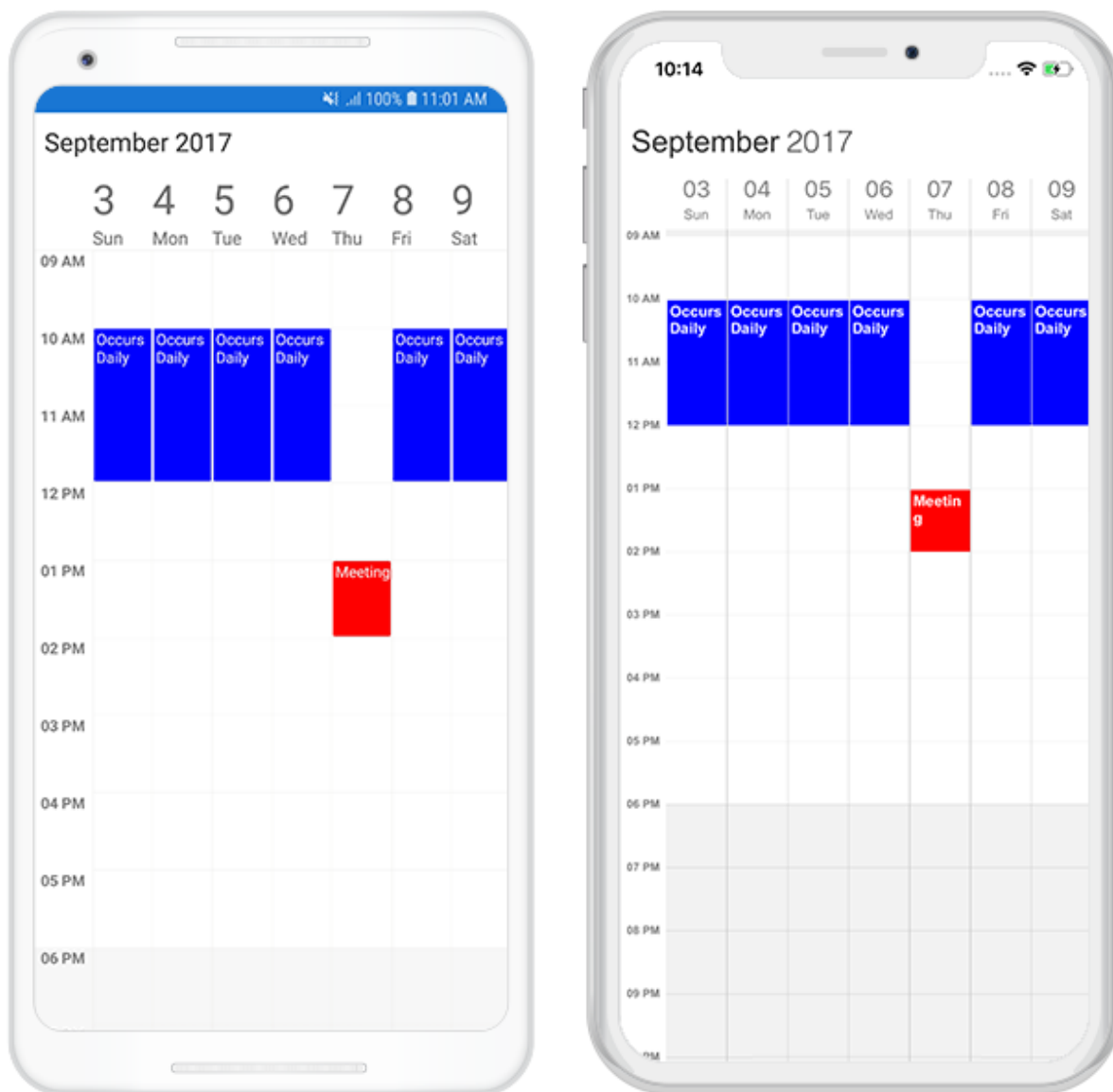
```
// Add exception appointment to the current recurrence pattern
var exceptionAppointment = new ScheduleAppointment()
{
    StartTime = new DateTime(2017, 09, 07, 12, 0, 0),
    EndTime = new DateTime(2017, 09, 07, 14, 0, 0),
    Subject = "Meeting",
    Color = Color.Red,
    // Recurrence Id should be parent appointment object
    RecurrenceId = recurrenceAppointment,
    //Actual occurrence date
    ExceptionOccurrenceActualDate = exceptionDate
};
```

---

**NOTE**

---

- **RecurrenceId** should be a recurrence pattern appointment object.
- Exception appointment should be a normal appointment and should not be created as recurring appointment, since its occurrence is from recurrence pattern.
- **ExceptionOccurrenceActualDate** should be in Universal Time Coordinates (UTC) time zone.



#### Add exception appointment to recurrence pattern dynamically

You can also add exception appointment dynamically for added exception date by adding exception appointment to the schedule `DataSource` which is exception from the recurrence pattern appointment by using the `RecurrenceId` property which is used to map the exception appointment with recurrence pattern appointment and `ExceptionOccurrenceActualDate` property which is used to mention the actual pattern occurrence date of exception appointment of the `ScheduleAppointment` class.

#### C#

```
var recurrenceAppointment = scheduleAppointmentCollection[0];
var exceptionDate = new DateTime(2017, 09, 07);
// Add exception appointment to the current recurrence series
var exceptionAppointment = new ScheduleAppointment()
{
    StartTime = new DateTime(2017, 09, 07, 12, 0, 0),
```

```

EndTime = new DateTime(2017, 09, 07, 14, 0, 0),
Subject = "Meeting",
Color = Color.Red,
RecurrenceId = recurrenceAppointment, // set the parent appointment to
recurrence Id
//Actual occurrence date
ExceptionOccurrenceActualDate = exceptionDate
};
//Adding exception appointment in schedule appointment collection
scheduleAppointmentCollection.Add(exceptionAppointment);

```

## NOTE

- **RecurrenceId** should be a recurrence pattern appointment object.
- Exception appointment should be a normal appointment and should not be created as recurring appointment, since its occurrence from recurrence pattern.
- **ExceptionOccurrenceActualDate** should be in Universal Time Coordinates (UTC) time zone.

[Remove exception appointment from recurrence pattern](#)

You can directly remove the added exception appointment for recurrence pattern by removing it from schedule **DataSource**.

## C#

```

var exceptionAppointment = scheduleAppointmentCollection[1];
//Remove exception appointment from schedule appointment collection
scheduleAppointmentCollection.Remove(exceptionAppointment);

```

You can download the entire source code of this demo for Xamarin.Forms from here [RecurrenceExceptions](#).

[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 fields **RecurrenceExceptionDates**, **ActualDate**, **RecurrenceId**.

[Delete occurrence from recurrence pattern appointment or adding exception dates to recurrence pattern appointment](#)

You can delete any occurrence which is exception from the recurrence pattern appointment by using the [RecurrenceExceptionDatesMapping](#) property of **ScheduleAppointmentMapping** class which is used to map the exception dates to the schedule recurrence appointment. The deleted occurrence date will be considered as recurrence exception dates.

To add the exception dates in the recurrence series of custom appointment, add the **RecurrenceExceptionDates** property to custom class **Meeting**.

## C#

```

public ObservableCollection<DateTime> RecurrenceExceptionDates { get; set; }
= new ObservableCollection<DateTime>();

```

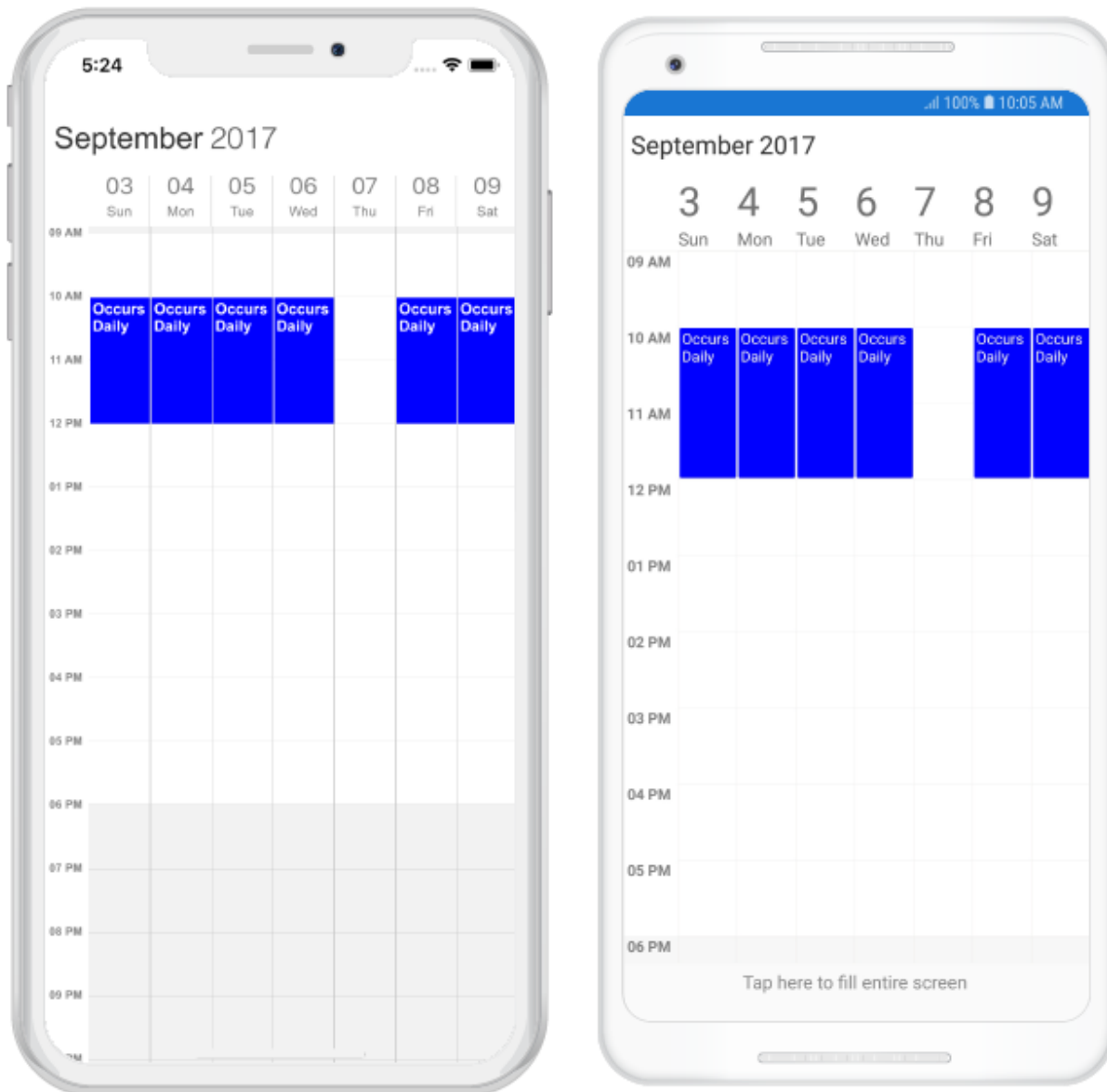
You should map this custom property `RecurrenceExceptionDates` of custom class with the `RecurrenceExceptionDatesMapping` property of `ScheduleAppointmentMapping` class to map the exception dates to the schedule appointment.

### C#

```
// data mapping for custom appointments.
dataMapping.RecurrenceExceptionDatesMapping = "RecurrenceExceptionDates";
// 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",
    EventColor = Color.Blue
};
// Creating recurrence rule
recurrenceAppointment.RecurrenceRule = "FREQ=DAILY;COUNT=20";
// Add RecurrenceExceptionDates to appointment.
recurrenceAppointment.RecurrenceExceptionDates = new
ObservableCollection<DateTime>()
{
    exceptionDate
};
```

### NOTE

- Exception dates should be in Universal Time Coordinates (UTC) time zone.
- You can also dynamically update the custom property `RecurrenceExceptionDates` collection.



[Delete occurrence from recurrence pattern dynamically or add exception dates to recurrence pattern dynamically](#)

You can also delete any occurrence from the recurrence pattern appointment by adding exception date to the `RecurrenceExceptionDates` custom property collection.

#### C#

```
var recurrenceAppointment = eventCollection[0];
recurrenceAppointment.RecurrenceExceptionDates.Add(new DateTime(2017, 09, 05));
```

[Add deleted occurrence to recurrence pattern dynamically or remove exception dates from recurrence pattern dynamically](#)

You can also add the deleted occurrence to the recurrence pattern appointment by removing exception date from the `RecurrenceExceptionDates` custom property collection.

#### C#

```
var recurrenceAppointment = eventCollection[0];
recurrenceAppointment.RecurrenceExceptionDates.RemoveAt(0);
```

#### NOTE

If you add the deleted occurrence to the recurrence pattern by removing exception date when any [exception appointment](#) has been created for the mentioned exception date, the respective exception appointment will be deleted by matching with `RecurrenceId` and `ActualDate` from the schedule `DataSource` and recurrence pattern appointment created for that exception date.

[Add all deleted occurrence to recurrence pattern dynamically or clear exception dates from recurrence pattern dynamically](#)

You can also add all deleted occurrence to the recurrence pattern appointment by clearing the exception dates from the `RecurrenceExceptionDates` custom property collection.

#### C#

```
var recurrenceAppointment = eventCollection[0];
recurrenceAppointment.RecurrenceExceptionDates.Clear();
```

[Add exception appointment to recurrence pattern](#)

You can change any occurrence appointment which is exception from the recurrence pattern appointment by using the [RecurrenceIdMapping](#) property of `ScheduleAppointmentMapping` class which is used to map the custom exception appointment with schedule recurrence series appointment and [ExceptionOccurrenceActualDateMapping](#) property of `ScheduleAppointmentMapping` class which is used to mention the actual series occurrence date of exception appointment of schedule recurrence appointment.

For adding custom exception appointment to the recurrence series, add the `ActualDate` and `RecurrenceId` properties to custom class `Meeting`.

#### C#

```
public DateTime ActualDate { get; set; }
public object RecurrenceId { get; set; }
```

You should map this custom property `RecurrenceId` of `Meeting` with the `RecurrenceIdMapping` property of `ScheduleAppointmentMapping` class which is used to map the exception appointment with schedule recurrence series appointment `Meeting`.

You should also map this custom property `ActualDate` of `Meeting` with the `ExceptionOccurrenceActualDateMapping` property of `ScheduleAppointmentMapping` class which is used to mention the actual series occurrence date of exception appointment with schedule recurrence appointment.



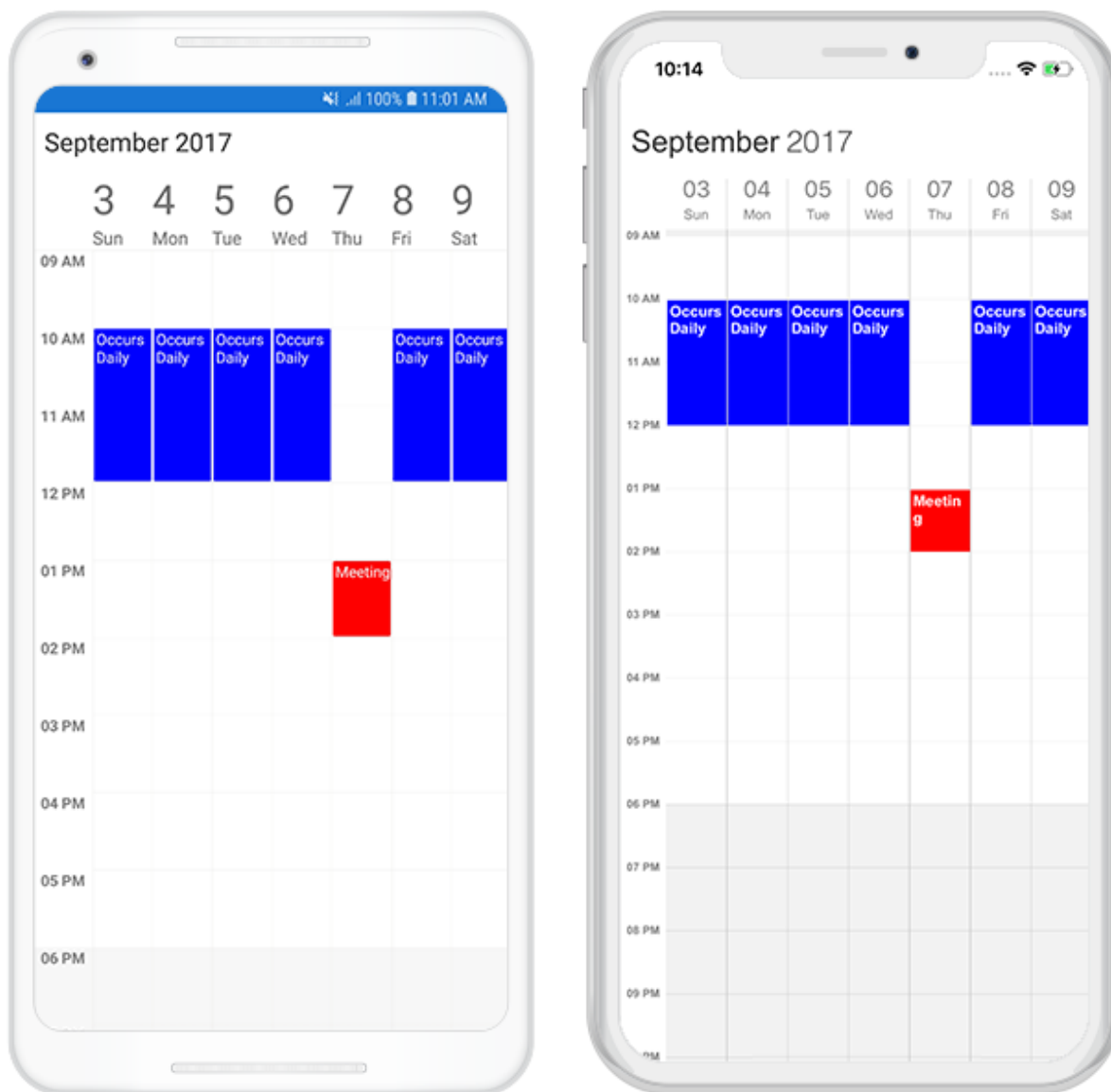
You should add the created exception recurrence appointment to the schedule `DataSource`.

### C#

```
// Create the new exception date.
var exceptionDate = new DateTime(2017, 09, 07);
//Adding schedule appointment in schedule appointment collection
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",
    EventColor = Color.Blue
};
recurrenceAppointment.RecurrenceRule = "FREQ=DAILY;COUNT=20";
// Add RecurrenceExceptionDates to appointment.
recurrenceAppointment.RecurrenceExceptionDates = new
ObservableCollection<DateTime>()
{
    exceptionDate
};
// Add exception appointment to the current recurrence series
var exceptionAppointment = new Meeting()
{
    From = new DateTime(2017, 09, 07, 12, 0, 0),
    To = new DateTime(2017, 09, 07, 14, 0, 0),
    EventName = "Meeting",
    EventColor = Color.Red,
    RecurrenceID = recurrenceAppointment,
    ActualDate = exceptionDate
};
```

### NOTE

- `RecurrenceId` should be a recurrence pattern appointment object.
- Exception appointment should be a normal appointment and should not be created as recurring appointment, since its occurrence from recurrence pattern.
- `ActualDate` should be in Universal Time Coordinates (UTC) time zone.



#### Add exception appointment to recurrence pattern dynamically

You can also add exception appointment dynamically for added exception date by adding exception appointment to the schedule `DataSource` by using the `RecurrenceIdMapping` property of `ScheduleAppointmentMapping` class which is used to map the custom exception appointment with schedule recurrence series appointment and `ExceptionOccurrenceActualDateMapping` property of the `ScheduleAppointmentMapping` class which is used to mention the actual series occurrence date of exception appointment of schedule recurrence appointment.

#### C#

```
var recurrenceAppointment = eventCollection[0];
var exceptionDate = new DateTime(2017, 09, 07);
// Add exception appointment to the current recurrence series
var exceptionAppointment = new Meeting()
{
```

```

From = new DateTime(2017, 09, 07, 12, 0, 0),
To = new DateTime(2017, 09, 07, 14, 0, 0),
EventName = "Meeting",
EventColor = Color.Red,
RecurrenceID = recurrenceAppointment, // set the parent appointment to
recurrence Id.
//Actual occurrence date
ActualDate = exceptionDate
};
eventCollection.Add(exceptionAppointment);

```

## NOTE

- **RecurrenceId** should be a recurrence pattern appointment object.
- Exception appointment should be a normal appointment and should not be created as recurring appointment, since its occurrence from recurrence pattern.
- **ActualDate** should be in Universal Time Coordinates (UTC) time zone.

### Remove exception appointment from recurrence pattern

You can directly remove the added exception appointment for recurrence pattern by removing from the schedule **DataSource**.

## C#

```

var exceptionAppointment = eventCollection[1];
eventCollection.Remove(exceptionAppointment);

```

You can download the entire source code of this demo for Xamarin.Forms from here [RecurrenceExceptions](#).

### Get visible appointments

You can get the list of visible appointments by using [GetVisibleAppointments](#) method available in schedule. It is applicable for all schedule views.

### Get visible appointments from given date time range

You can get the visible appointments in schedule by passing the start and end **DateTime** range to [GetVisibleAppointments](#) method.

## C#

```

List<ScheduleAppointment> visibleAppointments =
schedule.GetVisibleAppointments(new DateTime(2017, 05, 08, 10, 0, 0), new
DateTime(2017, 05, 12, 10, 0, 0));

```

### Get visible appointments from date time

You can get the visible appointments in schedule by passing **DateTime** to [GetVisibleAppointments](#) method.

## C#

```

List<ScheduleAppointment> visibleAppointments =
schedule.GetVisibleAppointments(new DateTime(2017, 05, 08, 10, 0, 0));

```

---

**NOTE**

---

- You can get the visible appointments after rendering the schedule only.
- The specified start/end Time ranges should lie on schedule current visible dates range.
- `GetVisibleAppointments` method will always returns the `List<ScheduleAppointment>` even if it has a custom appointment collection.

#### Suspend and resume the appointment update

Schedule allows you to suspend and resume the appointment UI update while performing collection changes (Add/Remove/Reset). [SuspendAppointmentUpdate](#) method will suspend appointment UI rendering until you resume it when large number of data added dynamically in schedule `DataSource` to avoid each time updating UI when collection changes. After data added dynamically in schedule, you can call [ResumeAppointmentUpdate](#) to update the appointment UI rendering.

**C#**

```
// Creating an instance for schedule appointment collection
ScheduleAppointmentCollection scheduleAppointmentCollection = new
ScheduleAppointmentCollection();
//Adding schedule appointment in schedule appointment collection
var scheduleAppointment = new ScheduleAppointment()
{
    StartTime = new DateTime(2017, 05, 08, 10, 0, 0),
    EndTime = new DateTime(2017, 05, 08, 12, 0, 0),
    Subject = "Project Discussion",
    Color=Color.Red
};
scheduleAppointmentCollection.Add(scheduleAppointment);
schedule.DataSource = scheduleAppointmentCollection;
//Trigger the visible dates changed event.
schedule.VisibleDatesChangedEvent+= Schedule_VisibleDatesChangedEvent;
private void Schedule_VisibleDatesChangedEvent(object sender,
VisibleDatesChangedEventArgs e)
{
    // Suspends the Appointment Update.
    schedule.SuspendAppointmentUpdate();
    for (int i = 0; i < e.visibleDates.Count; i++)
    {
        var visibleDate = e.visibleDates[i].Date;
        var scheduleAppointment = new ScheduleAppointment()
        {
            StartTime = visibleDate.AddHours(10),
            EndTime = visibleDate.AddHours(12),
            Subject = visibleDate.ToString("dd/MM/yyyy"),
            Color = Color.Red
        };
        scheduleAppointmentCollection.Add(scheduleAppointment);
    }
    // Resumes the Appointment Update.
    schedule.ResumeAppointmentUpdate();
}
```

### Appearance Customization

The default appearance of the appointment can be customized by using the [AppointmentStyle](#) property and [AppointmentLoadedEvent](#). The event and property is used to customize or override the default template of the Appointments.

- [Customize appearance using Style](#)
- [Customize appearance using Event](#)
- [Customize appearance using Custom View](#)
- [Customize appearance using DataTemplate](#)
- [Customize appearance using DataTemplateSelector](#)

#### *Customize appearance using Style*

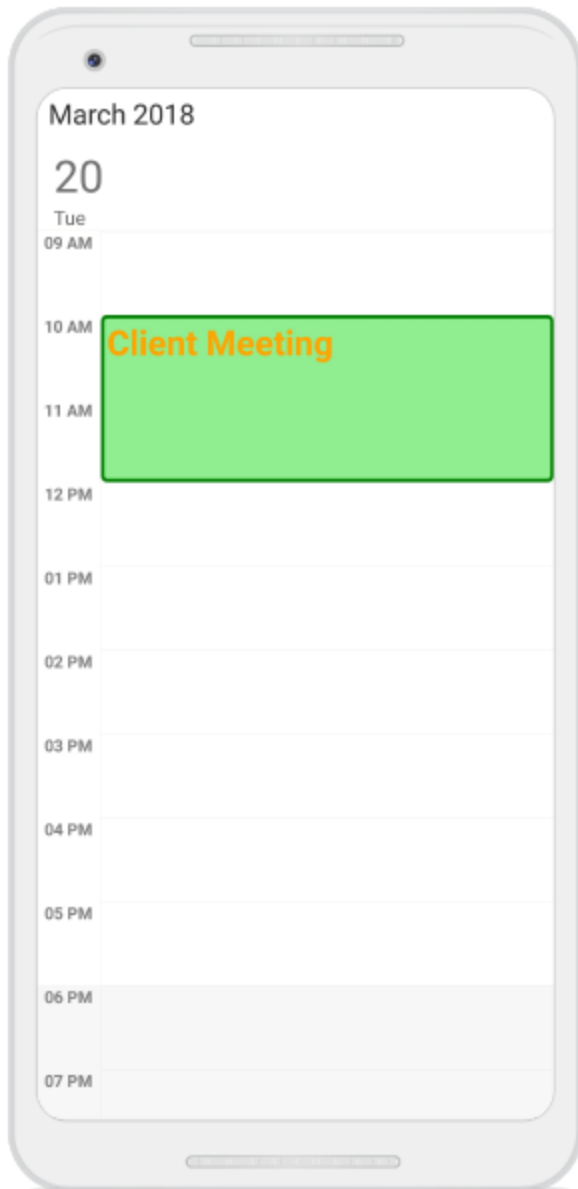
Schedule appointment can be customized by setting appointment style properties such as [TextColor](#), [FontFamily](#), [FontSize](#), [FontAttributes](#), [BorderColor](#), [BorderCornerRadius](#), [BorderWidth](#) to the [AppointmentStyle](#) property of [SfSchedule](#).

#### **XML**

```
<syncfusion:SfSchedule x:Name="schedule" ScheduleView="DayView"
DataSource="{Binding Meetings}">
<syncfusion:SfSchedule.AppointmentStyle>
<syncfusion:AppointmentStyle
BorderWidth="10"
TextColor="Red"
BorderCornerRadius="10"
FontSize = "25"
FontAttributes = "Bold"
FontFamily = "Arial"
BorderColor="Blue">
</syncfusion:AppointmentStyle>
</syncfusion:SfSchedule.AppointmentStyle>
</syncfusion:SfSchedule>
```

#### **C#**

```
//Creating Appointment style
AppointmentStyle appointmentStyle = new AppointmentStyle();
appointmentStyle.TextColor = Color.Red;
appointmentStyle.FontSize = 25;
appointmentStyle.FontAttributes = FontAttributes.Bold;
appointmentStyle.FontFamily = "Arial";
appointmentStyle.BorderColor = Color.Blue;
appointmentStyle.BorderCornerRadius = 12;
appointmentStyle.BorderWidth = 10;
//Setting Appointment Style
schedule.AppointmentStyle = appointmentStyle;
```



#### *Customize appearance using Event*

Schedule appointment can be customized during runtime using [OnAppointmentLoadedEvent](#). `ScheduleAppointment` style can be customized using the `appointmentStyle` property.

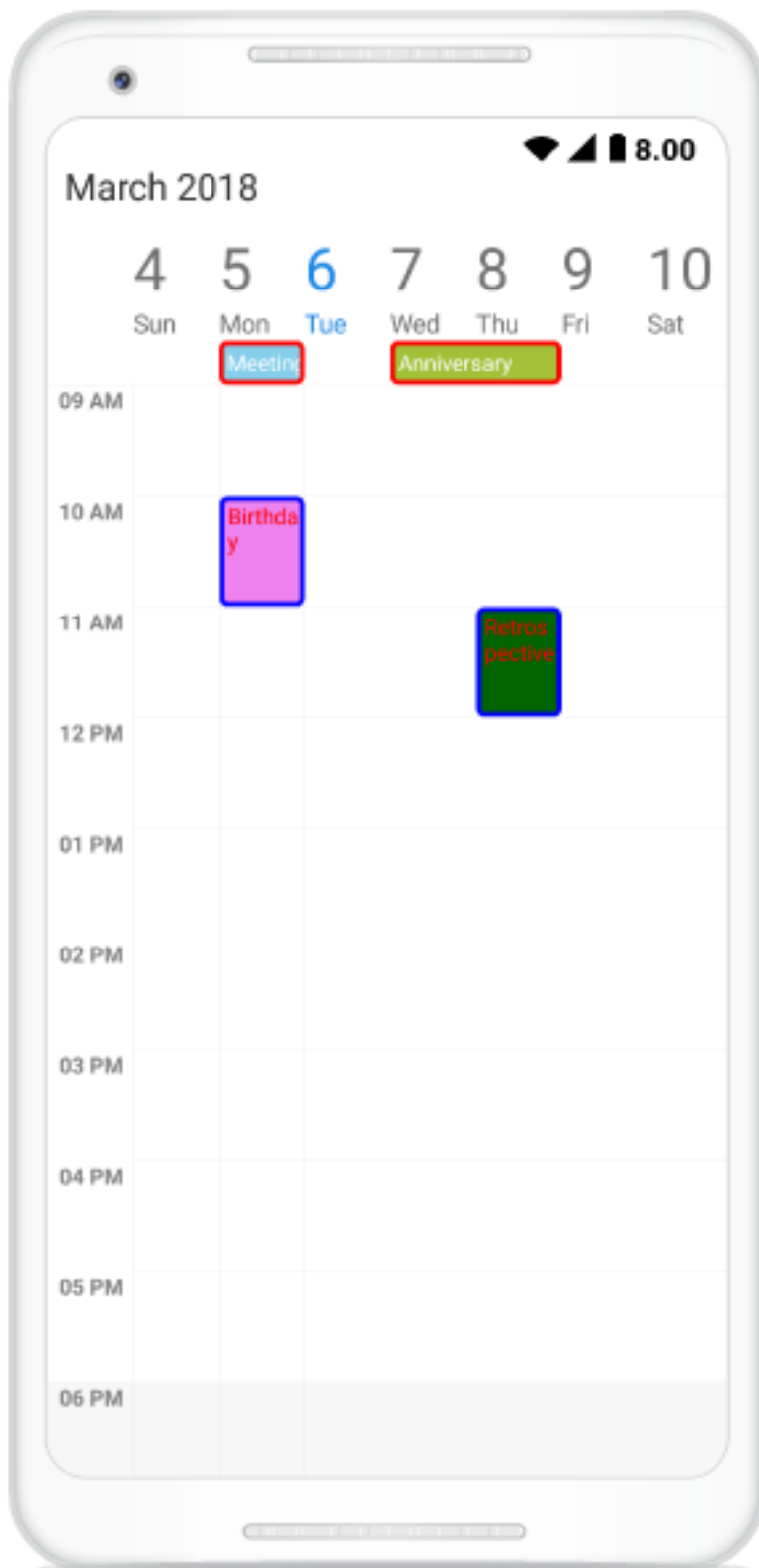
[AppointmentLoadedEventArgs](#) has below properties,

- [appointment](#) – Contains the appointments values.
- [appointmentStyle](#) – Gets and sets the appointments style.
- [view](#) - Sets the Custom UI for Appointments.
- [Bounds](#) – Contains the UI bounds of appointment.

#### **C#**

```
schedule.OnAppointmentLoadedEvent += Schedule_OnAppointmentLoadedEvent;
```

```
...  
private void Schedule_OnAppointmentLoadedEvent(object sender,  
AppointmentLoadedEventArgs args)  
{  
    if (args.appointment != null && (args.appointment as Meeting) != null &&  
        (args.appointment as Meeting).IsAllDay)  
    {  
        args.appointmentStyle.BorderColor = Color.Red;  
        args.appointmentStyle.TextColor = Color.White;  
        args.appointmentStyle.BorderCornerRadius = 12;  
        args.appointmentStyle.BorderWidth = 10;  
    }  
    else  
    {  
        args.appointmentStyle.BorderColor = Color.Blue;  
        args.appointmentStyle.TextColor = Color.Red;  
        args.appointmentStyle.BorderCornerRadius = 12;  
        args.appointmentStyle.BorderWidth = 10;  
    }  
}
```



---

**NOTE**

---



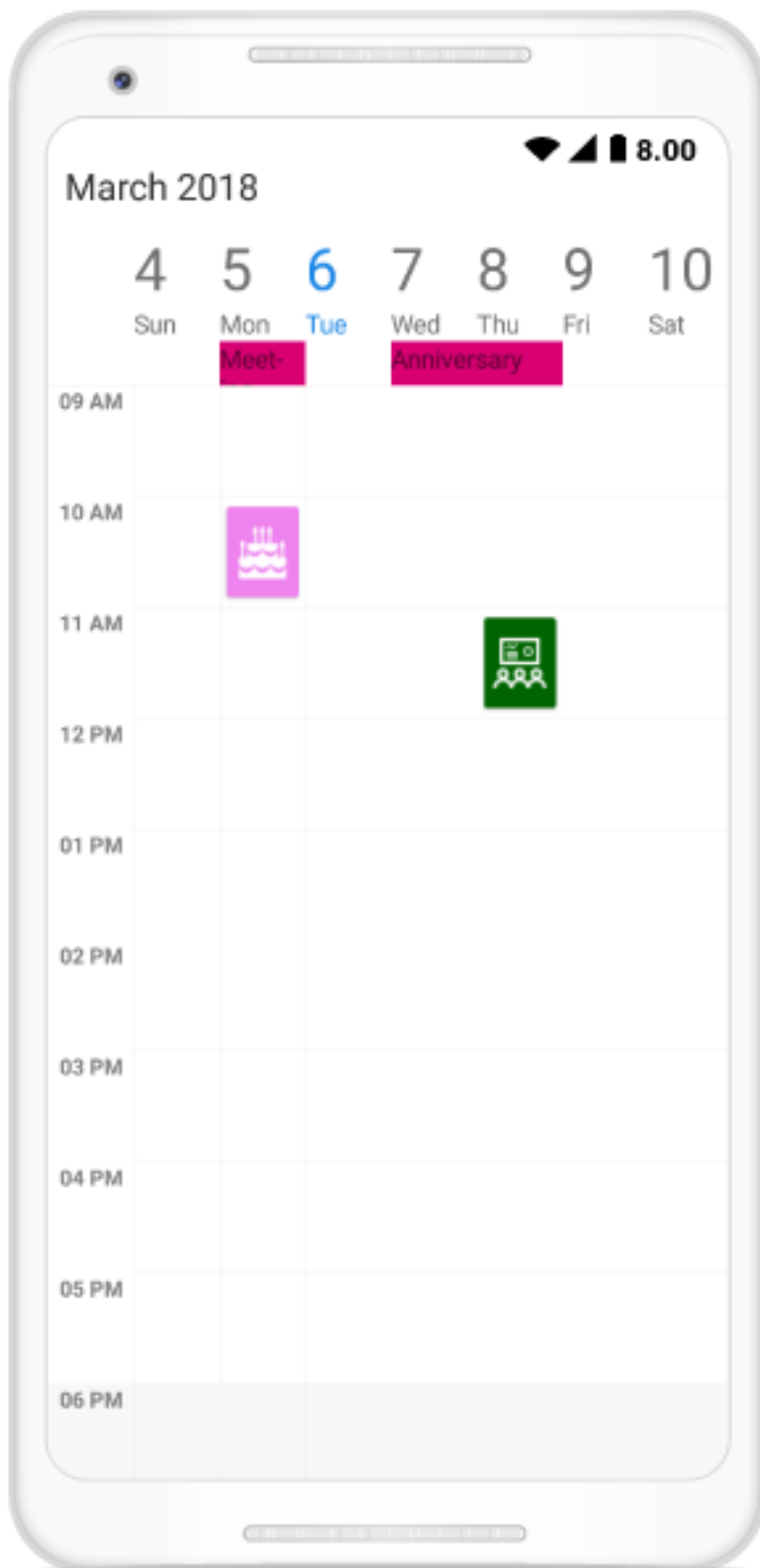
FontAttributes and FontFamily are native to the platform. Custom font and the font which are not available in the specified platform will not be applied.

#### Customize appearance using Custom View

Default appointment UI can be changed using `view` property passed through `AppointmentLoadedEventArgs`.

#### C#

```
schedule.OnAppointmentLoadedEvent += Schedule_OnAppointmentLoadedEvent;
private void Schedule_OnAppointmentLoadedEvent(object sender,
AppointmentLoadedEventArgs args)
{
    if(args.appointment == null || (args.appointment as Meeting) == null)
        return;
    if ((args.appointment as Meeting).IsAllDay)
    {
        Label label = new Label();
        label.BackgroundColor = (args.appointment as Meeting).Color;
        label.Text = (args.appointment as Meeting).EventName;
        args.view = label;
    }
    else if ((args.appointment as Meeting).EventName == "Retrospective" )
    {
        Button button = new Button();
        button.Image = "Meeting.png";
        button.BackgroundColor = (args.appointment as Meeting).EventName;
        args.view = button;
    }
    else
    {
        Button button = new Button();
        button.Image = "Cake.png";
        button.BackgroundColor = (args.appointment as Meeting).EventName;
        args.view = button;
    }
}
```



### Customize appearance using DataTemplate

The default appearance of the Appointment can be customized by using the [AppointmentTemplate](#) property of the Schedule.

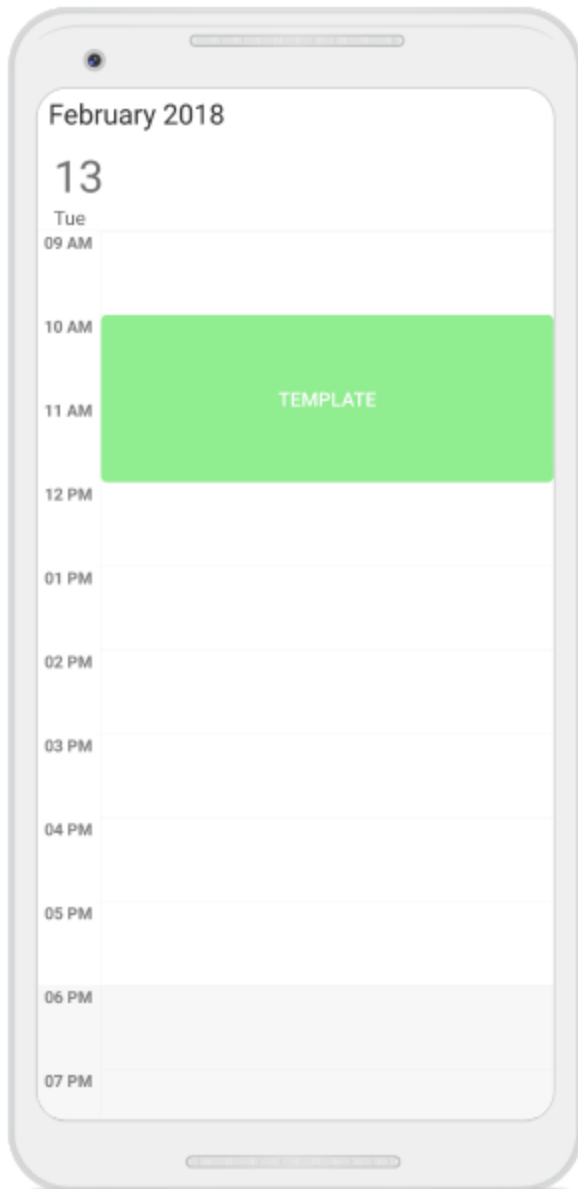
#### **XML**

```
<schedule:SfSchedule
x:Name="schedule"
AppointmentTemplate="{Binding AppointmentTemplate}">
<schedule:SfSchedule.BindingContext>
<samplelocal:DayAppointmentDataTemplate />
</schedule:SfSchedule.BindingContext>
</schedule:SfSchedule>
```

### Creating a DataTemplate

#### **C#**

```
public class DayAppointmentDataTemplate : DataTemplate
{
    public DataTemplate AppointmentTemplate { get; set; }
    public DayAppointmentDataTemplate()
    {
        AppointmentTemplate = new DataTemplate(() =>
        {
            return new Button
            {
                Text = "Template",
                TextColor = Color.White,
                BackgroundColor = Color.LightGreen
            };
        });
    }
}
```



### Customize appearance using DataTemplateSelector

**DataTemplateSelector** can be used to choose a **DataTemplate** at runtime based on the value of a data-bound to Schedule appointment property through **AppointmentTemplate**. It provides multiple DataTemplates to be enabled for Schedule appointments, to customize the appearance of particular Appointment.

#### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<samplelocal:AppointmentDataTemplateSelector
x:Key="appointmentDataTemplateSelector" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
```

```
<schedule:SfSchedule
AppointmentTemplate="{StaticResource appointmentDataTemplateSelector}">
<schedule:SfSchedule.BindingContext>
<samplelocal:AppointmentDataTemplateSelector />
</schedule:SfSchedule.BindingContext>
</schedule:SfSchedule>
</ContentPage.Content>
```

### Creating a DataTemplateSelector

#### C#

```
public class AppointmentDataTemplateSelector : DataTemplateSelector
{
    public DataTemplate DayAppointmentTemplate { get; set; }
    public DataTemplate AllDayAppointmentTemplate { get; set; }
    public AppointmentDataTemplateSelector()
    {
        DayAppointmentTemplate = new DataTemplate(typeof(DayAppointmentTemplate));
        AllDayAppointmentTemplate = new
        DataTemplate(typeof(AllDayAppointmentTemplate));
    }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject
    container)
    {
        if ((item as ScheduleAppointment).IsAllDay)
        return AllDayAppointmentTemplate;
        else
        return DayAppointmentTemplate;
    }
}
```

Used button to display day appointment and Label to display all day appointment.

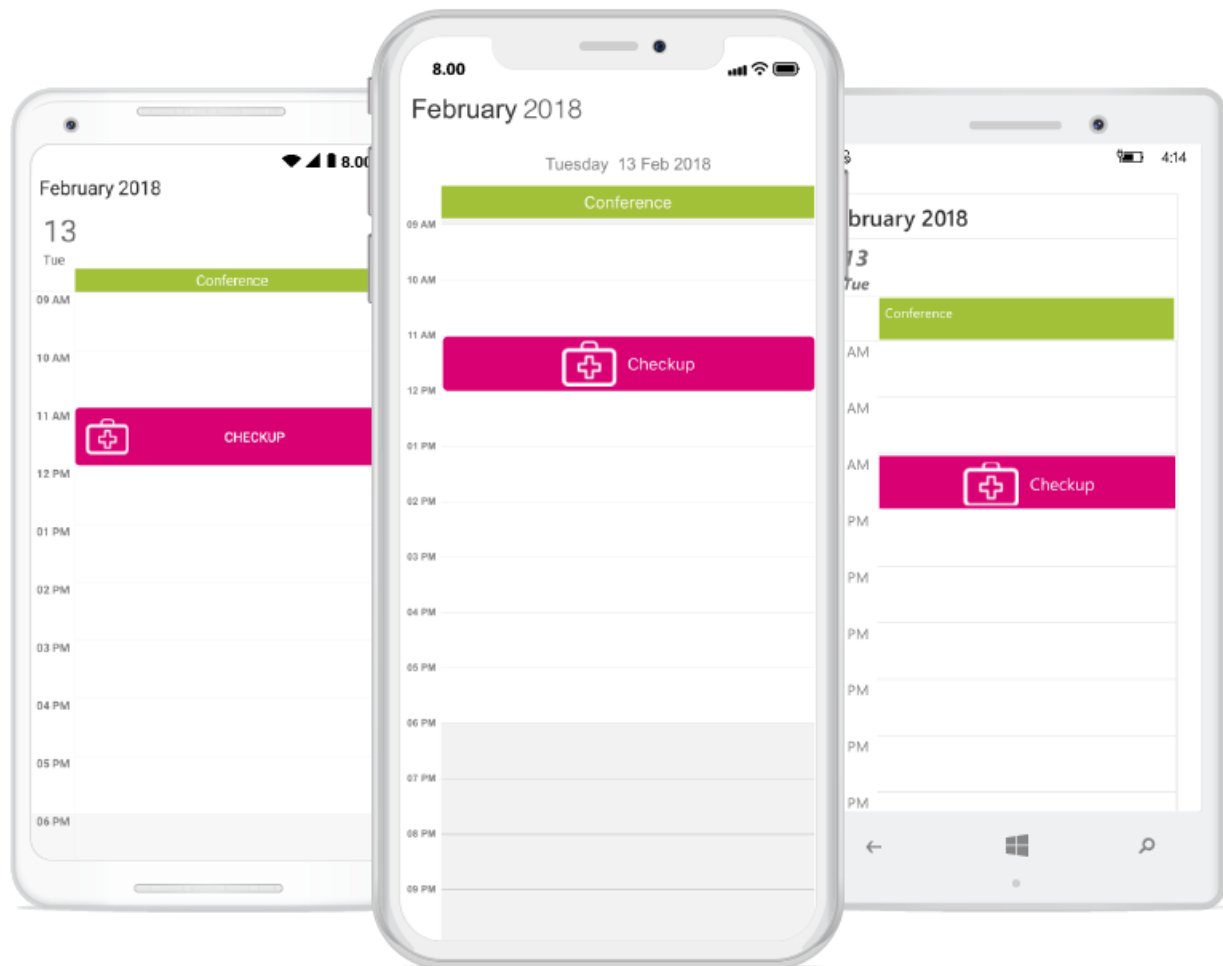
#### XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!--<Button as Template for day Appointment>-->
<Button xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="ScheduleUG.DayAppointmentTemplate"
BackgroundColor="{Binding Color}"
HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand"
Text="{Binding Subject}"
TextColor="White"
Image="{Binding Subject}">
</Button>
.....
<?xml version="1.0" encoding="UTF-8"?>
<!--<Label as Template for all day Appointment>-->
<Label xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="ScheduleUG.AllDayAppointmentTemplate"
BackgroundColor="{Binding Color}"
Text="{Binding Subject}"
HorizontalOptions="FillAndExpand"
```

```

VerticalOptions="CenterAndExpand"
TextColor="White" FontSize="15"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center">
</Label>

```



### Customize Font Appearance

you can change the appearance of Font by setting the [FontFamily](#) property of [AppointmentStyle](#) property in Schedule.

### XML

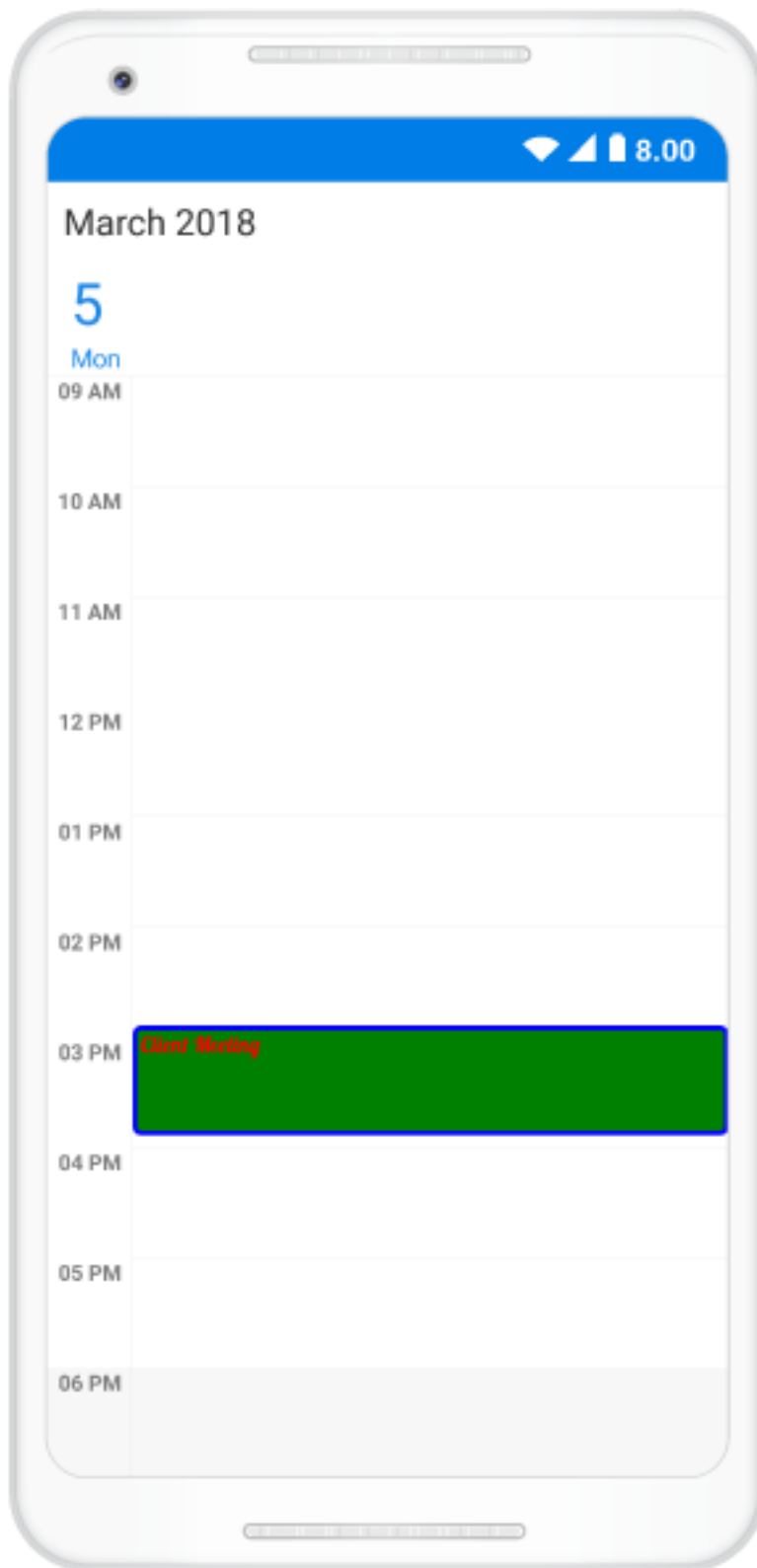
```

<schedule:AppointmentStyle.FontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf" WinPhone="Assets/Lobster-Regular.ttf#Lobster"
/>
</schedule:AppointmentStyle.FontFamily>

```

### C#

```
appointmentStyle.FontFamily = Device.OnPlatform("Lobster-Regular", "Lobster-  
Regular.ttf", "Assets/Lobster-Regular.ttf#Lobster");
```



Refer [this](#) to configure the custom fonts in Xamarin.Forms.



### Selection

Schedule control has built-in events to handle tapped, double tapped and long pressed touch actions.

- [CellTapped](#)
- [CellDoubleTapped](#)
- [CellLongPressed](#)

These events will be triggered while perform respective touch actions in timeslots, month cells and in appointments. All the three events contain the same argument [CellTappedEventArgs](#) which holds selected appointment and date time details in it.

- [Appointment](#) - Contains the selected appointment value when tapped on the appointment. It will be null when tapped on the timeslot. Selected occurrence of a recurring appointment's value will be same as the master appointment, except the date values. So selected occurrence's date can be obtained from the args.DateTime value.
- [Appointments](#) - Contains appointments value of Tapped month cell.
- [DateTime](#) - Contains selected time slot DateTime value.

---

N Occurrences can be handled from tapped event when single occurrence remains unmodified.

---

### C#

```
schedule.CellTapped += Schedule_CellTapped;
schedule.CellDoubleTapped += Schedule_CellDoubleTapped;
schedule.CellLongPressed += Schedule_CellLongPressed;
...
private void Schedule_CellTapped(object sender, CellTappedEventArgs e)
{
}
private void Schedule_CellDoubleTapped(object sender, CellTappedEventArgs e)
{
}
private void Schedule_CellLongPressed(object sender, CellTappedEventArgs e)
{
}
```

### Commands

Schedule commands allow data bindings to make method calls directly to a ViewModel, which supports tapped, double tapped, long pressed touch actions and visible date changed action.

- [CellTappedCommand](#)
- [CellDoubleTappedCommand](#)
- [CellLongPressedCommand](#)
- [VisibleDatesChangedCommand](#)

### XML

```
<schedule:SfSchedule
CellTappedCommand="{Binding ScheduleCellTapped}"
CellDoubleTappedCommand="{Binding ScheduleCellDoubleTapped}"
CellLongPressedCommand="{Binding ScheduleCellLongPressed}"
VisibleDatesChangedCommand="{Binding ScheduleVisibleDatesChanged}">
```

```
<schedule:SfSchedule.BindingContext>
<samplelocal:ScheduleViewModel />
</schedule:SfSchedule.BindingContext>
</schedule:SfSchedule>
```

## C#

```
public class ScheduleViewModel
{
    public ICommand ScheduleCellTapped { get; set; }
    public ICommand ScheduleCellDoubleTapped { get; set; }
    public ICommand ScheduleCellLongPressed { get; set; }
    public ICommand ScheduleVisibleDatesChanged { get; set; }
    public ScheduleViewModel()
    {
        ScheduleCellTapped = new Command<CellTappedEventArgs>(CellTapped);
        ScheduleCellDoubleTapped = new Command<CellTappedEventArgs>(DoubleTapped);
        ScheduleCellLongPressed = new Command<CellTappedEventArgs>(LongPressed);
        ScheduleVisibleDatesChanged = new
        Command<VisibleDatesChangedEventArgs>(VisibleDatesChanged);
    }
    private void CellTapped(CellTappedEventArgs args)
    {
        var selectedDateTime = args.Datetime;
    }
    private void DoubleTapped(CellTappedEventArgs args)
    {
        var selectedDateTime = args.Datetime;
    }
    private void LongPressed(CellTappedEventArgs args)
    {
        var selectedDateTime = args.Datetime;
    }
    private void VisibleDatesChanged(VisibleDatesChangedEventArgs args)
    {
        var visibleDates = args.visibleDates;
    }
}
```

### Selection customization

The default selection of an appointment can be customized by using [SelectionBorderColor](#), [SelectionTextColor](#) properties in [AppointmentStyle](#) property of [SfSchedule](#). The property is used to customize or override the default selection of the appointments.

**Note:** [BorderWidth](#) value must be set to highlight [SelectionBorderColor](#).

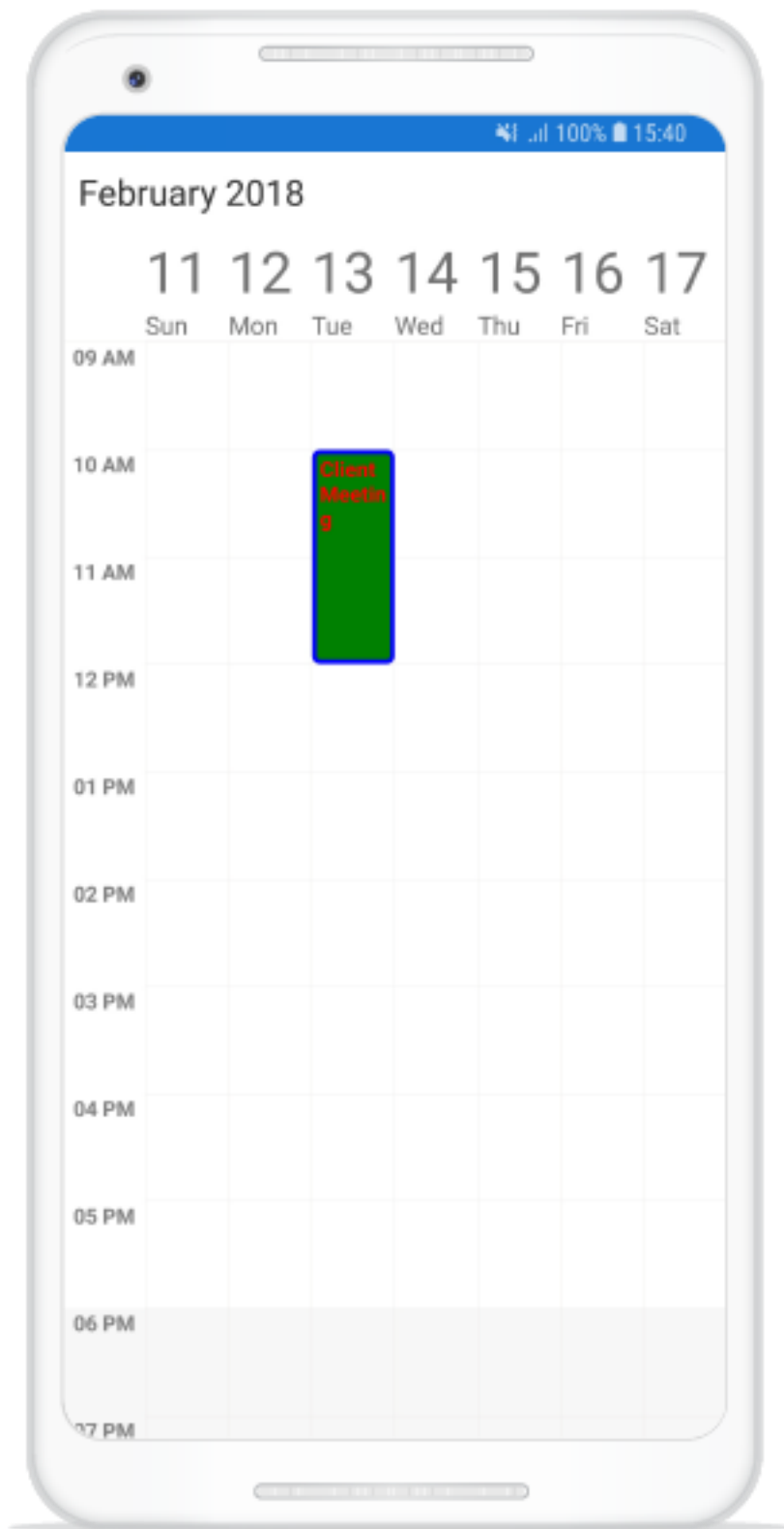
## XML

```
<syncfusion:SfSchedule x:Name="schedule" ScheduleView="DayView"
DataSource="{Binding Meetings}">
<syncfusion:SfSchedule.AppointmentStyle>
<syncfusion:AppointmentStyle
SelectionTextColor="Yellow"
SelectionBorderColor="Yellow">
</syncfusion:AppointmentStyle>
```

```
</syncfusion:SfSchedule.AppointmentStyle>  
</syncfusion:SfSchedule>
```

### **C#**

```
//Creating an appointment style  
AppointmentStyle appointmentStyle = new AppointmentStyle();  
appointmentStyle.SelectionBorderColor = Color.Yellow;  
appointmentStyle.SelectionTextColor = Color.Yellow;  
//Setting an appointment style  
schedule.AppointmentStyle = appointmentStyle;
```



### How to raise schedule events while using custom view for appointments?

You can raise schedule events such as `CellTapped`, `CellDoubleTapped`, `CellLongPressed` by using the `InputTransparent` property of custom view.

#### C#

```
// Raise cell tapped event.
schedule.CellTapped += Schedule_CellTapped;
// Raise Appointment Loaded event.
schedule.OnAppointmentLoadedEvent += Schedule_OnAppointmentLoadedEvent;
...
private void Schedule_OnAppointmentLoadedEvent(object sender,
AppointmentLoadedEventArgs e)
{
    var button = new Button { Text = "CustomView", BackgroundColor = Color.Blue
};
// Set input transparent to `true` to raise schedule inbuilt event.
button.InputTransparent = true;
e.view = button;
}
private void Schedule_CellTapped(object sender, CellTappedEventArgs e)
{
    DisplayAlert("", "Cell Tapping Raised", "ok");
}
```

#### NOTE

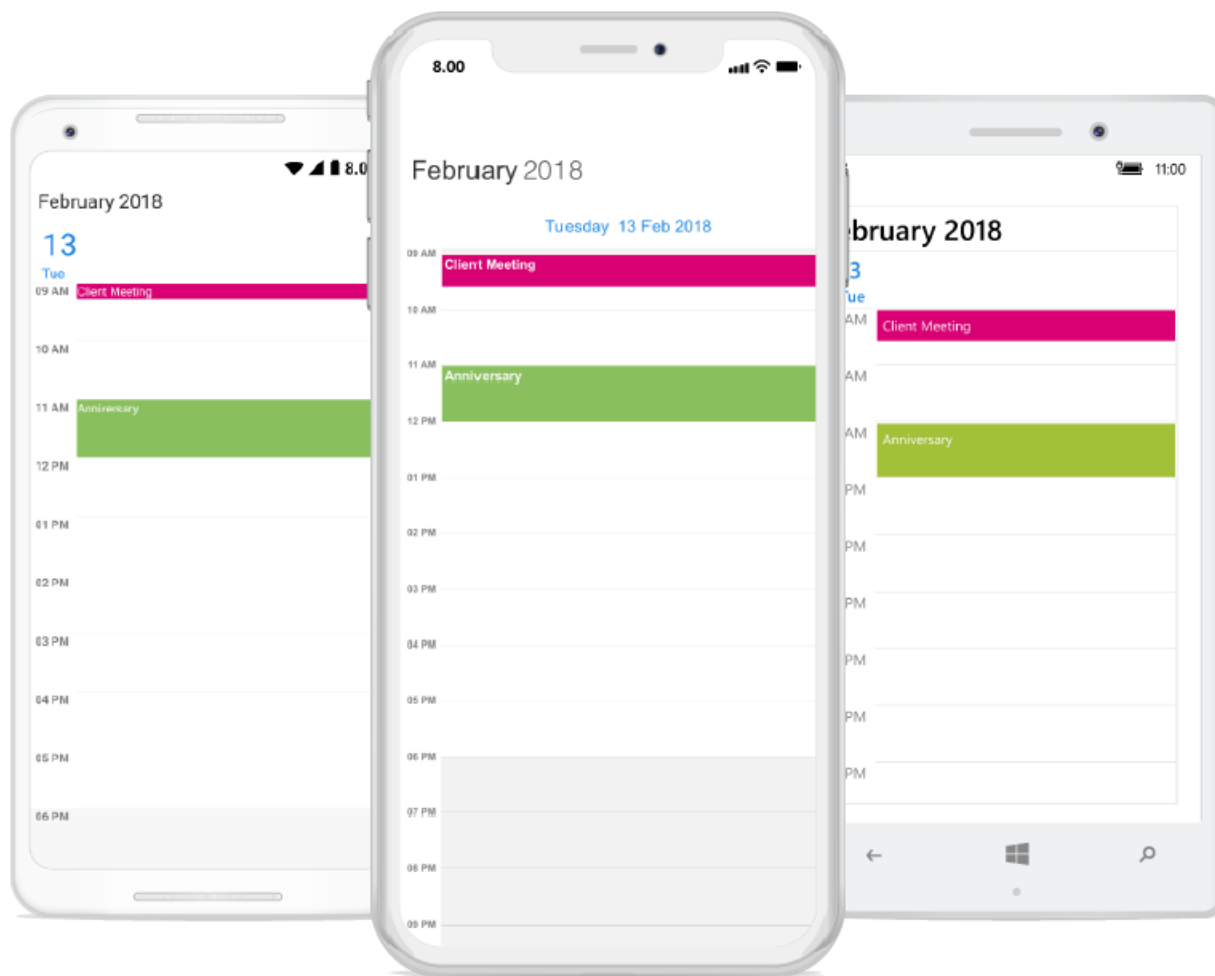
`InputTransparent` default value is false, custom view event will be raised by default.

#### Minimum Appointment Height

`MinHeight` of an appointment is to set an arbitrary height to appointments when it has minimum duration, so that the subject can be readable.

#### C#

```
schedule.ScheduleView = ScheduleView.DayView;
ScheduleAppointmentCollection scheduleAppointmentCollection = new
ScheduleAppointmentCollection();
scheduleAppointmentCollection.Add(new ScheduleAppointment()
{
    StartTime = new DateTime(2018, 2, 13, 09, 0, 0),
    EndTime = new DateTime(2018, 2, 13, 09, 0, 0),
    Subject = "Client Meeting",
    MinHeight = 30,
    Color = Color.FromHex("#FFD80073")
});
scheduleAppointmentCollection.Add(new ScheduleAppointment()
{
    StartTime = new DateTime(2018, 2, 13, 11, 0, 0),
    EndTime = new DateTime(2018, 2, 13, 12, 0, 0),
    Subject = "Anniversary",
    Color = Color.FromHex("#FFA2C139")
});
schedule.DataSource = scheduleAppointmentCollection;
this.Content = schedule;
```



---

**NOTE**

- **MinHeight** value will be set, when the an appointment height (duration) value lesser than MinHeight.
- Appointment height (duration) value will be set, when the appointment height (duration) value greater than **MinHeight**.
- TimeInterval value will be set, when Minimum Height greater than TimeInterval with lesser appointment height (duration).
- **MinHeight** has ScheduleAppointmentMapping Support.
- All day Appointment does not support **MinHeight**.

### Resource view

Resource view displays the resources as discrete views integrated with the scheduler to display appointments in all types of schedule views. It provides an intuitive user interface, which allows users to select single or multiple resources and display the events associated with the selected resources with efficient and effective utilization. Each resource can be assigned to a unique color to more easily identify the resource associated with an appointment.

You can add resources that can be assigned to appointments using the [ScheduleResources](#) property of SfSchedule. You need to set the [Name](#), [Id](#), and [Color](#) properties of [ScheduleResource](#) to create a resource.

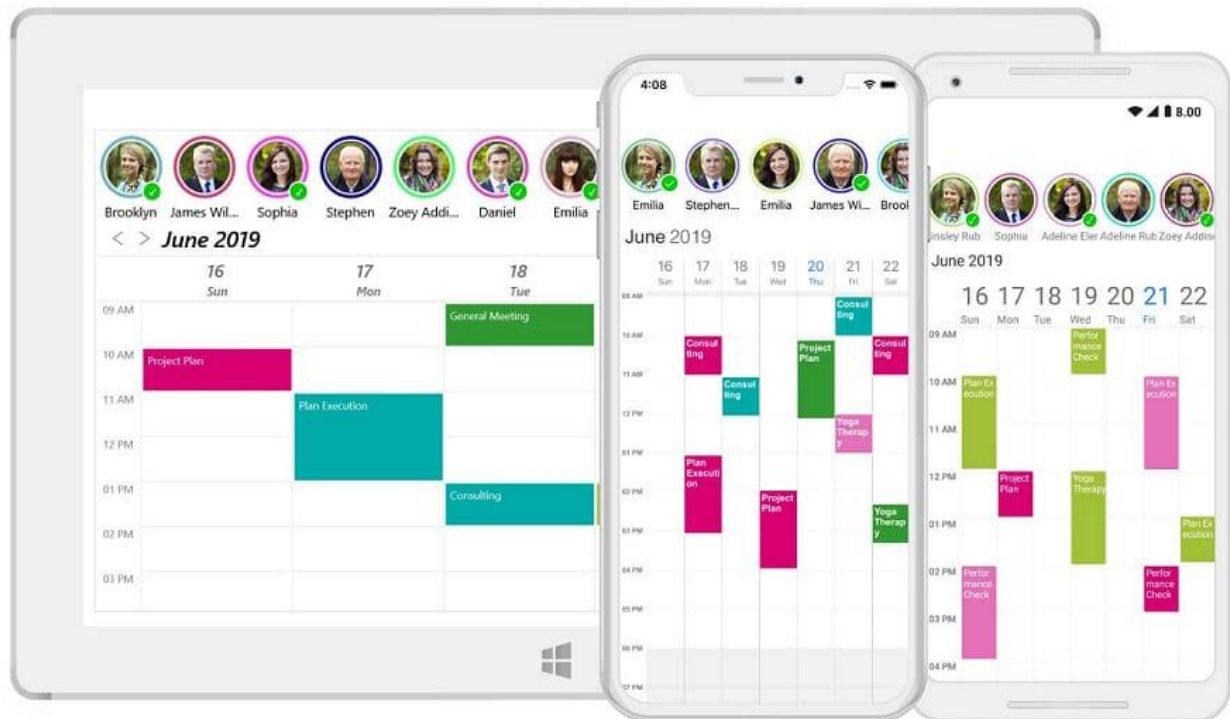
### XML

```
<syncfusion:SfSchedule
x:Name="schedule"
ScheduleView="WeekView"
ShowResourceView="True">
<syncfusion:SfSchedule.ScheduleResources>
<syncfusion:ScheduleResource
Name="Brooklyn"
Id="5601"
Color="#FF3399" />
</syncfusion:SfSchedule.ScheduleResources>
</syncfusion:SfSchedule>
```

### C#

```
// Creating an instance for schedule resource collection.
ObservableCollection<object> resources = new ObservableCollection<object>();
// Adding schedule resource in schedule resource collection.
resources.Add(new ScheduleResource()
{
    Name = "Brooklyn",
    Id = 5601,
    Color = Color.FromHex("#FF3399")
});
// Adding schedule resource collection to schedule resources of SfSchedule.
schedule.ScheduleResources = resources;
```

You can download the custom resource demo for Xamarin.Forms from [here](#).



### Resource view visibility

You can handle the visibility of resource view using the [ShowResourceView](#) bool property of SfSchedule. By default, value of this property is set to false.

### XML

```
<schedule:SfSchedule ScheduleView="WeekView" ShowResourceView="True" />
```

### C#

```
schedule.ScheduleView = ScheduleView.WeekView;
schedule.ShowResourceView = true;
```

### Assigning events for resources

You can associate Resources to the appointments by adding Id of resource in the [ResourceIds](#) property of ScheduleAppointment. Appointments associated with the selected resources will be displayed in the SfSchedule views.

### 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, 05, 08, 10, 0, 0),
    EndTime = new DateTime(2019, 05, 08, 12, 0, 0),
    Subject = "Meeting",
    Location = "Hutchison road",
```



```

ResourceIds = new ObservableCollection<object> { 5601, 5602 }
});
//Adding schedule appointment collection to DataSource of SfSchedule
schedule.DataSource = scheduleAppointmentCollection;

```

### Assigning custom events to resources

You can associate resources to the custom appointments using the equivalent field of **ResourceIds** in custom appointment class.

### Creating custom events

You can create a custom class **Event** with mandatory fields **From**, **To**, **EventName**, and **Resources**.

### 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; }
    public Color Color { get; set; }
    public ObservableCollection<object> Resources { get; set; }
}

```

### NOTE

You can inherit this class from **INotifyPropertyChanged** for dynamic changes in custom data.

You can map the properties of **Meeting** class with our **SfSchedule** control using **ScheduleAppointmentMapping**.

### XML

```

<syncfusion:SfSchedule x:Name="schedule" ScheduleView="DayView"
DataSource="{Binding Meetings}">
<syncfusion:SfSchedule.AppointmentMapping>
<syncfusion:ScheduleAppointmentMapping
SubjectMapping="EventName"
ColorMapping="Color"
StartTimeMapping="From"
EndTimeMapping="To"
ResourceIdsMapping="Resources">
</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.ColorMapping = "Color";
dataMapping.ResourceIdsMapping = "Resources";
schedule.AppointmentMapping = dataMapping;
```

You can associate resources to the custom events by adding **Id** of resource in the **Resources** property of custom appointment class.

### C#

```
// Creating an instance for custom appointment class.
Meeting meeting = new Meeting();
meeting.From = new DateTime(2017, 06, 11, 10, 0, 0);
meeting.To = meeting.From.AddHours(2);
meeting.EventName = "Client Meeting";
meeting.Color = Color.Green;
// Setting resources for an event.
meeting.Resources = new ObservableCollection<object> () {5601, 5604};
```

### NOTE

- You can also associate custom resources to the appointments by using the equivalent field of resource **Id** in custom resource class.
- All appointments will be displayed when the **ShowResourceView** property is set to false.
- If an appointment is mapped to a single resource, it will be displayed in resource **Color**. If an appointment is mapped to multiple resources, it will be displayed in the default appointment **Color**.

### Mapping

Schedule supports full data binding to any type of **IEnumerable** source. Specify the [ResourceMapping](#) attribute to map the properties in the underlying data source to the schedule resource.

Property Name	Description
<a href="#">Name</a>	Maps the property name of custom class, which is equivalent to Name in ScheduleResource.
<a href="#">Id</a>	Maps the property name of custom class, which is equivalent to Id in ScheduleResource.
<a href="#">Image</a>	Maps the property name of custom class, which is equivalent to Image in ScheduleResource.
<a href="#">Color</a>	Maps the property name of custom class, which is equivalent to Color in ScheduleResource.

### NOTE

Custom resource class should contain a mandatory field for resource **Id**.

#### Creating a custom resource

You can create a custom class **Employee** with required fields such as **Name**, **Id**, **Color**, and **DisplayPicture**.

### C#

```

/// <summary>
/// Represents custom data properties.
/// </summary>
public class Employee
{
    public string Name { get; set; }
    public object Id { get; set; }
    public Color Color { get; set; }
    public string DisplayPicture { get; set; }
}

```

## NOTE

You can inherit this class from `INotifyPropertyChanged` for dynamic changes in custom data.

You can map the properties of `Employee` class with our `SfSchedule` control using `ResourceMapping`.

## XML

```

<schedule:SfSchedule ScheduleView="WeekView" ShowResourceView="True">
  <schedule:SfSchedule.ResourceMapping>
    <schedule:ResourceMapping Name="Name"
      Id="Id"
      Color="Color"
      Image="DisplayPicture"/>
  </schedule:SfSchedule.ResourceMapping>
</schedule:SfSchedule>

```

## C#

```

// Creating an instance for resource mapping.
ResourceMapping resourceMapping = new ResourceMapping();
// Mapping the custom data fields.
resourceMapping.Name = "Name";
resourceMapping.Id = "Id";
resourceMapping.Color = "Color";
resourceMapping.Image = "DisplayPicture";
schedule.ResourceMapping = resourceMapping;

```

You can create a resource by setting `Id`, `Name`, `Color` and `DisplayPicture` of the `Employee` class.

Create resources of type `ObservableCollection<Employee>` and assign this resource collection to the `ScheduleResources` property of `SfSchedule`.

## C#

```

public ObservableCollection<object> Employees { get; set; }
// Creating an instance for collection of custom resources.
Employees = new ObservableCollection<object>();
// Creating an instance for custom appointment class.
Employee employee = new Employee();
employee.Name = "Kinsley Elena";
employee.Id = 5601;
employee.Color = Color.FromHex("#FFE671B8");
employee.DisplayPicture = "KinsleyElena.png";
// Adding a custom resource in custom resource collection.

```

```
Employees.Add(employee);  
// Adding a custom resource collection to schedule resources.  
schedule.ScheduleResources = Employees;
```

### Selection mode

The SfSchedule control provides support to select single or multiple resources using the [SelectionMode](#) property of [ResourceViewSettings](#) in SfSchedule.

### XML

```
<schedule:SfSchedule ScheduleView="WeekView" ShowResourceView="True">  
  <schedule:SfSchedule.ResourceViewSettings>  
    <schedule:ResourceViewSettings SelectionMode="Multiple"/>  
  </schedule:SfSchedule.ResourceViewSettings>  
</schedule:SfSchedule>
```

### C#

```
schedule.ScheduleView = ScheduleView.WeekView;  
schedule.ShowResourceView = true;  
ResourceViewSettings resourceViewSettings = new ResourceViewSettings();  
resourceViewSettings.SelectionMode = SelectionMode.Multiple;  
schedule.ResourceViewSettings = resourceViewSettings;
```



### Programmatic resource selection

You can programmatically select single or multiple resources by adding resources to the [SelectedResources](#) property of `SfSchedule`.

### C#

```
// Creating an instance for collection of selected resources.
ObservableCollection<object> selectedResources = new
ObservableCollection<object>();
// Adding selected resource in resource collection from the resources.
selectedResources.Add(resources.FirstOrDefault(resource => (resource as
ScheduleResource).Id.ToString() == "5601"));
selectedResources.Add(resources.FirstOrDefault(resource => (resource as
ScheduleResource).Id.ToString() == "5604"));
```

```
selectedResources.Add(resources.FirstOrDefault(resource => (resource as  
ScheduleResource).Id.ToString() == "5608"));  
// Adding selected resource collection to the selected resources of  
SfSchedule.  
schedule.SelectedResources = selectedResources;
```

You can clear the selection by removing the resource from `SelectedResources` or by setting `SelectedResources` to null.

### C#

```
var selectedResource = selectedResources.FirstOrDefault(resource =>  
(resource as ScheduleResource).Id.ToString() == "5604");  
// Removing selected resource in selected resources of SfSchedule.  
schedule.SelectedResources.Remove(selectedResource);
```

### Changing resource view height

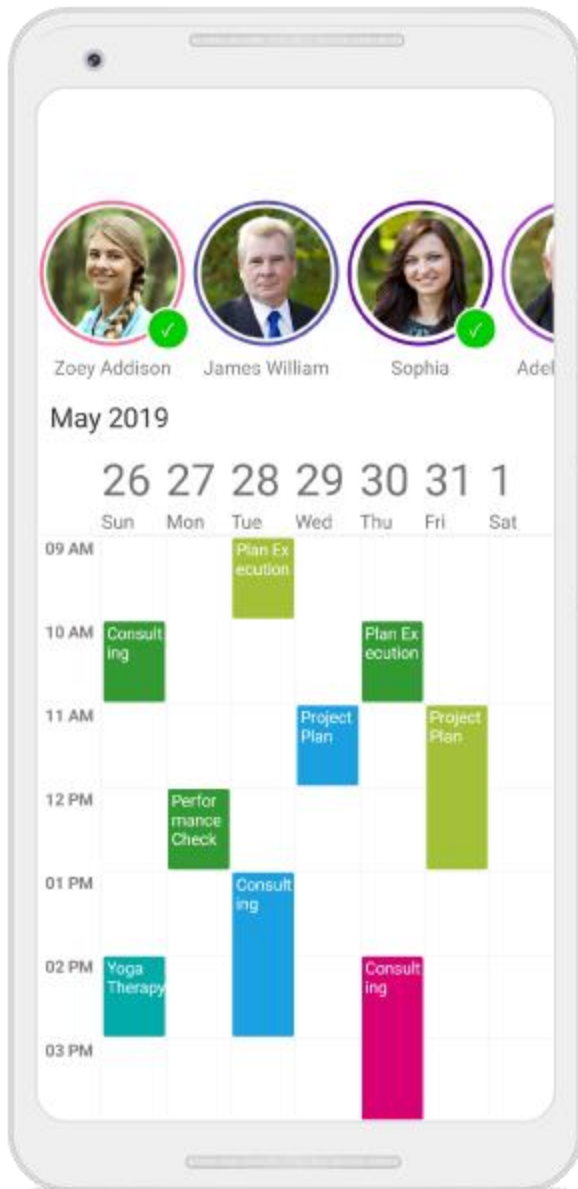
You can customize the height of the resource view using the [ResourceViewHeight](#) property of `SfSchedule`.

### XML

```
<schedule:SfSchedule ScheduleView="WeekView"  
ShowResourceView="True"  
ResourceViewHeight="200" />
```

### C#

```
schedule.ScheduleView = ScheduleView.WeekView;  
schedule.ShowResourceView = true;  
schedule.ResourceViewHeight = 200;
```



### Visible resource count

You can customize the number of visible resources in the current view using the [VisibleResourceCount](#) property of `ResourceViewSettings` in `SfSchedule`. By default, value of this property is set to -1.

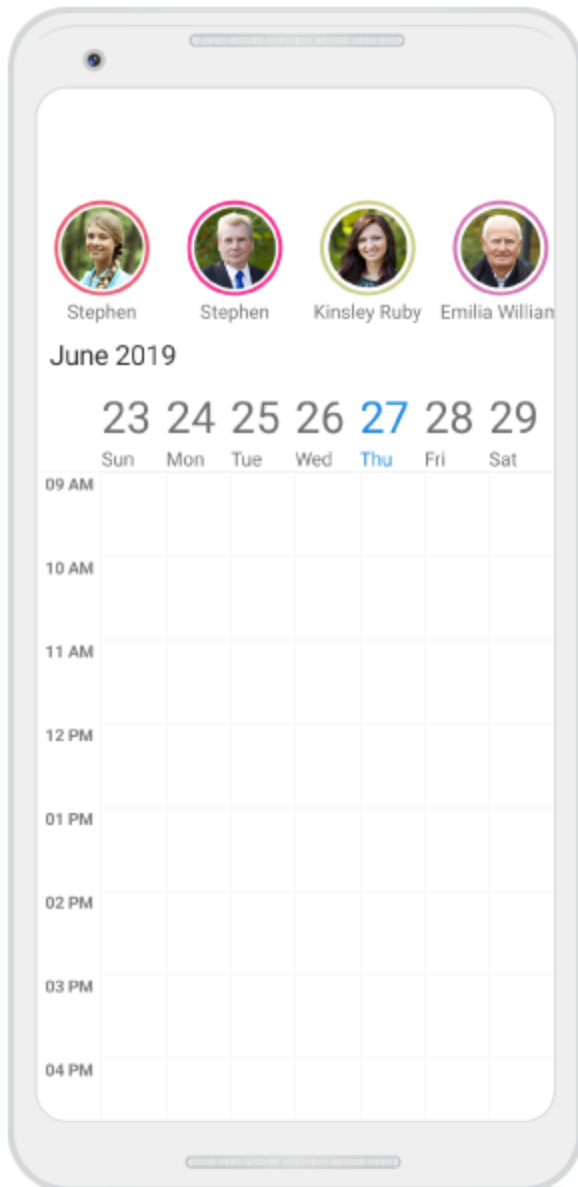
### XML

```
<schedule:SfSchedule ScheduleView="WeekView" ShowResourceView="True">
  <schedule:SfSchedule.ResourceViewSettings>
    <schedule:ResourceViewSettings>
      <schedule:ResourceViewSettings.VisibleResourceCount>
        <OnPlatform x:TypeArguments="x:Int32"
          iOS="5"
          Android="4"
          WinPhone="10" />
      </schedule:ResourceViewSettings.VisibleResourceCount>
    </schedule:ResourceViewSettings>
  </schedule:SfSchedule.ResourceViewSettings>
</schedule:SfSchedule>
```

```
</schedule:ResourceViewSettings>  
</schedule:SfSchedule.ResourceViewSettings>  
</schedule:SfSchedule>
```

## C#

```
ResourceViewSettings resourceViewSettings = new ResourceViewSettings();  
resourceViewSettings.VisibleResourceCount = Device.OnPlatform(5, 10, 5);  
schedule.ResourceViewSettings = resourceViewSettings;
```



### Resource item tapped event

You can handle the single tap action of resource view using the [ResourceItemTapped](#) event of [SfSchedule](#). This event occurs when a resource item is tapped. This event contains



[ResourceItemTappedEventArgs](#) argument, which holds the details of [SelectedResource](#) and [SelectedResources](#) in it.

### C#

```
schedule.ResourceItemTapped += OnResourceItemTapped;
...
private void OnResourceItemTapped(object sender, ResourceItemTappedEventArgs
e)
{
}
```

## Customization

### *Changing resource name label text color*

You can customize the text color of the resource name using the [DisplayLabelTextColor](#) property of [ResourceViewSettings](#) in [SfSchedule](#).

### XML

```
<schedule:SfSchedule ScheduleView="WeekView" ShowResourceView="True">
<schedule:SfSchedule.ResourceViewSettings>
<schedule:ResourceViewSettings DisplayLabelTextColor="#8490f9" />
</schedule:SfSchedule.ResourceViewSettings>
</schedule:SfSchedule>
```

### C#

```
ResourceViewSettings resourceViewSettings = new ResourceViewSettings();
resourceViewSettings.DisplayLabelTextColor = Color.FromHex("#8490f9");
schedule.ResourceViewSettings = resourceViewSettings;
```

### *Data template*

The default appearance of the resource can be customized using the [ResourceItemTemplate](#) property of the schedule. You can handle the default touch action such as selection of resource item using the [InputTransparent](#) property of the [Xamarin.Forms.VisualElement](#).

### XML

```
<schedule:SfSchedule x:Name="schedule"
ResourceItemTemplate="{Binding ResourceTemplate}">
<schedule:SfSchedule.BindingContext>
<samplelocal:ResourceDataTemplate />
</schedule:SfSchedule.BindingContext>
</schedule:SfSchedule>
```

### *Creating a DataTemplate*

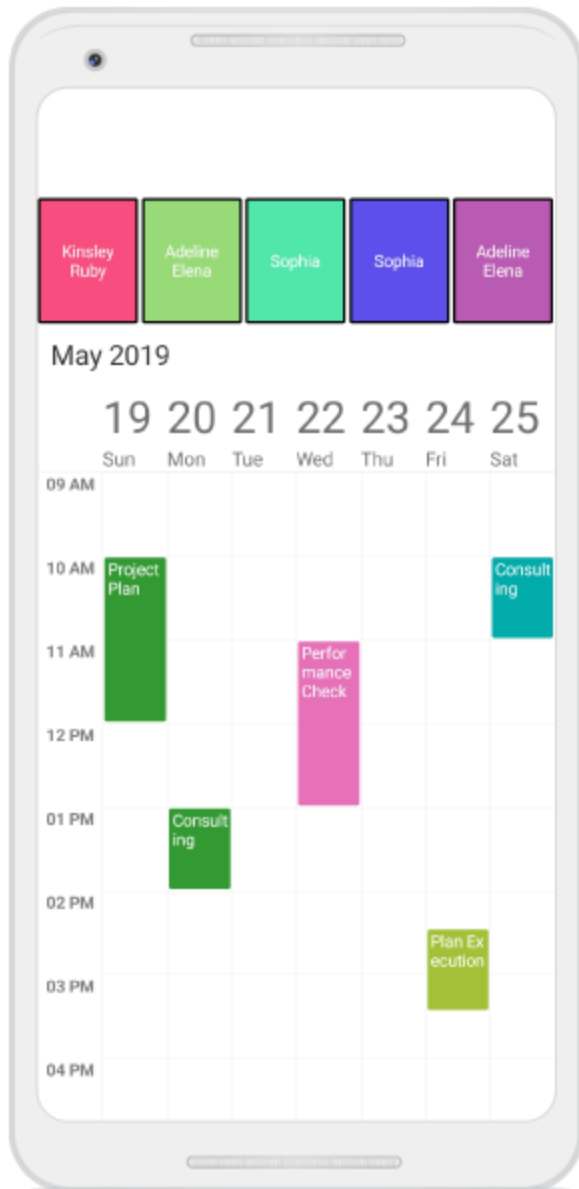
### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Button xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="ScheduleSample.ResourceTemplate"
Text="{Binding Name}"
TextColor="White"
```

```
FontSize="15"  
BackgroundColor="{Binding Color}"  
BorderColor="Black"  
BorderWidth="2">  
</Button>
```

## **C#**

```
public class ResourceDataTemplate : DataTemplate  
{  
    public DataTemplate ResourceTemplate { get; set; }  
    public ResourceDataTemplate()  
    {  
        ResourceTemplate = new DataTemplate(() =>  
        {  
            return new ResourceTemplate();  
        });  
    }  
}
```



### Template selector

`DataTemplateSelector` can be used to choose a `DataTemplate` at run time based on the value of data bound to the `ScheduleResource` property through `ResourceItemTemplate`. It provides multiple `DataTemplates` to be enabled for schedule resources to customize the appearance of a particular resource item. You can handle the default touch action such as selection of a resource item using the `InputTransparent` property of the `Xamarin.Forms.VisualElement`.

### XML

```
<ContentPage.Resources>
  <ResourceDictionary>
    <local:ResourceTemplateSelector x:Key="resourceDataTemplateSelector"/>
  </ResourceDictionary>
</ContentPage.Resources>
```

```
<ContentPage.Content>
<schedule:SfSchedule x:Name="schedule"
ScheduleView="WeekView"
ShowResourceView="True"
ResourceItemTemplate="{StaticResource resourceDataTemplateSelector}"/>
</ContentPage.Content>
```

### Creating a DataTemplateSelector

#### C#

```
public class ResourceTemplateSelector : DataTemplateSelector
{
    public DataTemplate AvailabelResourceTemplate { get; set; }
    public DataTemplate UnavailableResourceTemplate { get; set; }
    public ResourceTemplateSelector()
    {
        AvailabelResourceTemplate = new
        DataTemplate(typeof(AvailabelResourceTemplate));
        UnavailableResourceTemplate = new
        DataTemplate(typeof(UnavailableResourceTemplate));
    }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject
    container)
    {
        if ((item as ScheduleResource).Id.ToString() == "5601" ||
        (item as ScheduleResource).Id.ToString() == "5604")
        return UnavailableResourceTemplate;
        else
        return AvailabelResourceTemplate;
    }
}
```

Used button to display the resources

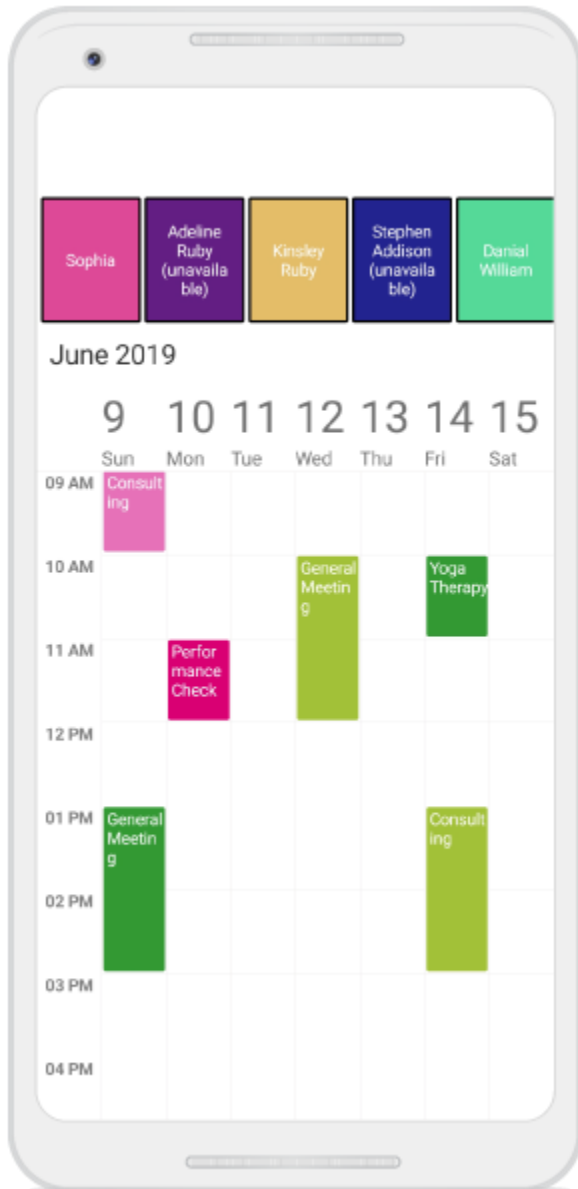
#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Button xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="ScheduleSample.AvailabelResourceTemplate"
Text="{Binding Name}"
TextColor="White"
FontSize="15"
BackgroundColor="{Binding Color}"
BorderColor="Black"
BorderWidth="2">
</Button>
.....
<?xml version="1.0" encoding="utf-8" ?>
<Button xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="ScheduleSample.UnavailableResource"
Text="{Binding Name, StringFormat= '\{0\} (unavailable)'}"
TextColor="White"
FontSize="15"
BackgroundColor="{Binding Color}"
```

```

BorderColor="Black"
BorderWidth="2">
</Button>

```



You can download the template selector demo for Xamarin.Forms from [here](#).

### Appointment Drag and Drop

Appointments can be rescheduled using the drag and drop operation. To perform drag-and-drop operations within the schedule, enable the [AllowAppointmentDrag](#) property of `SfSchedule`.

#### XML

```

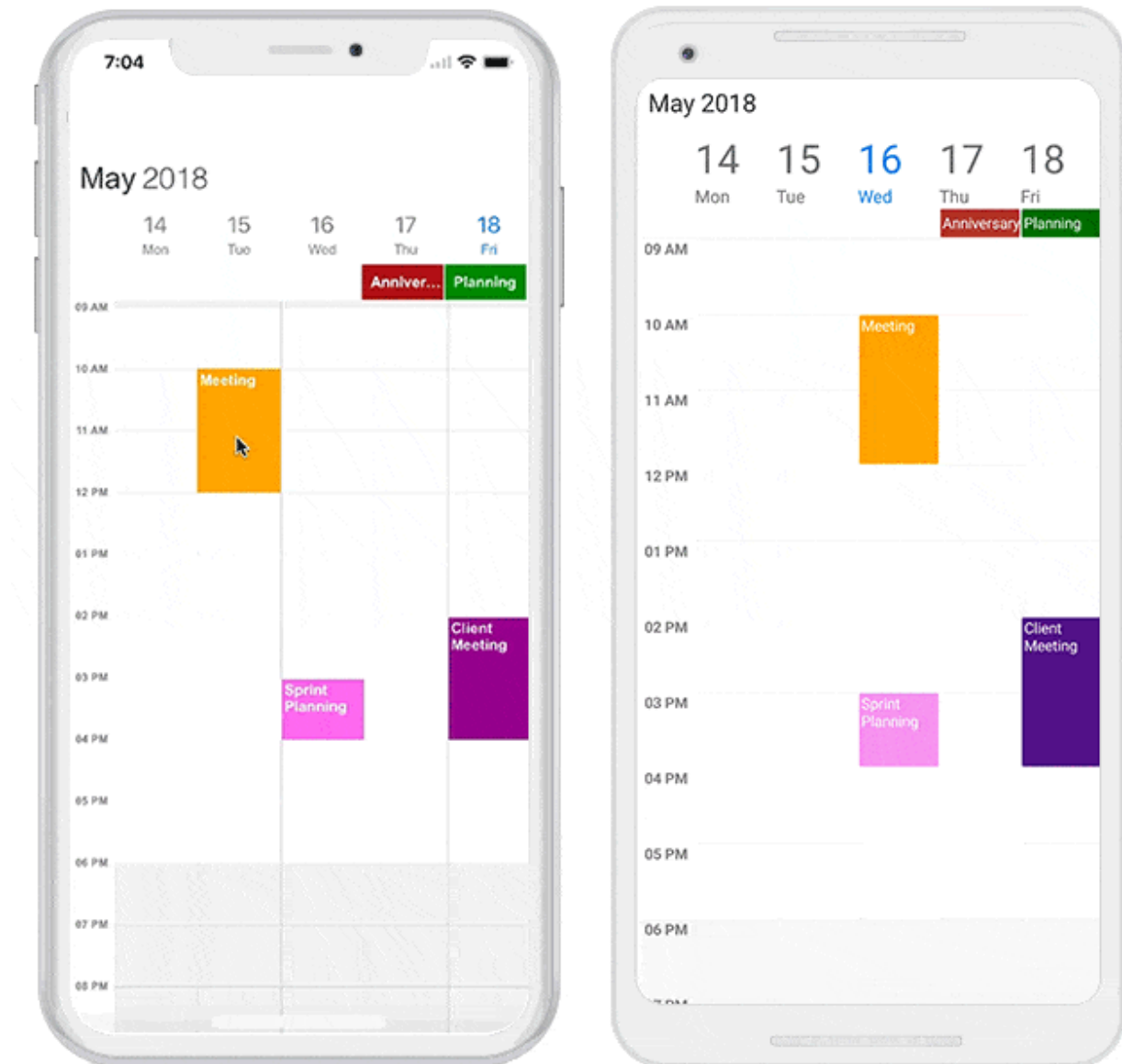
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView"
AllowAppointmentDrag="true">

```

```
</schedule:SfSchedule>
```

### C#

```
schedule.AllowAppointmentDrag = true;
```



### Handle dragging based on the appointment

Using [AppointmentDragStarting](#) event, you can get the appointment details and handle whether the appointment can be draggable or not. This event will be triggered when the appointment is started dragging. The [AppointmentDragStartingEventArgs](#) argument contains the following properties.

[Appointment](#) - Gets the dragged appointment details.

[Cancel](#)- Appointment dragging can be handled (enable/disable) using this boolean property.

**C#**

```
schedule.AppointmentDragStarting += Schedule_AppointmentDragStarting;
...
private void Schedule_AppointmentDragStarting(object sender,
AppointmentDragStartingEventArgs e)
{
    var appointment = e.Appointment;
    e.Cancel = false;
}
```

*Disabling dragging when the appointment is AllDay appointment*

Using Cancel property in the AppointmentDragStartingEventArgs argument of Schedule AppointmentDragStarting event, you can enable/disable the appointment dragging based on the requirement. In the below code, appointment dragging is disabled when the appointment is AllDay appointment.

**C#**

```
schedule.AppointmentDragStarting += Schedule_AppointmentDragStarting;
...
private void Schedule_AppointmentDragStarting(object sender,
AppointmentDragStartingEventArgs e)
{
    var appointment = e.Appointment as ScheduleAppointment;
    if (appointment.IsAllDay)
    {
        e.Cancel = true;
    }
}
```

*Get the dragging appointment position*

Using [AppointmentDragOver](#) event, you can get the dragging appointment details, position and time of the particular location. The event will be continuously triggered when the appointment is being dragged. The [AppointmentDragEventArgs](#) argument contains the following properties.

[Appointment](#) - Gets the details of the appointment to be dropped.

[DraggingPoint](#)- Gets the dragging point (X, Y) of the appointment in Schedule.

[DraggingTime](#)- Gets the dragging time of the appointment in Schedule

**C#**

```
schedule.AppointmentDragOver += Schedule_AppointmentDragOver;
...
private void Schedule_AppointmentDragOver(object sender,
AppointmentDragEventArgs e)
{
    var appointment = e.Appointment;
    var draggingPoint = e.DraggingPoint;
    var draggingTime = e.DraggingTime;
}
```

### *Displaying alert while dragging appointment over the blocked time slots*

Using `draggingPoint` and `draggingTime` properties in the `AppointmentDragEventArgs` of `Schedule AppointmentDragOver` event you can get the current position and time of dragging appointment. In the below code, Indicating the message while dragging over the `Schedule NonAccessibleBlock`.

#### **C#**

```
schedule.AppointmentDragOver += Schedule_AppointmentDragOver;
...
private void Schedule_AppointmentDragOver(object sender,
AppointmentDragEventArgs e)
{
    //// checking whether dragging appointment time within NonAccessibleBlock
    if (schedule.WorkWeekViewSettings.NonAccessibleBlocks[0].StartTime ==
        e.DraggingTime.Hour ||
        (schedule.WorkWeekViewSettings.NonAccessibleBlocks[0].StartTime - 1 ==
         e.DraggingTime.Hour && e.DraggingTime.Minute > 0))
    {
        label.Text = "Cannot be moved to blocked time slots";
    }
}
```

### Handle appointment dropping

Using `AppointmentDrop` event you can get the dropping appointment details, position, time and you can handle whether the appointment can be dropped to the specific position or not. This event will trigger after dropping the appointment. The `AppointmentDropEventArgs` argument contains the following properties.

[Appointment](#) - Gets the details of the appointment to be dropped.

[Cancel](#)- Appointment dropping can be handled (enable / disable) using this Boolean property.

[DropTime](#)- Gets the dropped time of the appointment in Schedule

#### **C#**

```
schedule.AppointmentDrop += Schedule_AppointmentDrop;
...
private void Schedule_AppointmentDrop(object sender,
AppointmentDropEventArgs e)
{
    var appointment = e.Appointment;
    e.Cancel = false;
    var dropTime = e.DropTime;
}
```

### *Disabling dropping when dropping appointment within the Non-Accessible region*

Using `Cancel` property in the `AppointmentDropEventArgs` argument of `Schedule AppointmentDrop` event, you can enable/disable the appointment dropping based on the requirement. In the below code, appointment dropping is disabled while dropping in the Non-Accessible block region.

#### **C#**

```
schedule.AppointmentDrop += Schedule_AppointmentDrop;
...
```



```
private void Schedule_AppointmentDrop(object sender,
AppointmentDropEventArgs e)
{
    //// checking whether dropping appointment time within NonAccessibleBlock
    if (schedule.WorkWeekViewSettings.NonAccessibleBlocks[0].StartTime ==
e.DropTime.Hour ||
(schedule.WorkWeekViewSettings.NonAccessibleBlocks[0].StartTime - 1 ==
e.DropTime.Hour && e.DropTime.Minute > 0))
    {
        e.Cancel = true;
    }
}
```

### Customizing the Drag and Drop environment

Using [DragDropSettings](#) property of schedule, you can handle the behavior of drag and drop in Schedule.

#### XML

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView">
<schedule:SfSchedule.DragDropSettings>
<schedule:DragDropSettings AllowNavigate="true" AllowScroll="true"
ShowTimeIndicator="true">
</schedule:DragDropSettings>
</schedule:SfSchedule.DragDropSettings>
</schedule:SfSchedule>
```

#### C#

```
DragDropSettings dragDropSettings = new DragDropSettings();
dragDropSettings.AllowNavigate = true;
dragDropSettings.AllowScroll = true;
var timeSpan = new TimeSpan(0, 0, 0, 1, 0);
dragDropSettings.AutoNavigationDelay = timeSpan;
dragDropSettings.ShowTimeIndicator = true;
dragDropSettings.TimeIndicatorStyle = timeIndicatorStyle;
schedule.DragDropSettings = dragDropSettings;
```

### Disabling navigation when dragging appointment

Using [AllowNavigate](#) boolean property can handle the Appointment dragging, whether navigate to next/previous view or not while dragging the appointment to the endpoint of the current view in Schedule. Default value of the [AllowNavigate](#) property is true and Schedule will navigate to next/previous view when dragging the appointment the endpoint of the current view.

#### C#

```
dragDropSettings.AllowNavigate = false;
```

### Handling navigation delay while holding dragged appointment

Using [AutoNavigationDelay](#) [TimeSpan](#) property can handle the navigation time when navigating to next/previous view while holding the dragged appointment.

#### C#

```
var timeSpan = new TimeSpan(0, 0, 0, 1, 0);
```

```
dragDropSettings.AutoNavigationDelay = TimeSpan;
```

#### *Disabling scroll when dragging appointment*

Using [AllowScroll](#) boolean property can handle the Appointment dragging, whether scroll (below/above) the Schedule or not while dragging the appointment to the endpoint of the current view in Schedule.

Default value of the [AllowScroll](#) property is true.

#### **C#**

```
dragDropSettings.AllowScroll = false;
```

#### *Disabling dragging time indicator*

[ShowTimeIndicator](#) - Using this boolean property can handle the time indicator whether it should visible or not, which shows the dragged appointment current position time in time text slots. Default value of the [ShowTimeIndicator](#) property is true.

#### **C#**

```
dragDropSettings.ShowTimeIndicator = false;
```

Customize appearance of dragging Time Indicator

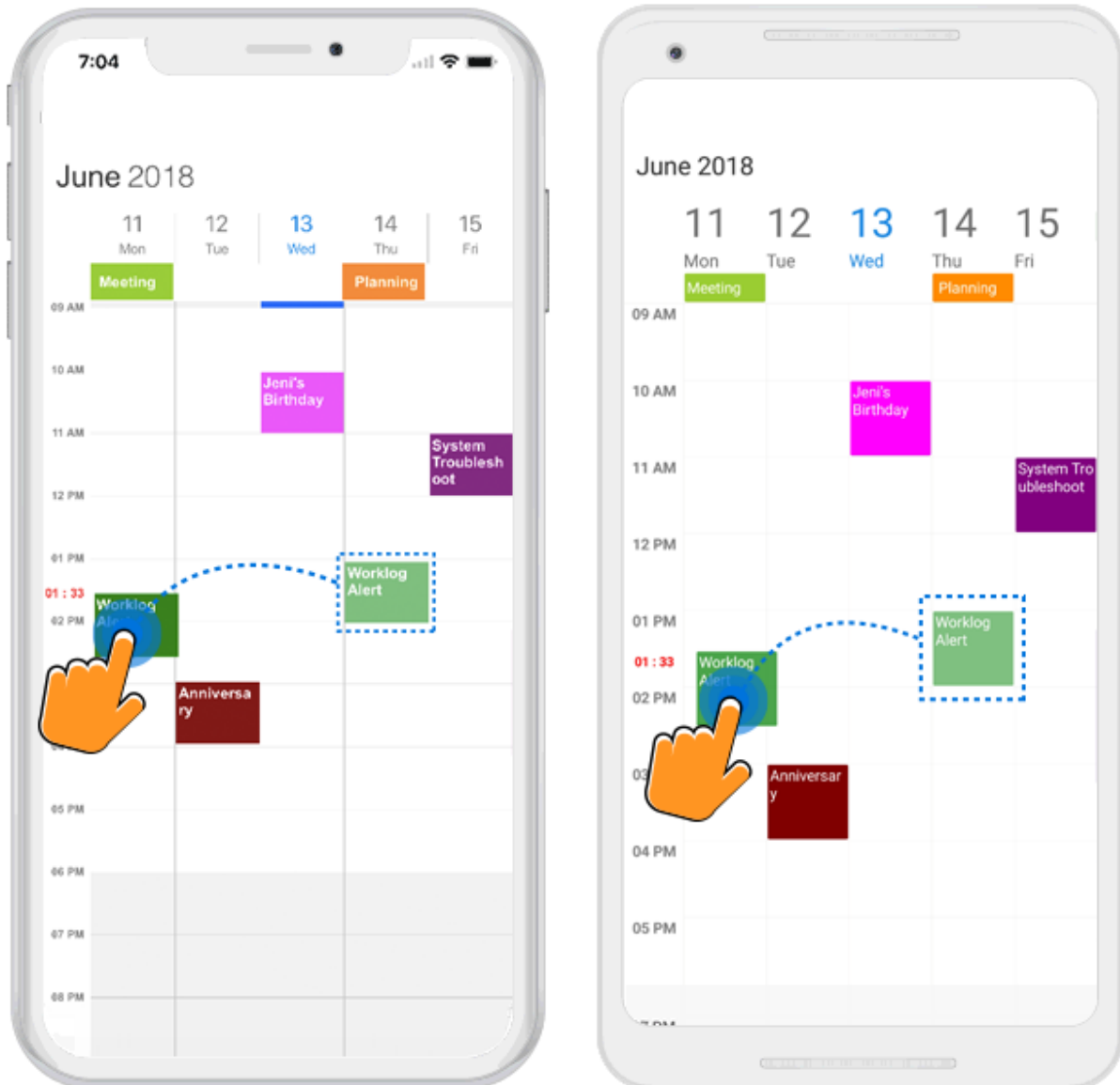
Using [TimeIndicatorStyle](#) property can handle the time indicator style which contains [TextColor](#), [TextSize](#) and [TextFormat](#).

#### **XML**

```
<schedule:SfSchedule x:Name="schedule" ScheduleView="WeekView">
  <schedule:SfSchedule.DragDropSettings>
    <schedule:DragDropSettings>
      <schedule:DragDropSettings.TimeIndicatorStyle>
        <schedule:TimeIndicatorStyle TextColor="Red" TextSize="13"
          TextFormat="hh:mm">
        </schedule:TimeIndicatorStyle>
      </schedule:DragDropSettings.TimeIndicatorStyle>
    </schedule:DragDropSettings>
  </schedule:SfSchedule.DragDropSettings>
</schedule:SfSchedule>
```

#### **C#**

```
TimeIndicatorStyle timeIndicatorStyle = new TimeIndicatorStyle();
timeIndicatorStyle.TextColor = Color.Red;
timeIndicatorStyle.TextSize = 15;
timeIndicatorStyle.TextFormat = "hh : mm";
dragDropSettings.TimeIndicatorStyle = timeIndicatorStyle;
```



## Notes

- While dropping appointment to **AllDay** panel from time slots, appointment start and end time will change to 12.00 AM.
- While dropping appointment to time slots from **AllDay** panel, appointment duration will change as one (1) hour from the dropped time.
- Doesn't support control to control drag and drop.

## Time Zone

Schedule allows you create appointments in various time zones and display them in users' time zone or any other time zone. You can use a 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 schedule time zone
- Display appointments at the same time everywhere regardless of client's time zone

We have added the following Time Zone's for the respective countries to cover all the time zone regions, you can use any of the time zone's from the following list for schedule 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

You can 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. You can set different time zones to the [StartTimeZone](#) and [EndTimeZone](#) properties.

You can 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, you can get exact appointment rendering time.

### C#

```
appointment.StartTimeZone = "India Standard Time";
appointment.EndTimeZone = "India Standard Time";
```

```
DateTime exactStartTime = appointment.StartTime;  
DateTime exactEndTime = appointment.EndTime;
```

## 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 you create an all-day appointment, 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.
- Schedule supports daylight saving time.
- The time zone support is applicable for custom appointments too, so you need to map the corresponding property.
- You can use `TimeZone` for custom appointments by mapping the [StartTimeZoneMapping](#) and [EndTimeZoneMapping](#) custom properties of [ScheduleAppointmentMapping](#).

### Display Appointments based on client's time zone

You can display the appointments based on the client's local time zone in schedule. 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 you each view your calendar, 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 schedule time zone

You can set specific time zone to schedule using the `TimeZone` property of schedule. 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 schedule time zone.

## XML

```
<syncfusion:SfSchedule  
  x:Name="schedule"  
  TimeZone="GMT Standard Time">  
</schedule:SfSchedule>
```

## C#

```
schedule.TimeZone = "GMT Standard Time";
```

### Display appointments at same time everywhere regardless of client's time zone

You can display appointments at the same time everywhere without considering the time zone when you set the `TimeZone` property of schedule, 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 schedule time zone and appointment's time zone.

For an example, consider, your local time zone is **India Standard Time**, if you drag an appointment from 9 AM and drop this on 1 PM and the schedule'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 schedule'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 schedule'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 schedule'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 schedule time zone to UTC time zone and the appointment's start time will be saved at 10.30 PM.

## Date Navigations

### Enabling Navigation

By default, Schedule views can be moved backwards and forwards using touch swipe gesture. This navigation gesture can be enabled or disabled by setting [EnableNavigation](#) property of **SfSchedule**. By default, it is enabled.

### XML

```
<schedule:SfSchedule EnableNavigation="False"/>
```

### C#

```
//disabling navigation gesture
schedule.EnableNavigation = false;
```

### Programmatically change to specific dates

Visible dates can be moved to specific date using [NavigateTo](#) method and [MoveToDate](#) property available in **SfSchedule**. It will move to any specific date if the schedule view is Day View, similarly it will move to the specific week if it is week view and to specific month if it is month view

### C#

```
//using NavigateTo
DateTime currentDate = DateTime.Now;
DateTime SpecificDate = new DateTime(currentDate.Year - 5, currentDate.Month - 3, currentDate.Day, 0, 0, 0);
Schedule.NavigateTo(SpecificDate);
//using MoveToDate
DateTime currentDate = DateTime.Now;
DateTime SpecificDate = new DateTime(currentDate.Year - 5, currentDate.Month - 3, currentDate.Day, 0, 0, 0);
Schedule.MoveToDate = SpecificDate;
```

---

**NOTE**

---

The specified date should lie between `MinDisplayDate` and `MaxDisplayDate`, if the specified date is greater than `MaxDisplayDate` then the view moved to `MaxDisplayDate` similarly if the specified date is lesser than the `MinDisplayDate` then the view moved to `MinDisplayDate`.

[Programmatically change to adjacent dates.](#)

By default the date can be navigated to next and previous view using touch gesture, by swiping the control in right to left and left to right direction. The view can be also changed programmatically using [Forward](#) and [Backward](#) method available in `SfSchedule`.

- Forward \* Backward

#### [Forward](#)

You can use the `Forward` method for viewing the next immediate visible dates in the `SfSchedule`. It will move to next month if the schedule view is month, similarly it will move to next week for week view and next day for day view.

#### **C#**

```
//Viewing next immediate visible dates  
schedule.Forward();
```

---

**NOTE**

---

Date can be navigated until it reaches the Min Max date.

#### [Backward](#)

You can use the `Backward` method for viewing the previous immediate visible dates in the `SfSchedule`. It will move to previous month if the schedule view is month, similarly it will move to previous week for week view and previous day for day view.

#### **C#**

```
//Viewing previous immediate visible dates  
schedule.Backward();
```

---

**NOTE**

---

Date can be navigated until it reaches the Min Max date.

#### [Range for visible dates](#)

Visible dates can be restricted between certain range of dates, using [MinDisplayDate](#) and [MaxDisplayDate](#) property in `SfSchedule`. It is applicable in all the schedule views.

#### [Minimum Display Date](#)

`MinDisplayDate` will restrict date navigations features of `Backward`, `MoveToDate` and also can't swipe the control using touch gesture beyond the min max date range. Thus, Inline and selection feature in month view will work only within the min max date range.

#### **C#**

```
int monthRange = 2; DateTime currentDate = DateTime.Now;  
//setting minimum display date
```

```
DateTime minDate = new DateTime(currentDate.Year, currentDate.Month -
monthRange, currentDate.Day, 10, 0, 0);
schedule.MinDisplayDate = minDate;
```

### Maximum Display Date

**MaxDisplayDate** will restrict date navigations features of **Forward**, **MoveToDate** and also can't swipe the control using touch gesture beyond the max date range. Thus, Inline and selection in month view will work only within the max date range.

### C#

```
int monthRange = 2; DateTime currentDate = DateTime.Now;
//setting maximum display date
DateTime maxDate = new DateTime(currentDate.Year, currentDate.Month +
monthRange, currentDate.Day, 10, 0, 0);
schedule.MaxDisplayDate = maxDate;
```

### VisibleDatesChanged event

You can get the visible dates of the Schedule using [VisibleDatesChangedEvent](#) in **SfSchedule**. It is applicable in all the schedule views. The event handler receives an argument of type [VisibleDatesChangedEventArgs](#) containing [visibleDates](#) and [Schedule](#).

### C#

```
schedule.VisibleDatesChangedEvent += Schedule_VisibleDatesChangedEvent;
.....
private void Schedule_VisibleDatesChangedEvent(object sender,
VisibleDatesChangedEventArgs e)
{
var visibleDates = e.visibleDates;
var Schedule = e.Schedule;
}
```

**VisibleDatesChangedEvent** will be triggered when view is swiped back or forth and also when schedule view is switched dynamically.

You can add appointments on demand based on the visible date range in this event by setting **DataSource** property to schedule in the **VisibleDatesChangedEvent**.

### C#

```
private void Schedule_VisibleDatesChangedEvent(object sender,
VisibleDatesChangedEventArgs e)
{
schedule.DataSource = scheduleAppointmentCollection;
}
```

You can also move to specific time of the day or current time of day when view is swiped by setting specific time in **MoveToDate** property in the **VisibleDatesChanged** event. Such that when the schedule view is swiped, it moves to the mentioned time.

### C#

```
schedule.VisibleDatesChangedEvent += Schedule_VisibleDatesChangedEvent;
```

```
...
private void Schedule_VisibleDatesChangedEvent(object sender,
VisibleDatesChangedEventArgs e)
{
    var date = DateTime.Now;
    var formatDate = e.visibleDates[0];
    schedule.MoveToDate = new DateTime(formatDate.Year, formatDate.Month,
formatDate.Day, date.Hour, date.Minute, date.Second);
}
```

## Localization

Schedule control is available with complete localization support. Localization can be specified by setting the [Locale](#) property of SfSchedule. In the format of **Language code**.

### Change default control language

Based on the **Locale** specified the strings in the control such as Date, time, days are localized accordingly.

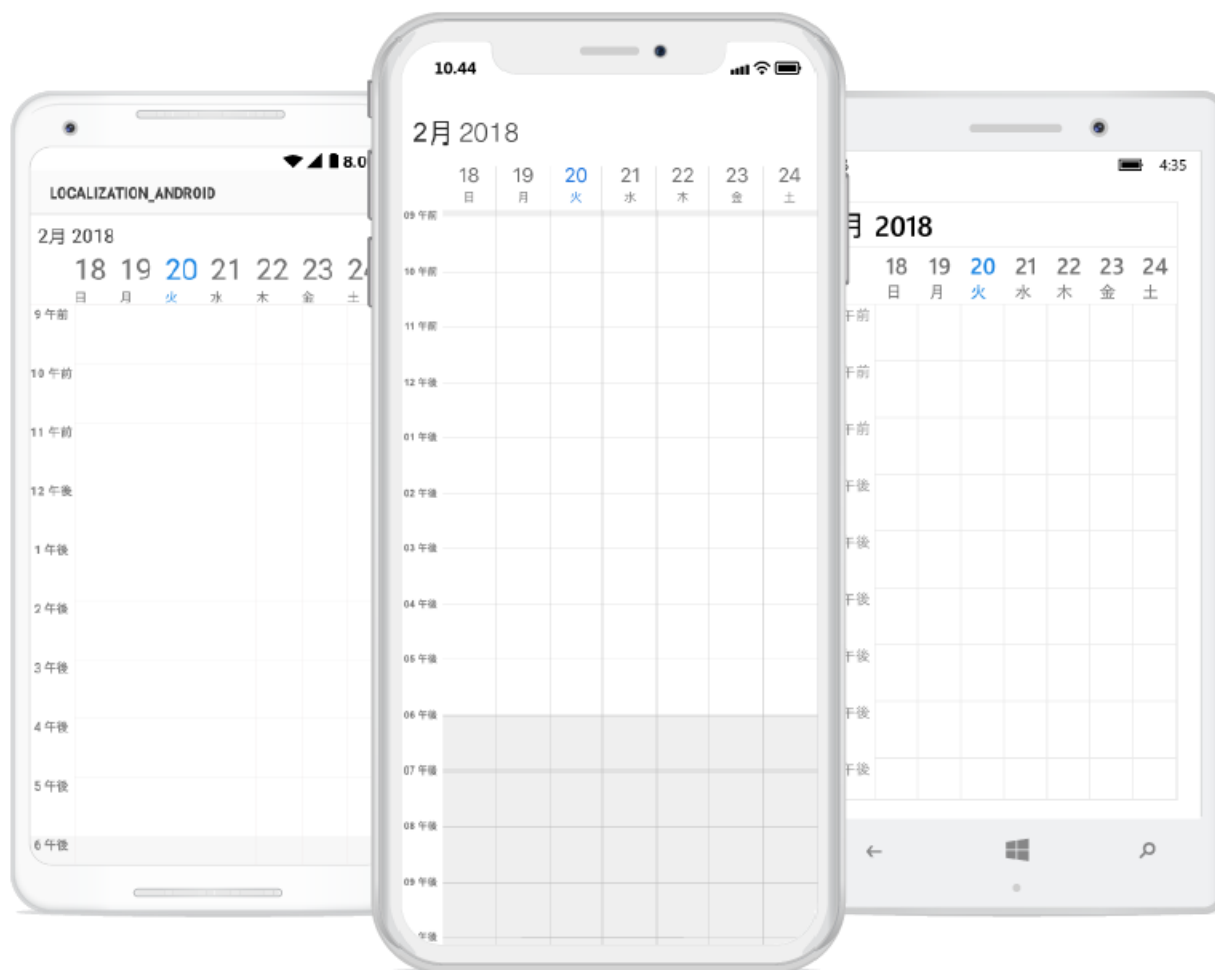
By default, schedule control is available with en locale, which is English.

### XML

```
<schedule:SfSchedule
x:Name="schedule"
ScheduleView="WeekView"
Locale="ja">
</schedule:SfSchedule>
```

### C#

```
//creating new instance for schedule
SfSchedule schedule = new SfSchedule();
//setting schedule view
schedule.ScheduleView = ScheduleView.WeekView;
//setting locale for the control
schedule.Locale = "ja";
```



Change custom texts in the control.

You can localize the custom strings used in the schedule control. For that you need to configure it for each platform separately.

*Localizing custom text in Android renderer.* Localizing custom text in iOS renderer. \* Localizing custom text in UWP renderer.

You can download the entire source code of this demo for Xamarin.Forms from here [Localization](#).

*Localizing custom text in Android renderer.*

You can localize custom text available in the control by adding equivalent localized string in the string.xml file.

### XML

```
<resources>
<string name="No_Appointments">Aucun événement</string>
<string name="No_SelectedDate">Aucune date sélectionnée</string>
<string name="No_Resources">Aucune ressource</string>
</resources>
```

Android can select and load resources from different directories, based on the current device configuration and locale, refer [here](#). For an example, if an application requires multiple languages you can follow the below steps.

The procedure for creating strings.xml files is as follows:

*Translate the strings.xml file to each language.* Create new folders under resource as values-ar, values-de, values-en and values-fr (The original values folder already exists). \* Place the translated strings.xml files in the respective folders.



---

**NOTE**

The corresponding Locale values folder updates only when the device language changes.

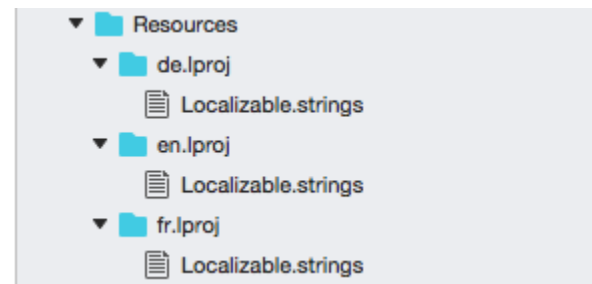
[Localizing custom text in iOS renderer.](#)

You can localize custom text available in the control by adding equivalent localized string in the Localizable.strings file, refer [here](#).



If an application requires multiple languages you can follow the below steps:

*Translate the Localizable.Strings file to each language.* Create new .lproj folders under resource as en.lproj, fr.lproj, de.lproj. \* Place the Localizable.Strings file in the respective .lproj folders.



---

**NOTE**

The corresponding <Language>.lproj folder updates only when the device language changes.

[Localizing custom text in UWP renderer.](#)

You can localize custom text available in the control by adding equivalent localized string in the Resources(.resw) file.

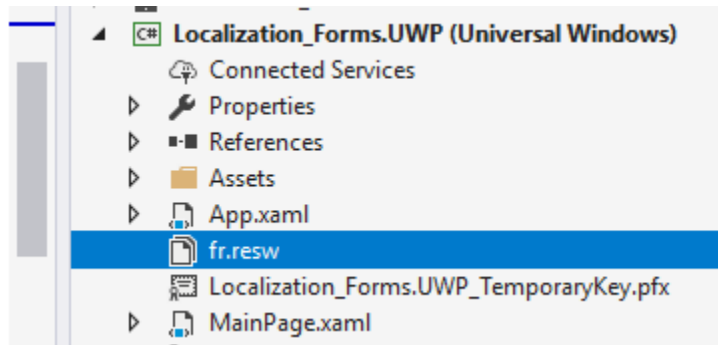
---

**NOTE**

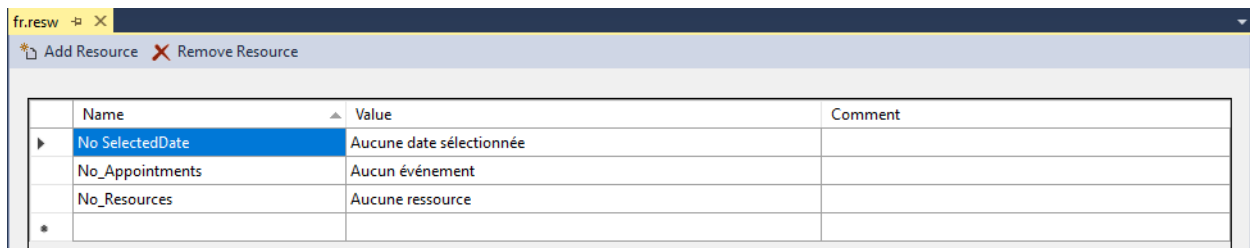
Here Resources(.resw) file name should be match with the given locale language code.

The procedure for creating Resources(.resw) file is as follows:

\* Create Resources(.resw) file in sample with C# culture standard name for example fr , de-DE and so.



\* Translate the custom string used in schedule to respective localized culture.

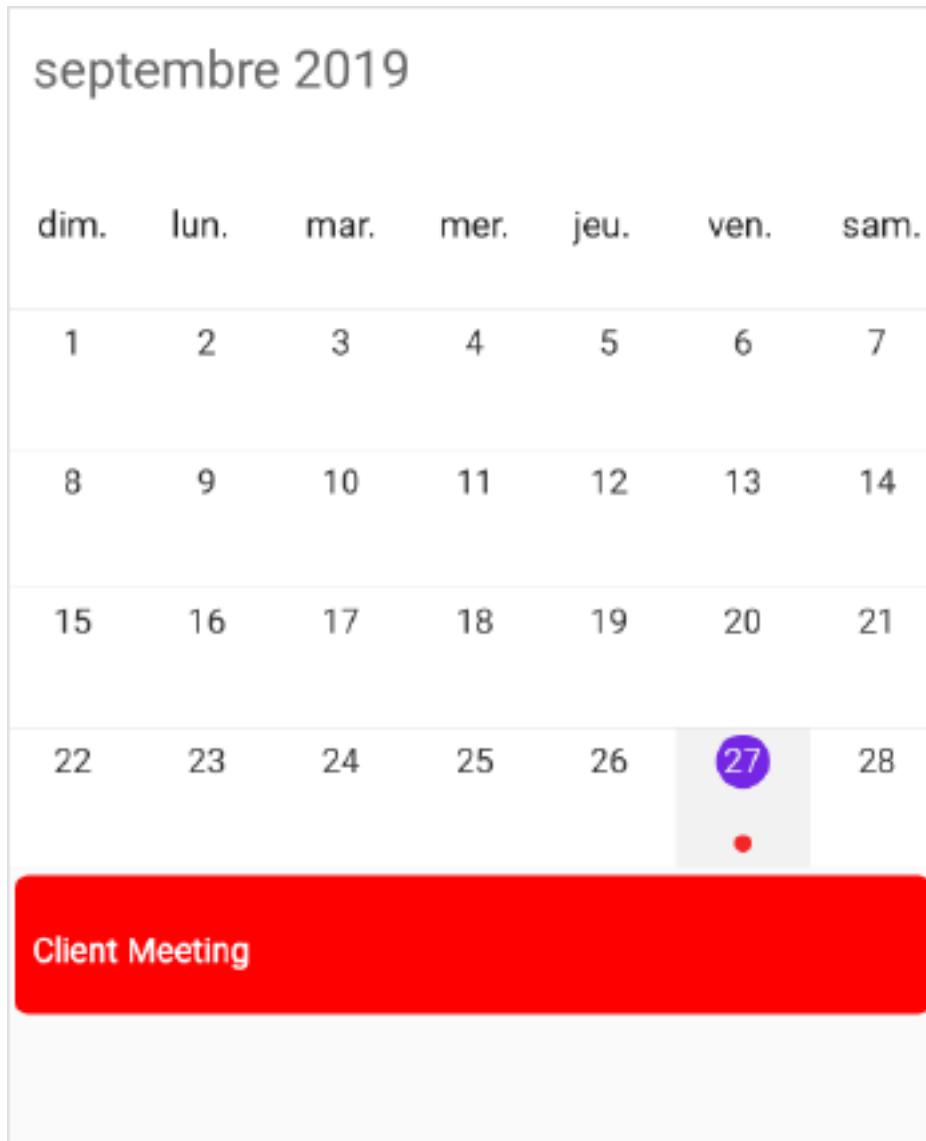


### Localizing custom strings from PCL

You can localize the custom strings (No SelectedDate, No Events, and No Resources) used in the schedule control from PCL. It can be achieved by providing the custom strings to the specific language resx file and handling the required culture with the locale using DependencyService instead of device language. Create a new resource manager based on the resources, and set it to the [Manager](#) property of [ScheduleResourceManager](#). In the following code, French has been set as Schedule locale as well as custom strings.

### C#

```
// Here ScheduleSample is a sample name.
ScheduleResourceManager.Manager = new
ResourceManager("ScheduleSample.Resources.Syncfusion.SfSchedule.Forms",
GetType().GetTypeInfo().Assembly);
if (schedule.Locale == "fr")
{
    if (Device.RuntimePlatform == Device.iOS || Device.RuntimePlatform ==
Device.Android)
    {
        DependencyService.Get<ILocalize>().SetLocale(new CultureInfo("fr"));
    }
    else
    {
        CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("fr");
    }
}
```



You can download the entire source code of this demo for Xamarin.Forms from here [LocaleFromPCL](#).

#### *Adding resx file*

You need to add the required resx files under the Resources folder in the PCL project and the filename should be `Syncfusion.SfSchedule.Forms.LanguageCode.resx`.

Example: For French, `Syncfusion.SfSchedule.Forms.fr.resx`

Now, set the Build Action as `EmbeddedResource` for `Syncfusion.SfSchedule.Forms.fr.resx` file and Build Action as `Compile` for `Syncfusion.SfSchedule.Forms.fr.Designer.cs` file.

#### **XML**

```
<data name="NoEvents" xml:space="preserve">
<value>Pas d'événements</value>
</data>
<data name="NoSelectedDate" xml:space="preserve">
```



```
<value>Aucune date sélectionnée</value>
</data>
<data name="NoResources" xml:space="preserve">
<value>Aucune ressource</value>
</data>
```

#### *Adding ILocalize interface in PCL*

You need to add the ILocalize interface to convert the platform-specific locales to a value supported in .NET cultures in the PCL project.

#### **C#**

```
namespace ScheduleLocale
{
    public interface ILocalize
    {
        CultureInfo GetCurrentCultureInfo();
        void SetLocale(CultureInfo cultureInfo);
    }
    public class PlatformCulture
    {
        public PlatformCulture(string platformCultureString)
        {
            if (String.IsNullOrEmpty(platformCultureString))
                throw new ArgumentException("Expected culture identifier",
                    "platformCultureString");
            PlatformString = platformCultureString.Replace("_", "-");
            var dashIndex = PlatformString.IndexOf("-", StringComparison.Ordinal);
            if (dashIndex > 0)
            {
                var parts = PlatformString.Split('-');
                LanguageCode = parts[0];
                LocaleCode = parts[1];
            }
            else
            {
                LanguageCode = PlatformString;
                LocaleCode = "";
            }
        }
        public string PlatformString { get; private set; }
        public string LanguageCode { get; private set; }
        public string LocaleCode { get; private set; }
        public override string ToString()
        {
            return PlatformString;
        }
    }
}
```

#### *Adding Localize class in Android and iOS project inheriting from ILocalize*

You need to add the Localize class in Android and iOS project by inheriting from ILocalize. You can get the CultureInfo and set the same to Schedule Locale by using `GetCurrentCultureInfo` and `SetLocale`.

Localize class for Android project,

**C#**

```

using System.Globalization;
using System.Threading;
[assembly: Xamarin.Forms.Dependency(typeof(ScheduleLocale.Droid.Localize))]
namespace ScheduleLocale.Droid
{
    public class Localize : ILocalize
    {
        public void SetLocale(CultureInfo cultureInfo)
        {
            Thread.CurrentThread.CurrentCulture = cultureInfo;
            Thread.CurrentThread.CurrentUICulture = cultureInfo;
        }
        public CultureInfo GetCurrentCultureInfo()
        {
            var netLanguage = "en";
            var androidLocale = Java.Util.Locale.Default;
            netLanguage = AndroidToDotnetLanguage(androidLocale.ToString().Replace("_",
            "-"));
            CultureInfo cultureInfo = null;
            try
            {
                cultureInfo = new CultureInfo(netLanguage);
            }
            catch
            {
                try
                {
                    var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
                    cultureInfo = new CultureInfo(fallback);
                }
                catch
                {
                    cultureInfo = new CultureInfo("en");
                }
            }
            return cultureInfo;
        }
        private string AndroidToDotnetLanguage(string androidLanguage)
        {
            var netLanguage = androidLanguage;
            switch (androidLanguage)
            {
                case "ms-BN": // "Malaysian (Brunei)" not supported .NET culture
                case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture
                case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture
                    netLanguage = "ms"; // closest supported
                    break;
                case "in-ID": // "Indonesian (Indonesia)" has different code in .NET
                    netLanguage = "id-ID"; // correct code for .NET
                    break;
                case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET
                    culture
                    netLanguage = "de-CH"; // closest supported
                    break;
            }
        }
    }
}

```

```

return netLanguage;
}
private string ToDotnetFallbackLanguage(PlatformCulture platformCulture)
{
var netLanguage = platformCulture.LanguageCode; // use the first part of the
identifier (two chars, usually);
switch (platformCulture.LanguageCode)
{
case "gsw":
netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
break;
}
return netLanguage;
}
}
}

```

Localize class for iOS project,

### **C#**

```

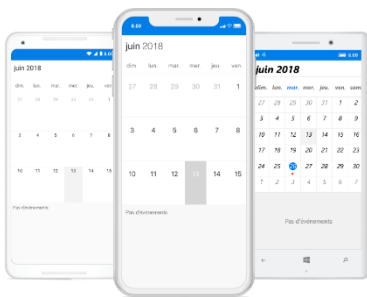
using System.Globalization;
using System.Threading;
using Foundation;
[assembly: Xamarin.Forms.Dependency(typeof(ScheduleLocale.iOS.Localize))]
namespace ScheduleLocale.iOS
{
public class Localize : ILocalize
{
public void SetLocale(CultureInfo cultureInfo)
{
Thread.CurrentThread.CurrentCulture = cultureInfo;
Thread.CurrentThread.CurrentUICulture = cultureInfo;
}
public CultureInfo GetCurrentCultureInfo()
{
var netLanguage = "en";
if (NSLocale.PreferredLanguages.Length > 0)
{
var pref = NSLocale.PreferredLanguages[0];
netLanguage = iOSToDotnetLanguage(pref);
}
CultureInfo cultureInfo = null;
try
{
cultureInfo = new CultureInfo(netLanguage);
}
catch
{
try
{
var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
cultureInfo = new CultureInfo(fallback);
}
catch
{
}
}
}
}

```

```

cultureInfo = new CultureInfo("en");
}
}
return cultureInfo;
}
private string iOSToDotnetLanguage(string iOSLanguage)
{
    var netLanguage = iOSLanguage;
    switch (iOSLanguage)
    {
        case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture
        case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture
            netLanguage = "ms"; // closest supported
            break;
        case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET
            culture
            netLanguage = "de-CH"; // closest supported
            break;
    }
    return netLanguage;
}
private string ToDotnetFallbackLanguage(PlatformCulture platCulture)
{
    var netLanguage = platCulture.LanguageCode; // use the first part of the
    identifier (two chars, usually);
    switch (platCulture.LanguageCode)
    {
        case "pt":
            netLanguage = "pt-PT"; // fallback to Portuguese (Portugal)
            break;
        case "gsw":
            netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
            break;
    }
    return netLanguage;
}
}
}
}

```



## NOTE

For UWP project, no sample level changes required.

## Right to left(RTL)

SfSchedule supports to change the layout direction of the control in the right-to-left direction by setting the [FlowDirection](#) to `RightToLeft` or by changing the device language.

### XML

```
<schedule:SfSchedule FlowDirection="RightToLeft">
</schedule:SfSchedule>
```

### C#

```
schedule.FlowDirection = FlowDirection.RightToLeft;
```

### Note

For implementing the `FlowDirection` in the control, Xamarin.Forms package version must be 3.0 and above. Please refer [RightToLeft](#) to get more details about `RightToLeft` flow direction in Xamarin.Forms.

### Android

For Android, add `android:supportsRtl="true"` in your application tag of `AndroidManifest.xml` file, and make sure your `MinSDKVersion` is 17+. By changing the device language / enabling the device's Force RTL layout can achieve the `RightToLeft` layout direction in Schedule.

### XML

```
<manifest ... >
<uses-sdk android:minSdkVersion="17" ... />
<application ... android:supportsRtl="true">
</application>
</manifest>
```

### iOS

For iOS, add the `RightToLeft` language in the `CFBundleLocalizations` section of your `Info.plist` file, and make sure you're targeting iOS 9+.

### XML

```
<resources>
<key>CFBundleDevelopmentRegion</key>
<string>en</string>
<key>CFBundleLocalizations</key>
<array>
<string>en</string>
<string>ar</string>
</array>
</resources>
```

Localization native development region	String en
▼ Localizations	Array (2 items)
	String en
	String ar

### UWP

For UWP, you need to set `FlowDirection` to `RightToLeft` in the `MainPage.cs` file of the `UWP` project.

### C#

```
public MainPage ()
{
    ...
    this.FlowDirection = FlowDirection.RightToLeft;
    LoadApplication (new App ());
    ...
}
```

### Theming

Theming is a set of resources which are used to provide the consistency look for schedule.

You can modify the default appearance of schedule using this support. By default schedule have default theme resources and it's located in `Syncfusion.Xamarin.Core` assembly. You need to merge them in your application's resource to apply the theme.

In the below code you can see the default color and key value for the default resources.

### XML

```
<!--DayViewSettings-->
<Color x:Key="SfScheduleDayViewNonWorkingHoursTimeSlotColor">#FAFAFA</Color>
<Color
x:Key="SfScheduleDayViewNonWorkingHoursTimeSlotBorderColor">#F0F0F0</Color>
<Color x:Key="SfScheduleDayViewTimeSlotColor">#FFFFFF</Color>
<Color x:Key="SfScheduleDayViewTimeSlotBorderColor">#F0F0F0</Color>
<Color x:Key="SfScheduleDayViewVerticalLineColor">#F0F0F0</Color>
<Color x:Key="SfScheduleDayViewAllDayAppointmentLayoutColor">#FFFFFF</Color>
<!--WeekViewSettings-->
<Color
x:Key="SfScheduleWeekViewNonWorkingHoursTimeSlotColor">#FAFAFA</Color>
<Color
x:Key="SfScheduleWeekViewNonWorkingHoursTimeSlotBorderColor">#F0F0F0</Color>
<Color x:Key="SfScheduleWeekViewTimeSlotColor">#FFFFFF</Color>
<Color x:Key="SfScheduleWeekViewTimeSlotBorderColor">#F0F0F0</Color>
<Color x:Key="SfScheduleWeekViewVerticalLineColor">#F0F0F0</Color>
<Color
x:Key="SfScheduleWeekViewAllDayAppointmentLayoutColor">#FFFFFF</Color>
<!--WorkWeekViewSettings-->
<Color
x:Key="SfScheduleWorkWeekViewNonWorkingHoursTimeSlotColor">#FAFAFA</Color>
<Color
x:Key="SfScheduleWorkWeekViewNonWorkingHoursTimeSlotBorderColor">#F0F0F0</Co
lor>
<Color x:Key="SfScheduleWorkWeekViewTimeSlotColor">#FFFFFF</Color>
<Color x:Key="SfScheduleWorkWeekViewTimeSlotBorderColor">#F0F0F0</Color>
<Color x:Key="SfScheduleWorkWeekViewVerticalLineColor">#F0F0F0</Color>
<Color
x:Key="SfScheduleWorkWeekViewAllDayAppointmentLayoutColor">#FFFFFF</Color>
<!--TimelineViewSettings-->
<Color x:Key="SfScheduleTimelineViewTimeSlotColor">#FFFFFF</Color>
<Color x:Key="SfScheduleTimelineViewTimeSlotBorderColor">#F0F0F0</Color>
```

```

<!--MonthViewSettings-->
<Color x:Key="SfScheduleMonthViewTodayBackground">#1470D0</Color>
<Color x:Key="SfScheduleMonthViewSelectionTextColor">#000000</Color>
<!--DayLabelSettings-->
<Color x:Key="SfScheduleDayViewLabelTimeLabelColor">#707070</Color>
<!--WeekLabelSettings-->
<Color x:Key="SfScheduleWeekViewLabelTimeLabelColor">#707070</Color>
<!--WorkWeekLabelSettings-->
<Color x:Key="SfScheduleWorkWeekViewLabelTimeLabelColor">#707070</Color>
<!--TimelineLabelSettings-->
<Color x:Key="SfScheduleTimelineViewLabelTimeLabelColor">#707070</Color>
<!--HeaderStyle-->
<Color x:Key="SfScheduleHeaderBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfScheduleHeaderTextColor">#707070</Color>
<!--ViewHeaderStyle-->
<Color x:Key="SfScheduleViewHeaderDayTextColor">#707070</Color>
<Color x:Key="SfScheduleViewHeaderBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfScheduleViewHeaderDateTextColor">#707070</Color>
<!--SelectionStyle-->
<Color x:Key="SfScheduleSelectionBackgroundColor">#F5F5F5</Color>
<Color x:Key="SfScheduleSelectionBorderColor">#F0F0F0</Color>
<!--AppointmentStyle-->
<Color x:Key="SfScheduleAppointmentTextColor">#FFFFFF</Color>
<Color x:Key="SfScheduleAppointmentBorderColor">#FFFFFF</Color>
<Color x:Key="SfScheduleAppointmentSelectionBorderColor">#424242</Color>
<Color x:Key="SfScheduleAppointmentSelectionTextColor">#FFFFFF</Color>
<!--MonthViewCellStyle-->
<Color x:Key="SfScheduleMonthCellTextColor">#414141</Color>
<Color x:Key="SfScheduleMonthCellBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfScheduleMonthCellTodayBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfScheduleMonthCellTodayTextColor">#FFFFFF</Color>
<Color x:Key="SfScheduleMonthCellPreviousMonthTextColor">#C2C2C2</Color>
<Color
x:Key="SfScheduleMonthCellPreviousMonthBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfScheduleMonthCellNextMonthBackgroundColor">#FFFFFF</Color>
<Color x:Key="SfScheduleMonthCellNextMonthTextColor">#C2C2C2</Color>
<!--WeekNumberStyle-->
<Color x:Key="SfScheduleMonthViewWeekNumberTextColor">#707070</Color>
<Color x:Key="SfScheduleMonthViewWeekNumberBackgroundColor">#FFFFFF</Color>
<!--AgendaViewStyle-->
<Color x:Key="SfScheduleMonthAgendaViewSubjectFontColor">#414141</Color>
<Color x:Key="SfScheduleMonthAgendaViewBackgroundColor">#F5F5F5</Color>
<Color x:Key="SfScheduleMonthAgendaViewTimeFontColor">#707070</Color>
<Color x:Key="SfScheduleMonthAgendaViewDateFontColor">#707070</Color>
<!--TimeIndicatorStyle-->
<Color x:Key="SfScheduleTimeIndicatorTextColor">#6D25E8</Color>

```

You can apply the default theme (light theme) or customized theme to schedule.

Default theme (light theme)

You need to apply the syncfusion theme dictionaries in your application to view the default theme (light theme).

#### XML

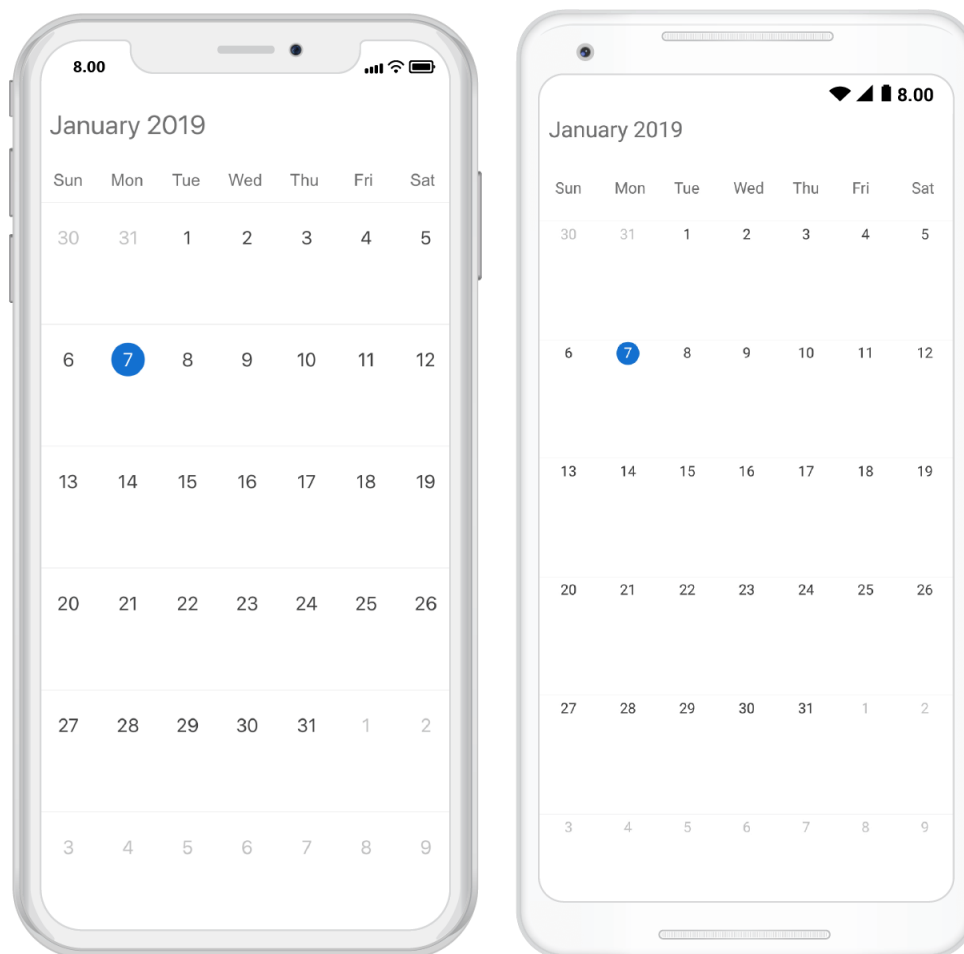
```
<?xml version="1.0" encoding="utf-8"?>
```

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:ScheduleTheme" x:Class="ScheduleTheme.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfSchedule.XForms;assembly=Syncfusion.SfSchedule.XForms
"
xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms">
<ContentPage.Resources>
<syncTheme:SyncfusionThemeDictionary>
<syncTheme:SyncfusionThemeDictionary.MergedDictionaries>
<syncTheme:LightTheme x:Name="LightTheme" />
</syncTheme:SyncfusionThemeDictionary.MergedDictionaries>
</syncTheme:SyncfusionThemeDictionary>
</ContentPage.Resources>
<syncfusion:SfSchedule x:Name="schedule">
</syncfusion:SfSchedule>
</ContentPage>

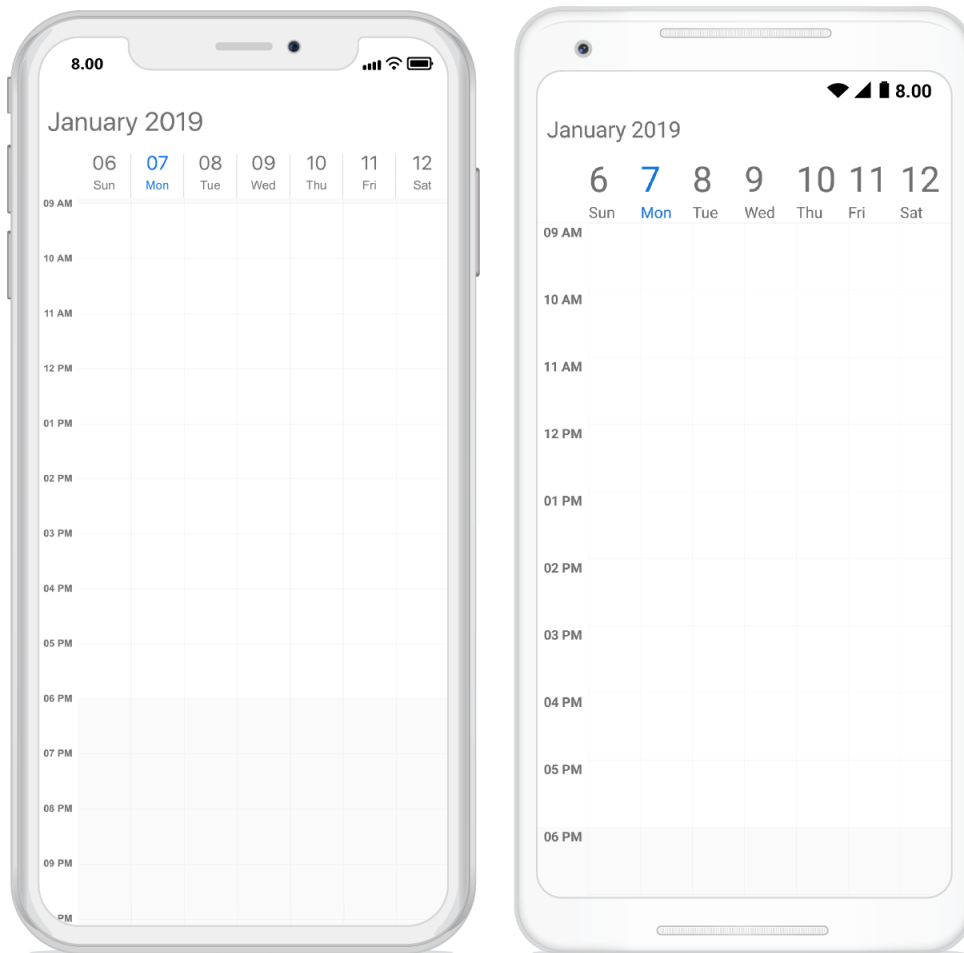
```

### Month view

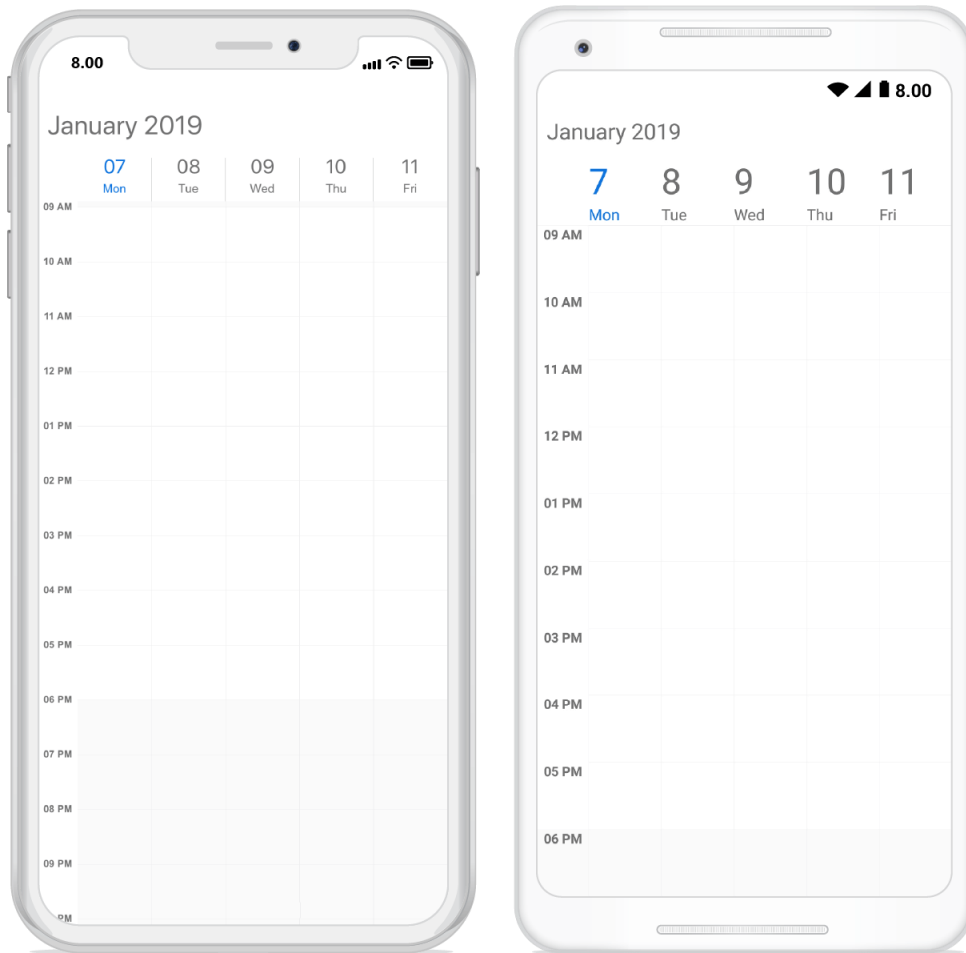


### Week view

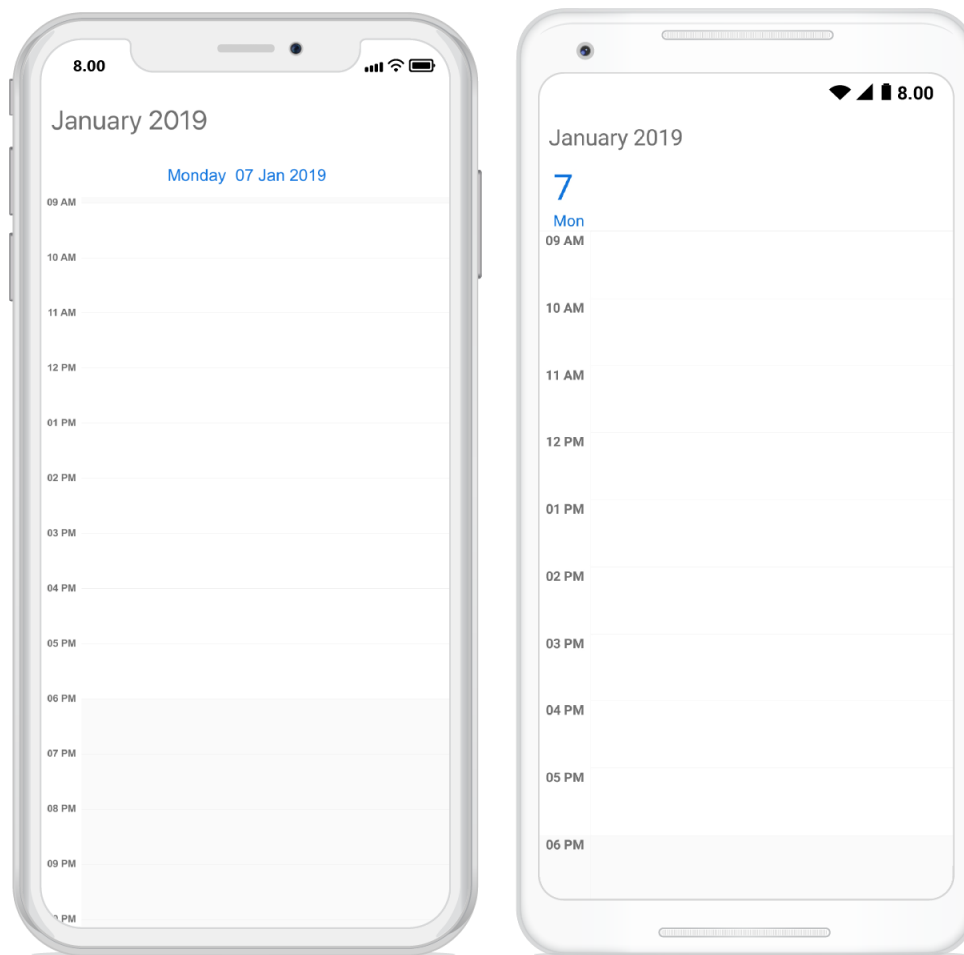




Work Week view



Day view



### Customizing the default theme

You can customize the default theme by overriding the existing key and set the new value.

#### XML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:ScheduleTheme" x:Class="ScheduleTheme.MainPage"
xmlns:syncfusion="clr-
namespace:Syncfusion.SfSchedule.XForms;assembly=Syncfusion.SfSchedule.XForms
"
xmlns:syncTheme="clr-
namespace:Syncfusion.XForms.Themes;assembly=Syncfusion.Core.XForms">
<ContentPage.Resources>
<syncTheme:SyncfusionThemeDictionary>
<syncTheme:SyncfusionThemeDictionary.MergedDictionaries>
<syncTheme:LightTheme x:Name="LightTheme" />
<syncfusion:SfScheduleStyles />
<ResourceDictionary>
<Color x:Key="SfScheduleHeaderTextColor">Red</Color>
<Color x:Key="SfScheduleViewHeaderDayTextColor">Blue</Color>
<Color x:Key="SfScheduleViewHeaderDateTextColor">Blue</Color>
```

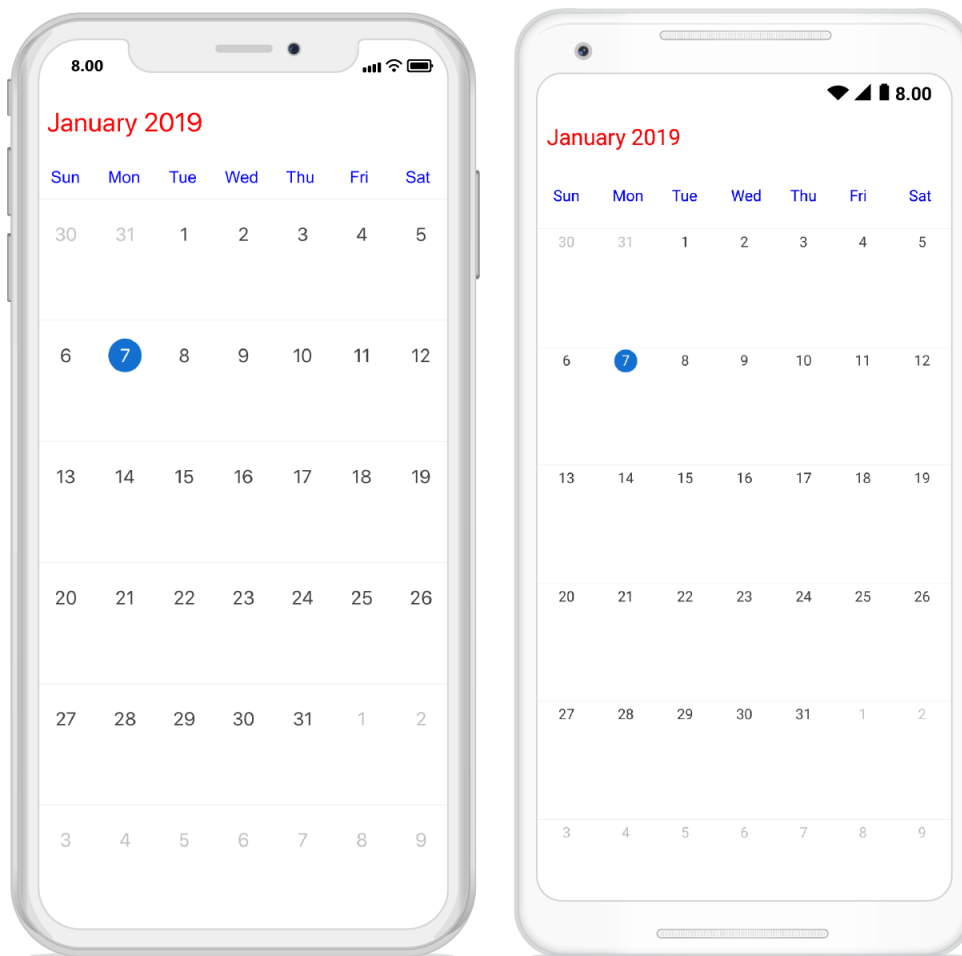
```
</ResourceDictionary>
</syncTheme:SyncfusionThemeDictionary.MergedDictionaries>
</syncTheme:SyncfusionThemeDictionary>
</ContentPage.Resources>
<syncfusion:SfSchedule x:Name="schedule">
</syncfusion:SfSchedule>
</ContentPage>
```

---

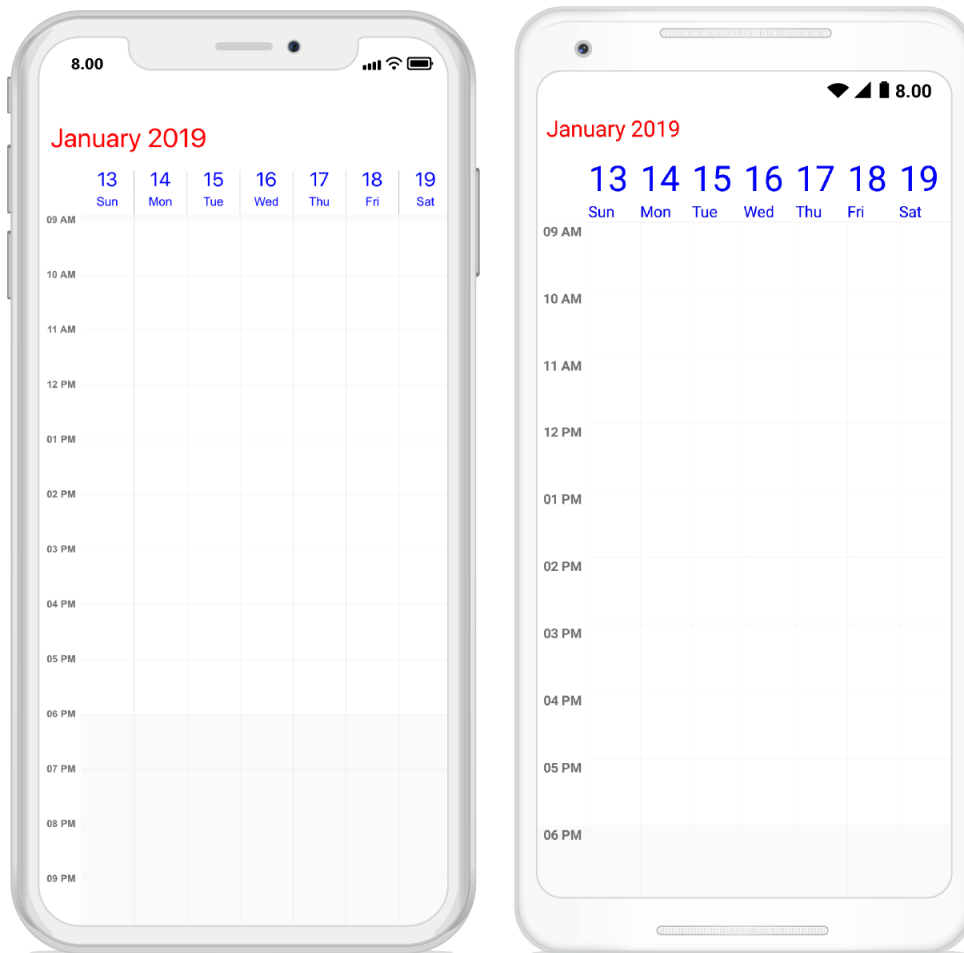
**NOTE**

Xamarin.Forms version should be above 3 while customize the default theme.

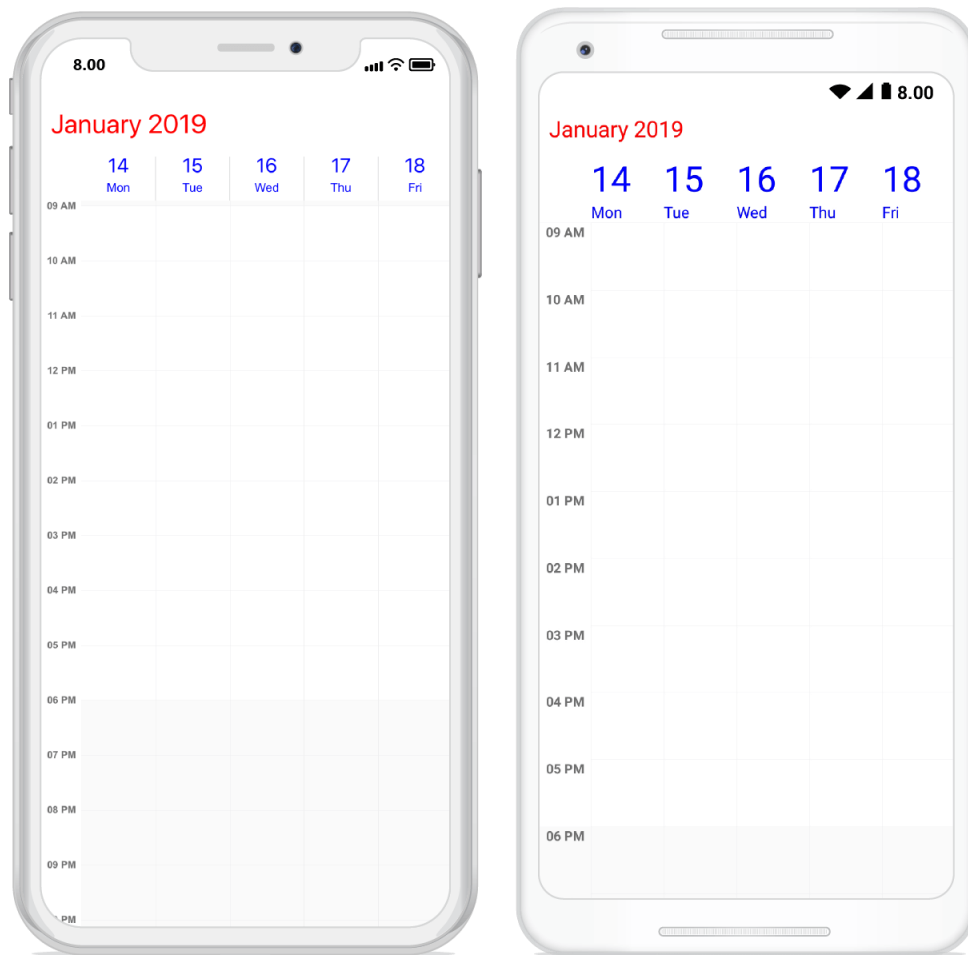
Month view



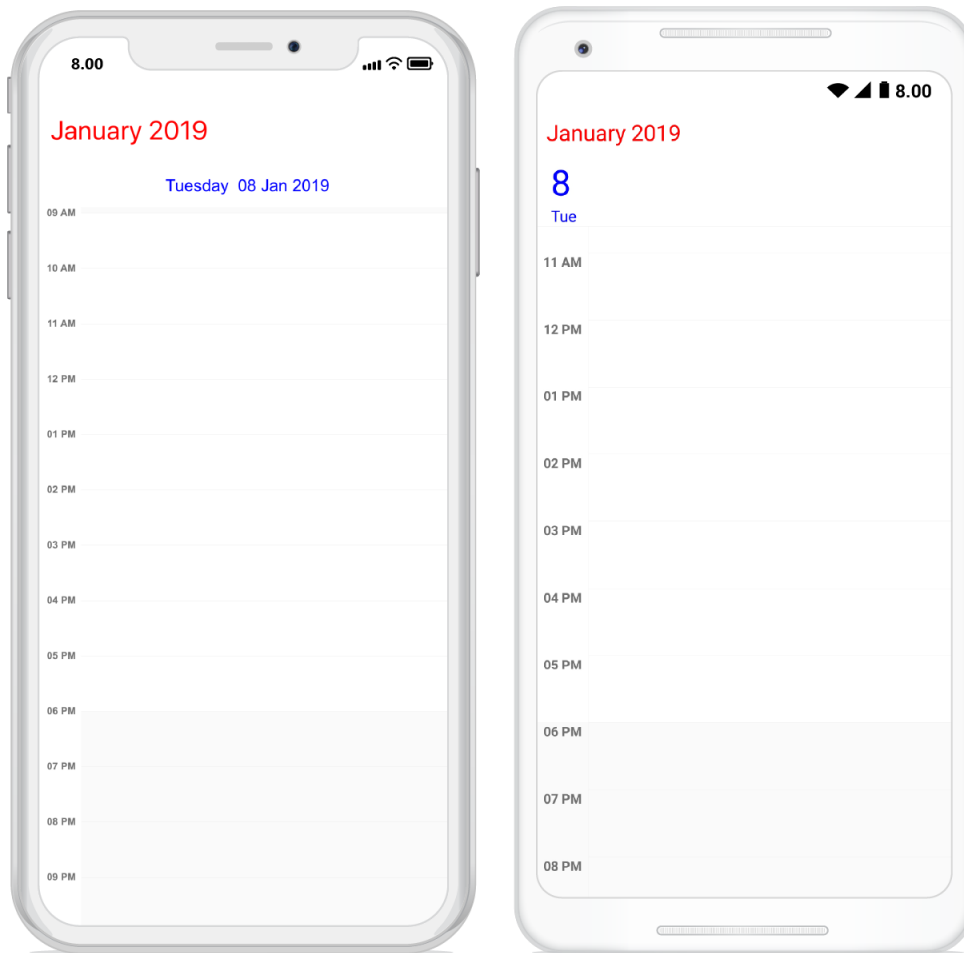
Week view



Work Week view



Day view



You can download the entire source code of this demo for Xamarin.Forms from here [ScheduleTheme](#).

### AutomationId

The **SfSchedule** control has built-in **AutomationId** for inner elements. Please find the following table of Automation IDs for inner elements. To keep unique **AutomationId**, these inner elements' AutomationIds are updated based on the control's **AutomationId**. For example, if you set **SfSchedule AutomationId** as **SfSchedule.AutomationId = EventScheduler**, then the Automation framework will interact with the month header as **EventSchedulerAugust 2019**. The following screenshots denote the AutomationIds for inner elements.

### Month view

View	AutomationId Format	Example
MonthCell	dd/MMMM/yyyy	01/July/2019
Month Header	MMMM yyyy	August 2019
Inline View	hh a Subject dd/MMMM/yyyy	10 AM Consulting 22/August/2019

Agenda View	hh a Subject dd/MMMM/yyyy	11 AM Planning 22/August/2019
No Events	No Events text	No Events

September 2019

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

11

Wed

Support  
09:00:AM - 11:00:AM

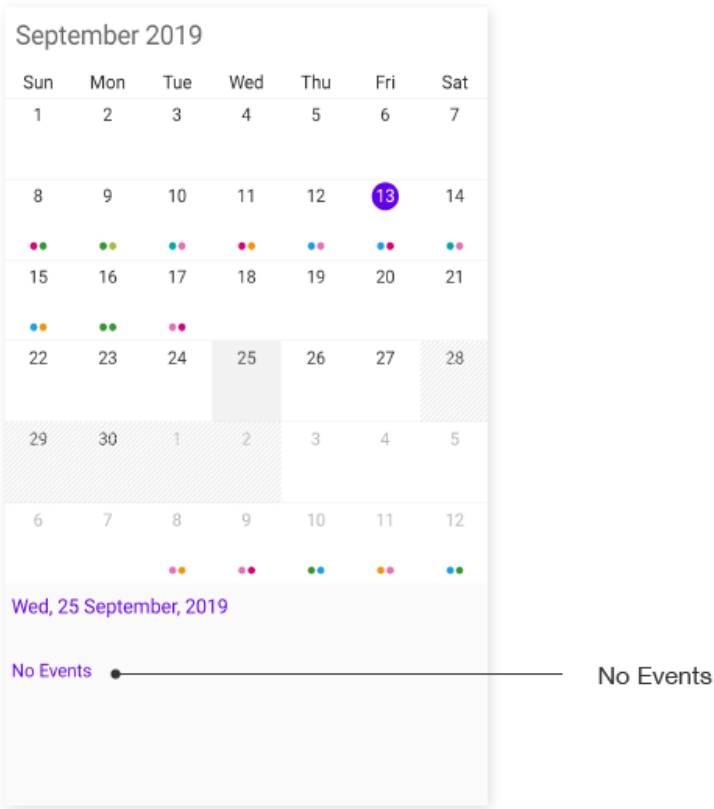
Consulting  
02:00:PM - 04:00:PM

EventSchedulerSeptember 2019

EventScheduler11/September/2019

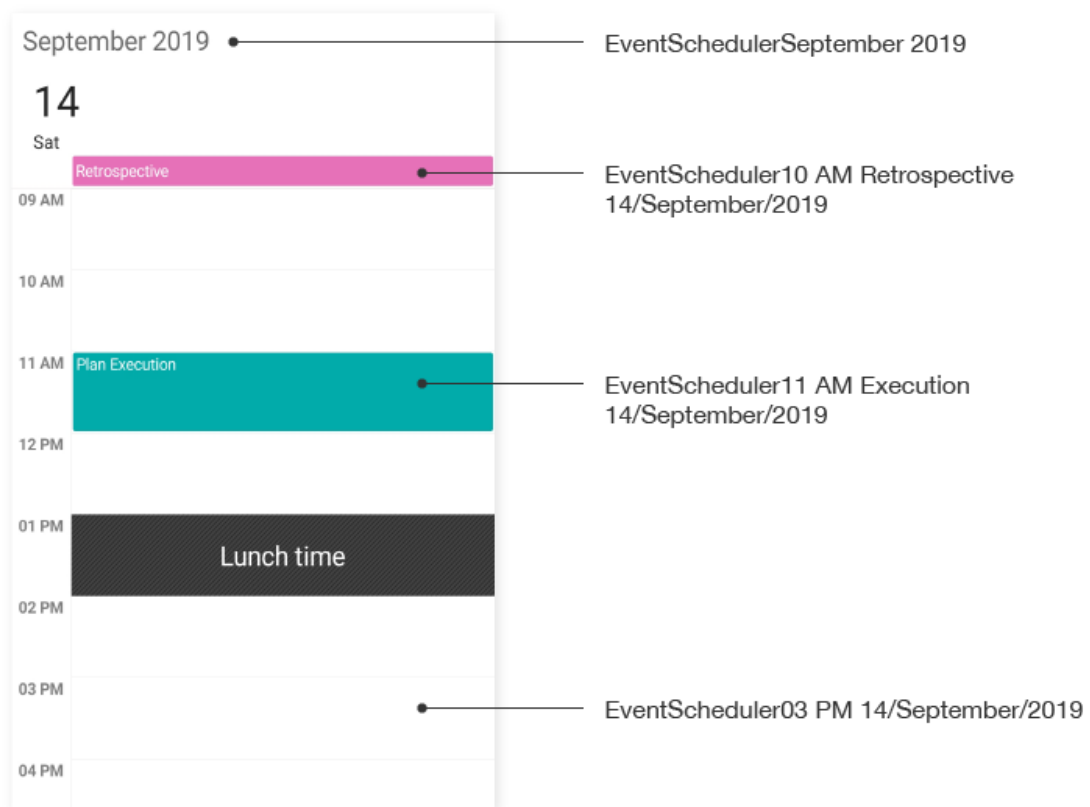
EventScheduler09 AM Support  
11/September/2019

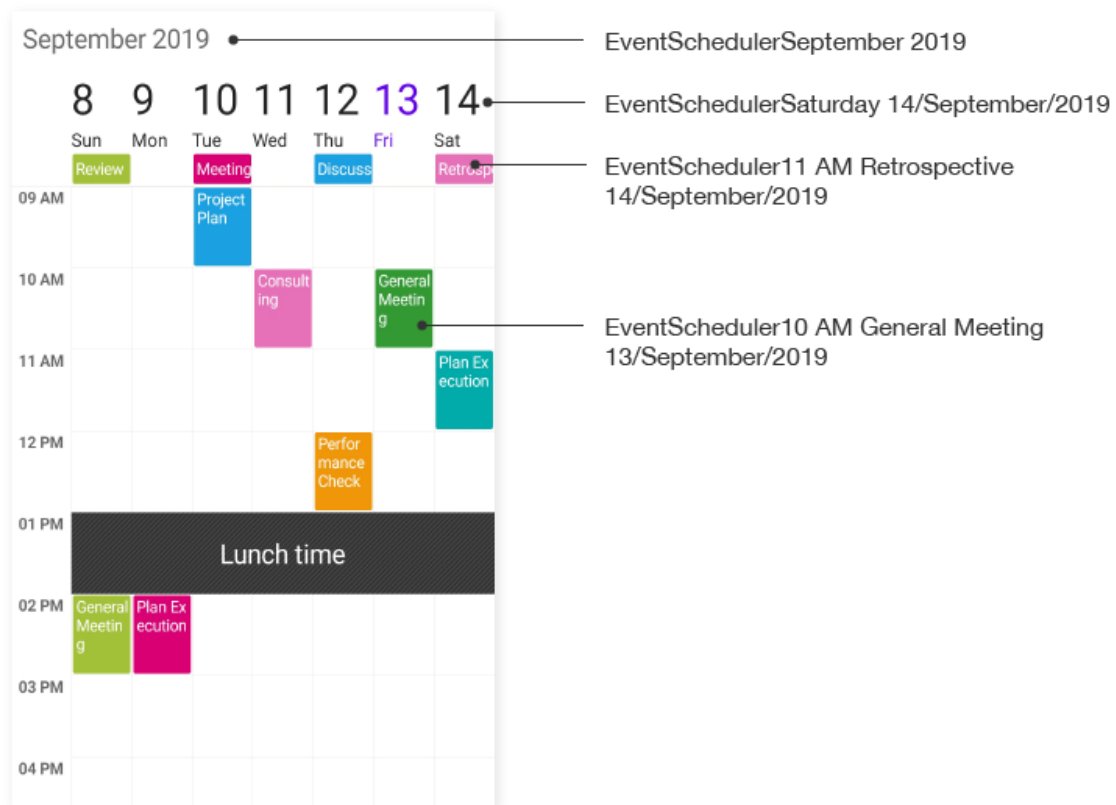


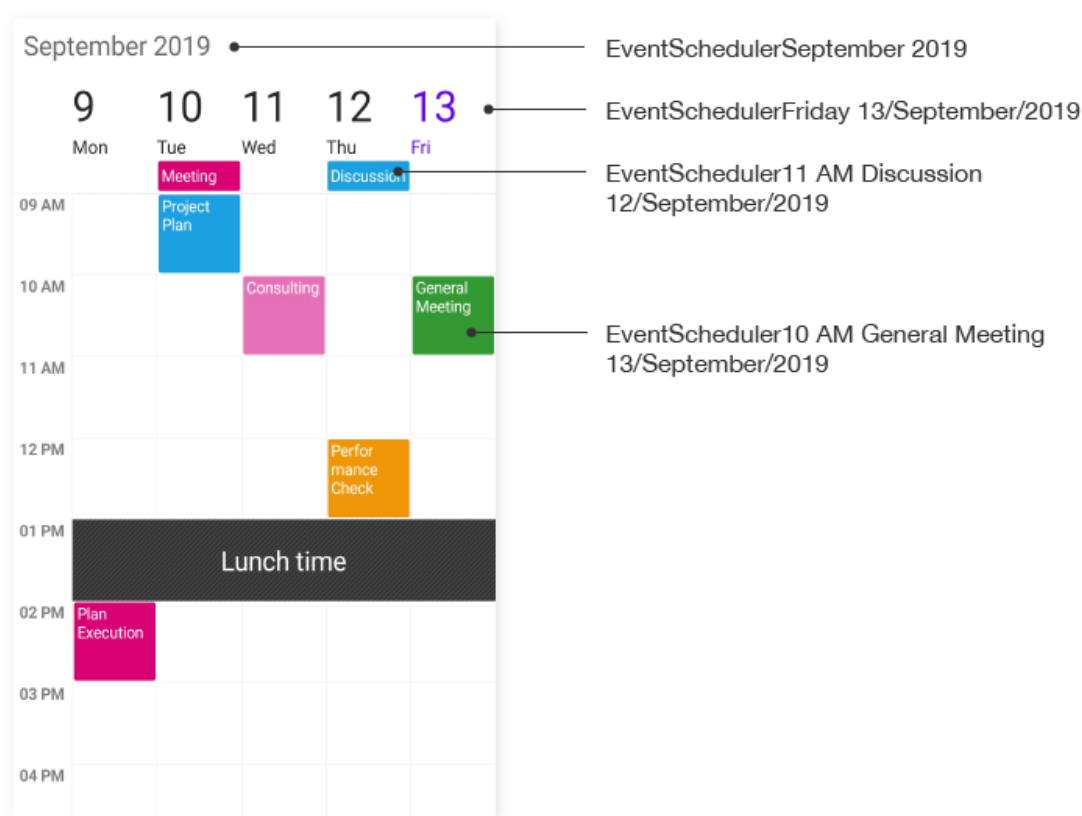


Day, week and workweek views

View	AutomationId Format	Example
TimeslotView(Only for DayView)	hh a dd/MMMM/yyyy	09 AM 22/August/2019
Appointment	hh a Subject dd/MMMM/yyyy	10 AM Consulting 22/August/2019
Header	MMMM yyyy	August 2019
ViewHeader	dddd dd/MMMM/yyyy	Sunday 22/August/2019
AllDay	hh a Subject dd/MMMM/yyyy	10 AM Consulting 22/August/2019

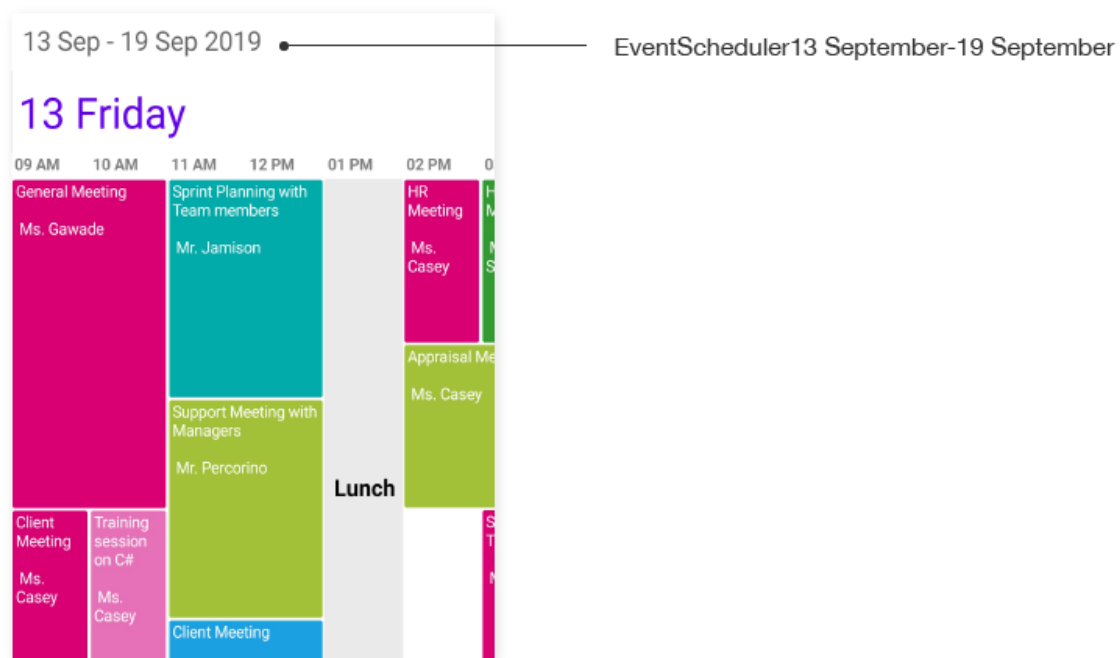
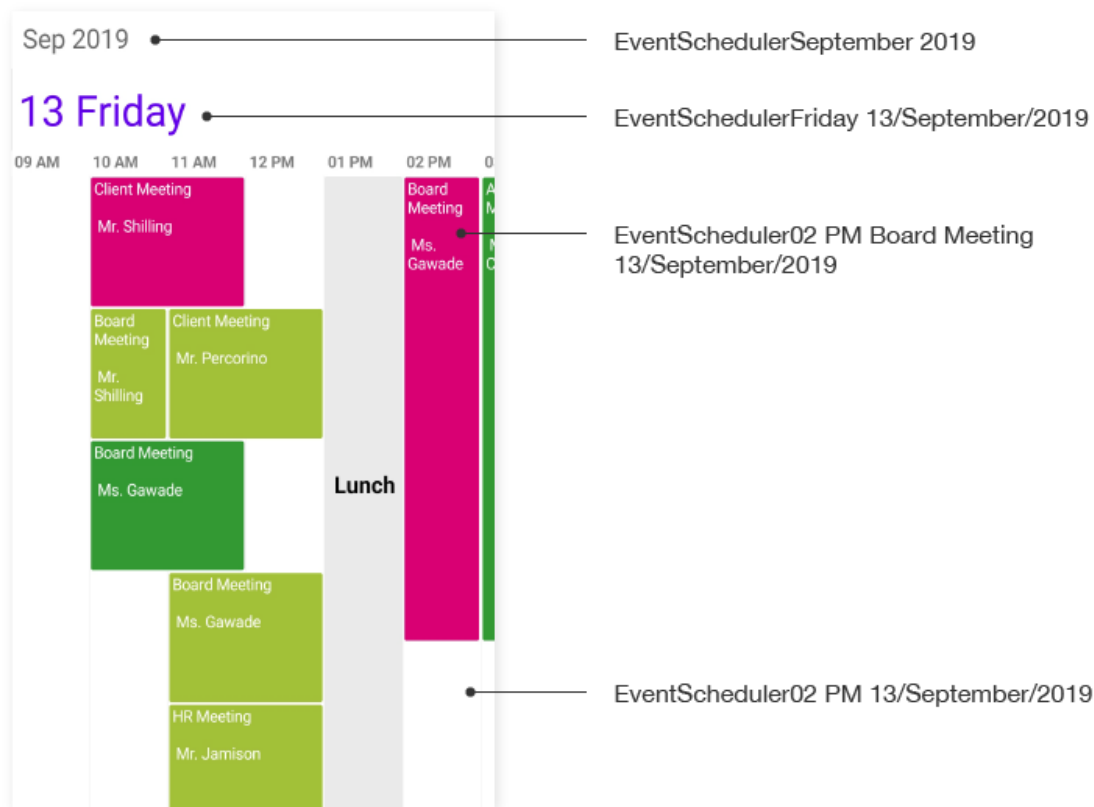






Timeline view

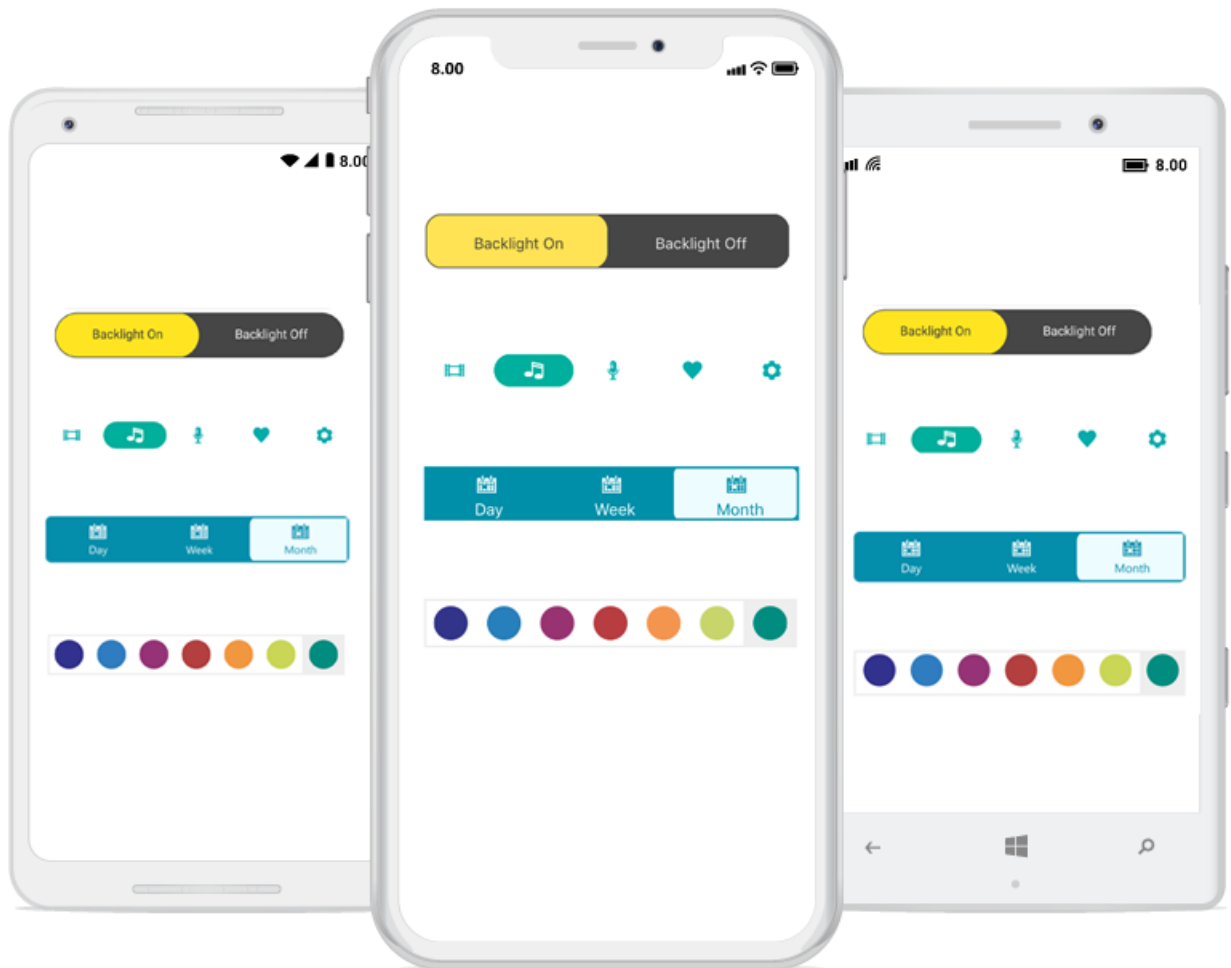
View	AutomationId Format	Example
TimeslotView(Only applicable if days count is 1)	hh a dd/MMMM/yyyy	09 AM 22/August/2019
Appointment	hh a Subject dd/MMMM/yyyy	10 AM Consulting 22/August/2019
Header	MMMM yyyy	August 2019
Header - DaysCount	dd MMMM - dd MMMM yyyy	04 December - 08 December 2018
ViewHeader	dddd dd/MMMM/yyyy	Sunday 22/August/2019



## SfSegmentedControl

### Overview

The segmented control for Xamarin.Forms provides a simple way to choose from a linear set of two or more segments, each of which functions as a mutually exclusive button.



### Key Features

- Supports displaying the segments with text, font icons, or both.
- Supports scrolling and aligning the segments equally within the available space.
- Populates the segments from a collection of strings and views along with the objects of built-in classes.
- Supports customizing text and other UI elements.

### Getting Started

This section provides an overview for working with the segmented control for Xamarin.Forms. Walk through the entire process of creating a real-world application with the SfSegmentedControl.

### Assembly deployment

After installing the Essential Studio for Xamarin, find all the required assemblies in installation folders, {Syncfusion Essential Studio Installed location}\Essential Studio\16.2.0.41\Xamarin\lib}.

E.g., C:\Program Files (x86)\Syncfusion\Essential Studio\16.2.0.41\Xamarin\lib

---

**Note:** Assemblies can be found in an unzipped package location in Mac.

---

### Adding SfSegmentedControl reference

You can add SfSegmentedControl reference using one of the following methods:

#### Method 1: Adding SfSegmentedControl reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Buttons). To add SfSegmentedControl to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Buttons](https://www.nuget.org/packages/Syncfusion.Xamarin.Buttons), and then install it.

![Adding SfSegmentedControl reference from NuGet](images/getting-started/Adding SfSegmentedControl reference.png)

**Note:**

- Install the same version of SfSegmentedControl NuGet in all the projects.
- In addition, you need to install the [Syncfusion.Xamarin.Buttons.WPF]() package for Xamarin.Forms WPF platform only.

#### Method 2: Adding SfSegmentedControl reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfSegmentedControl control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfSegmentedControl assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Buttons.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll

UWP	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
WPF	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.WPF.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.WPF.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

#### Launching the SfSegmentedControl on each platform

To use the segmented control inside an application, each platform application must initialize the segmented control renderer. This initialization steps vary from platform to platform, and it is discussed in the following sections:

##### *Android and UWP*

The Android and UWP launches the SfSegmentedControl without any initialization, and it is enough to only initialize the Xamarin.Forms Framework to launch the application.

**Note:** If you are adding the references from toolbox, this step is not needed.

##### *iOS*

To launch the segmented control in iOS, call the SfSegmentedControlRenderer.Init() in the FinishedLaunching overridden method of the AppDelegate class after the Xamarin.Forms Framework has been initialized and before the LoadApplication is called, as demonstrated in the following code example.

##### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.XForms.iOS.Buttons.SfSegmentedControlRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

##### *ReleaseMode issue in UWP platform*

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in **Release Mode**.



The above problem can be resolved by initializing the SfSegmentedControl assemblies in `App.xaml.cs` in UWP project as like in below code snippet.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.Buttons.SfSegmentedControlRenderer()).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

#### *Additional step for WPF*

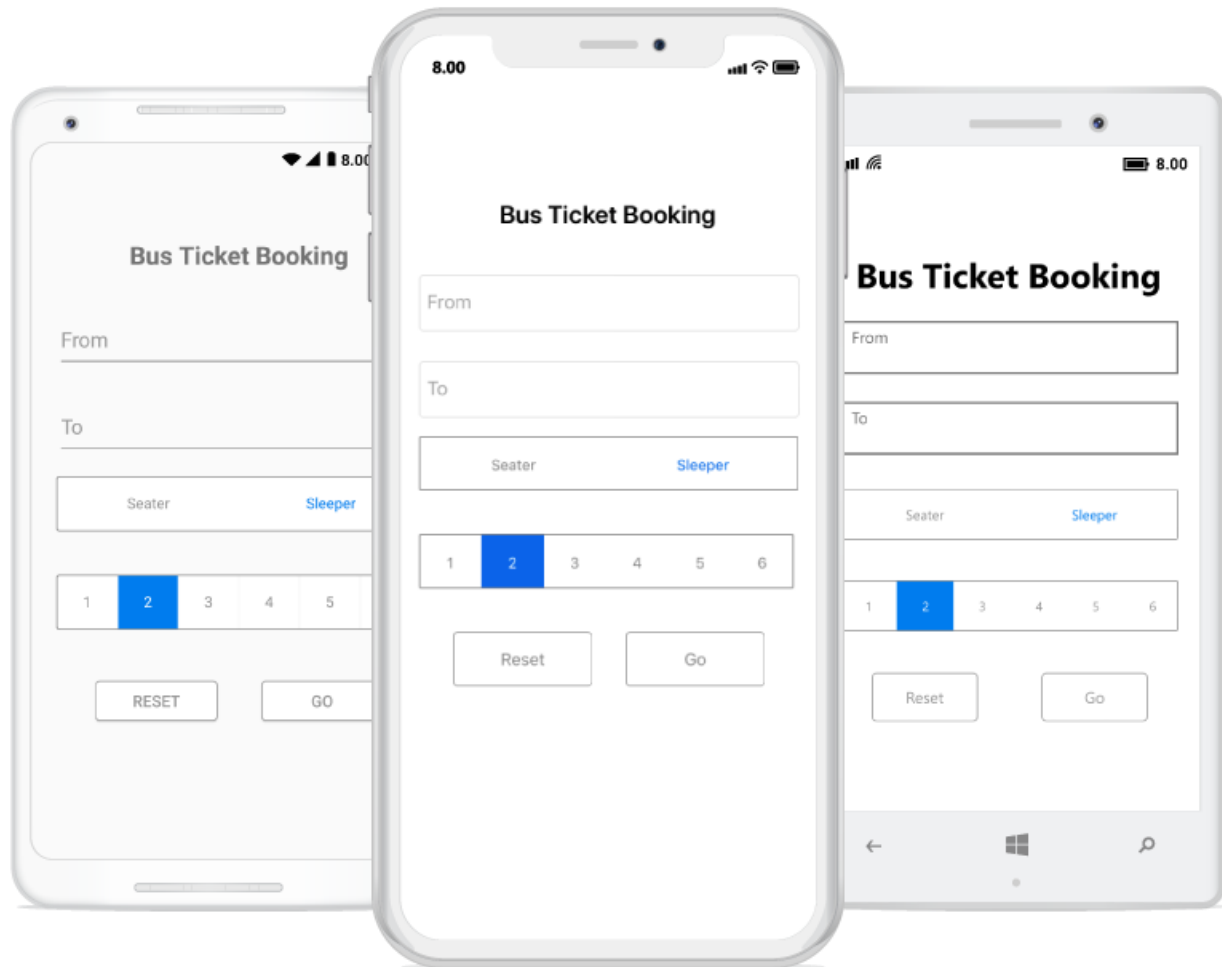
To launch the segmented control in WPF, call the `SfSegmentedControlRenderer.Init()` method in the `MainWindow` constructor of the `MainWindow` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

#### C#

```
public partial class MainWindow : FormsApplicationPage
{
    public MainWindow()
    {
        InitializeComponent();
        Forms.Init();
        Syncfusion.XForms.WPF.Buttons.SfSegmentedControlRenderer.Init();
        LoadApplication(new App());
    }
}
```

#### Creating an application with segmented control.

This section explains how to create a bus ticket booking module by using syncfusion segmented control. The control can be configured entirely in C# code or by using XAML markup. The following screenshot illustrates the output of Segmented control on iOS, Android and UWP devices.



### Creating a project

Create a new BlankApp (Xamarin.Forms.Portable) application in Visual Studio for Xamarin.Forms.

### Adding SfSegmentedControl in Xamarin.Forms

Add the required assembly references to the PCL and renderer projects as discussed in the [Assembly deployment section](#).

Import the control namespace as shown in the following code.

#### XML

```
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
```

#### C#

```
using Syncfusion.XForms.Buttons;
```

Set the control to content in `ContentPage`.

#### XML

```
<? xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<buttons:SfSegmentedControl />
</ContentPage.Content>
</ContentPage>
```

**C#**

```
using Syncfusion.XForms.Buttons;
using Xamarin.Forms;
namespace GettingStarted
{
    public partial class MainPage : ContentPage
    {
        private SfSegmentedControl segmentedControl;
        public MainPage()
        {
            InitializeComponent();
            segmentedControl = new SfSegmentedControl();
            this.Content = segmentedControl;
        }
    }
}
```

*Adding supportive views to the application.*

For the completeness of the ticket booking application, few framework controls are added to the application, to get the data from the user.

ViewModel class for the Entry which we have used in our View.

**C#**

```
using Syncfusion.XForms.Buttons;
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Xamarin.Forms;
namespace SegmentGettingStarted
{
    class ViewModel : INotifyPropertyChanged
    {
        private string fromText="";
        public string FromText
        {
            get { return fromText; }
            set { fromText = value; NotifyPropertyChanged("FromText"); }
        }
        private string toText = "";
        public string ToText
        {

```

```

get { return toText; }
set { toText = value; NotifyPropertyChanged("ToText"); }
}
public ViewModel()
{
}
public event PropertyChangedEventHandler PropertyChanged;
private void NotifyPropertyChanged([CallerMemberName] String propertyName =
    "")
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
}
}
}

```

View can be created by the following code snippet.

#### XML

```

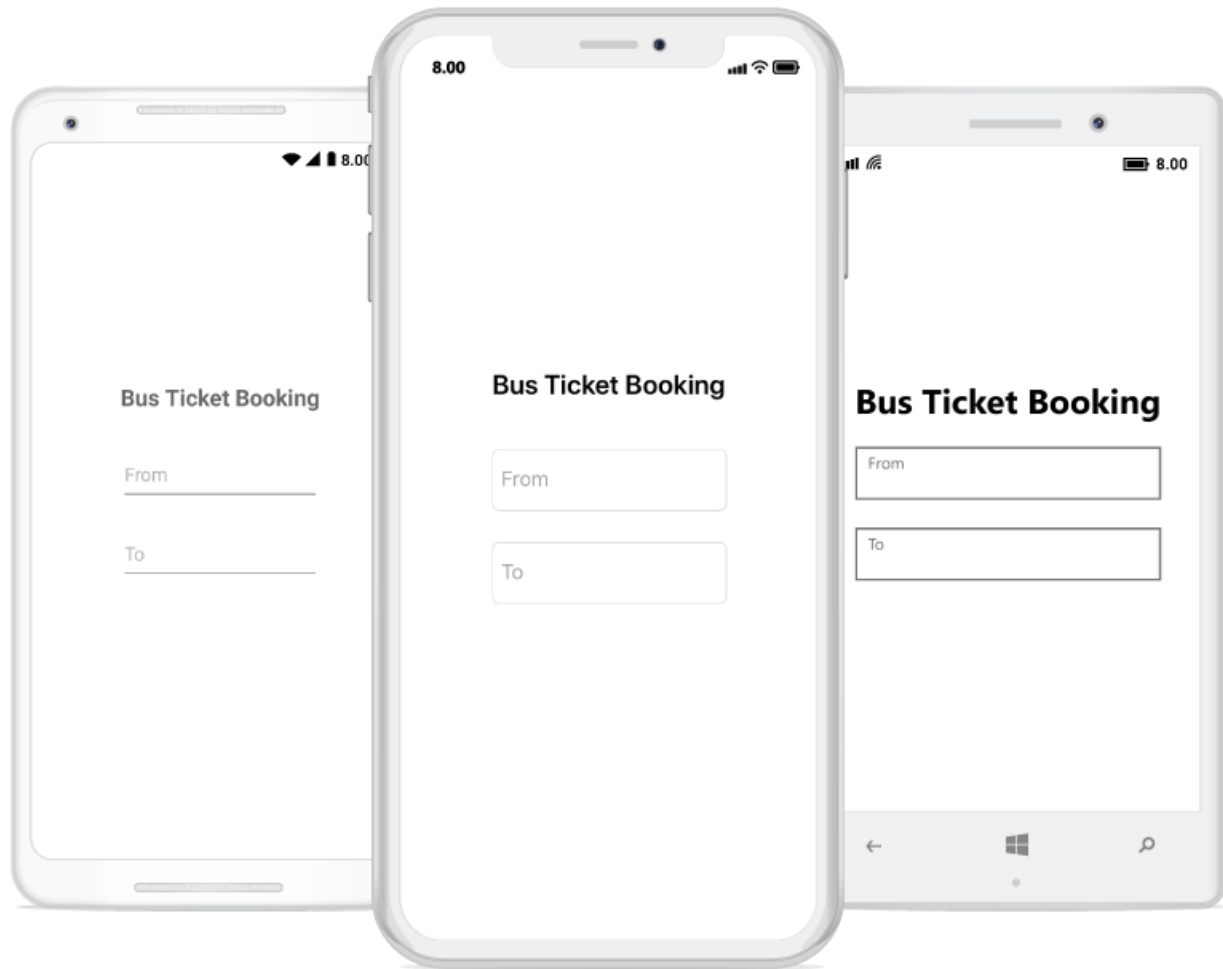
<? xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SegmentGettingStarted"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:segmentCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
x:Class="SegmentGettingStarted.MainPage">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<StackLayout
HorizontalOptions = "Center"
VerticalOptions="Center"
Padding="20,0,20,0">
<Label
Text="Bus Ticket Booking"
FontSize="Large"
FontAttributes="Bold"
HeightRequest="50"
HorizontalOptions="Center"
VerticalOptions="Center"/>
<Entry
Placeholder="From"
Text="{Binding FromText,Mode=TwoWay}"
HeightRequest="50"
Margin="0,10,0,10"/>
<Entry
Placeholder="To"
Text="{Binding ToText}"
HeightRequest="50"
Margin="0,10,0,10"/>
</StackLayout>

```

`</ContentPage>`

**C#**

```
using Syncfusion.XForms.Buttons;
using System.Collections.Generic;
using Xamarin.Forms;
namespace SegmentGettingStarted
{
    public partial class MainPage : ContentPage
    {
        StackLayout stack;
        ViewModel viewModel;
        Label headerLabel;
        Entry fromEntry, toEntry;
        public MainPage()
        {
            InitializeComponent();
            stack = new StackLayout();
            viewModel = new ViewModel();
            stack.HorizontalOptions = LayoutOptions.Center;
            stack.VerticalOptions = LayoutOptions.Center;
            stack.Padding = new Thickness(20, 0, 20, 0);
            //Label to denote the header part of application
            headerLabel = new Label();
            headerLabel.Text = "Bus Ticket Booking";
            headerLabel.FontAttributes = FontAttributes.Bold;
            headerLabel.HeightRequest = 50;
            headerLabel.FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label));
            headerLabel.HorizontalOptions = LayoutOptions.Center;
            headerLabel.VerticalOptions = LayoutOptions.Center;
            //Entry to enter the origin location
            fromEntry = new Entry();
            fromEntry.Placeholder = "From";
            fromEntry.Text = viewModel.FromText;
            fromEntry.HeightRequest = 50;
            fromEntry.Margin = new Thickness(0, 10, 0, 10);
            //Entry to enter the destination location
            toEntry = new Entry();
            toEntry.Placeholder = "To";
            toEntry.Text = viewModel.ToText;
            toEntry.HeightRequest = 50;
            toEntry.Margin = new Thickness(0, 10, 0, 10);
            stack.Children.Add(headerLabel);
            stack.Children.Add(fromEntry);
            stack.Children.Add(toEntry);
            this.Content = stack;
        }
    }
}
```



### Adding data/Items to SfSegmentedControl

We can add the data inside the segmented control in 3 different ways.

1. String data
2. SfSegmentItem
3. Custom View

Items inside the segmented control can be added through the `ItemsSource` property of `SfSegmentedControl`, which holds the collection/list of items.

#### Adding data as a String

With the help of `Xamarin.Forms` `System.Collections.Generic` we can add string data as `ItemsSource` to `SfSegmentedControl`.

#### C#

```
using Syncfusion.XForms.Buttons;  
using System;  
using System.Collections.ObjectModel;  
using System.ComponentModel;  
using System.Runtime.CompilerServices;
```

```

using Xamarin.Forms;
namespace SegmentGettingStarted
{
    class ViewModel : INotifyPropertyChanged
    {
        private string fromText="";
        public string FromText
        {
            get { return fromText; }
            set { fromText = value; NotifyPropertyChanged("FromText"); }
        }
        private string toText = "";
        public string ToText
        {
            get { return toText; }
            set { toText = value; NotifyPropertyChanged("ToText"); }
        }
        public ViewModel()
        {
        }
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged([CallerMemberName] String propertyName =
            "")
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }
    }
}

```

## XML

```

<? xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns = "http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SegmentGettingStarted"
xmlns:buttons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:segmentCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
x:Class="SegmentGettingStarted.MainPage">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<StackLayout
HorizontalOptions = "Center"
VerticalOptions="Center"
Padding="20,0,20,0">
<Label
Text="Bus Ticket Booking"
FontSize="Large"
FontAttributes="Bold"
HeightRequest="50"
HorizontalOptions="Center"

```

```

VerticalOptions="Center"/>
<Entry
Placeholder="From"
Text="{Binding FromText,Mode=TwoWay}"
HeightRequest="50"
Margin="0,10,0,10"/>
<Entry
Placeholder="To"
Text="{Binding ToText}"
HeightRequest="50"
Margin="0,10,0,10"/>
<buttons:SfSegmentedControl
SelectionTextColor = "White"
HeightRequest="80"
VisibleSegmentsCount="6"
Color="Transparent"
BorderColor="#929292"
SelectedIndex="1"
FontColor="#929292"
BackgroundColor="Transparent" >
<segmentCollection:List x:TypeArguments="x:String">
<x:String>1</x:String>
<x:String>2</x:String>
<x:String>3</x:String>
<x:String>4</x:String>
<x:String>5</x:String>
<x:String>6</x:String>
</segmentCollection:List>
</buttons:SfSegmentedControl>
</StackLayout>
</ContentPage>

```

## C#

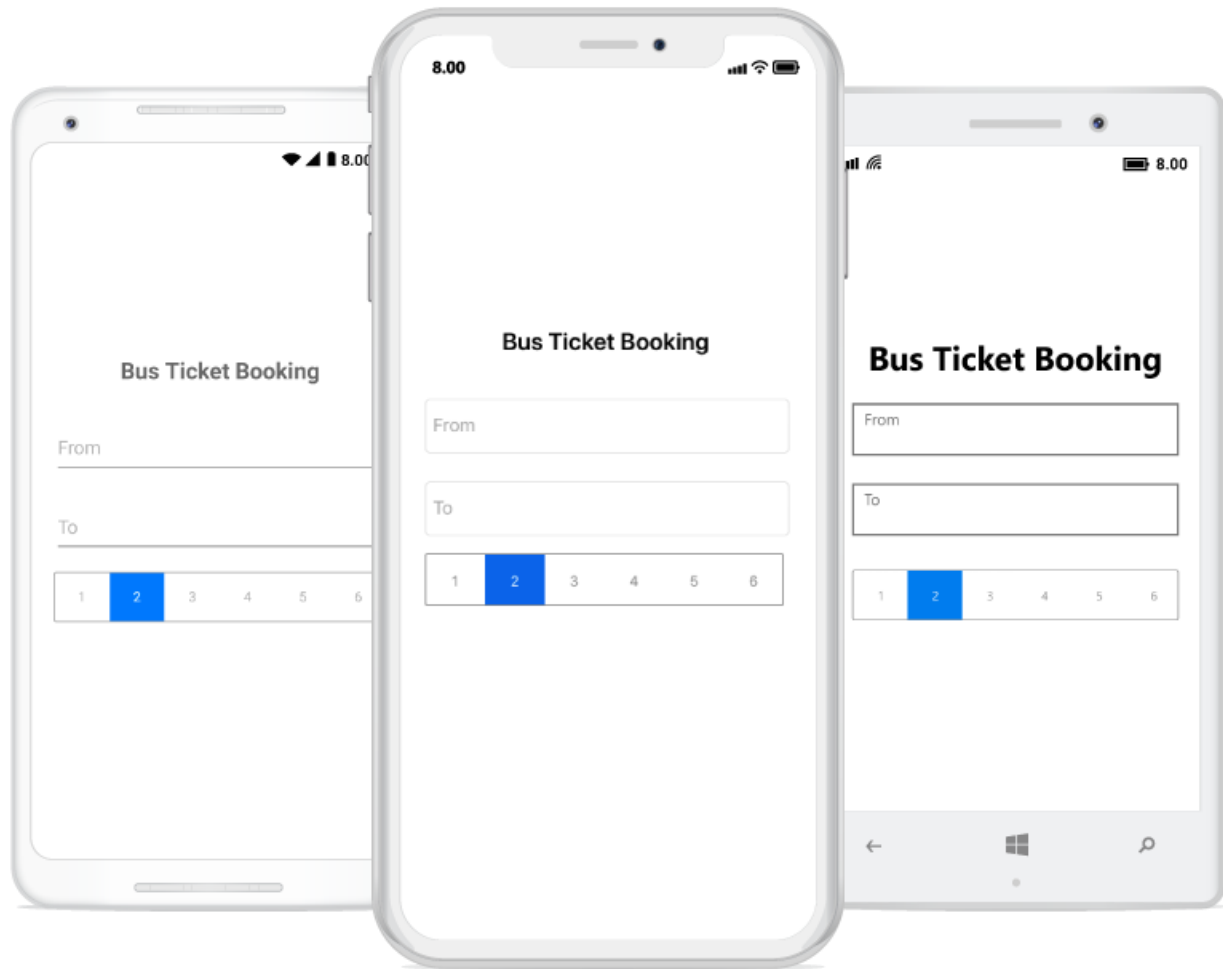
```

using Syncfusion.XForms.Buttons;
using System.Collections.Generic;
using Xamarin.Forms;
namespace SegmentGettingStarted
{
public partial class MainPage : ContentPage
{
SfSegmentedControl segmentedControl;
StackLayout stack;
ViewModel viewModel;
Label headerLabel;
Entry fromEntry, toEntry;
public MainPage()
{
InitializeComponent();
stack = new StackLayout();
viewModel = new ViewModel();
stack.HorizontalOptions = LayoutOptions.Center;
stack.VerticalOptions = LayoutOptions.Center;
stack.Padding = new Thickness(20, 0, 20, 0);
//Label to denote the header part of application
headerLabel = new Label();
}
}

```



```
headerLabel.Text = "Bus Ticket Booking";
headerLabel.FontAttributes = FontAttributes.Bold;
headerLabel.HeightRequest = 50;
headerLabel.FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label));
headerLabel.HorizontalOptions = LayoutOptions.Center;
headerLabel.VerticalOptions = LayoutOptions.Center;
//Entry to enter the origin location
fromEntry = new Entry();
fromEntry.Placeholder = "From";
fromEntry.Text = viewModel.FromText;
fromEntry.HeightRequest = 50;
fromEntry.Margin = new Thickness(0, 10, 0, 10);
//Entry to enter the destination location
toEntry = new Entry();
toEntry.Placeholder = "To";
toEntry.Text = viewModel.ToText;
toEntry.HeightRequest = 50;
toEntry.Margin = new Thickness(0, 10, 0, 10);
//Adding item to the SegmentedControl as String
segmentedControl = new SfSegmentedControl();
segmentedControl.SelectionTextColor = Color.White;
segmentedControl.HeightRequest = 80;
segmentedControl.Color = Color.Transparent;
segmentedControl.BorderColor = Color.FromHex("#929292");
segmentedControl.FontColor = Color.FromHex("#929292");
segmentedControl.SelectedIndex = 1;
segmentedControl.BackgroundColor = Color.Transparent;
segmentedControl.VisibleSegmentsCount = 6;
segmentedControl.DisplayMode = SegmentDisplayMode.Text;
List<string> list = new List<string>
{
    "1", "2", "3", "4", "5", "6"
};
segmentedControl.ItemsSource = list;
stack.Children.Add(headerLabel);
stack.Children.Add(fromEntry);
stack.Children.Add(toEntry);
stack.Children.Add(segmentedControl);
this.Content = stack;
}
}
}
```



### *Adding data as a SfSegmentItem*

By using **SfSegmentItem** class, we can add data inside the segmented control.

In ViewModel add the below given code to get the respective items in SfSegmentedControl.

#### **C#**

```
using Syncfusion.XForms.Buttons;
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Xamarin.Forms;
namespace SegmentGettingStarted
{
    class ViewModel : INotifyPropertyChanged
    {
        private string fromText="";
        public string FromText
        {
            get { return fromText; }
            set { fromText = value; NotifyPropertyChanged("FromText"); }
        }
    }
}
```

```

private string toText = "";
public string ToText
{
    get { return toText; }
    set { toText = value; NotifyPropertyChanged("ToText"); }
}
public ViewModel()
{
    itemCollection = new ObservableCollection<SfSegmentItem>
    {
        new SfSegmentItem() { Text = "Seater"},
        new SfSegmentItem() { Text = "Sleeper"},
    };
}
public event PropertyChangedEventHandler PropertyChanged;
private void NotifyPropertyChanged([CallerMemberName] String propertyName =
    "")
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
private ObservableCollection<SfSegmentItem> itemCollection = new
    ObservableCollection<SfSegmentItem>();
public ObservableCollection<SfSegmentItem> ItemCollection
{
    get { return itemCollection; }
    set { itemCollection = value; }
}
}

```

## XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:SegmentGettingStarted"
    xmlns:buttons="clr-
        namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
    xmlns:segmentCollection="clr-
        namespace:System.Collections.Generic;assembly=netstandard"
    x:Class="SegmentGettingStarted.MainPage">
    <ContentPage.BindingContext>
    <local:ViewModel x:Name="viewModel" />
    </ContentPage.BindingContext>
    <StackLayout
        HorizontalOptions="Center"
        VerticalOptions="Center"
        Padding="20,0,20,0">
        <Label
            Text="Bus Ticket Booking"
            FontSize="Large"
            FontAttributes="Bold"
            HeightRequest="50"

```

```

HorizontalOptions="Center"
VerticalOptions="Center"/>
<Entry
Placeholder="From"
Text="{Binding FromText,Mode=TwoWay}"
HeightRequest="50"
Margin="0,10,0,10"/>
<Entry
Placeholder="To"
Text="{Binding ToText}"
HeightRequest="50"
Margin="0,10,0,10"/>
<buttons:SfSegmentedControl
x:Name="segment"
HeightRequest="80"
VisibleSegmentsCount="2"
Color="White"
SelectionTextColor="#007CEE"
FontColor="#929292"
BorderColor="#929292"
SelectedIndex="1"
ItemsSource="{Binding ItemCollection,Mode=TwoWay}"
BackgroundColor="Transparent">
</buttons:SfSegmentedControl>
<buttons:SfSegmentedControl
SelectionTextColor="White"
HeightRequest="80"
VisibleSegmentsCount="6"
Color="Transparent"
BorderColor="#929292"
SelectedIndex="1"
FontColor="#929292"
BackgroundColor="Transparent" >
<segmentCollection:List x:TypeArguments="x:String">
<x:String>1</x:String>
<x:String>2</x:String>
<x:String>3</x:String>
<x:String>4</x:String>
<x:String>5</x:String>
<x:String>6</x:String>
</segmentCollection:List>
</buttons:SfSegmentedControl>
</StackLayout>
</ContentPage>

```

## C#

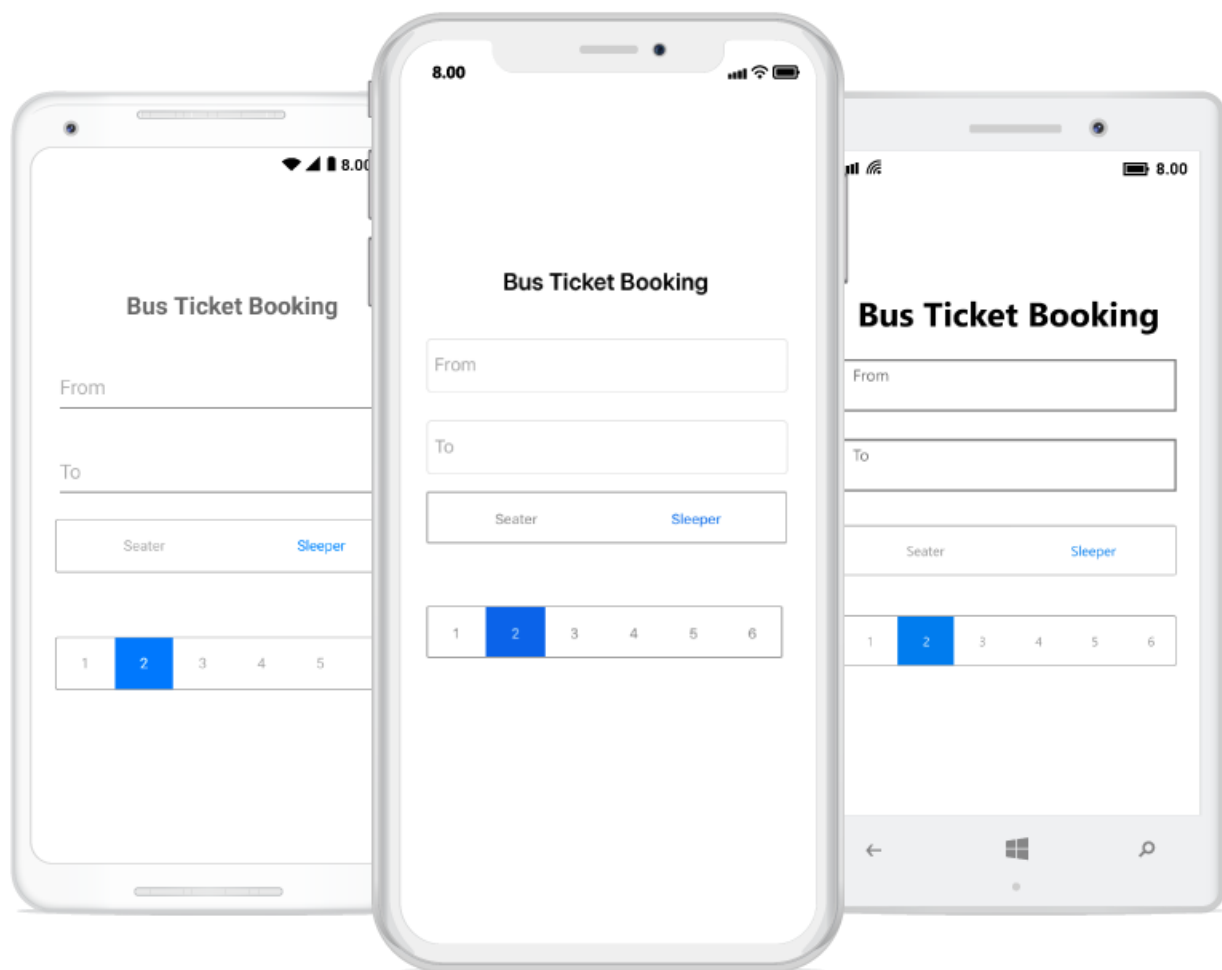
```

using Syncfusion.XForms.Buttons;
using System.Collections.Generic;
using Xamarin.Forms;
namespace SegmentGettingStarted
{
    {
        public partial class MainPage : ContentPage
        {
            SfSegmentedControl segmentedControl, segment;
            StackLayout stack;

```

```
ViewModel viewModel;
Label headerLabel;
Entry fromEntry, toEntry;
public MainPage()
{
    InitializeComponent();
    stack = new StackLayout();
    viewModel = new ViewModel();
    stack.HorizontalOptions = LayoutOptions.Center;
    stack.VerticalOptions = LayoutOptions.Center;
    stack.Padding = new Thickness(20, 0, 20, 0);
    //Label to denote the header part of application
    headerLabel = new Label();
    headerLabel.Text = "Bus Ticket Booking";
    headerLabel.FontAttributes = FontAttributes.Bold;
    headerLabel.HeightRequest = 50;
    headerLabel.FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label));
    headerLabel.HorizontalOptions = LayoutOptions.Center;
    headerLabel.VerticalOptions = LayoutOptions.Center;
    //Entry to enter the origin location
    fromEntry = new Entry();
    fromEntry.Placeholder = "From";
    fromEntry.Text = viewModel.FromText;
    fromEntry.HeightRequest = 50;
    fromEntry.Margin = new Thickness(0, 10, 0, 10);
    //Entry to enter the destination location
    toEntry = new Entry();
    toEntry.Placeholder = "To";
    toEntry.Text = viewModel.ToText;
    toEntry.HeightRequest = 50;
    toEntry.Margin = new Thickness(0, 10, 0, 10);
    //Adding item to the SegmentedControl as String
    segmentedControl = new SfSegmentedControl();
    segmentedControl.SelectionTextColor = Color.White;
    segmentedControl.HeightRequest = 80;
    segmentedControl.Color = Color.Transparent;
    segmentedControl.BorderColor = Color.FromHex("#929292");
    segmentedControl.FontColor = Color.FromHex("#929292");
    segmentedControl.SelectedIndex = 1;
    segmentedControl.BackgroundColor = Color.Transparent;
    segmentedControl.VisibleSegmentsCount = 6;
    segmentedControl.DisplayMode = SegmentDisplayMode.Text;
    List<string> list = new List<string>
    {
        "1", "2", "3", "4", "5", "6"
    };
    segmentedControl.ItemsSource = list;
    //Adding item to SegmentedControl as SegmentItem
    segment = new SfSegmentedControl();
    segment.SelectionTextColor = Color.FromHex("#007CEE");
    segment.VisibleSegmentsCount = 2;
    segment.Color = Color.White;
    segment.BorderColor = Color.FromHex("#929292");
    segment.SelectedIndex = 1;
    segment.FontColor = Color.FromHex("#929292");
    segment.BackgroundColor = Color.Transparent;
    segment.BindingContext = viewModel;
```

```
segment.ItemsSource = viewModel.ItemCollection;
SelectionIndicatorSettings select = new SelectionIndicatorSettings();
select.Color = Color.White;
segment.SelectionIndicatorSettings = select;
stack.Children.Add(headerLabel);
stack.Children.Add(fromEntry);
stack.Children.Add(toEntry);
stack.Children.Add(segment);
stack.Children.Add(segmentedControl);
this.Content = stack;
}
}
}
```



#### *Adding data as Custom View.*

We can add any custom view to the segmented control

In ViewModel add the below given code to get the respective items/collection in SfSegmentedControl.

#### **C#**

```
using Syncfusion.XForms.Buttons;
```

```

using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Xamarin.Forms;
namespace SegmentGettingStarted
{
    class ViewModel : INotifyPropertyChanged
    {
        private string fromText="";
        public string FromText
        {
            get { return fromText; }
            set { fromText = value; NotifyPropertyChanged("FromText"); }
        }
        private string toText = "";
        public string ToText
        {
            get { return toText; }
            set { toText = value; NotifyPropertyChanged("ToText"); }
        }
        public ViewModel()
        {
            itemCollection = new ObservableCollection<SfSegmentItem>
            {
                new SfSegmentItem() { Text = "Seater"},
                new SfSegmentItem() { Text = "Sleeper"},
            };
            ViewCollection = new ObservableCollection<View>
            {
                ResetViewButton,
                GoViewButton
            };
        }
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged([CallerMemberName] String propertyName =
            "")
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }
        private ObservableCollection<SfSegmentItem> itemCollection = new
            ObservableCollection<SfSegmentItem>();
        private ObservableCollection<View> viewCollection = new
            ObservableCollection<View>();
        public ObservableCollection<SfSegmentItem> ItemCollection
        {
            get { return itemCollection; }
            set { itemCollection = value; }
        }
        public ObservableCollection<View> ViewCollection
        {
            get { return viewCollection; }
            set { viewCollection = value; }
        }
    }
}

```

```

private Button ResetViewButton = new Button
{
    Text = "Reset",
    TextColor = Color.FromHex("#979797"),
    BackgroundColor = Color.White,
    BorderColor = Color.FromHex("#979797"),
    BorderWidth = 1,
    CornerRadius = 4,
    HeightRequest = 50,
    VerticalOptions = LayoutOptions.Center
};
private Button GoViewButton = new Button
{
    Text = "Go",
    TextColor = Color.FromHex("#979797"),
    BackgroundColor = Color.White,
    BorderColor = Color.FromHex("#979797"),
    BorderWidth = 1,
    CornerRadius = 4,
    HeightRequest = 50,
    VerticalOptions = LayoutOptions.Center
};
}
}

```

## XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SegmentGettingStarted"
xmlns:buttons="clr-namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:segmentCollection="clr-namespace:System.Collections.Generic;assembly=netstandard"
x:Class="SegmentGettingStarted.MainPage">
<ContentPage.BindingContext>
<local:ViewModel x:Name="viewModel" />
</ContentPage.BindingContext>
<StackLayout
HorizontalOptions="Center"
VerticalOptions="Center"
Padding="20,0,20,0">
<Label
Text="Bus Ticket Booking"
FontSize="Large"
FontAttributes="Bold"
HeightRequest="50"
HorizontalOptions="Center"
VerticalOptions="Center"/>
<Entry
Placeholder="From"
Text="{Binding FromText,Mode=TwoWay}"
HeightRequest="50"
Margin="0,10,0,10"/>
<Entry

```



```

Placeholder="To"
Text="{Binding ToText}"
HeightRequest="50"
Margin="0,10,0,10"/>
<buttons:SfSegmentedControl
x:Name="segment"
HeightRequest="80"
VisibleSegmentsCount="2"
Color="White"
SelectionTextColor="#007CEE"
FontColor="#929292"
BorderColor="#929292"
SelectedIndex="1"
ItemsSource="{Binding ItemCollection,Mode=TwoWay}"
BackgroundColor="Transparent">
</buttons:SfSegmentedControl>
<buttons:SfSegmentedControl
SelectionTextColor="White"
HeightRequest="80"
VisibleSegmentsCount="6"
Color="Transparent"
BorderColor="#929292"
SelectedIndex="1"
FontColor="#929292"
BackgroundColor="Transparent" >
<segmentCollection:List x:TypeArguments="x:String">
<x:String>1</x:String>
<x:String>2</x:String>
<x:String>3</x:String>
<x:String>4</x:String>
<x:String>5</x:String>
<x:String>6</x:String>
</segmentCollection:List>
</buttons:SfSegmentedControl>
<buttons:SfSegmentedControl
BorderColor="Transparent"
HeightRequest="80"
HorizontalOptions="Center"
x:Name="segmentView"
VisibleSegmentsCount="2"
Color="Transparent"
ItemsSource="{Binding ViewCollection}"
SegmentPadding="30">
</buttons:SfSegmentedControl>
</StackLayout>
</ContentPage>

```

## C#

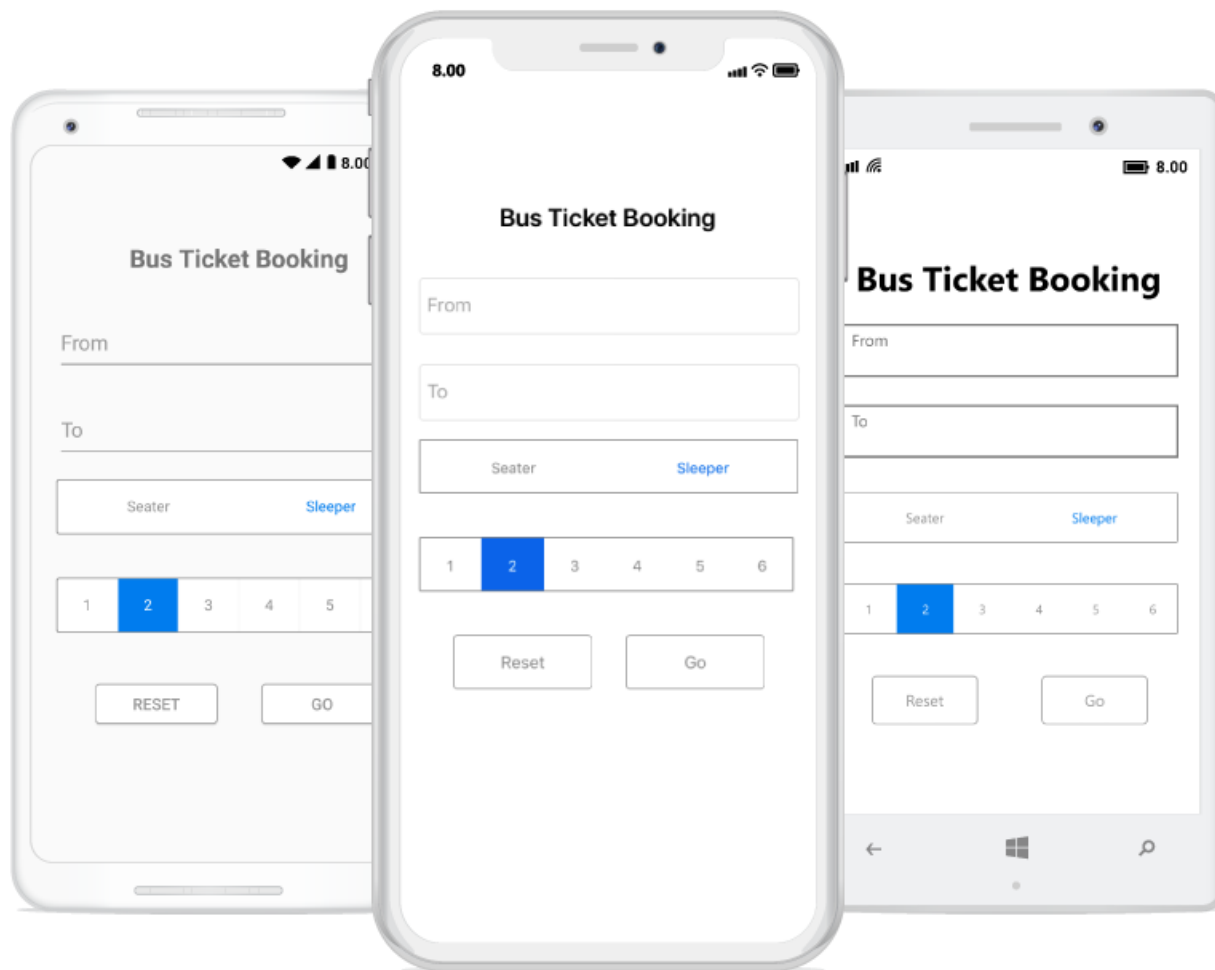
```

using Syncfusion.XForms.Buttons;
using System.Collections.Generic;
using Xamarin.Forms;
namespace SegmentGettingStarted
{
    public partial class MainPage : ContentPage
    {

```

```
SfSegmentedControl segmentedControl, segment, segmentView;
StackLayout stack;
ViewModel viewModel;
Label headerLabel;
Entry fromEntry, toEntry;
public MainPage()
{
    InitializeComponent();
    stack = new StackLayout();
    viewModel = new ViewModel();
    stack.HorizontalOptions = LayoutOptions.Center;
    stack.VerticalOptions = LayoutOptions.Center;
    stack.Padding = new Thickness(20, 0, 20, 0);
    //Label to denote the header part of application
    headerLabel = new Label();
    headerLabel.Text = "Bus Ticket Booking";
    headerLabel.FontAttributes = FontAttributes.Bold;
    headerLabel.HeightRequest = 50;
    headerLabel.FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label));
    headerLabel.HorizontalOptions = LayoutOptions.Center;
    headerLabel.VerticalOptions = LayoutOptions.Center;
    //Entry to enter the origin location
    fromEntry = new Entry();
    fromEntry.Placeholder = "From";
    fromEntry.Text = viewModel.FromText;
    fromEntry.HeightRequest = 50;
    fromEntry.Margin = new Thickness(0, 10, 0, 10);
    //Entry to enter the destination location
    toEntry = new Entry();
    toEntry.Placeholder = "To";
    toEntry.Text = viewModel.ToText;
    toEntry.HeightRequest = 50;
    toEntry.Margin = new Thickness(0, 10, 0, 10);
    //Adding item to the segmented control as String
    segmentedControl = new SfSegmentedControl();
    segmentedControl.SelectionTextColor = Color.White;
    segmentedControl.HeightRequest = 80;
    segmentedControl.Color = Color.Transparent;
    segmentedControl.BorderColor = Color.FromHex("#929292");
    segmentedControl.FontColor = Color.FromHex("#929292");
    segmentedControl.SelectedIndex = 1;
    segmentedControl.BackgroundColor = Color.Transparent;
    segmentedControl.VisibleSegmentsCount = 6;
    segmentedControl.DisplayMode = SegmentDisplayMode.Text;
    List<string> list = new List<string>
    {
        "1", "2", "3", "4", "5", "6"
    };
    segmentedControl.ItemsSource = list;
    //Adding item to segmented control as SegmentItem
    segment = new SfSegmentedControl();
    segment.SelectionTextColor = Color.FromHex("#007CEE");
    segment.VisibleSegmentsCount = 2;
    segment.Color = Color.White;
    segment.BorderColor = Color.FromHex("#929292");
    segment.SelectedIndex = 1;
    segment.FontColor = Color.FromHex("#929292");
```

```
segment.BackgroundColor = Color.Transparent;
segment.BindingContext = viewModel;
segment.ItemsSource = viewModel.ItemCollection;
SelectionIndicatorSettings select = new SelectionIndicatorSettings();
select.Color = Color.White;
segment.SelectionIndicatorSettings = select;
//Adding item to the segmented control as View
segmentView = new SfSegmentedControl();
segmentView.BindingContext = viewModel;
segmentView.ItemsSource = viewModel.ViewCollection;
segmentView.BorderColor = Color.Transparent;
segmentView.HeightRequest = 80;
segmentView.HorizontalOptions = LayoutOptions.Center;
segmentView.VisibleSegmentsCount = 2;
segmentView.Color = Color.Transparent;
segmentView.SegmentPadding = 30;
SelectionIndicatorSettings selectionIndicator = new
SelectionIndicatorSettings();
selectionIndicator.Color = Color.Transparent;
selectionIndicator.Position = SelectionIndicatorPosition.Fill;
selectionIndicator.StrokeThickness = 10;
segmentView.SelectionIndicatorSettings = selectionIndicator;
stack.Children.Add(headerLabel);
stack.Children.Add(fromEntry);
stack.Children.Add(toEntry);
stack.Children.Add(segment);
stack.Children.Add(segmentedControl);
stack.Children.Add(segmentView);
this.Content = stack;
}
}
}
```



### Customizing segmented control appearance

*Share space equally to all the items.*

To share the Item space equally to segmented control, set the number of segment item that has to be visible on the available screen width and that can be distributed in the available space through the `VisibleSegmentsCount` property of `SfSegmentedControl`.

#### XML

```
<buttons:SfSegmentedControl VisibleSegmentsCount="5">
```

#### C#

```
SegmentedControl.VisibleSegmentsCount = 5;
```

### Display Mode

We can change the appearance of the segmented control by using the `DisplayMode` property of `SfSegmentedControl`. We can set the `DisplayMode` to either Image or Text or ImageWithText.

#### XML

```
<buttons:SfSegmentedControl DisplayMode="Text"/>
```

**C#**

```
SegmentedControl.DisplayMode = SegmentDisplayMode.Text;
```

**Customizing selection indicator appearance**

The Selection indicator can be used to indicate the selected index of the segmented control. It can be customized with the built-in APIs that are available in the `SelectionIndicatorSettings` property of `SfSegmentedControl`.

To know more about customizing selection indicator refer this [feature link](#)

**Handle click events**

`SfSegmentedControl` has `SelectionChanged` event, using this we can perform operation based on our needs.

**XML**

```
<buttons:SfSegmentedControl
x:Name="SegmentedControl"
SelectionChanged="SegmentedControl_SelectionChanged" />
```

**C#**

```
SegmentedControl.SelectionChanged += SegmentedControl_SelectionChanged;
```

**C#**

```
private void SegmentedControl_SelectionChanged(object sender,
Syncfusion.XForms.Buttons.SelectionChangedEventArgs e)
{
    var currentValue = e.Index; //To get the value of current selected index.
}
```

**Note:** For custom view user need to handle click event manually for the view which have been used.

eg. For Button we have to use its "Click" event.

The below given code can be included on the Custom view viewModel to get the click event output.

**C#**

```
public ViewModel()
{
    ResetViewButton.Clicked += ResetViewButton_Clicked;
    GoViewButton.Clicked += GoViewButton_Clicked;
}
private void GoViewButton_Clicked(object sender, EventArgs e)
{
    if (FromText == "")
    {
        SendMessgae("Please enter your origin city.");
    }
    else if (ToText == "")
```

```

{
    SendMessage("Please enter your destination city.");
}
else
    SendMessage("Your ticket has been booked.");
}
private void ResetViewButton_Clicked(object sender, EventArgs e)
{
    FromText = "";
    ToText = "";
    Application.Current.MainPage.DisplayAlert("Status", "Fields has been
    refreshed", null, "Ok");
}
internal void SendMessage(string message)
{
    Application.Current.MainPage.DisplayAlert(message, null, "Ok");
}

```

Note: Getting started sample can be downloaded from this [link](#)

### Populating data source

The segmented control can be populated from a collection of strings, views, or a collection of objects in a built-in class.

#### String collection

The segmented control provides the collection of strings as a data source.

#### XML

```

Namespace:
xmlns:sys="clr-namespace:System.Collections.Generic;assembly=netstandard"
...
<ContentPage.Content>
<buttons:SfSegmentedControl
VisibleSegmentsCount="3"
SelectedIndex="2"
BorderColor="#3F3F3F"
FontColor="Black"
Color="Transparent"
SelectionTextColor="#02A0AE">
<sys:List x:TypeArguments="x:String">
<x:String>Formals</x:String>
<x:String>Casuals</x:String>
<x:String>Trendy</x:String>
</sys:List>
</buttons:SfSegmentedControl>
</ContentPage.Content>

```

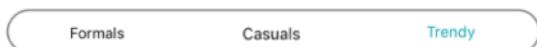
#### C#

```

SfSegmentedControl segmentedControl = new SfSegmentedControl();
List<string> clothList = new List<string>
{
    "Formals", "Casuals", "Trendy"
};
segmentedControl.ItemsSource = clothList;

```

```
segmentedControl.Color = Color.Transparent;
segmentedControl.VisibleSegmentsCount = 3;
segmentedControl.SelectedIndex = 2;
segmentedControl.BorderColor = Color.FromHex("#3F3F3F");
segmentedControl.FontColor = Color.Black;
segmentedControl.SelectionTextColor = Color.FromHex("#02A0AE");
this.Content = segmentedControl;
```



### Segment items

The segmented control customizes the text or icons, or use other built-in customization options available for the segments. Segment item collections can also be used.

The items inside the ItemsSource can be added in the code behind as below.

### C#

```
public class ViewModel
{
    public ObservableCollection<SfSegmentItem> SegmentItems { get; set; }
    public ViewModel()
    {
        SegmentItems = new ObservableCollection<SfSegmentItem>
        {
            new SfSegmentItem() { Text="XS", FontColor=Color.FromHex("#3F3F3F") },
            new SfSegmentItem() { Text="S", FontColor=Color.FromHex("#3F3F3F") },
            new SfSegmentItem() { Text="M", FontColor=Color.FromHex("#3F3F3F") },
            new SfSegmentItem() { Text="L", FontColor=Color.FromHex("#3F3F3F") },
            new SfSegmentItem() { Text="XL", FontColor=Color.FromHex("#3F3F3F") },
        };
    }
}
```

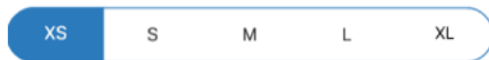
### XML

```
<ContentPage.Content>
<buttons:SfSegmentedControl
    CornerRadius="20"
    BorderColor="#2C7BBC"
    SelectionTextColor="White"
    Color="White"
    FontColor="#3F3F3F"
    VisibleSegmentsCount="5"
    ItemsSource="{Binding SegmentItems}">
    <buttons:SfSegmentedControl.BindingContext>
    <local:ViewModel/>
    </buttons:SfSegmentedControl.BindingContext>
</buttons:SfSegmentedControl>
```

```
</ContentPage.Content>
```

**C#**

```
SfSegmentedControl segmentedControl = new SfSegmentedControl();
segmentedControl.BindingContext = new ViewModel();
segmentedControl.SetBinding(SfSegmentedControl.ItemsSourceProperty,
"SegmentItems");
segmentedControl.CornerRadius = 20;
segmentedControl.Color = Color.White;
segmentedControl.VisibleSegmentsCount = 5;
segmentedControl.BorderColor = Color.FromHex("#2C7BBC");
segmentedControl.FontColor = Color.FromHex("#3F3F3F");
segmentedControl.SelectionTextColor = Color.White;
this.Content = segmentedControl;
```



## Custom views

Custom views or images can be added as segments in the segmented control.

The items inside the ItemsSource can be added in the code behind as below.

**C#**

```
public class ViewModel
{
    public ObservableCollection<View> ViewItems { get; set; }
    private Button ResetViewButton = new Button
    {
        Text = "Reset",
        TextColor = Color.FromHex("#979797"),
        BackgroundColor = Color.White,
        BorderColor = Color.FromHex("#979797"),
        BorderWidth = 1,
        HeightRequest = 50,
        VerticalOptions = LayoutOptions.Center
    };
    private Button GoViewButton = new Button
    {
        Text = "Go",
        TextColor = Color.FromHex("#979797"),
        BackgroundColor = Color.White,
        BorderColor = Color.FromHex("#979797"),
        BorderWidth = 1,
        HeightRequest = 50,
        VerticalOptions = LayoutOptions.Center
    };
    public ViewModel()
```



```
{
ViewItems = new ObservableCollection<View>
{
ResetViewButton,
GoViewButton
};
}
}
```

**XML**

```
<buttons:SfSegmentedControl
BorderColor="Transparent"
HeightRequest="80"
HorizontalOptions="Center"
x:Name="segmentedControl"
VisibleSegmentsCount="2"
Color="Transparent"
ItemsSource="{Binding ViewItems}"
SegmentPadding="30">
<buttons:SfSegmentedControl.BindingContext>
<local:ViewModel/>
</buttons:SfSegmentedControl.BindingContext>
<buttons:SfSegmentedControl.SelectionIndicatorSettings>
<buttons:SelectionIndicatorSettings Color="Transparent"/>
</buttons:SfSegmentedControl.SelectionIndicatorSettings>
</buttons:SfSegmentedControl>
```

**C#**

```
SfSegmentedControl segmentedControl = new SfSegmentedControl();
segmentedControl.BindingContext = new ViewModel();
segmentedControl.SetBinding(SfSegmentedControl.ItemsSourceProperty,
"ViewItems");
segmentedControl.BorderColor = Color.Transparent;
segmentedControl.HeightRequest = 80;
segmentedControl.HorizontalOptions = LayoutOptions.Center;
segmentedControl.VisibleSegmentsCount = 2;
segmentedControl.Color = Color.Transparent;
segmentedControl.SegmentPadding = 30;
segmentedControl.SelectionIndicatorSettings = new
SelectionIndicatorSettings()
{
Color = Color.Transparent,
Position = SelectionIndicatorPosition.Fill,
StrokeThickness = 10
};
```



## Selection changed

The selection changed event occurs when there is a change from one segment item to another in the segmented control. It can be handled by two ways.

### User interface

When users navigate from one item to another, selection is changed, so that the `SelectedIndex` value is updated to the new index of the item. The segmented control provides the `SelectionChanged` event, which is triggered when the selection is changed with the `SelectionChangedEventArgs`.

**Index** - Gets the current index value of the selected item.

### XML

```
<buttons:SfSegmentedControl x:Name = "segmentedControl"
    SelectionChanged="Handle_SelectionChanged"/>
```

### C#

```
segmentedControl.SelectionChanged += Handle_SelectionChanged;
void Handle_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    segmentedControl.BorderColor = UIColor.Red;
}
```

## Selected Index through programmatically.

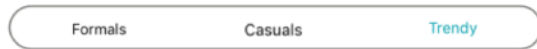
Users can set the default value programmatically for the selection to be placed. The selection is updated based on the index value given for the `SelectedIndex`.

### XML

```
<buttons:SfSegmentedControl SelectedIndex="2"/>
```

### C#

```
segmentedControl.SelectedIndex = 2;
```



## Display mode

Depending on application, different scenarios may require for effective communication. The segmented control supports these three options: icons, text, or a combination of both.

### Text

Items populated in the segmented control will be displayed as text by default.

Add the following namespace for loading data collection in string.

### XML

```
xmlns:segmentCollection="clr-
namespace:System.Collections.Generic;assembly=netstandard"
```

### XML

```
<buttons:SfSegmentedControl
x:Name="segmentedControl"
Margin="10,0"
CornerRadius="15"
SegmentHeight="50"
BorderColor="Transparent"
SelectedIndex="1"
Color="#048EAC"
FontSize="22"
DisplayMode="Text"
VisibleSegmentsCount="3"
FontColor="FFFFFF"
SelectionTextColor="#048EAC"
VerticalOptions="Center"
HorizontalOptions="Center">
<segmentCollection:List x:TypeArguments="x:String">
<x:String>Day</x:String>
<x:String>Week</x:String>
<x:String>Month</x:String>
</segmentCollection:List>
</buttons:SfSegmentedControl>
```

### C#

```
public partial class SegmentedControlSample : ContentPage
{
    SfSegmentedControl segmentedControl;
    public SegmentedControlSample()
    {
        InitializeComponent();
        segmentedControl = new SfSegmentedControl();
    }
}
```

```

List<String> periodsList = new List<String>
{
    "Day", "Week", "Month"
};
segmentedControl.ItemsSource = periodsList;
segmentedControl.DisplayMode = SegmentDisplayMode.Text;
segmentedControl.Color = Color.FromHex("#048EAC");
segmentedControl.SegmentHeight= 50;
segmentedControl.VisibleSegmentsCount = 3;
segmentedControl.CornerRadius = 15;
segmentedControl.HeightRequest = 50;
segmentedControl.SelectedIndex = 1;
segmentedControl.BorderColor = Color.Transparent;
segmentedControl.FontColor = Color.FromHex("#FFFFFF");
segmentedControl.FontSize = 22;
segmentedControl.VerticalOptions = LayoutOptions.Center;
segmentedControl.HorizontalOptions = LayoutOptions.Center;
segmentedControl.SelectionTextColor = Color.FromHex("#048EAC");
this.Content = segmentedControl;
}
}

```



### Image

Items populated in the segmented control will be displayed as icons.

The data source of the segmented control can be set as follows.

### C#

```

public class ViewModel : INotifyPropertyChanged
{
    private ObservableCollection<SfSegmentItem> imageCollection = new
    ObservableCollection<SfSegmentItem>();
    public ObservableCollection<SfSegmentItem> ImageCollection
    {
        get { return imageCollection; }
        set { imageCollection = value; }
    }
    public ViewModel()
    {
        ImageCollection = new ObservableCollection<SfSegmentItem>
        {
            new SfSegmentItem() { IconFont = "6", FontIconFontColor=Color.FromHex("#FFFFFF"), FontColor=Color.FromHex("#FFFFFF") },
            new SfSegmentItem() { IconFont = "6", FontIconFontColor=Color.FromHex("#FFFFFF"), FontColor=Color.FromHex("#FFFFFF") },
            new SfSegmentItem() { IconFont = "6", FontIconFontColor=Color.FromHex("#FFFFFF"), FontColor=Color.FromHex("#FFFFFF") },
        }
    }
}

```

```
};
}
}
```

**XML**

```
<buttons:SfSegmentedControl
x:Name="segmentedControl"
DisplayMode="Image"
SelectedIndex="1"
ItemsSource="{Binding ImageCollection}"
VisibleSegmentsCount="3"
SelectionTextColor="#FFFFFF"
/>
```

**C#**

```
public partial class SegmentedControlSample : ContentPage
{
    private ViewModel viewModel = new ViewModel();
    SfSegmentedControl segmentedControl;
    public SegmentedControlSample()
    {
        segmentedControl = new SfSegmentedControl();
        segmentedControl.BindingContext = viewModel;
        segmentedControl.ItemsSource = viewModel.ImageCollection;
        segmentedControl.DisplayMode = SegmentDisplayMode.Image;
        segmentedControl.SelectedIndex = 1;
        segmentedControl.VisibleSegmentsCount = 3;
        segmentedControl.SelectionTextColor = Color.FromHex("#FFFFFF");
        this.Content = segmentedControl;
    }
}
```

**Image with text**

Items populated in the segmented control will be displayed as icons with text.

The data source of the segmented control can be set as follows.

**C#**

```
public class ViewModel : INotifyPropertyChanged
{
    private ObservableCollection<SfSegmentItem> imageTextCollection = new
    ObservableCollection<SfSegmentItem>();
    public ObservableCollection<SfSegmentItem> ImageTextCollection
```

```

{
    get { return imageTextCollection; }
    set { imageTextCollection = value; }
}
public ViewModel()
{
    Image_textCollection = new ObservableCollection<SfSegmentItem>
    {
        new SfSegmentItem() { IconFont = "6", FontIconFontColor=Color.FromHex("#FFFFFF"), FontColor=Color.FromHex("#FFFFFF"), Text = "Day"},
        new SfSegmentItem() { IconFont = "6", FontIconFontColor=Color.FromHex("#FFFFFF"), FontColor=Color.FromHex("#FFFFFF"), Text = "Week"},
        new SfSegmentItem() { IconFont = "6", FontIconFontColor=Color.FromHex("#FFFFFF"), FontColor=Color.FromHex("#FFFFFF"), Text = "Month"
    };
}
}
}

```

**XML**

```

<buttons:SfSegmentedControl
x:Name="segmentedControl"
Margin="10,0"
CornerRadius="15"
SelectedIndex="1"
SelectedIndex="1"
ItemsSource = "{Binding ImageTextCollection}"
DisplayMode="ImageWithText"
VisibleSegmentsCount="3"
FontIconFontFamily = "segment.ttf"
/>

```

**C#**

```

public partial class SegmentedControlSample : ContentPage
{
    private ViewModel viewModel = new ViewModel();
    private SfSegmentedControl segmentedControl;
    public SegmentedControlSample()
    {
        segmentedControl = new SfSegmentedControl();
        segmentedControl.BindingContext = viewModel;
        segmentedControl.ItemsSource = viewModel.ImageTextCollection;
        segmentedControl.DisplayMode = SegmentDisplayMode.ImageWithText;
        segmentedControl.SelectedIndex = 1;
        segmentedControl.VisibleSegmentsCount = 3;
        segmentedControl.SelectionTextColor = Color.FromHex("#FFFFFF");
        segmentedControl.FontIconFontFamily = "segment.ttf";
        this.Content = segmentedControl;
    }
}

```



### How to set the font icons using ttf file?

You can refer this [link](#) for getting the font icons. Add the font file to your application by using the following steps for each platform:

#### Adding font file for iOS

1. Add the font family inside **Resource** folder iOS project.
2. Add the font file with the following build action: **BundleResource**.
3. Update the **Info.plist** file (fonts that are provided by application, UIAppFonts, or key).

#### Adding font file for Android

Add the font file to the **Assets** folder in the application project, and set the following build action: **AndroidAsset**.

#### Adding font file for UWP

Add the font family inside the application project of UWP.

**Note:** For iOS alone, FontFamily property is declared without succeeding with .ttf and for android and UWP platform font family name is defined followed by .ttf.

#### XML

```
FontIconFontFamily = "segment"
```

#### C#

```
segmentedControl.FontIconFontFamily = "segment";
```

### Indicating the selected item

The segmented control indicates the selected item by differentiating it with text color of the item or using selection strip.

#### Selection text color

You can change the text color of the selected item to desired color. The selected item's text color can be customized using the **SelectionTextColor** property.

#### XML

```
<buttons:SfSegmentedControl SelectionTextColor="#02A0AE"/>
```

#### C#

```
segmentedControl.SelectionTextColor = Color.FromHex("#02A0AE");
```



### Selection strip

A selection strip is used to indicate the selected item in the segmented control. The selection strip can be customized in many forms.

#### Position

The position of the selection indicator can be customized in different ways.

#### Top

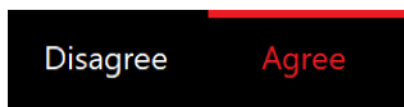
The selection strip can be displayed as a line with customizable color and thickness. It can be positioned at the top of an item.

#### XML

```
<buttons:SfSegmentedControl.SelectionIndicatorSettings>  
<buttons:SelectionIndicatorSettings  
Position="Top">  
</buttons:SelectionIndicatorSettings>  
</buttons:SfSegmentedControl.SelectionIndicatorSettings>
```

#### C#

```
SelectionIndicatorSettings selectionIndicator = new  
SelectionIndicatorSettings();  
selectionIndicator.Position = Position.Top;  
segmentedControl.SelectionIndicatorSettings = selectionIndicator;
```



#### Bottom

Like top placement, selection strip can be customized by its color and thickness and can be positioned at the bottom of an item.

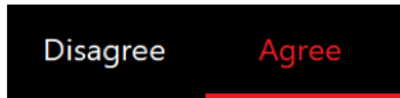
#### XML

```
<buttons:SfSegmentedControl.SelectionIndicatorSettings>  
<buttons:SelectionIndicatorSettings  
Position="Bottom">  
</buttons:SelectionIndicatorSettings>  
</buttons:SfSegmentedControl.SelectionIndicatorSettings>
```



### C#

```
SelectionIndicatorSettings selectionIndicator = new  
SelectionIndicatorSettings();  
selectionIndicator.Position = Position.Bottom;  
segmentedControl.SelectionIndicatorSettings = selectionIndicator;
```



### Fill

The selection strip can be placed over a segment item to indicate the selection. You can customize its color to highlight the item.

### XML

```
<buttons:SfSegmentedControl.SelectionIndicatorSettings>  
<buttons:SelectionIndicatorSettings  
Position="Fill">  
</buttons:SelectionIndicatorSettings>  
</buttons:SfSegmentedControl.SelectionIndicatorSettings>
```

### C#

```
SelectionIndicatorSettings selectionIndicator = new  
SelectionIndicatorSettings();  
selectionIndicator.Position = Position.Fill;  
segmentedControl.SelectionIndicatorSettings = selectionIndicator;
```



### Border

The selection strip can be set as a border to highlight the selected item.

### XML

```
<buttons:SfSegmentedControl.SelectionIndicatorSettings>  
<buttons:SelectionIndicatorSettings  
Position="Border">  
</buttons:SelectionIndicatorSettings>  
</buttons:SfSegmentedControl.SelectionIndicatorSettings>
```

## C#

```
SelectionIndicatorSettings selectionIndicator = new  
SelectionIndicatorSettings();  
selectionIndicator.Position = Position.Border;  
segmentedControl.SelectionIndicatorSettings = selectionIndicator;
```



## Color

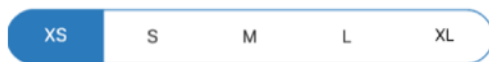
The background color of the selection strip can be customized using the **Color** property of **SelectionIndicatorSettings**.

## XML

```
<buttons:SfSegmentedControl.SelectionIndicatorSettings>  
<buttons:SelectionIndicatorSettings  
Color="#2C7BBC">  
</buttons:SelectionIndicatorSettings>  
</buttons:SfSegmentedControl.SelectionIndicatorSettings>
```

## C#

```
SelectionIndicatorSettings selectionIndicator = new  
SelectionIndicatorSettings();  
selectionIndicator.Color = Color.FromHex("#2C7BBC");  
segmentedControl.SelectionIndicatorSettings = selectionIndicator;
```



## Thickness

The border thickness of the selection strip can be customized using the **Thickness** property of **SelectionIndicatorSettings**.

## XML

```
<buttons:SfSegmentedControl.SelectionIndicatorSettings>  
<buttons:SelectionIndicatorSettings  
StrokeThickness="10">  
</buttons:SelectionIndicatorSettings>
```

```
</buttons:SfSegmentedControl.SelectionIndicatorSettings>
```

### C#

```
SelectionIndicatorSettings selectionIndicator = new
SelectionIndicatorSettings();
selectionIndicator.StrokeThickness = 10;
segmentedControl.SelectionIndicatorSettings = selectionIndicator;
```

### Handling multiple segments

The segmented control handles segmented items with space distributed for the items in two ways. When the available space in the segmented control is not equally distributed, the items beyond the edges of the control can be viewed by scrolling the panel.

### Visible segment counts

The segmented control displays items for view based on the count that is given for

**VisibleSegmentCount**.

### XML

```
<buttons:SfSegmentedControl
SegmentHeight="20"
VisibleSegmentsCount="4"
SelectedIndex="2"
FontColor="Black"
Color="Transparent"
DisplayMode = "Text"
SelectionTextColor="#02A0AE">
<segmentCollection:List x:TypeArguments="x:String">
<x:String>Item1</x:String>
<x:String>Item2</x:String>
<x:String>Item3</x:String>
<x:String>Item4</x:String>
<x:String>Item5</x:String>
<x:String>Item6</x:String>
<x:String>Item7</x:String>
<x:String>Item8</x:String>
<x:String>Item9</x:String>
</segmentCollection:List>
</buttons:SfSegmentedControl>
```

### C#

```
public partial class SegmentedControlSample : ContentPage
{
    SfSegmentedControl segmentedControl;
    public SegmentedControlSample()
    {
        InitializeComponent();
        segmentedControl = new SfSegmentedControl();
        List<String> itemsList = new List<String>
        {
            "Item1", "Item2", "Item3", "Item4", "Item5", "Item6", "Item7", "Item8", "Item9"
        };
    }
}
```

```
segmentedControl.ItemsSource = itemsList;  
segmentedControl.Color = Color.Transparent;  
segmentedControl.VisibleSegmentsCount = 4;  
segmentedControl.SelectedIndex = 2;  
segmentedControl.FontColor = Color.Black;  
segmentedControl.SelectionTextColor = Color.FromHex("#02A0AE");  
this.Content = segmentedControl;  
}  
}
```



### Segment width

Users can use the `SegmentWidth` property to display the segmented items within the given width instead `VisibleSegmentCount`.

#### XML

```
<buttons:SfSegmentedControl SegmentWidth = "80"/>
```

#### C#

```
segmentedControl.SegmentWidth = 80;
```

### Customization

The segmented control supports customizing segment color, text color, icon size, selection color, and more. This control also supports enabling the segments to fit your application's theme. It can be customized in the following areas.

#### Text appearance

The text inside the segmented control can be customized by its font size, color, and its font family.

#### Font family

You can customize the font family of the segmented item using the `FontFamily` property.

#### XML

```
<buttons:SfSegmentedControl>  
  <buttons:SfSegmentedControl.FontFamily>  
  <OnPlatform x:TypeArguments="x:String">  
    <On Platform="iOS" Value="Helvetica" />  
    <On Platform="Android" Value="Roboto" />  
    <On Platform="UWP" Value="Helvetica" />  
  </OnPlatform>  
</FontFamily>  
</SfSegmentedControl>
```

```
</OnPlatform>
</buttons:SfSegmentedControl.FontFamily>
</buttons:SfSegmentedControl >
```

### C#

```
segmentedControl.FontFamily = Device.RuntimePlatform == Device.iOS ?
"Helvetica" :
Device.RuntimePlatform == Device.Android ? "Roboto" : "Helvetica";
```

#### Font color

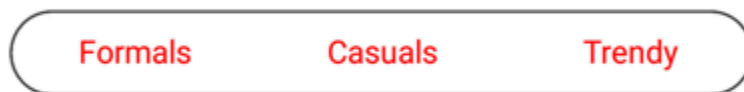
You can customize the text color of the segmented item using the **FontColor** property.

### XML

```
<buttons:SfSegmentedControl FontColor="Red" />
```

### C#

```
segmentedControl.SelectionTextColor = Color.Red;
```



#### Font size

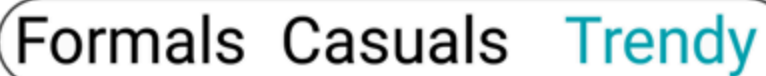
You can change the text size of the segmented item using the **FontSize** property.

### XML

```
<buttons:SfSegmentedControl FontSize="20" />
```

### C#

```
segmentedControl.FontSize = 20;
```



### Border

You can customize the border by their color and thickness.

#### Border color

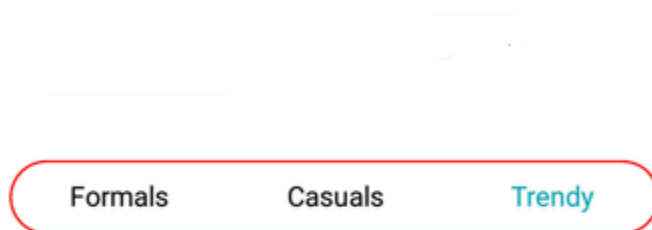
You can customize the border color of all the items in the segmented control.

#### XML

```
<buttons:SfSegmentedControl BorderColor="Red" />
```

#### C#

```
segmentedControl.BorderColor = Color.Red;
```



#### Border thickness

You can customize the width of the border using the `BorderThickness` property.

#### XML

```
<buttons:SfSegmentedControl BorderThickness="5" />
```

#### C#

```
segmentedControl.BorderThickness = 5;
```



### Padding

The segmented control handles padding between the items.

### Segment padding

Spacing between the segmented items in the control can be customized using the `SegmentPadding`.

### XML

```
<buttons:SfSegmentedControl SegmentPadding="15" />
```

### C#

```
segmentedControl.SegmentPadding = 15;
```



### Corner radius

The segmented control provides corner radius support for the segmented items and strip.

### Item radius

The segmented control customizes the corner radius for each segmented item.

### XML

```
<buttons:SfSegmentedControl SegmentCornerRadius="15" />
```

### C#

```
segmentedControl.SegmentCornerRadius = 15;
```



#### Selection strip radius

The segmented control customizes corner radius for selection strip using the `CornerRadius` inside the `SelectionIndicatorSetting` class.

#### XML

```
<buttons:SfSegmentedControl>
  <buttons:SfSegmentedControl.SelectionIndicatorSettings>
    <buttons:SelectionIndicatorSettings
      CornerRadius="15">
    </buttons:SelectionIndicatorSettings>
  </buttons:SfSegmentedControl.SelectionIndicatorSettings>
</buttons:SfSegmentedControl>
```

#### C#

```
SelectionIndicatorSettings selectionIndicator = new
SelectionIndicatorSettings();
selectionIndicator.CnerRadius = 15;
segmentedControl.SelectionIndicatorSettings = selectionIndicator;
```



#### Control radius

The segmented control also handles corner radius for border line of the whole control.

#### XML

```
<buttons:SfSegmentedControl CornerRadius="15" />
```

#### C#

```
segmentedControl.CnerRadius = 15;
```





### Color

The segmented control allows users to customize the background color of the segmented items and background color of the control.

### Item color

You can customize the background color of each segmented item using the `Color` property in the `SelectionIndicatorSettings` class.

### XML

```
<buttons:SfSegmentedControl.SelectionIndicatorSettings>  
<buttons:SelectionIndicatorSettings  
Color="#FF355088">  
</buttons:SelectionIndicatorSettings>  
</buttons:SfSegmentedControl.SelectionIndicatorSettings>
```

### C#

```
SelectionIndicatorSettings selectionIndicator = new  
SelectionIndicatorSettings();  
selectionIndicator.Color = Color.FromHex("#FF355088");  
segmentedControl.SelectionIndicatorSettings = selectionIndicator;
```



### Control color

You can customize the background color of the control by setting value for the `Color` property.

### XML

```
<buttons:SfSegmentedControl Color="#02A0AE" />
```

### C#

```
segmentedControl.Color = Color.FromHex("#02A0AE");
```

Formals

Casuals

Trendy

### Scrolling in segmented control programmatically

The SegmentedControl allows programmatic scrolling based on index and item using the `ScrollTo` methods mentioned as follows.

#### *ScrollTo(index, scrollToPosition)*

This method is used to scroll the segment item based on given index and `[ScrollToPosition]()` value.

#### **C#**

```
segmentedControl.ScrollTo(5,
    Syncfusion.XForms.Buttons.ScrollToPosition.Start);
```

#### *ScrollTo(item, scrollToPosition)*

This method is used to scroll the segment item based on the given data or `SfSegmentItem` and `[ScrollToPosition]()` value.

#### **C#**

```
segmentedControl.ScrollTo(viewModel.Items[5],
    Syncfusion.XForms.Buttons.ScrollToPosition.Start);
```

### AutomationId

The segmented control supports automating each segment item in control using the `AutomationId` property. The `AutomationId` value given for control will be appended with each segment item's text value, and it can be used for writing automation scripts. The following code snippet explains how to set `AutomationId` value to the segmented control.

#### **XML**

```
<syncfusion:SfSegmentedControl AutomationId="SegmentedControl1"
    ItemsSource="{Binding Segments}" />
```

#### **C#**

```
ViewModel viewModel = new ViewModel();
SfSegmentedControl sfSegmented = new SfSegmentedControl();
sfSegmented.AutomationId = "SegmentedControl1";
sfSegmented.ItemsSource = viewModel.Segments;
public class ViewModel
```

```
{
public ObservableCollection<SfSegmentItem> Segments { get; set; }
public ViewModel()
{
Segments = new ObservableCollection<SfSegmentItem>();
Segments.Add(new SfSegmentItem() { Text = "Max" });
Segments.Add(new SfSegmentItem() { Text = "Roger" });
Segments.Add(new SfSegmentItem() { Text = "Evans" });
Segments.Add(new SfSegmentItem() { Text = "John" });
Segments.Add(new SfSegmentItem() { Text = "Peter" });
Segments.Add(new SfSegmentItem() { Text = "Mike" });
}
}
```

The following table shows that the AutomationId value is set to each segment item in control for the above code snippet.

Segment item	AutomationId
Max	SegmentedControl1_Max
Roger	SegmentedControl1_Roger
Evans	SegmentedControl1_Evans
John	SegmentedControl1_John
Peter	SegmentedControl1_Peter
Mike	SegmentedControl1_Mike

## How to

### Clear the default selection in SfSegmentedControl

The SfSegmentedControl provides support to clear the default selection on segments by setting the value of **SelectedIndex** as negative or beyond the collection count.

**Note:** By default, it selects the 0th indexed item.

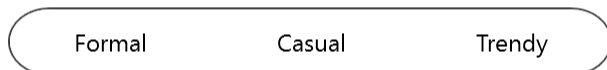
### XML

```
Namespace:
xmlns:sys="clr-namespace:System.Collections.Generic;assembly=netstandard"
...
<buttons:SfSegmentedControl
SelectedIndex="-1"
BorderColor="#3F3F3F"
FontColor="Black"
HorizontalOptions="Center"
VerticalOptions="Center"
SegmentHeight="32"
CornerRadius="20"
VisibleSegmentsCount="3">
<buttons:SfSegmentedControl.ItemsSource>
<sys:List x:TypeArguments="x:String">
```

```
<x:String>Formal</x:String>
<x:String>Casual</x:String>
<x:String>Trendy</x:String>
</sys:List>
</buttons:SfSegmentedControl.ItemsSource>
</buttons:SfSegmentedControl>
```

## C#

```
SfSegmentedControl segmentedControl = new SfSegmentedControl()
{
    SelectedIndex = -1,
    BorderColor = Color.FromHex("#3F3F3F"),
    FontColor = Color.Black,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    SegmentHeight = 32,
    CornerRadius = 20,
    VisibleSegmentsCount = 3,
};
segmentedControl.ItemsSource = new List<String>()
{
    "Formal", "Casual", "Trendy"
};
```



## Autoscrolling the selected segment item

Auto scrolling for selected item change can be enabled by setting the value of `[AutoScrollSelectedItem]()` property to true. You can set the scroll position of segment item using the `[ScrollToPosition]()` property. The default value for `[AutoScrollSelectedItem]()` is false, and the default value for `[ScrollToPosition]()` is `[MakeVisible]()`. The following options are available in `[ScrollToPosition]()`:

- `[MakeVisible]()` - Scrolls to the selected segment item to make it visible in the control. If the item is already visible, scrolling will not occur.
- `[Start]()` - Scrolls to the selected segment item at the start of the control.
- `[Center]()` - Scrolls to the selected segment item at the center of the control.
- `[End]()` - Scrolls to selected segment item at the end of the control.

## XML

```
<buttons:SfSegmentedControl x:Name = "segmentedControl" SelectedIndex="5"
AutoScrollSelectedItem="True" ScrollToPosition="Center"/>
```

## C#

```
segmentedControl.SelectedIndex = 5;  
segmentedControl.AutoScrollSelectedItem = true;  
segmentedControl.ScrollToPosition =  
Syncfusion.XForms.Buttons.ScrollToPosition.Center;
```

# SfShimmer

## Overview

The SfShimmer control can be used to improve the responsiveness of an application by showing a modern shimmer effect when data is being loaded in the background. This control comes with six built-in shimmer types.



## Key features

- Supports custom shimmer types.
- Provides shimmer effects from multiple directions.
- Supports shimmer color and shimmer wave color customization.

## Getting Started

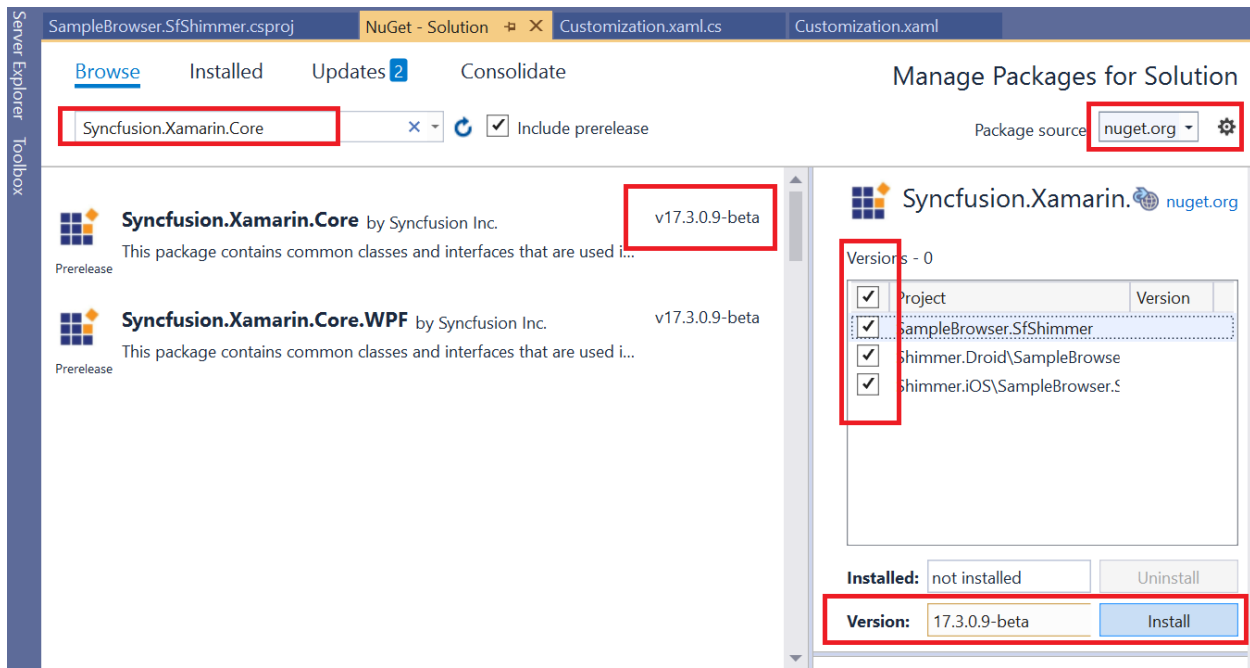
This section explains the steps required to configure the shimmer.

### Adding SfShimmer reference

You can add SfShimmer reference using one of the following methods:

#### Method 1: Adding SfShimmer reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.Core). To add SfShimmer to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Core](https://www.nuget.org/packages/Syncfusion.Xamarin.Core), and then install it.



**Note:** SfShimmer supports for Android and iOS.

### Method 2: Adding SfShimmer reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfShimmer control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfShimmer assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location : {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you will have to include a license key in your projects. Refer to [Syncfusion license key](#) to learn about registering Syncfusion license key in your Xamarin application to use our components.

### Launching an application on each platform with SfShimmer

To use the shimmer inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and are discussed in the following sections:

---

**Note:** If you are adding the references from toolbox, the following steps are not needed.

---

#### iOS

To launch the shimmer in iOS, call the `SfShimmerRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    SfShimmerRenderer.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### Android

Android platform does not require any additional configuration to render the shimmer control.

#### Initializing shimmer

Import the [SfShimmer](#) control namespace in respective page as demonstrated in the following code sample.

#### XML

```
xmlns:shimmer="clr-namespace:Syncfusion.XForms.Shimmer;assembly=Syncfusion.Core.XForms"
```

#### C#

```
using Syncfusion.XForms.Shimmer;
```

Then initialize shimmer as shown below,

#### XML

```
<shimmer:SfShimmer x:Name="shimmer" VerticalOptions="Fill"
IsLoaded="{Binding IsLoaded}">
<shimmer:SfShimmer.Content>
<StackLayout>
<Label Text="Content is loaded!" HorizontalOptions="CenterAndExpand"
VerticalOptions="CenterAndExpand"/>
</StackLayout>
</shimmer:SfShimmer.Content>
</shimmer:SfShimmer>
```

#### C#

```
SfShimmer shimmer = new SfShimmer();
shimmer.VerticalOptions = LayoutOptions.Fill;
shimmer.SetBinding(SfShimmer.IsLoadedProperty, "IsLoaded");
var stackLayout = new StackLayout();
var label = new Label();
label.Text = "Content is loaded!";
label.HorizontalOptions = LayoutOptions.CenterAndExpand;
label.VerticalOptions = LayoutOptions.CenterAndExpand;
stackLayout.Children.Add(label);
shimmer.Content = stackLayout;
```



**Note:** The SfShimmer has different shimmer types. The default shimmer type is **Persona**.

#### Loading shimmer content

By enabling the [IsLoaded](#) property of [SfShimmer](#), shimmer content is loaded.

#### XML

```
<shimmer:SfShimmer x:Name="shimmer" VerticalOptions="FillAndExpand" IsLoaded="true">
  <shimmer:SfShimmer.Content>
    <StackLayout>
      <Label Text="Content is loaded!"/>
    </StackLayout>
  </shimmer:SfShimmer.Content>
</shimmer:SfShimmer>
```

#### C#

```
shimmer = new SfShimmer()
{
    IsLoaded = true,
    VerticalOptions = LayoutOptions.FillAndExpand,
    Content = new Label
    {
        Text = "Content is loaded!"
    }
}
```



```
};  
this.Content = shimmer;
```

## Shimmer Types

### Built-in types

The following different built-in shimmer types are available in Shimmer:

- Persona
- Profile
- Article
- Video
- Feed
- Shopping

You can use the built-in shimmer types by setting the [Type](#) of [SfShimmer](#).

### XML

```
<shimmer:SfShimmer x:Name="shimmer" VerticalOptions="FillAndExpand"  
Type="Profile">  
  <shimmer:SfShimmer.Content>  
    <StackLayout>  
      <Label Text="Content is loaded!" HorizontalOptions="CenterAndExpand"  
VerticalOptions="CenterAndExpand"/>  
    </StackLayout>  
  </shimmer:SfShimmer.Content>  
</shimmer:SfShimmer>
```

### C#

```
shimmer = new SfShimmer()  
{  
    Type = ShimmerTypes.Profile,  
    VerticalOptions = LayoutOptions.FillAndExpand,  
    Content = new Label  
    {  
        Text = "Content is loaded!"  
    }  
};  
this.Content = shimmer;
```



### Custom view

You can customize the shimmer using your own view using the [CustomView](#) property of [SfShimmer](#).

### XML

```
<shimmer:SfShimmer.CustomView>
  <Grid
    x:Name="customView"
    Padding="10" ColumnSpacing="15" RowSpacing="15"
    HorizontalOptions="FillAndExpand"
    VerticalOptions="Center">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <border:SfBorder Grid.RowSpan="2"
      BackgroundColor="{Binding Color,Source={x:Reference shimmer}}"
      BorderColor="Transparent"
      CornerRadius="40"
      HeightRequest="70"
      VerticalOptions="Center"
      WidthRequest="70">
    </border:SfBorder>
    <border:SfBorder Grid.Row="0" Grid.Column="1"
      BackgroundColor="{Binding Color,Source={x:Reference shimmer}}"
      BorderColor="Transparent"
      HeightRequest="30"
      VerticalOptions="Center" />
    <border:SfBorder Grid.Row="1" Grid.Column="1"
      BackgroundColor="{Binding Color,Source={x:Reference shimmer}}"
      BorderColor="Transparent"
```

```
HeightRequest="30"  
VerticalOptions="Center" />  
</Grid>  
</shimmer:SfShimmer.CustomView>
```

## C#

```
var grid = new Grid  
{  
    Padding = 10,  
    ColumnSpacing = 15,  
    RowSpacing = 15,  
    HorizontalOptions = LayoutOptions.FillAndExpand,  
    VerticalOptions = LayoutOptions.Center,  
    ColumnDefinitions = new ColumnDefinitionCollection()  
    {  
        new ColumnDefinition { Width = GridLength.Auto },  
        new ColumnDefinition { Width = GridLength.Star }  
    },  
    RowDefinitions = new RowDefinitionCollection()  
    {  
        new RowDefinition { Height = GridLength.Auto },  
        new RowDefinition { Height = GridLength.Auto }  
    }  
};  
var imageBorder = new SfBorder  
{  
    BorderColor = Color.Transparent,  
    CornerRadius = 40,  
    HeightRequest = 70,  
    WidthRequest = 70,  
    VerticalOptions = LayoutOptions.Center  
};  
imageBorder.SetBinding(SfBorder.BackgroundColorProperty, new Binding {  
    Source = shimmer, Path = "Color" });  
var nameBorder = new SfBorder  
{  
    BorderColor = Color.Transparent,  
    HeightRequest = 30,  
    VerticalOptions = LayoutOptions.Center  
};  
nameBorder.SetBinding(SfBorder.BackgroundColorProperty, new Binding { Source  
= shimmer, Path = "Color" });  
var description = new SfBorder  
{  
    BorderColor = Color.Transparent,  
    HeightRequest = 30,  
    VerticalOptions = LayoutOptions.Center  
};  
description.SetBinding(SfBorder.BackgroundColorProperty, new Binding {  
    Source = shimmer, Path = "Color" });  
grid.Children.Add(imageBorder, 0, 0);  
Grid.SetRowSpan(imageBorder, 2);  
grid.Children.Add(nameBorder, 1, 0);  
grid.Children.Add(description, 1, 1);  
shimmer.CustomView = grid;
```



---

**Note:** Currently, [CustomView](#) will support only in Android.

---

### Customization of Shimmer

The Shimmer control supports options to customize the wave color, shimmer color, wave direction, wave animation duration, and more. This section explains how to customize the shimmer control.

#### WaveDirection

The [WaveDirection](#) property of [SfShimmer](#) is used to change the direction of shimmer wave. The following different wave directions are available in [SfShimmer](#):

- Default
- LeftToRight
- TopToBottom
- RightToLeft
- BottomToTop

#### XML

```
<shimmer:SfShimmer x:Name="shimmer" VerticalOptions="FillAndExpand"
WaveDirection="LeftToRight">
  <shimmer:SfShimmer.Content>
    <StackLayout>
      <Label Text="Content is loaded!"/>
    </StackLayout>
  </shimmer:SfShimmer.Content>
</shimmer:SfShimmer>
```

#### C#

```
shimmer = new SfShimmer()
{
    WaveDirection = WaveDirection.LeftToRight,
    VerticalOptions = LayoutOptions.FillAndExpand,
    Content = new Label
    {
        Text = "Content is loaded!"
    }
}
```

```

}
};
this.Content = shimmer;

```

### Color

The [Color](#) property of [SfShimmer](#) is used to customize the color of shimmer.

### XML

```

<shimmer:SfShimmer x:Name="shimmer" VerticalOptions="FillAndExpand"
Color="#F4E2EE">
<shimmer:SfShimmer.Content>
<StackLayout>
<Label Text="Content is loaded!"/>
</StackLayout>
</shimmer:SfShimmer.Content>
</shimmer:SfShimmer>

```

### C#

```

shimmer = new SfShimmer()
{
Color = Color.FromHex("#F4E2EE"),
VerticalOptions = LayoutOptions.FillAndExpand,
Content = new Label
{
Text = "Content is loaded!"
}
};
this.Content = shimmer;

```



### WaveColor

The [WaveColor](#) property of [SfShimmer](#) is used to customize the color of wave.

### XML

```

<shimmer:SfShimmer x:Name="shimmer" VerticalOptions="FillAndExpand"
WaveColor="#B8D4F2">
<shimmer:SfShimmer.Content>
<StackLayout>
<Label Text="Content is loaded!"/>

```

```

</StackLayout>
</shimmer:SfShimmer.Content>
</shimmer:SfShimmer>

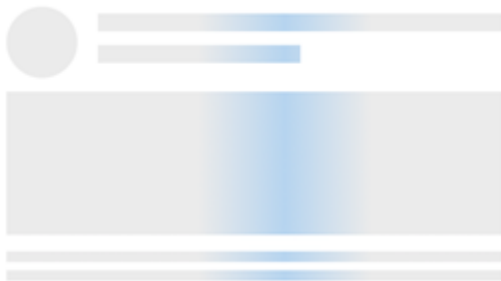
```

## C#

```

shimmer = new SfShimmer()
{
    WaveColor = Color.FromHex("#B8D4F2"),
    VerticalOptions = LayoutOptions.FillAndExpand,
    Content = new Label
    {
        Text = "Content is loaded!"
    }
};
this.Content = shimmer;

```



## WaveWidth

The [WaveWidth](#) property of [SfShimmer](#) is used to customize the width of wave.

## XML

```

<shimmer:SfShimmer x:Name="shimmer" VerticalOptions="FillAndExpand"
WaveColor="#B8D4F2" WaveWidth="100">
<shimmer:SfShimmer.Content>
<StackLayout>
<Label Text="Content is loaded!"/>
</StackLayout>
</shimmer:SfShimmer.Content>
</shimmer:SfShimmer>

```

## C#

```

shimmer = new SfShimmer()
{
    WaveWidth = 100,
    WaveColor = Color.FromHex("#B8D4F2"),
    VerticalOptions = LayoutOptions.FillAndExpand,
    Content = new Label
    {
        Text = "Content is loaded!"
    }
};
this.Content = shimmer;

```



### AnimationDuration

You can control the duration of wave animation using the [AnimationDuration](#) property of [SfShimmer](#). The default value of [AnimationDuration](#) is 1000 ms.

### XML

```
<shimmer:SfShimmer x:Name="shimmer" VerticalOptions="FillAndExpand"
AnimationDuration="2000">
  <shimmer:SfShimmer.Content>
    <StackLayout>
      <Label Text="Content is loaded!"/>
    </StackLayout>
  </shimmer:SfShimmer.Content>
</shimmer:SfShimmer>
```

### C#

```
shimmer = new SfShimmer()
{
    AnimationDuration = 2000,
    VerticalOptions = LayoutOptions.FillAndExpand,
    Content = new Label
    {
        Text = "Content is loaded!"
    }
};
this.Content = shimmer;
```

## SfSparkline

### Overview

Essential Sparkline for Xamarin.Forms is a very small chart, typically drawn without axes or coordinates. It presents the general shape of data's in a simple and highly condensed way.



### Key features

- Data Binding support
- Range Band support
- Supports 4 different types of sparkline.
- Marker support for line and area sparkline.
- Empty point support.

### Getting Started

This section explains you the steps required to populate the Sparkline with data.

#### Adding SfSparkline reference

You can add SfSparkline reference using one of the following methods:

##### Method 1: Adding SfSparkline reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfSparkline). To add SfSparkline to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfSparkline](https://www.nuget.org/packages/Syncfusion.Xamarin.SfSparkline), and then install it.

!{Adding SfSparkline reference from NuGet}{Getting-Started-image/Adding SfSparkline reference.png}

**Note:** Install the same version of SfSparkline NuGet in all the projects.

##### Method 2: Adding SfSparkline reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfSparkline control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

##### Method 3: Adding SfSparkline assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfSparkline.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfSparkline.Android.dll Syncfusion.SfSparkline.XForms.Android.dll



	Syncfusion.SfSparkline.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfSparkline.iOS.dll Syncfusion.SfSparkline.XForms.iOS.dll Syncfusion.SfSparkline.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfSparkline.UWP.dll Syncfusion.SfSparkline.XForms.UWP.dll Syncfusion.SfSparkline.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

#### Launching an application on each platform with SfSparkline.

To use the SfSparkline control inside an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch the SfSparkline in iOS, call the `SfSparklineRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework has been initialized and before the `LoadApplication` is called, as demonstrated in the following code example.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.SfSparkline.XForms.iOS.SfSparklineRenderer.Init ();
    LoadApplication (new App ());
    ...
}
```

### Universal Windows Platform (UWP)

You need to initialize the sparkline view assemblies in App.xaml.cs in UWP project as demonstrated in the following code samples. This is required to deploy the application with sparkline in Release mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    // Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.SfSparkline.XForms.UWP.SfSparkline
    Renderer).GetTypeInfo().Assembly);
    // Replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Android

The Android platform does not require any additional configuration to render the sparkline.

#### Initialize view model

Now, let us define a simple data model that represents a data point in [SfSparkline](#).

#### C#

```
public class Model
{
    public double Performance { get; set; }
}
```

Next, create a view model class and initialize a list of `Model` objects as shown below,

#### C#

```
public class ViewModel
{
    public List<Model> Data { get; set; }
    public ViewModel()
    {
        Data = new List<Model>()
        {
            new Model { Performance = 3000 },
            new Model { Performance = 5000 },
            new Model { Performance = -3000 },
            new Model { Performance = -4000 },
            new Model { Performance = 2000 },
            new Model { Performance = 3000 }
        };
    }
}
```

Set the `ViewModel` instance as the `BindingContext` of your Page; this is done to bind properties of `ViewModel` to [SfSparkline](#).

**Note:** Add namespace of `ViewModel` class in your XAML page if you prefer to set `BindingContext` in XAML.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="SparklineDemo.MainPage"
xmlns:sparkline="clr-
namespace:Syncfusion.SfSparkline.XForms;assembly=Syncfusion.SfSparkline.XFor
ms"
xmlns:local="clr-namespace:SparklineDemo">
<ContentPage.BindingContext>
<local:ViewModel></local:ViewModel>
</ContentPage.BindingContext>
</ContentPage>
```

### C#

```
this.BindingContext = new ViewModel();
```

Populate Sparkline with data

Import the [SfSparkline](#) namespace as shown below in your respective page,

### XML

```
xmlns:sparkline="clr-
namespace:Syncfusion.SfSparkline.XForms;assembly=Syncfusion.SfSparkline.XFor
ms"
```

### C#

```
using Syncfusion.SfSparkline.XForms;
```

Bind the Data property of the above `ViewModel` to the [SfSparkline.ItemsSource](#) property as shown below.

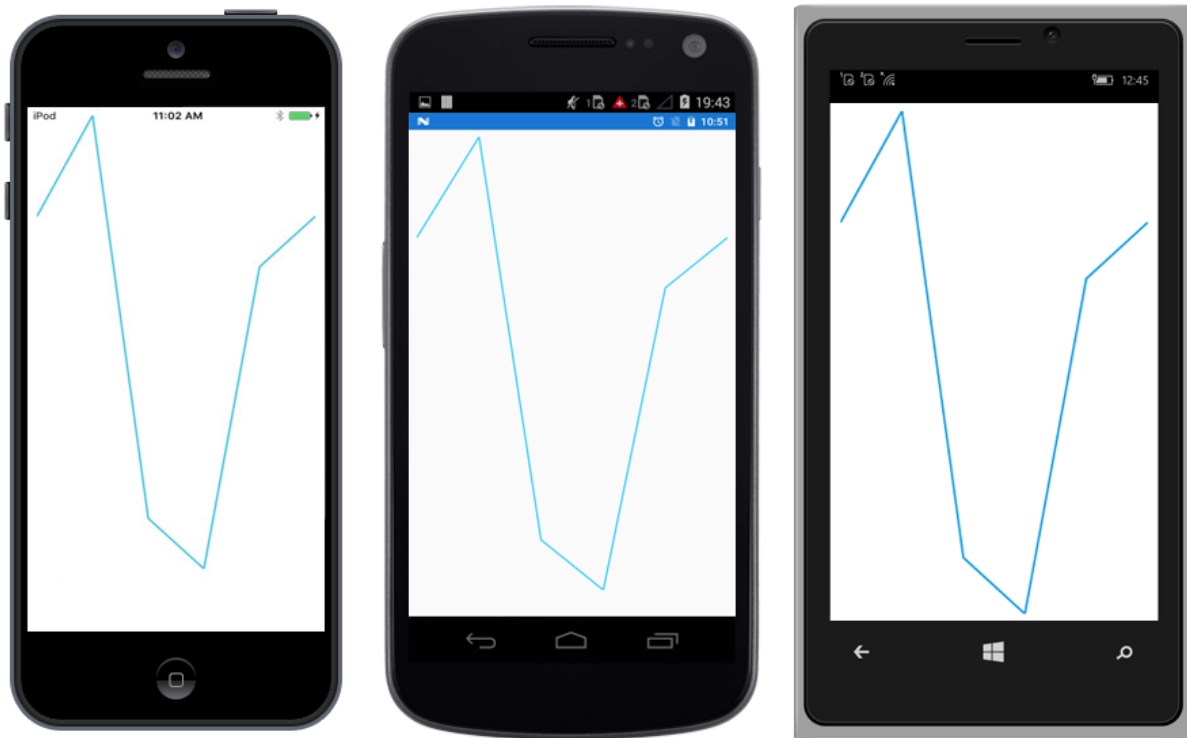
**Note:** You need to set `YBindingPath` property, so that [SfSparkline](#) would fetch values from the respective property in the data model to plot the Sparkline.

### XML

```
<sparkline:SfLineSparkline ItemsSource="{Binding Data}"
YBindingPath="Performance">
</sparkline:SfLineSparkline >
```

### C#

```
SfLineSparkline lineSparkline = new SfLineSparkline();
lineSparkline.YBindingPath = "Performance";
lineSparkline.SetBinding(SfSparklineBase.ItemsSourceProperty, "Data");
```



You can find the complete getting started sample from this [link](#).

## Sparkline Types

### Line Sparkline

[SfLineSparkline](#) is used for identifying patterns and trends in the data such as seasonal effects, large changes and turning points over a period of time.

- [StrokeWidth](#) - used to change the stroke width of the sparkline.
- [StrokeColor](#) - used to change the stroke color of the sparkline.
- [MinimumYValue](#) - used to set the minimum value of the y-axis in sparkline.
- [MaximumYValue](#) - used to set the maximum value of the y-axis in sparkline.

The following code is used to create the [SfLineSparkline](#).

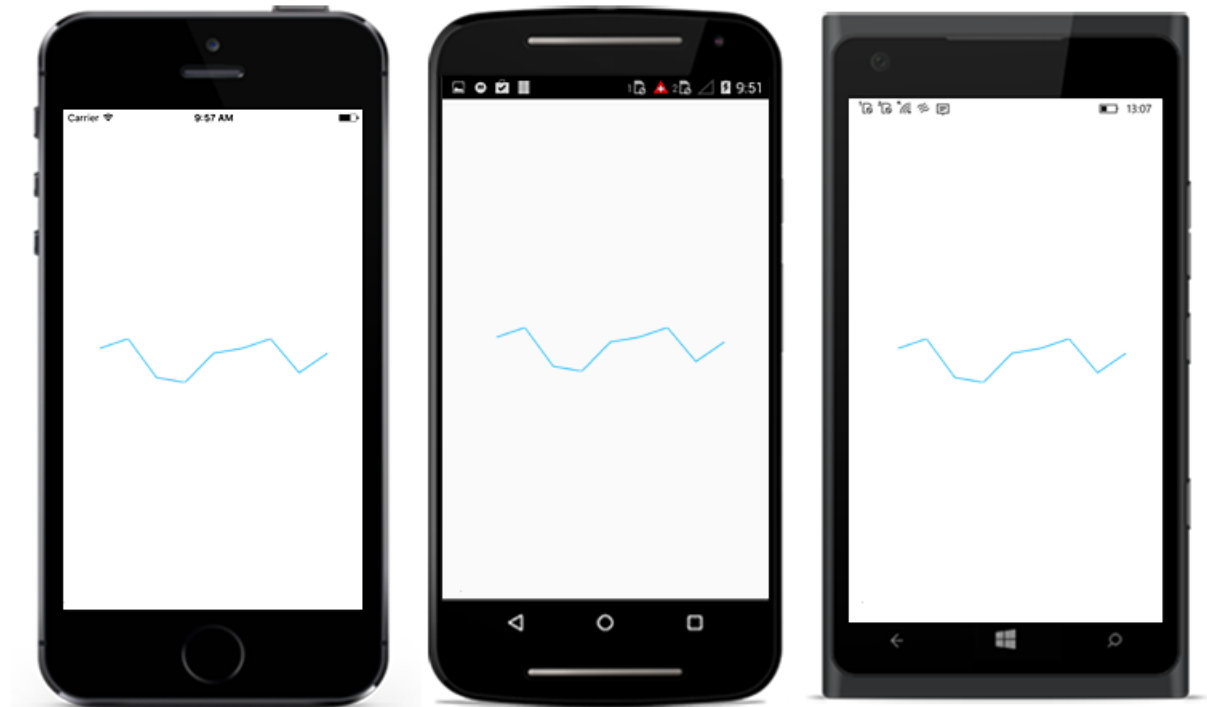
### XML

```
<sparkline:SfLineSparkline ItemsSource = "{Binding Data}"  
YBindingPath = "Performance">  
</sparkline:SfLineSparkline>
```

### C#

```
SfLineSparkline lineSparkline = new SfLineSparkline()  
{  
    ItemsSource = viewModel.Data,
```

```
YBindingPath = "Performance"
};
```



### Column Sparkline

[SfColumnSparkline](#) is very similar to a line sparkline in the sense that it is designed to show different values of two or more subjects but instead of using lines it is using horizontal and vertical bars that represent a different value.

- [StrokeWidth](#) - used to change the stroke width of the sparkline.
- [StrokeColor](#) - used to change the stroke color of the sparkline.
- [MinimumYValue](#) - used to set the minimum value of the y-axis in sparkline.
- [MaximumYValue](#) - used to set the maximum value of the y-axis in sparkline.
- [Color](#) - used to change the interior color of the column.

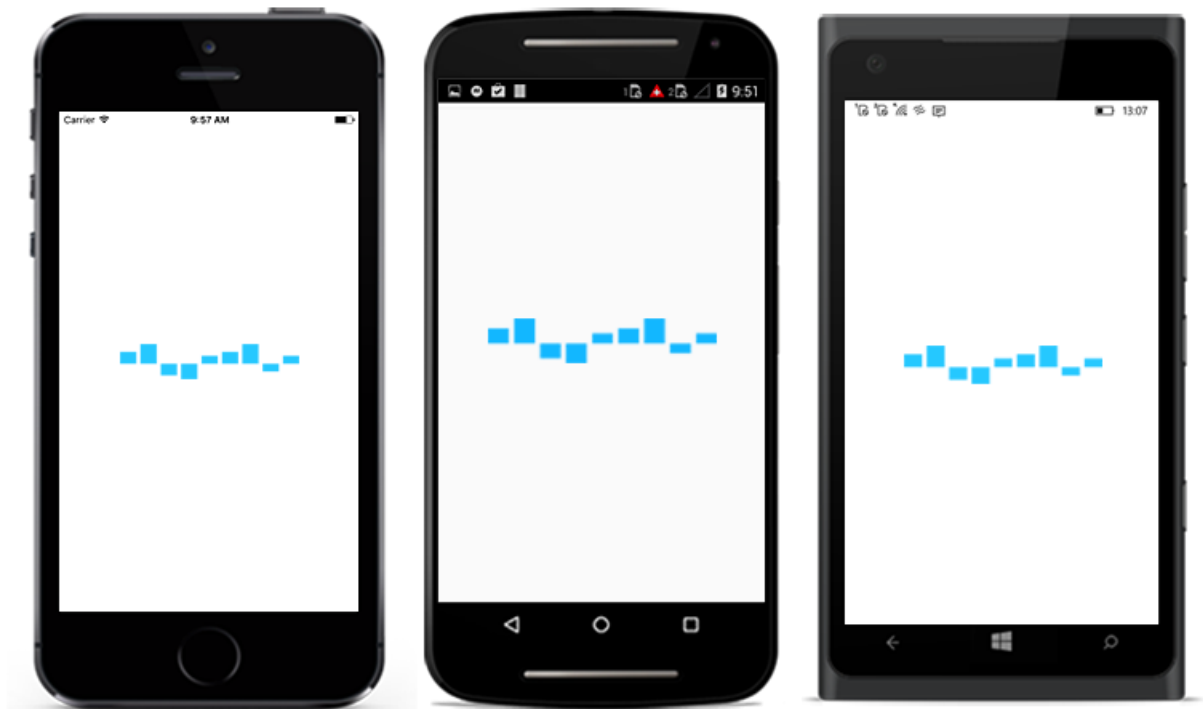
The following code is used to create the [SfColumnSparkline](#).

#### XML

```
<sparkline:SfColumnSparkline ItemsSource = "{Binding Data}"
YBindingPath = "Performance">
</sparkline:SfColumnSparkline>
```

#### C#

```
SfColumnSparkline columnSparkline = new SfColumnSparkline()
{
    ItemsSource = viewModel.Data,
    YBindingPath = "Performance"
};
```



### Area Sparkline

[SfAreaSparkline](#) is used to emphasize a change in values. This is primarily used when the magnitude of the trend is to be communicated rather than individual data values.

- [StrokeWidth](#) - used to change the stroke width of the sparkline.
- [StrokeColor](#) - used to change the stroke color of the sparkline.
- [MinimumYValue](#) - used to set the minimum value of the y-axis in sparkline.
- [MaximumYValue](#) - used to set the maximum value of the y-axis in sparkline.
- [Color](#) - used to change the color of interior area.

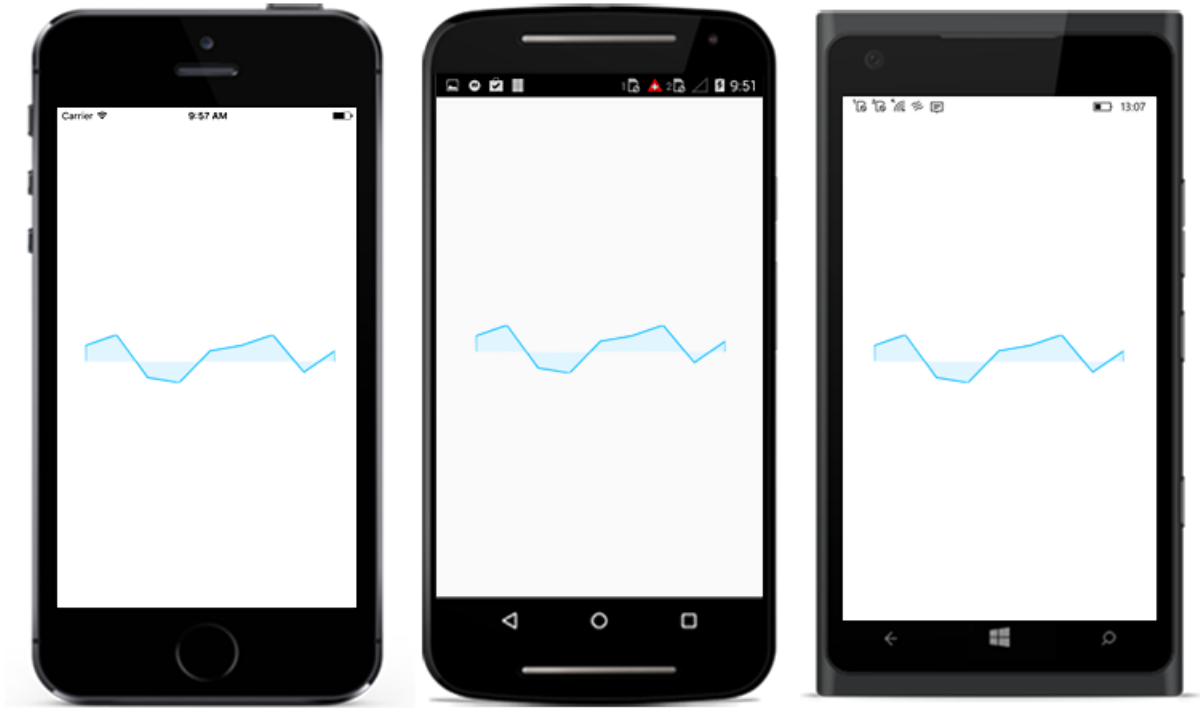
The following code is used to create the [SfAreaSparkline](#).

#### XML

```
<sparkline:SfAreaSparkline ItemsSource = "{Binding Data}"  
YBindingPath = "Performance">  
</sparkline:SfAreaSparkline>
```

#### C#

```
SfAreaSparkline areaSparkline = new SfAreaSparkline()  
{  
    ItemsSource = viewModel.Data,  
    YBindingPath = "Performance"  
};
```



### WinLoss Sparkline

[SfWinLossSparkline](#) is used to show whether each value is positive or negative visualizing a Win/Loss scenario. You can use the following properties to customize the appearance.

- [StrokeWidth](#) - used to change the stroke width of the sparkline.
- [StrokeColor](#) - used to change the stroke color of the sparkline.
- [MinimumYValue](#) - used to set the minimum value of the y-axis in sparkline.
- [MaximumYValue](#) - used to set the maximum value of the y-axis in sparkline.
- [Color](#) - used to change the interior color of the positive columns.
- [NeutralPointColor](#) - used to change the interior color of the neutral columns.

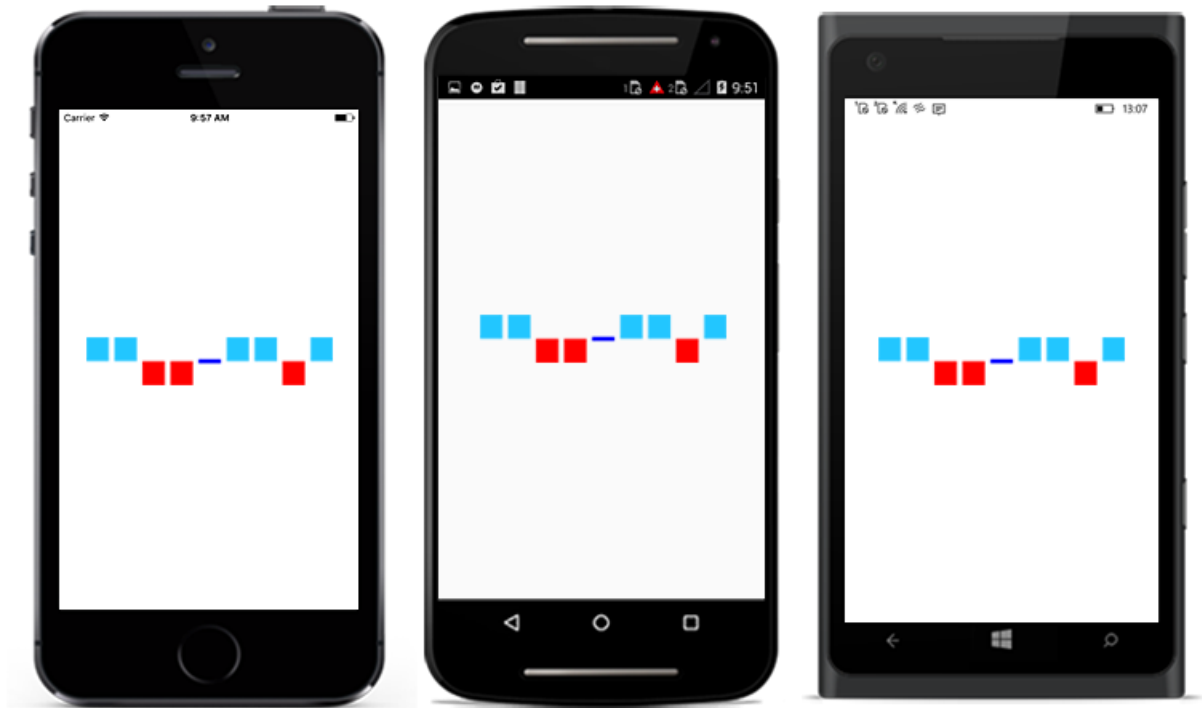
The following code is used to create the [SfWinLossSparkline](#).

#### XML

```
<sparkline:SfWinLossSparkline ItemsSource = "{Binding Data}"
YBindingPath = "Performance">
</sparkline:SfWinLossSparkline >
```

#### C#

```
SfWinLossSparkline winLossSparkline = new SfWinLossSparkline()
{
    ItemsSource = viewModel.Data,
    YBindingPath = "Performance"
};
```



## Range Band

This feature is used to highlight a particular region in the sparkline along Y axis.

- [RangeBandStart](#) - used to configure the start range band value in Y axis.
- [RangeBandEnd](#) - used to configure the end range band values in Y axis.
- [RangeBandColor](#) - used to change the color for range band.

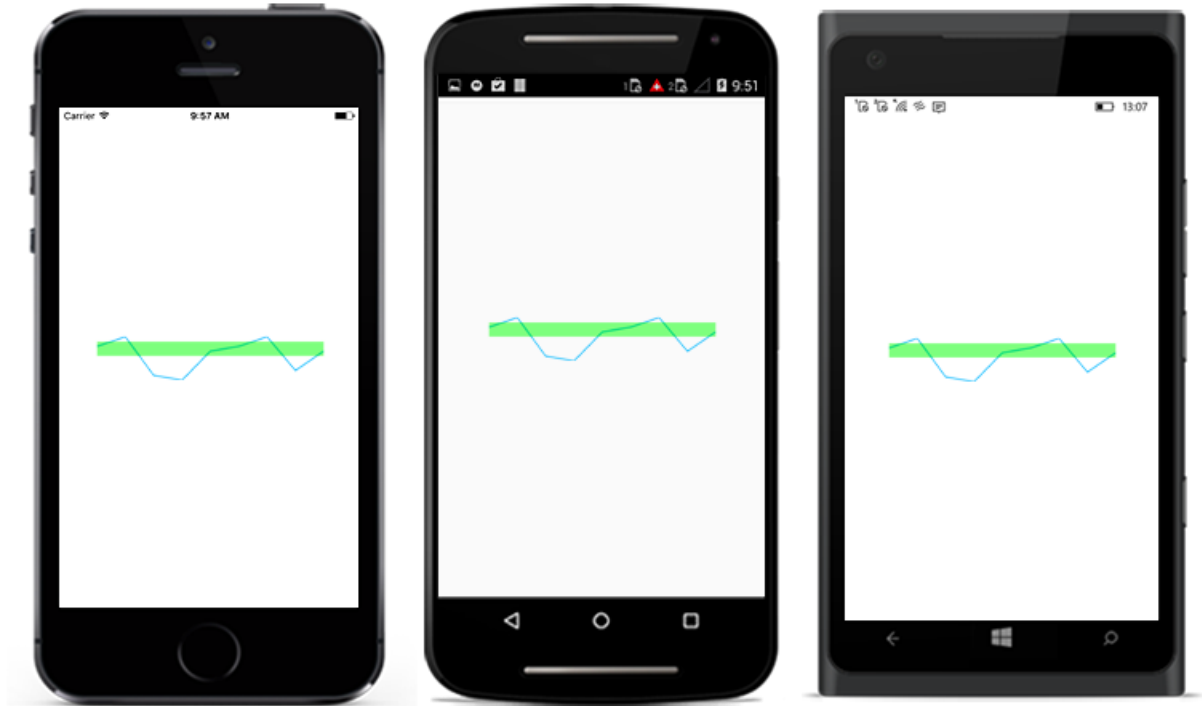
## XML

```
<sparkline:SfLineSparkline ItemsSource="{Binding Data}"
YBindingPath="Performance"
RangeBandStart="4000"
RangeBandEnd="1000"
RangeBandColor="Green">
</sparkline:SfLineSparkline>
```

## C#

```
SfLineSparkline lineSparkline = new SfLineSparkline()
{
    YBindingPath = "Performance",
    ItemsSource = viewModel.Data,
    RangeBandStart = 4000,
    RangeBandEnd = 1000,
    RangeBandColor = Color.FromRgba(0, 255, 0, 100)
};
```





## Markers

[Markers](#) are used to highlight the data point in [SfLineSparkline](#) and [SfAreaSparkline](#). You can use the following properties to customize the appearance.

- [IsVisible](#) - used to change the visibility of the marker.
- [Width](#) - used to change the width of the marker.
- [Height](#) - used to change the height of the marker.
- [Color](#) - used to change the color of the marker.

## XML

```
<sparkline:SfLineSparkline ItemsSource="{Binding Data}"
YBindingPath="Performance">
  <sparkline:SfLineSparkline.Marker>
    <sparkline:MarkerBase IsVisible="True"
Width="15"
Height="15"/>
  </sparkline:SfLineSparkline.Marker>
</sparkline:SfLineSparkline>
```

## C#

```
SfLineSparkline lineSparkline = new SfLineSparkline()
{
    YBindingPath = "Performance",
    ItemsSource = viewModel.Data,
    Marker = new MarkerBase()
    {
        IsVisible = true,
        Width = 15,
```

```
Height = 15
}
};
```



### Customize data points

Color of the first, last, high, low and negative data points can be customized using the following properties.

- [FirstPointColor](#) - used to change the first point color of the sparkline.
- [LastPointColor](#) - used to change the last point color of the sparkline.
- [HighPointColor](#) - used to change the high point color of the sparkline.
- [LowPointColor](#) - used to change the low point color of the sparkline.
- [NegativePointsColor](#) - used to change the negative point color of the sparkline.

**Note:** [NegativePointsColor](#) is applicable for [SfColumnSparkline](#) and [SfWinLossSparkline](#) alone.

Code snippet to customize the markers

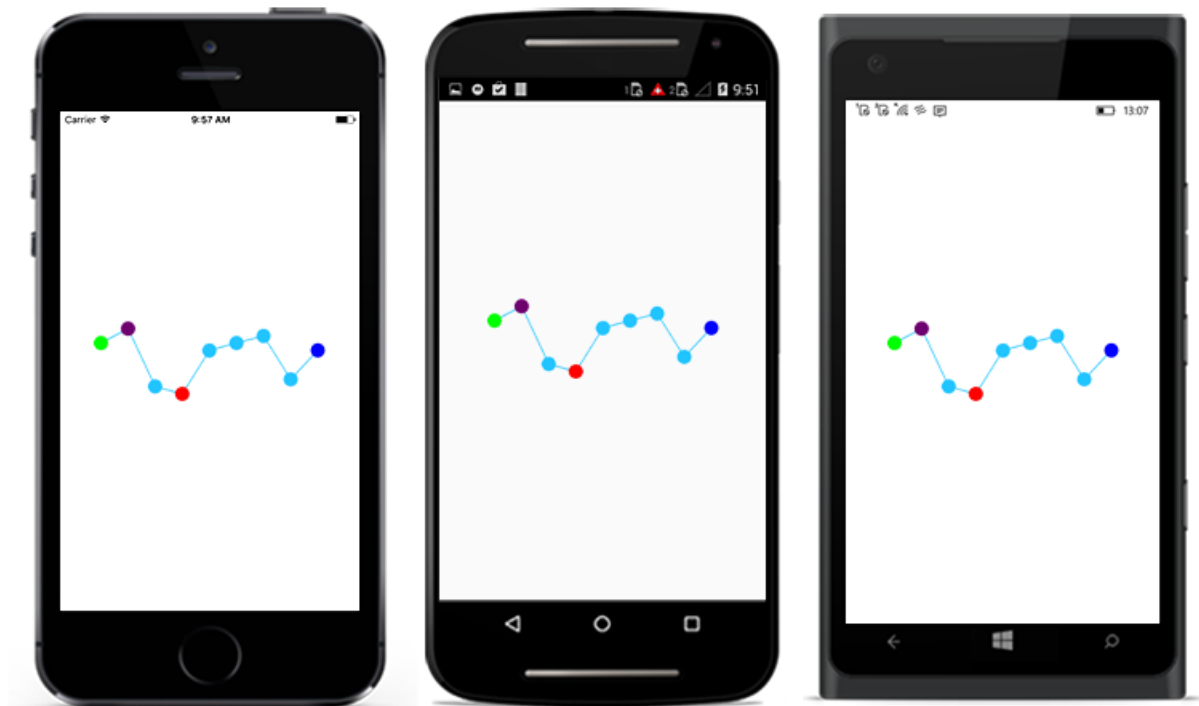
#### XML

```
<sparkline:SfLineSparkline ItemsSource = "{Binding Data}"
YBindingPath = "Performance"
FirstPointColor="Green"
LastPointColor="Blue"
HighPointColor="Purple"
LowPointColor="Red">
<sparkline:SfLineSparkline.Marker>
<sparkline:MarkerBase IsVisible="True"
Width= "15"
Height= "15"/>
```

```
</sparkline:SfLineSparkline.Marker>  
</sparkline:SfLineSparkline>
```

### C#

```
SfLineSparkline lineSparkline = new SfLineSparkline()  
{  
    YBindingPath = "Performance",  
    ItemsSource = viewModel.Data,  
    Marker = new MarkerBase()  
    {  
        IsVisible = true,  
        Width = 15,  
        Height = 15  
    },  
    FirstPointColor = Color.Green,  
    LastPointColor = Color.Blue,  
    HighPointColor = Color.Purple,  
    LowPointColor = Color.Red  
};
```



Code snippet to customize the segments

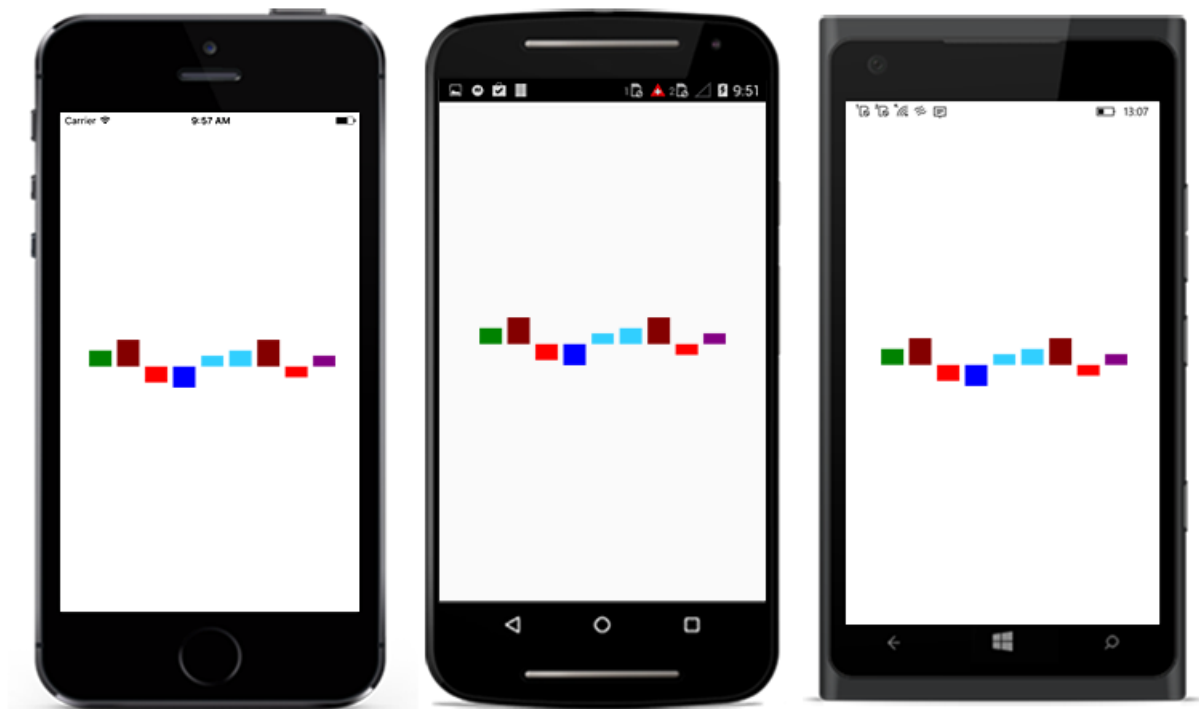
### XML

```
<sparkline:SfColumnSparkline ItemsSource="{Binding Data}"  
    YBindingPath="Performance"  
    FirstPointColor="Green"  
    LastPointColor="Purple"  
    HighPointColor="Maroon"  
    LowPointColor="Blue"
```

```
NegativePointsColor="Red">  
</sparkline:SfColumnSparkline>
```

## C#

```
SfColumnSparkline columnSparkline = new SfColumnSparkline()  
{  
    YBindingPath = "Performance",  
    ItemsSource = viewModel.Data,  
    FirstPointColor = Color.Green,  
    LastPointColor = Color.Purple,  
    HighPointColor = Color.Maroon,  
    LowPointColor = Color.Blue,  
    NegativePointsColor = Color.Red  
};
```



## Sparkline Axis

Axis of the sparkline can be configured and customized using following properties. This feature is applicable for all the sparkline types except [SfWinLossSparkline](#).

- [IsVisible](#) - used to change the visibility of the axis.
- [StrokeColor](#) - used to change the color of the axis.
- [StrokeWidth](#) - used to change the width of the axis.
- [AxisOrigin](#) - used to change the origin (positive or negative) of the axis.

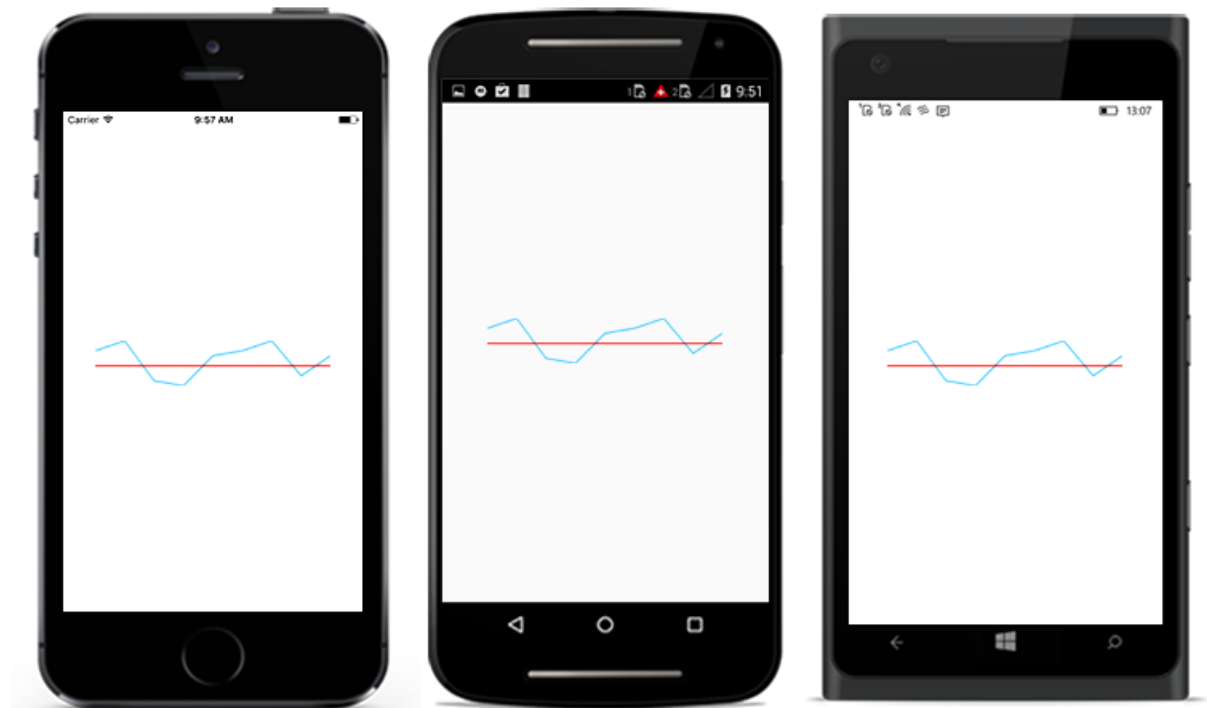
## XML

```
<sparkline:SfLineSparkline ItemsSource="{Binding Data}"  
    YBindingPath="Performance">
```

```
<sparkline:SfLineSparkline.Axis>  
<sparkline:SparklineAxis IsVisible="true"  
StrokeColor="Red"/>  
</sparkline:SfLineSparkline.Axis>  
</sparkline:SfLineSparkline>
```

## C#

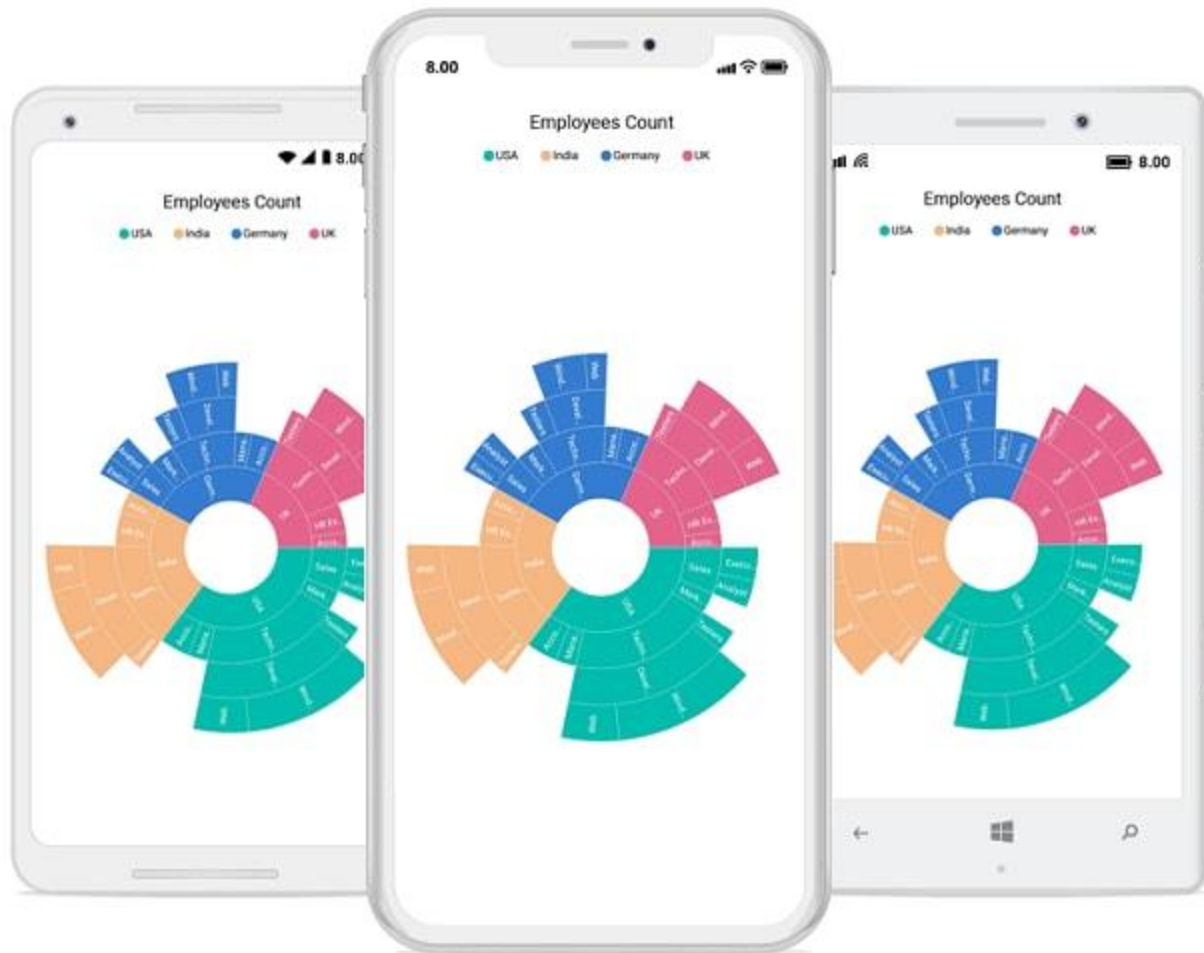
```
SfLineSparkline lineSparkline = new SfLineSparkline()  
{  
    YBindingPath = "Performance",  
    ItemsSource = viewModel.Data,  
    Axis = new SparklineAxis()  
    {  
        IsVisible = true,  
        StrokeColor = Color.Red,  
    }  
};
```



## SfSunburstChart

### Overview

The sunburst chart is used for visualizing hierarchical data. The circle at the center represents the root level of hierarchy, and the outer circles represent the higher levels of hierarchy.



### Key features

- Visualizes hierarchical data.
- Supports data labels for better readability.
- Supports legends with toggle selection.
- Supports palette.
- Supports drill-down feature that allows you to explore each level of hierarchy in detail.
- Provides interactive selection support that allows you to select or highlight the segments in the hierarchy with the following selection modes, such as Child, Group, Parent and Single.
- Supports animation.
- Supports tooltip to provide more information about the segments.

### Getting Started

This section explains the steps required to configure the [SfSunburstChart](#) and populate it with data, data labels, legends, and title. This section covers only the minimal features that needed to get started with the sunburst chart.

#### Adding SfSunburstChart reference

You can add SfSunburstChart reference using one of the following methods:

### Method 1: Adding SfSunburstChart reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfSunburstChart to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfSunburstChart](#), and then install it.

!{Adding SfSunburstChart reference from NuGet}(Getting-Started\_images/Adding SfSunburstChart reference.png)

---

**Note:** Install the same version of SfSunburstChart NuGet in all the projects.

---

### Method 2: Adding SfSunburstChart reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfSunburstChart control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfSunburstChart assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfSunburstChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfSunburstChart.Android.dll Syncfusion.SfSunburstChart.XForms.Android.dll Syncfusion.SfSunburstChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfSunburstChart.iOS.dll Syncfusion.SfSunburstChart.XForms.iOS.dll Syncfusion.SfSunburstChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfSunburstChart.UWP.dll Syncfusion.SfSunburstChart.XForms.UWP.dll Syncfusion.SfSunburstChart.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license](#)

[key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching an application on each platform with SfSunburstChart.

To use the SfSunburstChart control inside an application, each platform requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, this step is not needed.

#### iOS

To launch the SfSunburstChart in iOS, call the `SfSunburstChartRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` Framework has been initialized and before the `LoadApplication` is called, as demonstrated in the following code example.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.SfSunburstChart.XForms.iOS.SfSunburstChartRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

#### Universal Windows Platform (UWP)

You need to initialize the sunburst view assemblies in `App.xaml.cs` in UWP project as demonstrated in the following code samples. This is required to deploy the application with sunburst in Release mode in UWP platform.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    // Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.SfSunburstChart.XForms.UWP.SfSunburstChartRenderer).GetTypeInfo().Assembly);
    // Replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

#### Android

The Android platform does not require any additional configuration to render the sunburst.



### Initialize sunburst chart

Import the [SfSunburstChart](#) namespace as shown as follows.

#### XML

```
xmlns:sunburst="clr-
namespace:Syncfusion.SfSunburstChart.XForms;assembly=Syncfusion.SfSunburstCh
art.XForms"
```

#### C#

```
using Syncfusion.SfSunburstChart.XForms;
```

Then, initialize an empty sunburst chart as shown as follows.

#### XML

```
<sunburst:SfSunburstChart>
</sunburst:SfSunburstChart>
```

#### C#

```
SfSunburstChart sunburst = new SfSunburstChart();
this.Content = sunburst;
```

### Initialize view model

In this section, data in the following table is used for demonstration.

Country	Job description	Job group	Job role	Employees count
United States	Sales	Executive		50
United States	Sales	Analyst		40
United States	Marketing			40
United States	Technical	Testers		35
United States	Technical	Developers	Windows	175
United States	Technical	Developers	Web	70
United States	Management			40
United States	Accounts			60
India	Technical	Testers		33
India	Technical	Developers	Windows	125
India	Technical	Developers	Web	60
India	HR Executives			70
India	Accounts			45

Germany	Sales	Executive		30
Germany	Sales	Analyst		40
Germany	Marketing			50
Germany	Technical	Testers		40
Germany	Technical	Developers	Windows	65
Germany	Technical	Developers	Web	27
Germany	Management			33
Germany	Accounts			55
UK	Technical	Testers		25
UK	Technical	Developers	Windows	96
UK	Technical	Developers	Web	55
UK	HR executives			60
UK	Accounts			30

Define a data model that represents the above data in [SfSunburstChart](#).

### C#

```
public class SunburstModel
{
    public string JobDescription { get; set; }
    public string JobGroup { get; set; }
    public string JobRole { get; set; }
    public double EmployeesCount { get; set; }
    public string Country { get; set; }
}
```

Then, create a view model class, and initialize a list of SunburstModel objects as follows.

### C#

```
public class SunburstViewModel
{
    public ObservableCollection<SunburstModel> DataSource { get; set; }
    public SunburstViewModel()
    {
        this.DataSource = new ObservableCollection<SunburstModel>
        {
            new SunburstModel { Country = "USA", JobDescription = "Sales",
                                JobGroup="Executive", EmployeesCount = 50 },
            new SunburstModel { Country = "USA", JobDescription = "Sales", JobGroup =
                                "Analyst", EmployeesCount = 40 },
            new SunburstModel { Country = "USA", JobDescription = "Marketing",
                                EmployeesCount = 40 },
        }
    }
}
```

```

new SunburstModel { Country = "USA", JobDescription = "Technical", JobGroup
= "Testers", EmployeesCount = 35 },
new SunburstModel { Country = "USA", JobDescription = "Technical", JobGroup
= "Developers", JobRole = "Windows", EmployeesCount = 175 },
new SunburstModel { Country = "USA", JobDescription = "Technical", JobGroup
= "Developers", JobRole = "Web", EmployeesCount = 70 },
new SunburstModel { Country = "USA", JobDescription = "Management",
EmployeesCount = 40 },
new SunburstModel { Country = "USA", JobDescription = "Accounts",
EmployeesCount = 60 },
new SunburstModel { Country = "India", JobDescription = "Technical",
JobGroup = "Testers", EmployeesCount = 33 },
new SunburstModel { Country = "India", JobDescription = "Technical",
JobGroup = "Developers", JobRole = "Windows", EmployeesCount = 125 },
new SunburstModel { Country = "India", JobDescription = "Technical",
JobGroup = "Developers", JobRole = "Web", EmployeesCount = 60 },
new SunburstModel { Country = "India", JobDescription = "HR Executives",
EmployeesCount = 70 },
new SunburstModel { Country = "India", JobDescription = "Accounts",
EmployeesCount = 45 },
new SunburstModel { Country = "Germany", JobDescription = "Sales", JobGroup
= "Executive", EmployeesCount = 30 },
new SunburstModel { Country = "Germany", JobDescription = "Sales", JobGroup
= "Analyst", EmployeesCount = 40 },
new SunburstModel { Country = "Germany", JobDescription = "Marketing",
EmployeesCount = 50 },
new SunburstModel { Country = "Germany", JobDescription = "Technical",
JobGroup = "Testers", EmployeesCount = 40 },
new SunburstModel { Country = "Germany", JobDescription = "Technical",
JobGroup = "Developers", JobRole = "Windows", EmployeesCount = 65 },
new SunburstModel { Country = "Germany", JobDescription = "Technical",
JobGroup = "Developers", JobRole = "Web", EmployeesCount = 27 },
new SunburstModel { Country = "Germany", JobDescription = "Management",
EmployeesCount = 33 },
new SunburstModel { Country = "Germany", JobDescription = "Accounts",
EmployeesCount = 55 },
new SunburstModel { Country = "UK", JobDescription = "Technical", JobGroup =
"Testers", EmployeesCount = 25 },
new SunburstModel { Country = "UK", JobDescription = "Technical", JobGroup =
"Developers", JobRole = "Windows", EmployeesCount = 96 },
new SunburstModel { Country = "UK", JobDescription = "Technical", JobGroup =
"Developers", JobRole = "Web", EmployeesCount = 55 },
new SunburstModel { Country = "UK", JobDescription = "HR Executives",
EmployeesCount = 60 },
new SunburstModel { Country = "UK", JobDescription = "Accounts",
EmployeesCount = 30 }
};
}
}

```

Set the `SunburstViewModel` instance as the `BindingContext` of your page to bind the properties of `SunburstViewModel` to [SfSunburstChart](#).

**Note:** Add the namespace of `SunburstViewModel` class in your XAML page if you set the `BindingContext` in XAML.

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:SunburstDemo"
xmlns:sunburst="clr-
namespace:Syncfusion.SfSunburstChart.XForms;assembly=Syncfusion.SfSunburstCh
art.XForms"
x:Class="SunburstDemo.MainPage">
<ContentPage.BindingContext>
<local:SunburstViewModel></local:SunburstViewModel>
</ContentPage.BindingContext>
</ContentPage>
```

**C#**

```
this.BindingContext = new SunburstViewModel();
```

Populate sunburst chart with data

Bind the DataSource property of the above SunburstViewModel to the [ItemsSource](#) property.

Then, add the [SunburstHierarchicalLevel](#) to [Levels](#) collection. Each hierarchy level is formed based on the property specified in [GroupMemberPath](#) property, and each arc segment size is calculated using the [ValueMemberPath](#) property.

**XML**

```
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount">
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
</sunburst:SfSunburstChart>
```

**C#**

```
SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburstChart.ValueMemberPath = "EmployeesCount";
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
this.Content = sunburstChart;
```



### Add title

You can add title to the sunburst chart to provide information to users about the data being plotted in the chart. You can set title using the [SfSunburstChart.Title](#) property.

### XML

```
<sunburst:SfSunburstChart>
<sunburst:SfSunburstChart.Title>
<sunburst:SunburstChartTitle x:Name="title" Text="Employees
Count"></sunburst:SunburstChartTitle>
</sunburst:SfSunburstChart.Title>
</sunburst:SfSunburstChart>
```

### C#

```
SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.Title = new SunburstChartTitle();
sunburstChart.Title.Text = "Employees Count";
this.Content = sunburstChart;
```

## Employees Count



### Add legend

You can enable legend using the [SfSunburstChart.Legend](#) property.

#### XML

```
<sunburst:SfSunburstChart>
  <sunburst:SfSunburstChart.Legend>
    <sunburst:SunburstChartLegend x:Name="legend" IsVisible="True" >
    </sunburst:SunburstChartLegend>
  </sunburst:SfSunburstChart.Legend>
</sunburst:SfSunburstChart>
```

#### C#

```
SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.Legend = new SunburstChartLegend();
sunburstChart.Legend.IsVisible = true;
this.Content = sunburstChart;
```



#### Add data labels

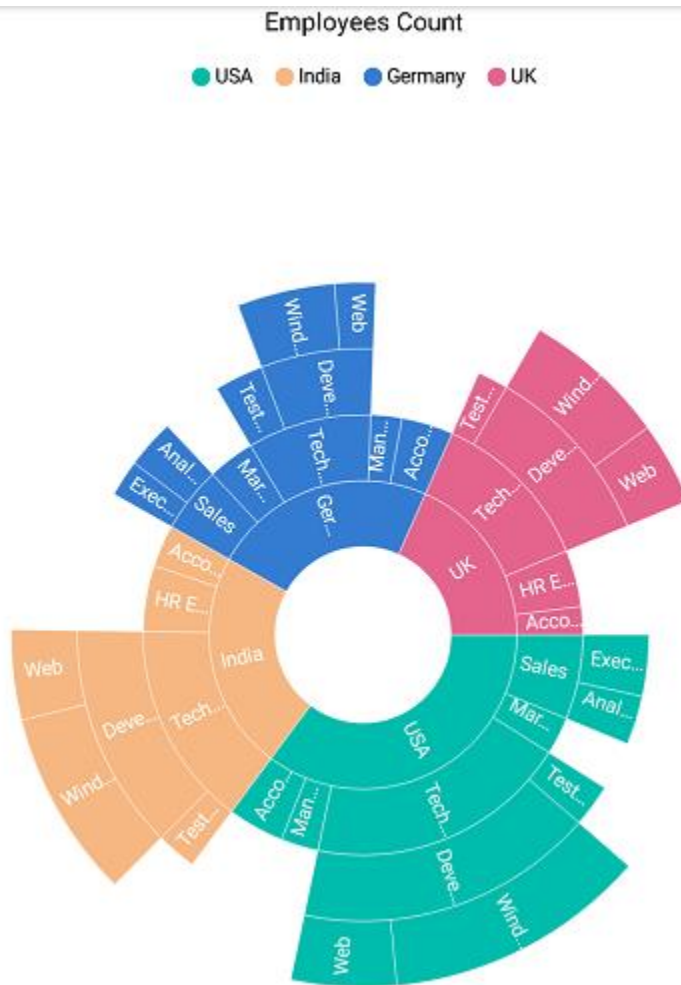
You can add data labels to improve the readability of the sunburst chart. Data labels can be added using the [SfSunburstChart.DataLabel](#) property.

#### XML

```
<sunburst:SfSunburstChart>
  <sunburst:SfSunburstChart.DataLabel>
    <sunburst:SunburstChartDataLabel x:Name="dataLabel" ShowLabel="True">
    </sunburst:SunburstChartDataLabel>
  </sunburst:SfSunburstChart.DataLabel>
</sunburst:SfSunburstChart>
```

#### C#

```
SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.DataLabel = new SunburstChartDataLabel();
sunburstChart.DataLabel.ShowLabel = true;
this.Content = sunburstChart;
```



Below snippet is the complete code for generating the final output.

#### XML

```
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount">
  <sunburst:SfSunburstChart.Levels>
    <sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
    <sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
    <sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
    <sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
  </sunburst:SfSunburstChart.Levels>
  <sunburst:SfSunburstChart.Title>
    <sunburst:SunburstChartTitle x:Name="title" Text="Employees Count"
    ></sunburst:SunburstChartTitle>
  </sunburst:SfSunburstChart.Title>
  <sunburst:SfSunburstChart.Legend>
    <sunburst:SunburstChartLegend x:Name="legend" IsVisible="True" >
    </sunburst:SunburstChartLegend>
  </sunburst:SfSunburstChart.Legend>
  <sunburst:SfSunburstChart.DataLabel>
    <sunburst:SunburstChartDataLabel x:Name="dataLabel"
    ShowLabel="True"></sunburst:SunburstChartDataLabel>
```



```
</sunburst:SfSunburstChart.DataLabel>  
</sunburst:SfSunburstChart>
```

## C#

```
this.BindingContext = new SunburstViewModel();  
SfSunburstChart sunburstChart = new SfSunburstChart();  
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");  
sunburstChart.ValueMemberPath = "EmployeesCount";  
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =  
"Country" });  
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =  
"JobDescription" });  
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =  
"JobGroup" });  
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =  
"JobRole" });  
sunburstChart.Title = new SunburstChartTitle();  
sunburstChart.Title.Text = "Employees Count";  
sunburstChart.Legend = new SunburstChartLegend();  
sunburstChart.Legend.IsVisible = true;  
sunburstChart.DataLabel = new SunburstChartDataLabel();  
sunburstChart.DataLabel.ShowLabel = true;  
this.Content = sunburstChart;
```

The following screenshot depicts the final output.



You can find the complete getting started sample from this [link](#).

## Levels

The sunburst chart is used to display hierarchical data. More than one hierarchical data can be added to the [Levels](#) collection of the sunburst chart. Each level of the hierarchy is represented by a circle.

The following code shows how to add hierarchical levels in the Levels collection.

### XML

```
<sunburst:SfSunburstChart.Levels>
  <sunburst:SunburstHierarchicalLevel />
</sunburst:SfSunburstChart.Levels>
```

### C#

```
SunburstHierarchicalLevel level = new SunburstHierarchicalLevel();
sunburstChart.Levels.Add(level);
```

### Group member path

The [GroupMemberPath](#) is a string property that is used to map the group category value in the sunburst [ItemsSource](#).

The following code shows how to map the group member path.

#### XML

```
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Level1"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Level2"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Level3"/>
</sunburst:SfSunburstChart.Levels>
```

#### C#

```
SunburstHierarchicalLevel level1 = new SunburstHierarchicalLevel();
level1.GroupMemberPath = "Level1";
SunburstHierarchicalLevel level2 = new SunburstHierarchicalLevel();
level2.GroupMemberPath = "Level2";
SunburstHierarchicalLevel level3 = new SunburstHierarchicalLevel();
level3.GroupMemberPath = "Level3";
sunburstChart.Levels.Add(level1);
sunburstChart.Levels.Add(level2);
sunburstChart.Levels.Add(level3);
```

The following code specifies the levels for data model specified in the getting started section.

#### XML

```
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount">
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
</sunburst:SfSunburstChart>
```

#### C#

```
SfSunburstChart sunburst = new SfSunburstChart();
sunburst.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburst.ValueMemberPath = "EmployeesCount";
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
this.Content = sunburstChart;
```



### Legend

Legends are used to represent the first level (i.e root level) of categories in the sunburst chart.

The following code explains how to initialize the legends.

#### XML

```
<sunburst:SfSunburstChart.Legend>  
<sunburst:SunburstChartLegend>  
</sunburst:SunburstChartLegend>  
</sunburst:SfSunburstChart.Legend>
```

#### C#

```
SunburstChartLegend legend = new SunburstChartLegend();  
sunburstChart.Legend = legend;
```

### Visibility

The visibility of legends can be controlled using the [IsVisible](#) property.

The following code shows how to control the visibility of legend.

**XML**

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstChartLegend IsVisible="True" >
</sunburst:SunburstChartLegend>
</sunburst:SfSunburstChart.Legend>
```

**C#**

```
SunburstChartLegend legend = new SunburstChartLegend();
legend.IsVisible = true;
sunburstChart.Legend = legend;
```

**Position**

Legends can be docked at the top, right, left, or bottom position using the [LegendPosition](#) property.

The following code shows customizing the legend position.

**XML**

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstChartLegend x:Name="legend" IsVisible="True"
LegendPosition="Left" >
```

```
</sunburst:SunburstChartLegend>
</sunburst:SfSunburstChart.Legend>
```

**C#**

```
SunburstChartLegend legend = new SunburstChartLegend();
legend.IsVisible = true;
legend.LegendPosition = SunburstDockPosition.Left;
sunburstChart.Legend = legend;
```

**Legend icon types**

Legend icon shapes can be customized using the [IconType](#) property. The IconType property provides several predefined shapes. The default legend icon type is circle.

The following predefined shapes are available in the IconType property:

- Circle
- Cross
- Diamond
- Pentagon
- Rectangle

- Triangle.

**XML**

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstChartLegend x:Name="legend" IsVisible="True"
IconType="Diamond" >
</sunburst:SunburstChartLegend>
</sunburst:SfSunburstChart.Legend>
```

**C#**

```
SunburstChartLegend legend = new SunburstChartLegend();
legend.IsVisible = true;
legend.IconType = SunburstLegendIcon.Diamond;
sunburstChart.Legend = legend;
```

**Icon size customization**

The size of the legend icon can be customized using the [IconHeight](#) and [IconWidth](#) properties.

**XML**



```

<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstChartLegend x:Name="legend" IsVisible="True"
IconHeight="15"
IconWidth="15" IconType="Diamond">
</sunburst:SunburstChartLegend>
</sunburst:SfSunburstChart.Legend>

```

## C#

```

SunburstChartLegend legend = new SunburstChartLegend();
legend.IsVisible = true;
legend.IconType = SunburstLegendIcon.Diamond;
legend.IconHeight = 15;
legend.IconWidth = 15;
sunburstChart.Legend = legend;

```



## Label style

Legend label can be customized using the following properties available in [LabelStyle](#):

- [TextColor](#): Customizes the text color of the label.
- [FontSize](#): Customizes the font size of the label.



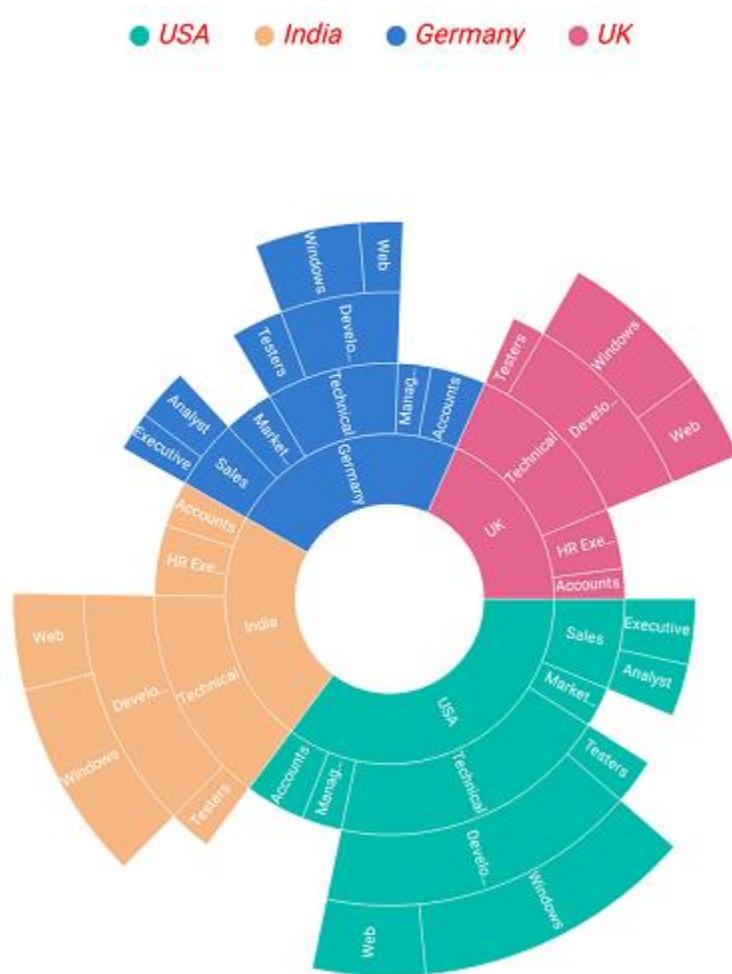
- [FontAttributes](#): Customizes the font attributes such as Bold or Italic.
- [Margin](#): Sets the specified margin for legend labels.
- [FontFamily](#): Sets the specified font family for labels.

### XML

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstChartLegend x:Name="legend" IsVisible="True" >
<sunburst:SunburstChartLegend.LabelStyle>
<sunburst:SunburstLegendLabelStyle x:Name="legendStyle"
FontAttributes="Italic" FontSize="14" Margin="5"
TextColor="Red" ></sunburst:SunburstLegendLabelStyle>
</sunburst:SunburstChartLegend.LabelStyle>
</sunburst:SunburstChartLegend>
</sunburst:SfSunburstChart.Legend>
```

### C#

```
sunburstChart.Legend = new SunburstChartLegend();
sunburstChart.Legend.IsVisible = true;
SunburstLegendLabelStyle labelStyle = new SunburstLegendLabelStyle();
labelStyle.FontAttributes = FontAttributes.Italic;
labelStyle.FontSize = 14;
labelStyle.TextColor = Color.Red;
labelStyle.Margin = new Thickness(5);
sunburstChart.Legend.LabelStyle = labelStyle;
```



### Item margin

Margin can be set to individual legend items using the [ItemMargin](#) property.

### XML

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstChartLegend x:Name="legend" IsVisible="True"
ItemMargin="3" >
</sunburst:SunburstChartLegend>
</sunburst:SfSunburstChart.Legend>
```

### C#

```
SunburstChartLegend legend = new SunburstChartLegend();
legend.IsVisible = true;
legend.ItemMargin = new Thickness(3, 3, 3, 3);
sunburstChart.Legend = legend;
```



Toggle selection

Sunburst segments can also be selected via legends.

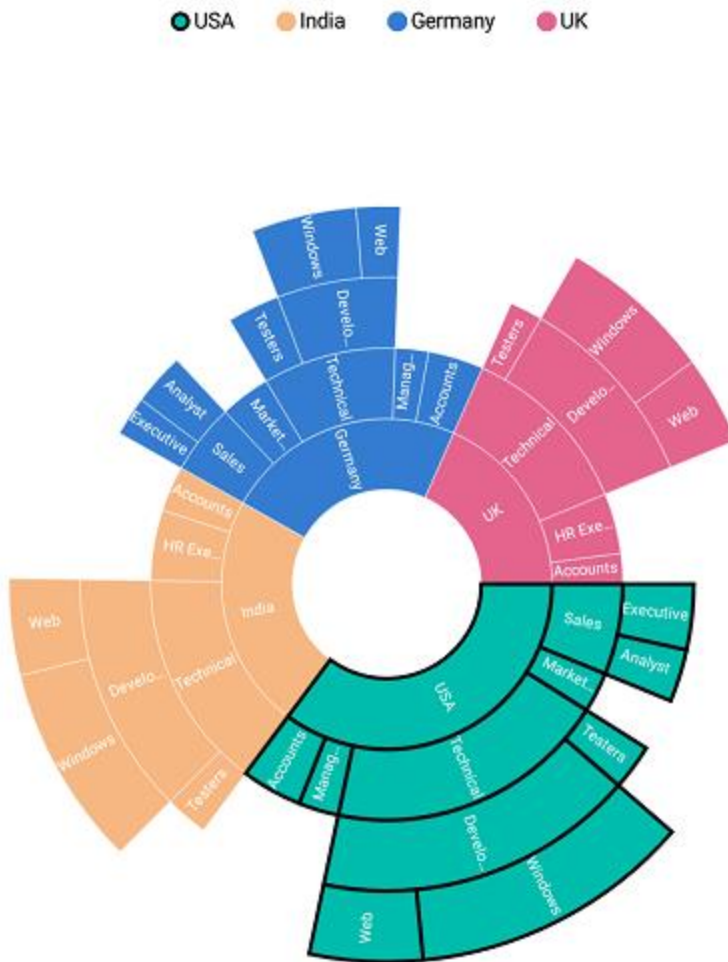
### XML

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstChartLegend x:Name="legend" IsVisible="True" >
</sunburst:SunburstChartLegend>
</sunburst:SfSunburstChart.Legend>
<sunburst:SfSunburstChart.SelectionSettings>
<sunburst:SelectionSettings EnableSelection="True"
SelectionDisplayMode="HighlightByStrokeColor">
</sunburst:SelectionSettings>
</sunburst:SfSunburstChart.SelectionSettings>
```

### C#

```
SunburstChartLegend legend = new SunburstChartLegend();
legend.IsVisible = true;
sunburstChart.Legend = legend;
SelectionSettings selection = new SelectionSettings();
selection.EnableSelection = true;
```

```
selection.SelectionDisplayMode =
SelectionDisplayMode.HighlightByStrokeColor;
sunburstChart.SelectionSettings = selection;
```



## Data label

Data labels are used to display information about segments. Data labels are enabled and disabled using the [ShowLabel](#) property. The default value of the ShowLabel property is true.

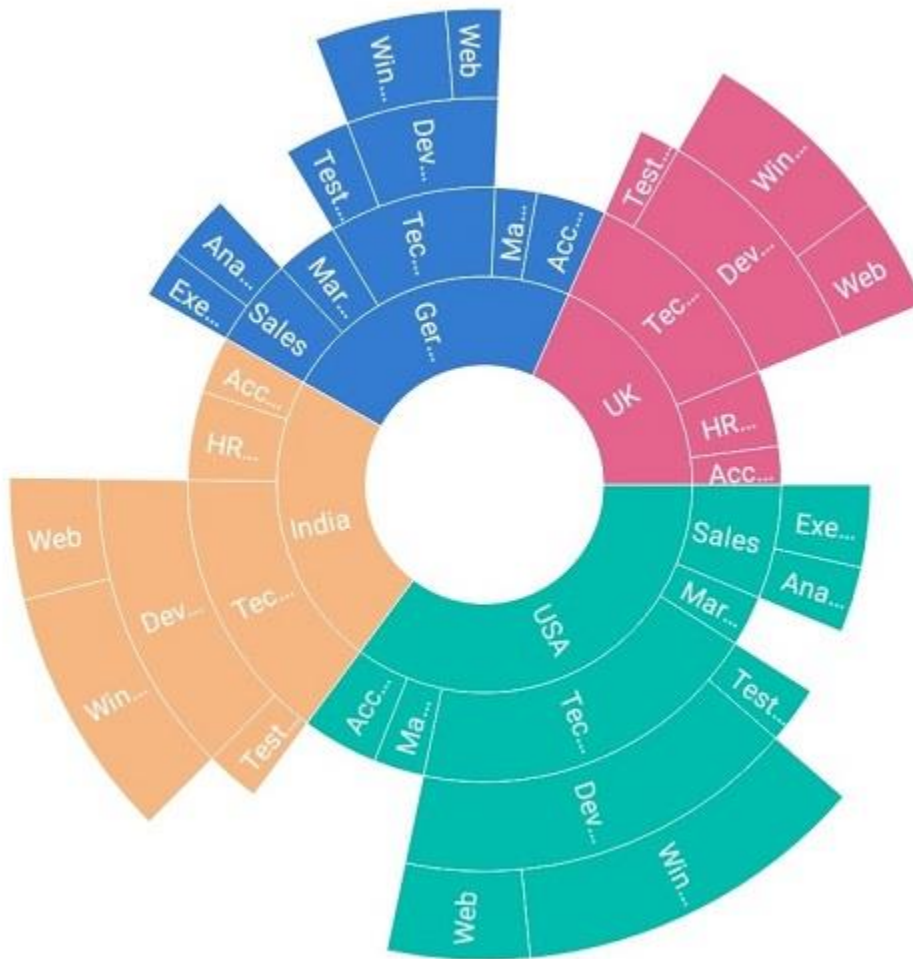
The following code explains how to initialize data labels.

### XML

```
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel"
ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
```

### C#

```
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
sunburstChart.DataLabel = label;
```



### Overflow Mode

When the data labels are large in size, they will overlap each other. To avoid overlapping, trim or hide the data labels using the [OverflowMode](#) property.

### Trim

The following code shows trimming the data labels.

### XML

```
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel" OverflowMode="Trim"
ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
```

### C#

```
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
label.OverflowMode = SunburstLabelOverflowMode.Trim;
sunburstChart.DataLabel = label;
```



### Hide

The following code shows hiding the data labels.

### XML

```
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel" ShowLabel="True"
OverflowMode="Hide"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
```

### C#

```
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
label.OverflowMode = SunburstLabelOverflowMode.Hide;
sunburstChart.DataLabel = label;
```



### Rotation Mode

The view of data labels can be customized using the [RotationMode](#) property. Data labels can be rotated to a angle for better readability. By default, the rotation mode is angle.

### Angle

The following code shows rotating a data label to an angle.

### XML

```
<sunburst:SfSunburstChart.DataLabel>
  <sunburst:SunburstChartDataLabel x:Name="dataLabel" RotationMode="Angle"
  ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
```

### C#

```
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
label.RotationMode = SunburstLabelRotationMode.Angle;
sunburstChart.DataLabel = label;
```





### Normal

The following code shows normal mode of data labels.

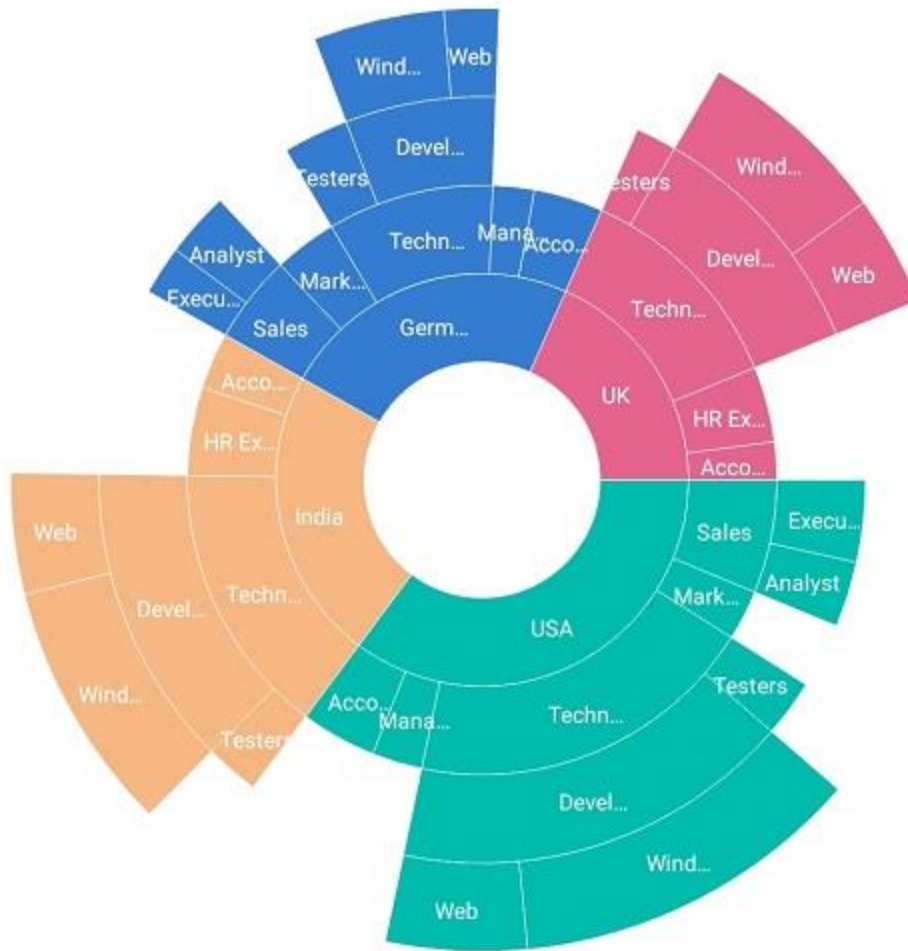
### XML

```
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel" RotationMode="Normal"
ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
```

### C#

```
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
label.RotationMode = SunburstLabelRotationMode.Normal;
sunburstChart.DataLabel = label;
```





### Customization

Data labels can be customized using the following properties.

- [TextColor](#) : Text color of the label can be changed.
- [FontSize](#) : Data label font size can be modified.
- [FontAttributes](#) : Font attributes such as bold or italic can be used.
- [FontFamily](#) : This modifies the font family of the labels.

### XML

```
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel" ShowLabel="True"
FontAttributes="Bold"
FontSize="10" TextColor="Red" FontFamily="ArialMT">
</sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
```

### C#

```
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
```

```
label.TextColor = Color.Red;
label.FontSize = 10;
label.FontAttributes = FontAttributes.Bold;
label.FontFamily = "ArialMT";
sunburstChart.DataLabel = label;
```



### Label text color

Data label text color will change automatically based on the color of the segments. Whatever be the color of the segments, text color of the label will change accordingly for better readability.

In the following code **Pineapple** palette is applied, hence data label takes the color based on the color of the segments.

### XML

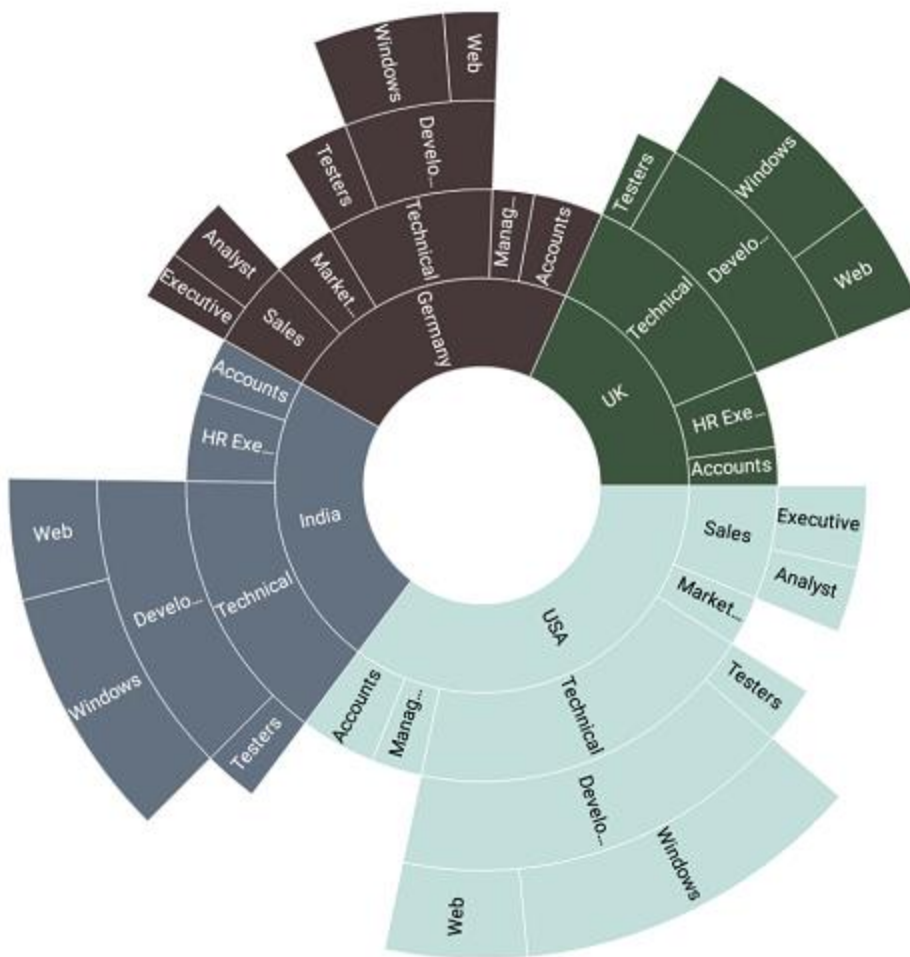
```
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel" ShowLabel="True" >
</sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
<sunburst:SfSunburstChart.ColorModel>
<sunburst:SunburstChartColorModel Palette="Pineapple">
</sunburst:SunburstChartColorModel>
```

```
</sunburst:SfSunburstChart.ColorModel>
```

### C#

```
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
sunburstChart.DataLabel = label;
SunburstChartColorModel colorModel = new SunburstChartColorModel();
colorModel.Palette = SunburstColorPalette.Pineapple;
sunburstChart.ColorModel = colorModel;
```

In the below image, first segment's data label is in black and others in white.



### Drill-down

The drill-down provides better visualization of hierarchy. Large set of data can be virtualized into minimal views. Each level of the segments can be drilled down. The sunburst chart provides animation along with the drill-down support. Toolbar will be enabled on drill-down that helps in performing zoom back and reset operations. The drill-down can be enabled or disabled using the [Enable](#) property in the drill-down settings.

**Information:** Double tapping the segment performs the drill-down operation.

The following code shows enabling the [DrilldownSettings](#).

#### XML

```
<sunburst:SfSunburstChart.DrilldownSettings>
<sunburst:DrilldownSettings Enable="True"></sunburst:DrilldownSettings>
</sunburst:SfSunburstChart.DrilldownSettings>
```

#### C#

```
DrilldownSettings drilldownSettings = new DrilldownSettings();
drilldownSettings.Enable = true;
sunburstChart.DrilldownSettings = drilldownSettings;
```



#### Positioning Toolbar

Toolbar can be positioned anywhere on the chart by specifying the [OffsetX](#) and [OffsetY](#) values. The offset values range from 0 to 1.

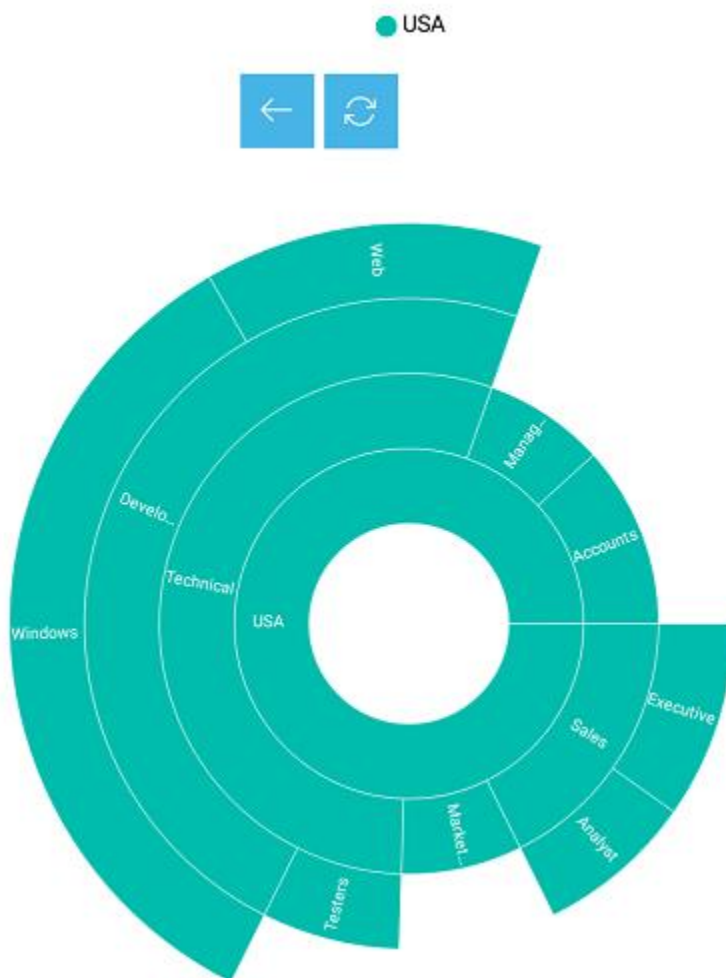
#### XML

```
<sunburst:SfSunburstChart.DrilldownSettings>
<sunburst:DrilldownSettings OffsetX="0.5" OffsetY="0"></sunburst:DrilldownSettings>
</sunburst:SfSunburstChart.DrilldownSettings>
```

```
Enable="True"></sunburst:DrilldownSettings>
</sunburst:SfSunburstChart.DrilldownSettings>
```

## C#

```
DrilldownSettings drilldownSettings = new DrilldownSettings();
drilldownSettings.Enable = true;
drilldownSettings.OffsetX = 0.5;
drilldownSettings.OffsetY = 0;
sunburstChart.DrilldownSettings = drilldownSettings;
```



## Toolbar alignment

The vertical and the horizontal alignments of the toolbar can be customized using the [ToolbarVerticalAlignment](#) and [ToolbarHorizontalAlignment](#) properties, respectively.

Both the alignment properties has the following enum types:

- Center: Toolbar takes the specified offset value as the center of the toolbar and get positioned.
- End: Toolbar takes the specified offset value as the start of the toolbar and get positioned.
- Start: Toolbar takes the specified offset value as the end of the toolbar and get positioned.

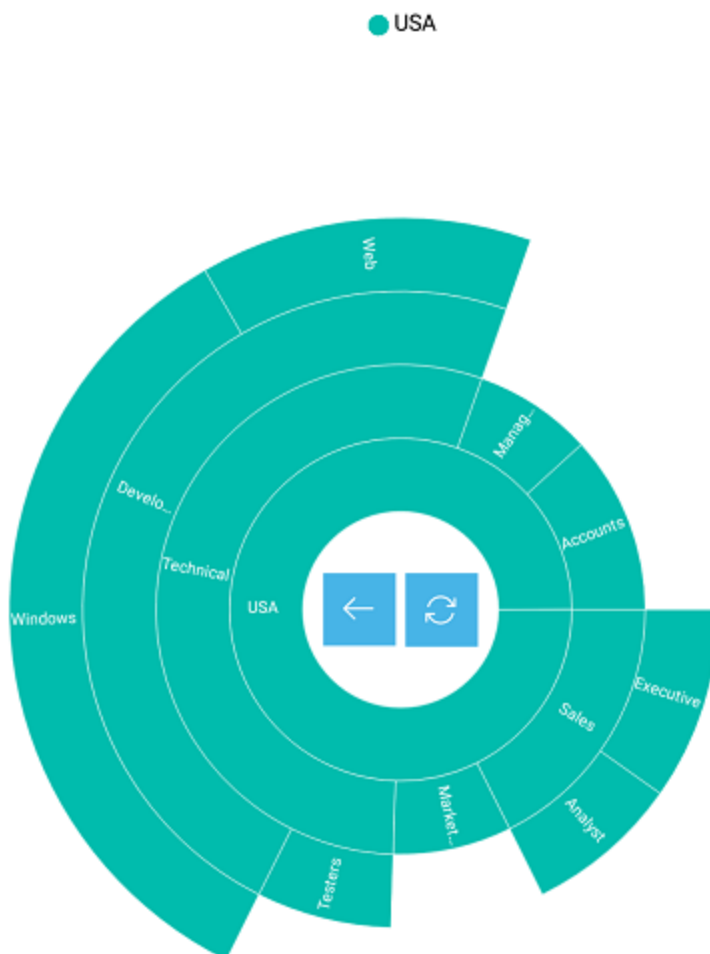
The following code shows the toolbar alignment.

#### XML

```
<sunburst:SfSunburstChart.DrilldownSettings>  
<sunburst:DrilldownSettings OffsetX="0.5" OffsetY="0.5" Enable="True"  
ToolbarHorizontalAlignment="Center"  
ToolbarVerticalAlignment="Center">  
</sunburst:DrilldownSettings>  
</sunburst:SfSunburstChart.DrilldownSettings>
```

#### C#

```
DrilldownSettings drilldownSettings = new DrilldownSettings();  
drilldownSettings.Enable = true;  
drilldownSettings.OffsetX = 0.5;  
drilldownSettings.OffsetY = 0.5;  
drilldownSettings.ToolbarHorizontalAlignment = ToolbarAlignment.Center;  
drilldownSettings.ToolbarVerticalAlignment = ToolbarAlignment.Center;  
sunburstChart.DrilldownSettings = drilldownSettings;
```



## Selection

The sunburst chart provides support to select or highlight the segments. Selection can be enabled with the help of [EnableSelection](#) property.

### Selection type

The [SelectionType](#) property allows you to select a segment based on the following categories:

- Child: Highlights the selected segment along with its children in all levels.
- Group: Highlights the entire group of the selected segment in a hierarchy.
- Parent: Highlights the parent of the selected segment in the hierarchy.
- Single: Highlights the selected segment alone.

The following code shows the **Child** selection type.

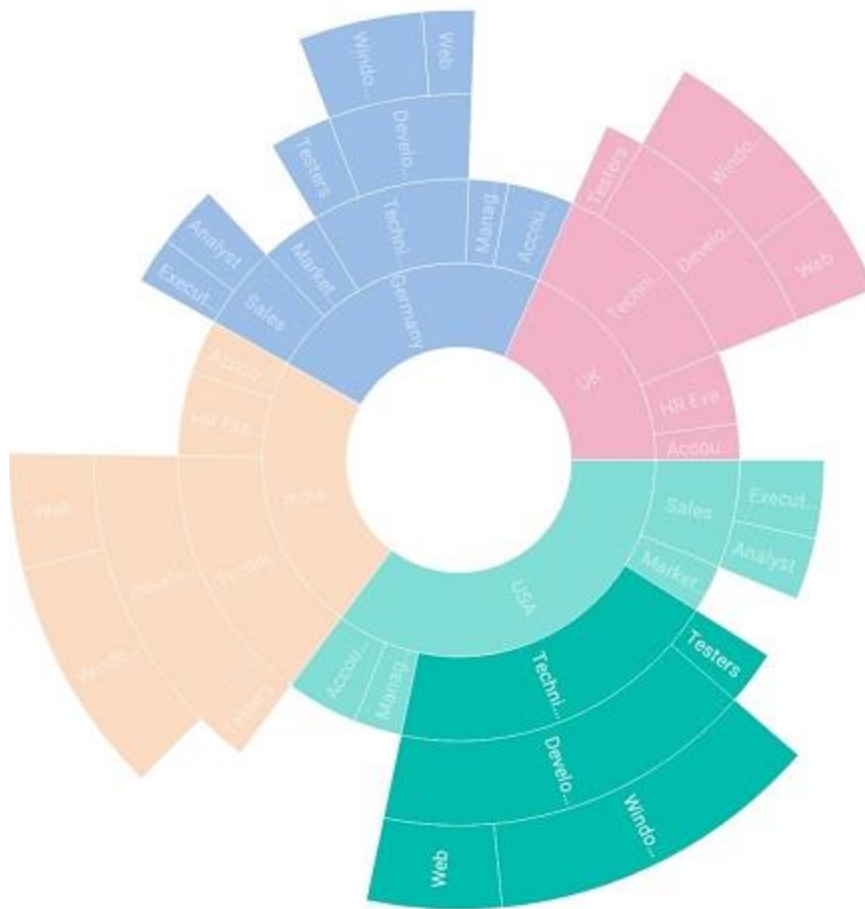
### XML

```
<sunburst:SfSunburstChart.SelectionSettings>  
<sunburst:SelectionSettings Opacity="0.5" EnableSelection="True"  
SelectionType="Child"></sunburst:SelectionSettings>  
</sunburst:SfSunburstChart.SelectionSettings>
```

### C#

```
SelectionSettings selection = new SelectionSettings();  
selection.Opacity = 0.5;  
selection.EnableSelection = true;  
selection.SelectionType = SelectionType.Child;  
sunburstChart.SelectionSettings = selection;
```





The following code shows the **Group** selection type.

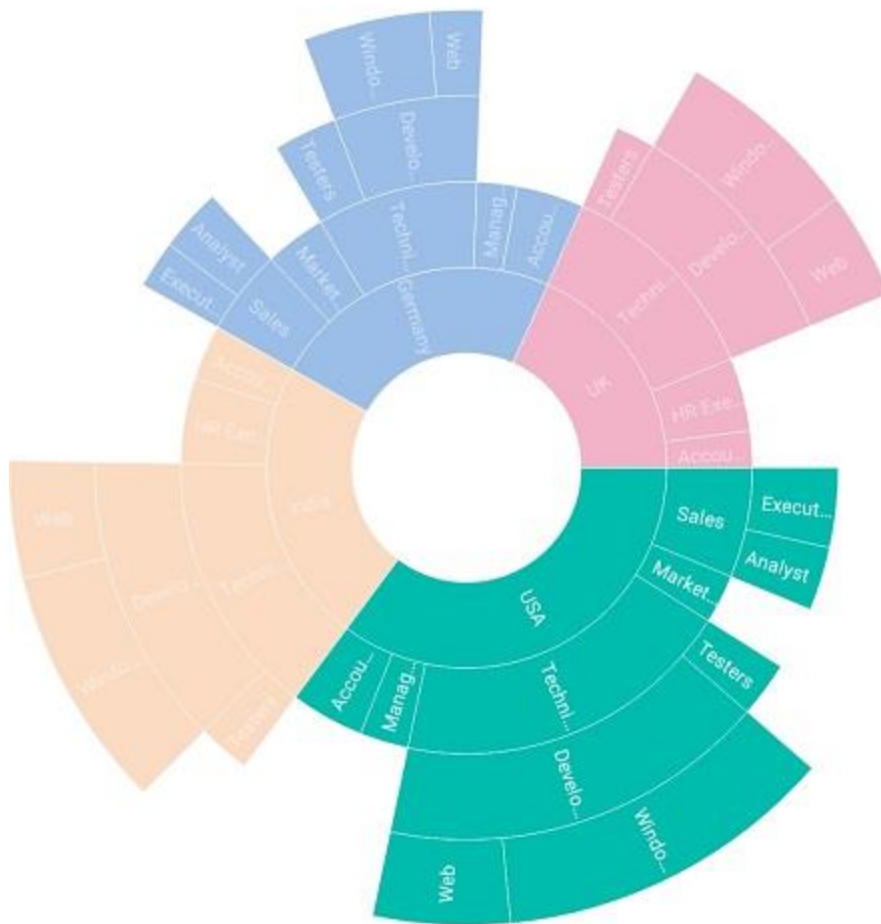
#### XML

```
<sunburst:SfSunburstChart.SelectionSettings>
<sunburst:SelectionSettings Opacity="0.5" EnableSelection="True"
SelectionType="Group"></sunburst:SelectionSettings>
</sunburst:SfSunburstChart.SelectionSettings>
```

#### C#

```
SelectionSettings selection = new SelectionSettings();
selection.Opacity = 0.5;
selection.EnableSelection = true;
selection.SelectionType = SelectionType.Group;
sunburstChart.SelectionSettings = selection;
```





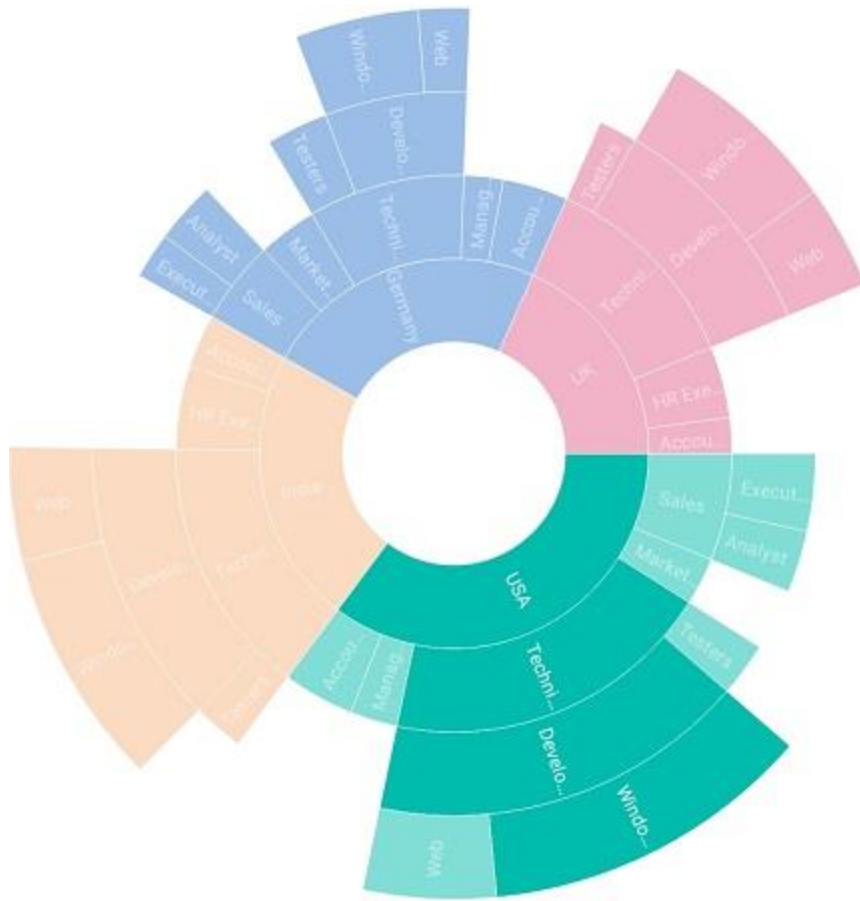
The following code shows the **Parent** selection type.

#### XML

```
<sunburst:SfSunburstChart.SelectionSettings>
<sunburst:SelectionSettings Opacity="0.5" EnableSelection="True"
SelectionType="Parent"></sunburst:SelectionSettings>
</sunburst:SfSunburstChart.SelectionSettings>
```

#### C#

```
SelectionSettings selection = new SelectionSettings();
selection.Opacity = 0.5;
selection.EnableSelection = true;
selection.SelectionType = SelectionType.Parent;
sunburstChart.SelectionSettings = selection;
```



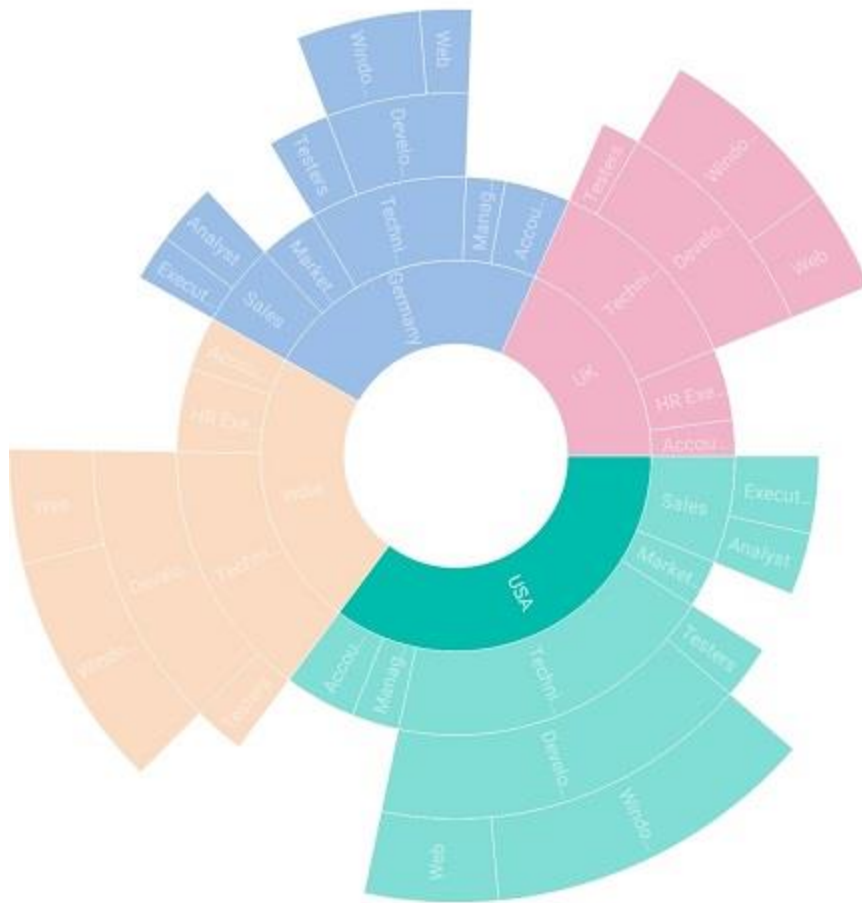
The following code shows the **Single** selection type.

#### XML

```
<sunburst:SfSunburstChart.SelectionSettings>
<sunburst:SelectionSettings Opacity="0.5" EnableSelection="True"
SelectionType="Single"></sunburst:SelectionSettings>
</sunburst:SfSunburstChart.SelectionSettings>
```

#### C#

```
SelectionSettings selection = new SelectionSettings();
selection.Opacity = 0.5;
selection.EnableSelection = true;
selection.SelectionType = SelectionType.Single;
sunburstChart.SelectionSettings = selection;
```



### Selection display mode

The [SelectionDisplayMode](#) property provides the following selection options to highlight the segments:

- By stroke
- By Color
- By opacity

### Opacity

This mode highlights the selected segment with the opacity specified in the [Opacity](#) property.

### XML

```
<sunburst:SfSunburstChart.SelectionSettings>
<sunburst:SelectionSettings Opacity="0.5" EnableSelection="True"
SelectionDisplayMode="HighlightByOpacity"
SelectionType="Group"></sunburst:SelectionSettings>
</sunburst:SfSunburstChart.SelectionSettings>
```

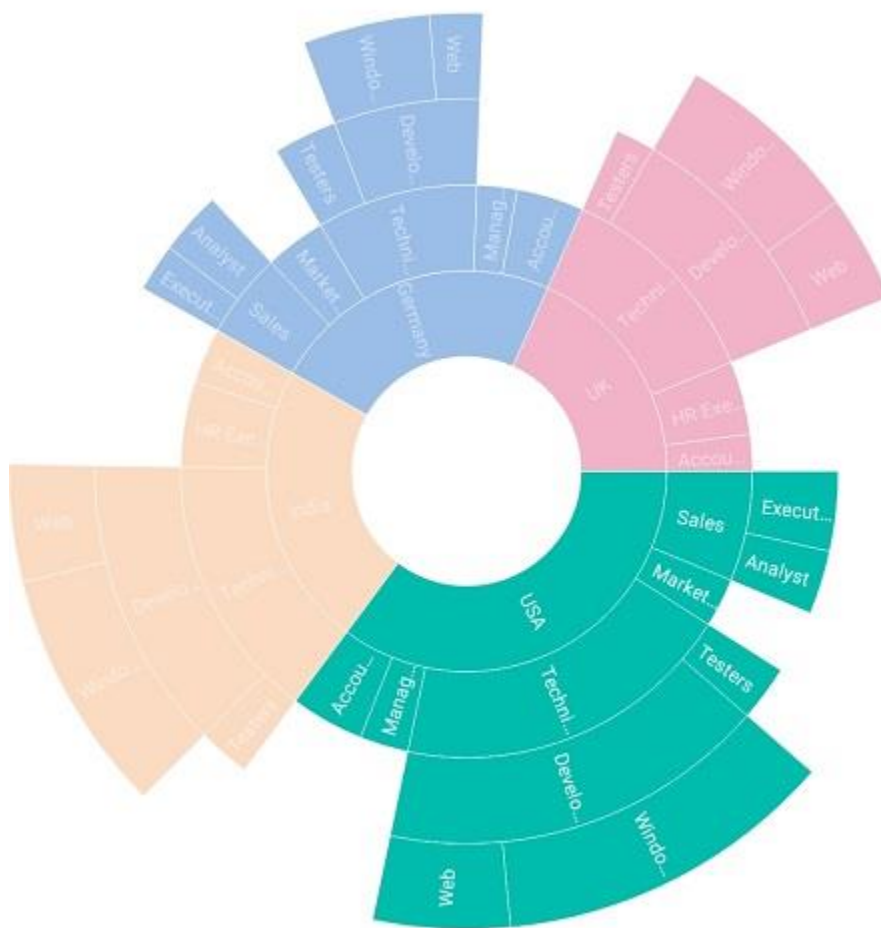
### C#

```
SelectionSettings selection = new SelectionSettings();
```

```

selection.EnableSelection = true;
selection.Opacity = 0.5;
selection.SelectionDisplayMode = SelectionDisplayMode.HighlightByOpacity;
selection.SelectionType = SelectionType.Group;
sunburstChart.SelectionSettings = selection;

```



### Color

This mode highlights the selected segment using the brush specified in the [SelectionBrush](#) property.

### XML

```

<sunburst:SfSunburstChart.SelectionSettings>
  <sunburst:SelectionSettings EnableSelection="True" SelectionBrush="Black"
    SelectionDisplayMode="HighlightByColor"
    SelectionType="Group"></sunburst:SelectionSettings>
</sunburst:SfSunburstChart.SelectionSettings>

```

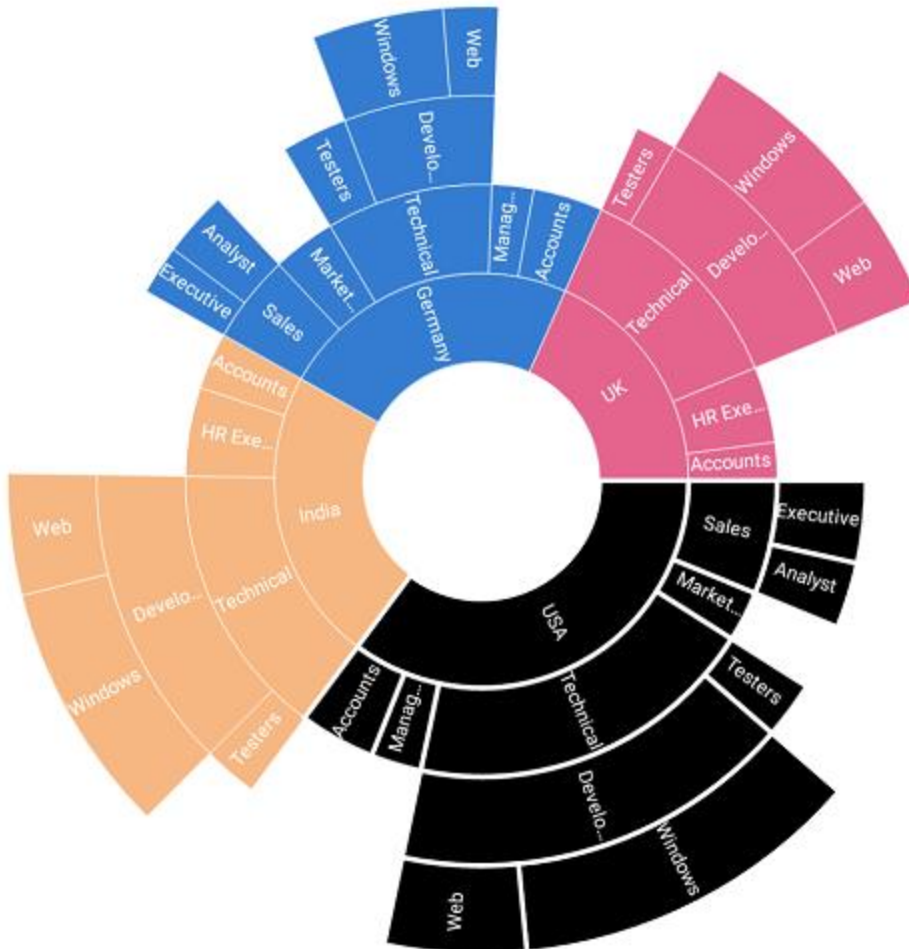
### C#

```

SelectionSettings selection = new SelectionSettings();
selection.EnableSelection = true;
selection.SelectionDisplayMode = SelectionDisplayMode.HighlightByColor;

```

```
selection.SelectionBrush = Color.Black;
selection.SelectionType = SelectionType.Group;
sunburstChart.SelectionSettings = selection;
```



### Stroke

This mode highlights the selected segment by applying stroke to it. The color and thickness of the stroke can be customized using the [SelectionStrokeBrush](#) and [SelectionStrokeWidth](#) properties.

### XML

```
<sunburst:SfSunburstChart.SelectionSettings>
  <sunburst:SelectionSettings EnableSelection="True"
    SelectionStrokeBrush="Black"
    SelectionDisplayMode="HighlightByStrokeColor"
    SelectionType="Group"></sunburst:SelectionSettings>
</sunburst:SfSunburstChart.SelectionSettings>
```

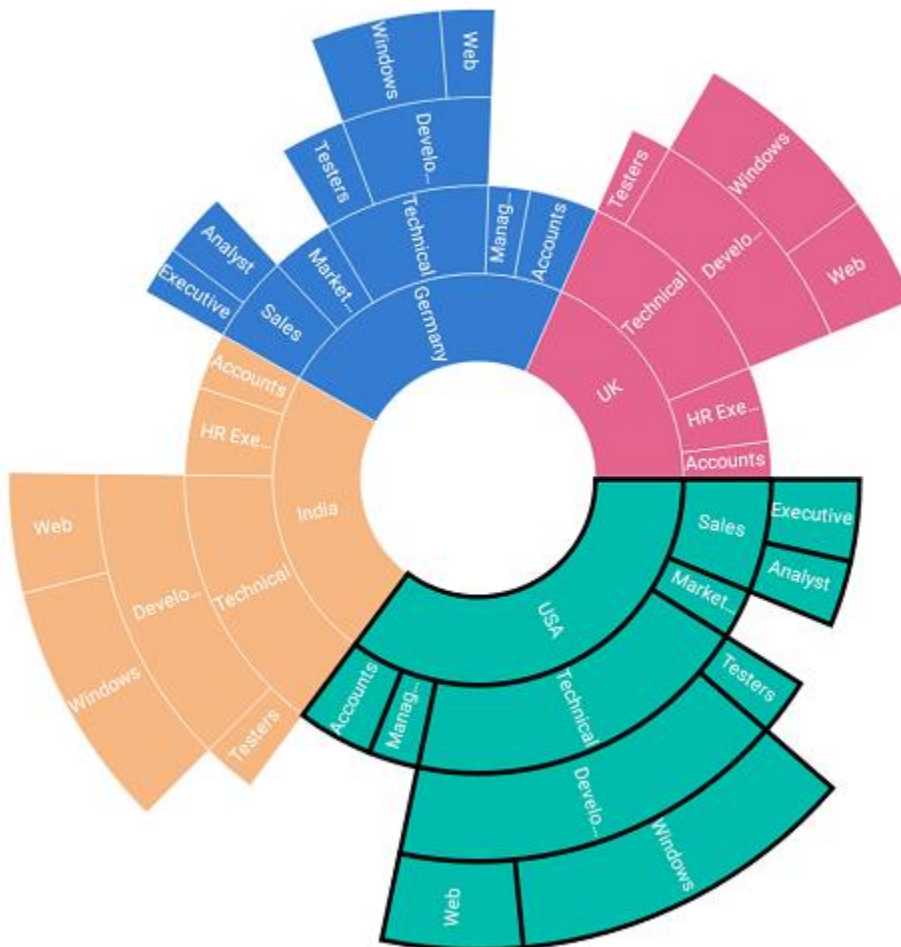
### C#

```
SelectionSettings selection = new SelectionSettings();
selection.EnableSelection = true;
```

```

selection.SelectionDisplayMode =
SelectionDisplayMode.HighlightByStrokeColor;
selection.SelectionStrokeBrush = Color.Black;
selection.SelectionType = SelectionType.Group;
sunburstChart.SelectionSettings = selection;

```



## Events

### *Selection Changed*

This event occurs whenever you select the segment. You can get the [SelectedSegment](#) and [IsSelected](#) properties details as argument from [SelectionChangedEventArgs](#) handler.

### XML

```

<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount"
SelectionChanged="SunburstChart_SelectionChanged">
<sunburst:SfSunburstChart.SelectionSettings>
<sunburst:SelectionSettings EnableSelection="True"
SelectionDisplayMode="HighlightByStrokeColor"
SelectionType="Single"></sunburst:SelectionSettings>
</sunburst:SfSunburstChart.SelectionSettings>

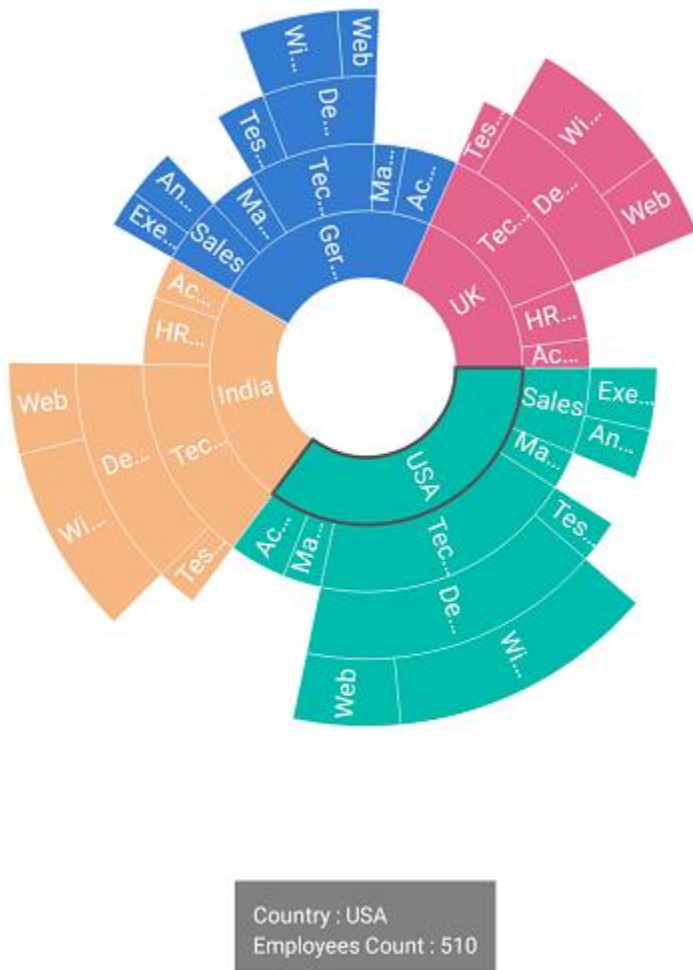
```

```
</sunburst:SfSunburstChart>
<StackLayout x:Name="stackLayout" IsVisible="false" Orientation="Vertical"
Spacing="0" Opacity="0.5" BackgroundColor="Black"
Grid.Row="1" Padding="10" HorizontalOptions="Center" VerticalOptions="End" >
<Label x:Name="countryLabel" FontSize="12" TextColor="White"/>
<Label x:Name="populationLabel" FontSize="12" TextColor="White"/>
</StackLayout>
```

## C#

```
private void SunburstChart_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (e.SelectedSegment != null)
    {
        stackLayout.IsVisible = true;
        if (!e.IsSelected)
        {
            stackLayout.IsVisible = false;
        }
        if (e.SelectedSegment.CurrentLevel == 0)
        {
            countryLabel.Text = "Country : " + e.SelectedSegment.Category;
            populationLabel.Text = "Employees Count : " + e.SelectedSegment.Value;
        }
        else if (e.SelectedSegment.CurrentLevel == 1)
        {
            countryLabel.Text = "JobDescription : " + e.SelectedSegment.Category;
            populationLabel.Text = "Employees Count : " + e.SelectedSegment.Value;
        }
        else if (e.SelectedSegment.CurrentLevel == 2)
        {
            countryLabel.Text = "JobGroup : " + e.SelectedSegment.Category;
            populationLabel.Text = "Employees Count : " + e.SelectedSegment.Value;
        }
        else
        {
            countryLabel.Text = "JobRole : " + e.SelectedSegment.Category;
            populationLabel.Text = "Employees Count : " + e.SelectedSegment.Value;
        }
    }
}
```





## Tooltip

Tooltip provides additional information about the segments in the sunburst chart. Tooltip is displayed by tapping the segment. By default, tooltip displays the corresponding segment's category and value. To enable the tooltip, set the [ShowTooltip](#) property to true.

The following code shows enabling the tooltip.

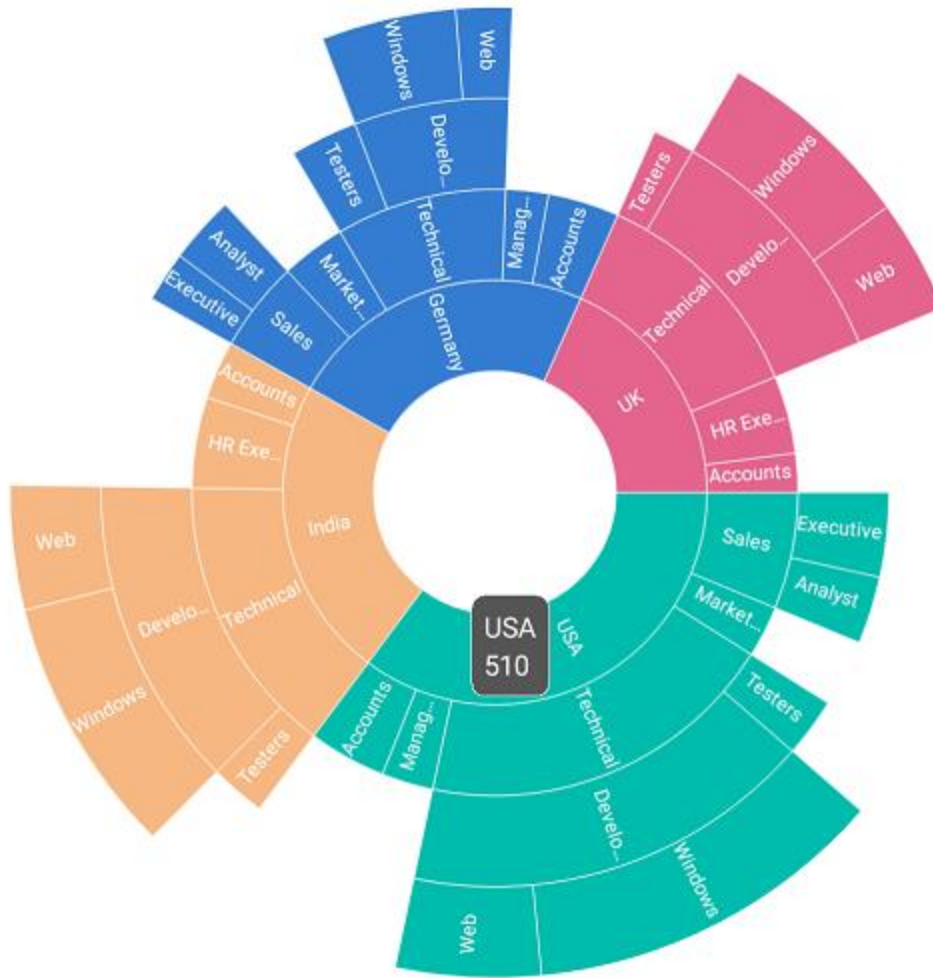
### XML

```
<sunburst:SfSunburstChart.TooltipSettings>
  <sunburst:SunburstTooltipSettings
    ShowTooltip="True"></sunburst:SunburstTooltipSettings>
</sunburst:SfSunburstChart.TooltipSettings>
```

### C#

```
SunburstTooltipSettings tooltipSettings = new SunburstTooltipSettings();
tooltipSettings.ShowTooltip = true;
sunburstChart.TooltipSettings = tooltipSettings;
```





### Customization

The appearance of the tooltip can be customized using the following properties:

- [TextColor](#): Customizes the text color of the tooltip.
- [BackgroundColor](#): Customizes the background color of the tooltip.
- [BorderColor](#): Customizes the border color of the tooltip.
- [BorderWidth](#): Customizes the border width of the tooltip.
- [Duration](#): Specifies the duration of the tooltip to be displayed.

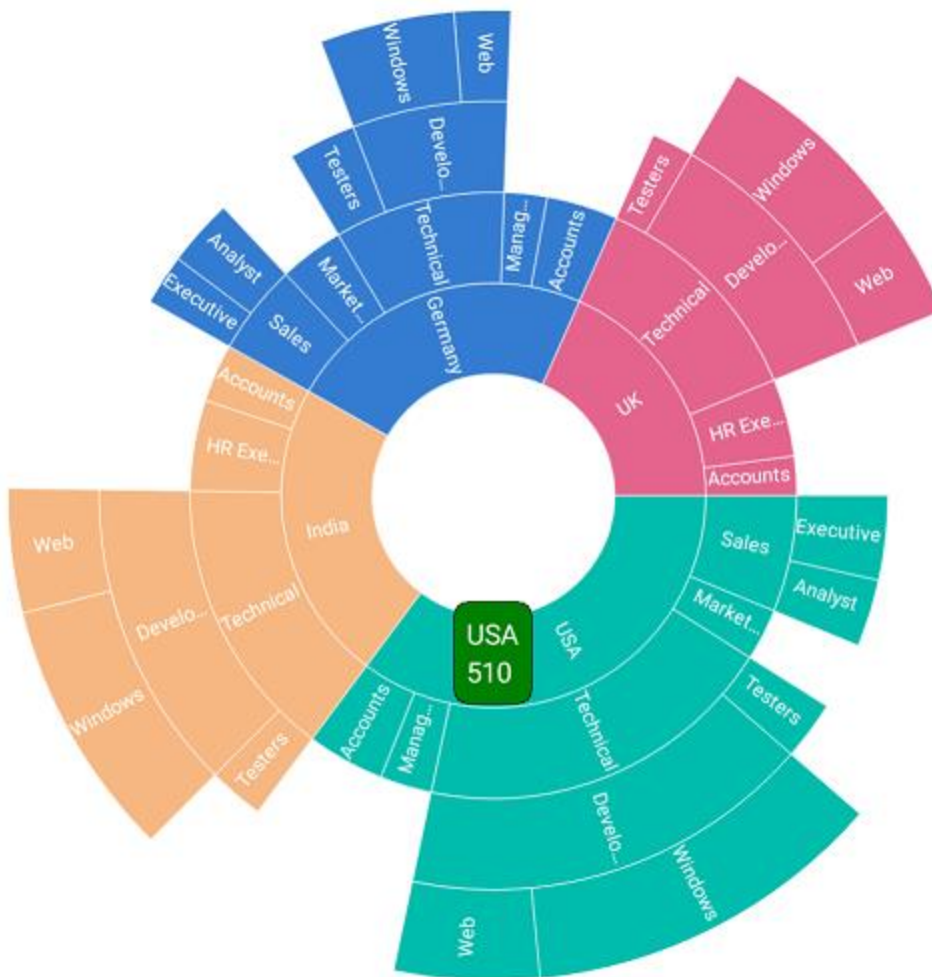
The following code shows all the above customizations.

### XML

```
<sunburst:SfSunburstChart.TooltipSettings>
<sunburst:SunburstTooltipSettings ShowTooltip="True" TextColor="White"
BackgroundColor="Green" BorderColor="Black" BorderWidth="1"
Duration="2000" ></sunburst:SunburstTooltipSettings>
</sunburst:SfSunburstChart.TooltipSettings>
```

### C#

```
SunburstTooltipSettings tooltipSettings = new SunburstTooltipSettings();
tooltipSettings.ShowTooltip = true;
tooltipSettings.TextColor = Color.White;
tooltipSettings.BackgroundColor = Color.Green;
tooltipSettings.BorderColor = Color.Black;
tooltipSettings.BorderWidth = 1;
tooltipSettings.Duration = 2000;
sunburstChart.TooltipSettings = tooltipSettings;
```



### Font customization

The font size can be customized using the [FontSize](#) property of tooltip.

### XML

```
<sunburst:SfSunburstChart.TooltipSettings>
<sunburst:SunburstTooltipSettings ShowTooltip="True" FontSize="20" />
</sunburst:SfSunburstChart.TooltipSettings>
```

### C#

```
SunburstTooltipSettings tooltipSettings = new SunburstTooltipSettings();
tooltipSettings.ShowTooltip = true;
```

```
tooltipSettings.FontSize = 20;
sunburstChart.TooltipSettings = tooltipSettings;
```



### Custom template

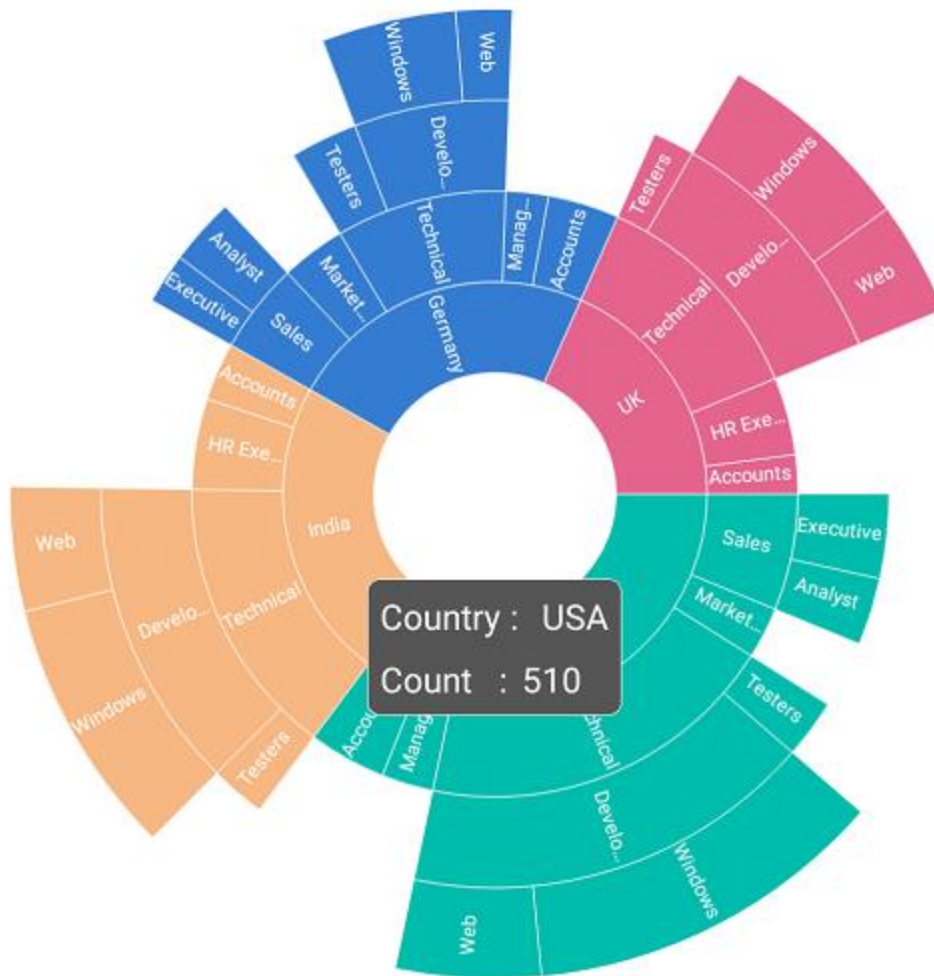
The sunburst chart provides options to design your own template for tooltip using the [TooltipTemplate](#) property.

### XML

```
<sunburst:SfSunburstChart.TooltipSettings>
  <sunburst:SunburstTooltipSettings ShowTooltip="True">
    <sunburst:SunburstTooltipSettings.TooltipTemplate>
      <DataTemplate>
        <StackLayout Orientation="Vertical">
          <StackLayout Orientation="Horizontal">
            <Label Text="Country : " TextColor="White"/>
            <Label Text="{Binding Category}" TextColor="White"/>
          </StackLayout>
          <StackLayout Orientation="Horizontal">
            <Label Text="Count : " TextColor="White"/>
            <Label Text="{Binding Value}" TextColor="White"/>
          </StackLayout>
        </StackLayout>
      </DataTemplate>
    </sunburst:SunburstTooltipSettings.TooltipTemplate>
  </sunburst:SunburstTooltipSettings>
</sunburst:SfSunburstChart.TooltipSettings>
```

**C#**

```
SunburstTooltipSettings tooltipSettings = new SunburstTooltipSettings();
tooltipSettings.ShowTooltip = true;
DataTemplate template = new DataTemplate(() =>
{
    StackLayout stack = new StackLayout() { Orientation =
    StackOrientation.Vertical };
    StackLayout categoryLayout = new StackLayout() { Orientation =
    StackOrientation.Horizontal };
    Label label = new Label() { Text = "Country :" };
    label.TextColor = Color.White;
    Label category = new Label();
    category.TextColor = Color.White;
    category.SetBinding(Label.TextProperty, "Category");
    categoryLayout.Children.Add(label);
    categoryLayout.Children.Add(category);
    StackLayout valueLayout = new StackLayout() { Orientation =
    StackOrientation.Horizontal };
    Label label1 = new Label() { Text = "Count :" };
    label1.TextColor = Color.White;
    Label value = new Label();
    value.TextColor = Color.White;
    value.SetBinding(Label.TextProperty, "Value");
    valueLayout.Children.Add(label1);
    valueLayout.Children.Add(value);
    stack.Children.Add(categoryLayout);
    stack.Children.Add(valueLayout);
    return stack;
});
tooltipSettings.TooltipTemplate = template;
sunburstChart.TooltipSettings = tooltipSettings;
```



## Animation

The sunburst chart provides animation on loading and whenever the item source changes. Animation can be enabled by setting the [EnableAnimation](#) property to true.

The following code shows enabling animation.

### XML

```
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount" EnableAnimation="True">
</sunburst:SfSunburstChart>
```

### C#

```
SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburstChart.ValueMemberPath = "EmployeesCount";
sunburstChart.EnableAnimation = true;
this.Content = sunburstChart;
```

## Duration

Animation duration can be controlled using the [AnimationDuration](#) property.

### XML

```
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}" AnimationDuration="2"
ValueMemberPath="EmployeesCount" EnableAnimation="True">
</sunburst:SfSunburstChart>
```

### C#

```
SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburstChart.ValueMemberPath = "EmployeesCount";
sunburstChart.EnableAnimation = true;
sunburstChart.AnimationDuration = 2;
this.Content = sunburstChart;
```

Below snippet is the complete code for generating the following output.

### XML

```
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}" AnimationDuration="2"
ValueMemberPath="EmployeesCount" EnableAnimation="True">
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel"
ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
</sunburst:SfSunburstChart>
```

### C#

```
SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburstChart.ValueMemberPath = "EmployeesCount";
sunburstChart.EnableAnimation = true;
sunburstChart.AnimationDuration = 2;
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
sunburstChart.DataLabel = label;
```

```
this.Content = sunburstChart;
```

The following screenshot depicts animation of the sunburst chart with the specified duration.

## Customization

The sunburst chart provides various customizing and styling options to enrich the application.

### Palettes

The sunburst chart provides options to apply different kinds of palettes using the [ColorModel](#).

The following palettes are available in the sunburst chart:

- Metro
- Natural
- Pineapple

- TomatoSpectrum
- Custom

The following code shows applying the TomatoSpectrum [Palette](#).

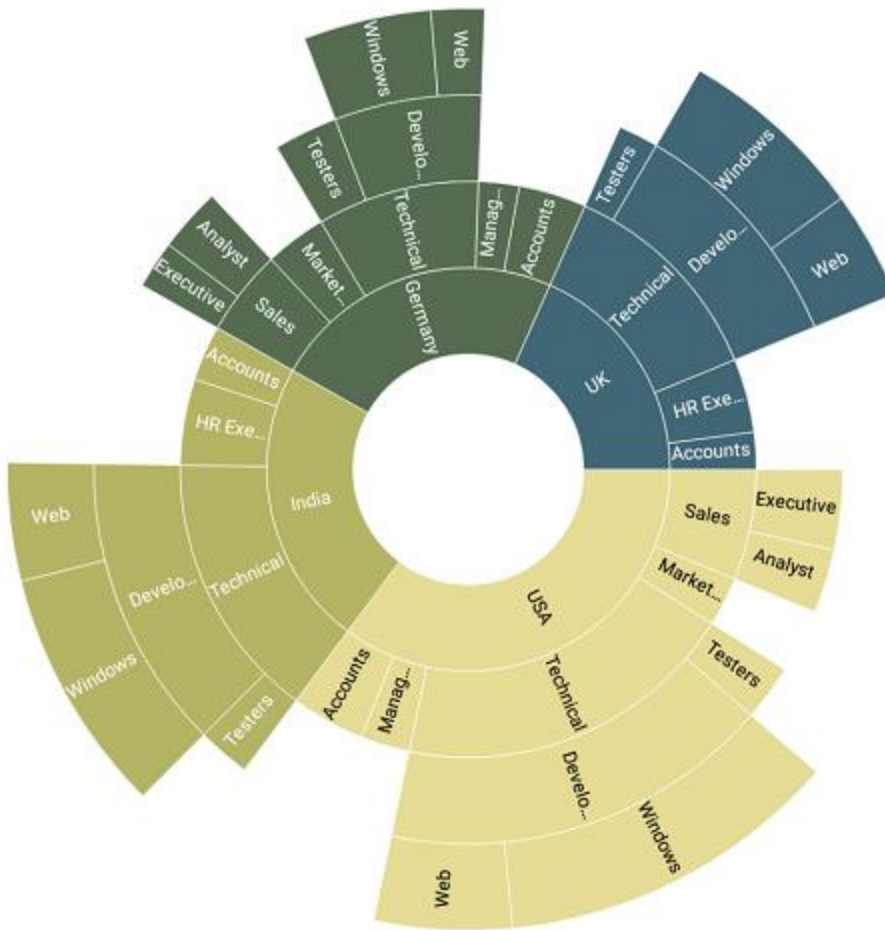
#### XML

```
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount" >
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel"
ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
<sunburst:SfSunburstChart.ColorModel>
<sunburst:SunburstChartColorModel
Palette="TomatoSpectrum"></sunburst:SunburstChartColorModel>
</sunburst:SfSunburstChart.ColorModel>
</sunburst:SfSunburstChart>
```

#### C#

```
SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburstChart.ValueMemberPath = "EmployeesCount";
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
SunburstChartColorModel colorModel = new SunburstChartColorModel();
colorModel.Palette = SunburstColorPalette.TomatoSpectrum;
sunburstChart.ColorModel = colorModel;
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
sunburstChart.DataLabel = label;
this.Content = sunburstChart;
```





### Angle

The start angle and end angle of the sunburst chart can be adjusted by using the [StartAngle](#) and [EndAngle](#) properties.

### XML

```
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount" StartAngle="180" EndAngle="360">
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel"
ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
</sunburst:SfSunburstChart>
```

### C#

```

SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburstChart.ValueMemberPath = "EmployeesCount";
sunburstChart.StartAngle = 180;
sunburstChart.EndAngle = 360;
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
sunburstChart.DataLabel = label;
this.Content = sunburstChart;

```



### Radius

The sunburst chart allows you to customize the radius by using the [Radius](#) property. The default value of this property is 0.9, and the value ranges from 0 to 1.

### XML

```

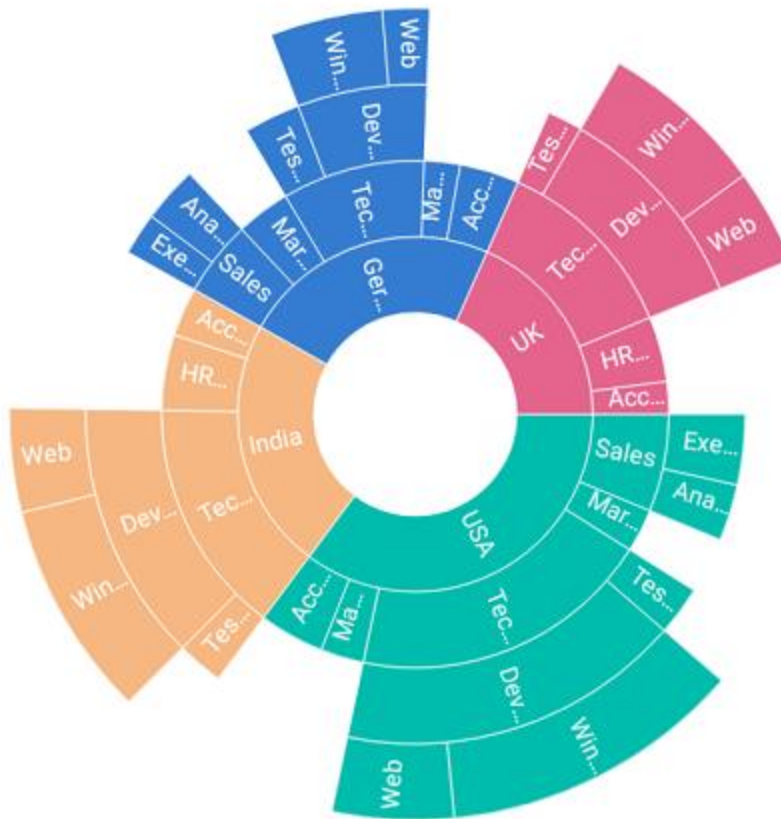
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount" Radius="0.6">
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>

```

```
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel"
ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
</sunburst:SfSunburstChart>
```

## C#

```
SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburstChart.ValueMemberPath = "EmployeesCount";
sunburstChart.Radius = 0.6;
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
sunburstChart.DataLabel = label;
this.Content = sunburstChart;
```



### Inner radius

The sunburst chart allows you to customize the inner radius using the [InnerRadius](#) property. The default value of this property is 0.2, and the value ranges from 0 to 1.

### XML

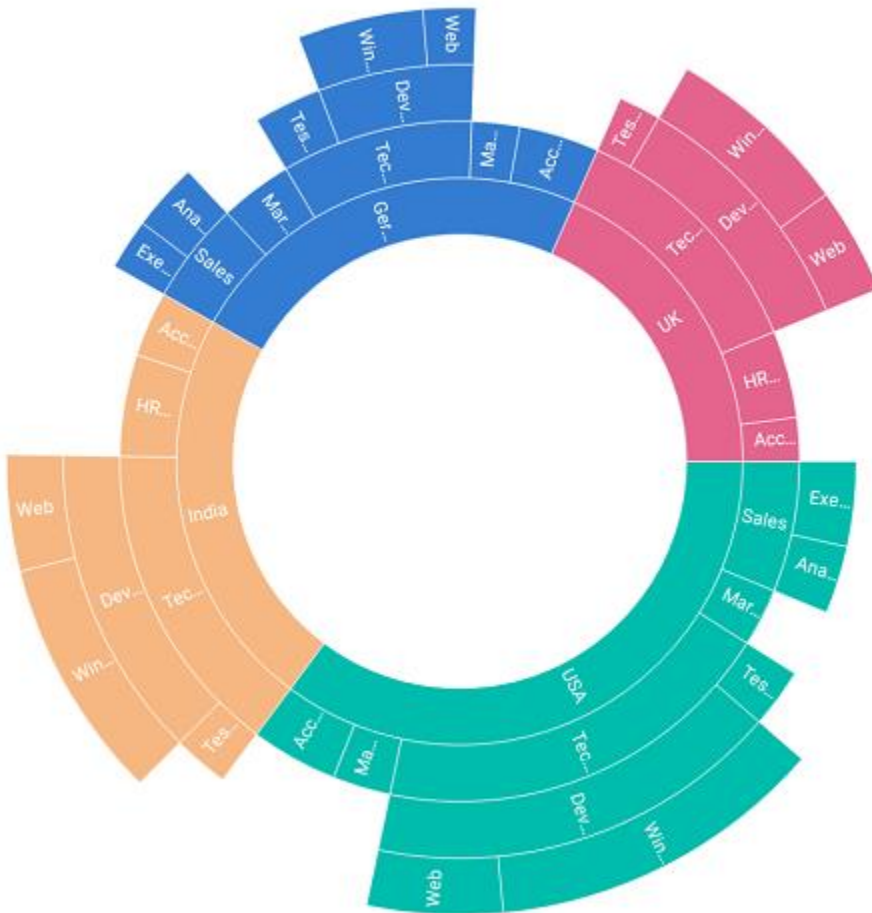
```
<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount" InnerRadius="0.5">
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel"
ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
</sunburst:SfSunburstChart>
```

### C#

```

SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburstChart.ValueMemberPath = "EmployeesCount";
sunburstChart.InnerRadius = 0.5;
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
sunburstChart.DataLabel = label;
this.Content = sunburstChart;

```



### Stroke customization

Stroke color and stroke width of the sunburst chart can be customized using [StrokeColor](#) and [StrokeWidth](#) properties respectively.

### XML

```

<sunburst:SfSunburstChart x:Name="sunburstChart" ItemsSource="{Binding
DataSource}"
ValueMemberPath="EmployeesCount" StrokeColor="Black" StrokeWidth="2">
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
<sunburst:SfSunburstChart.DataLabel>
<sunburst:SunburstChartDataLabel x:Name="dataLabel"
ShowLabel="True"></sunburst:SunburstChartDataLabel>
</sunburst:SfSunburstChart.DataLabel>
</sunburst:SfSunburstChart>

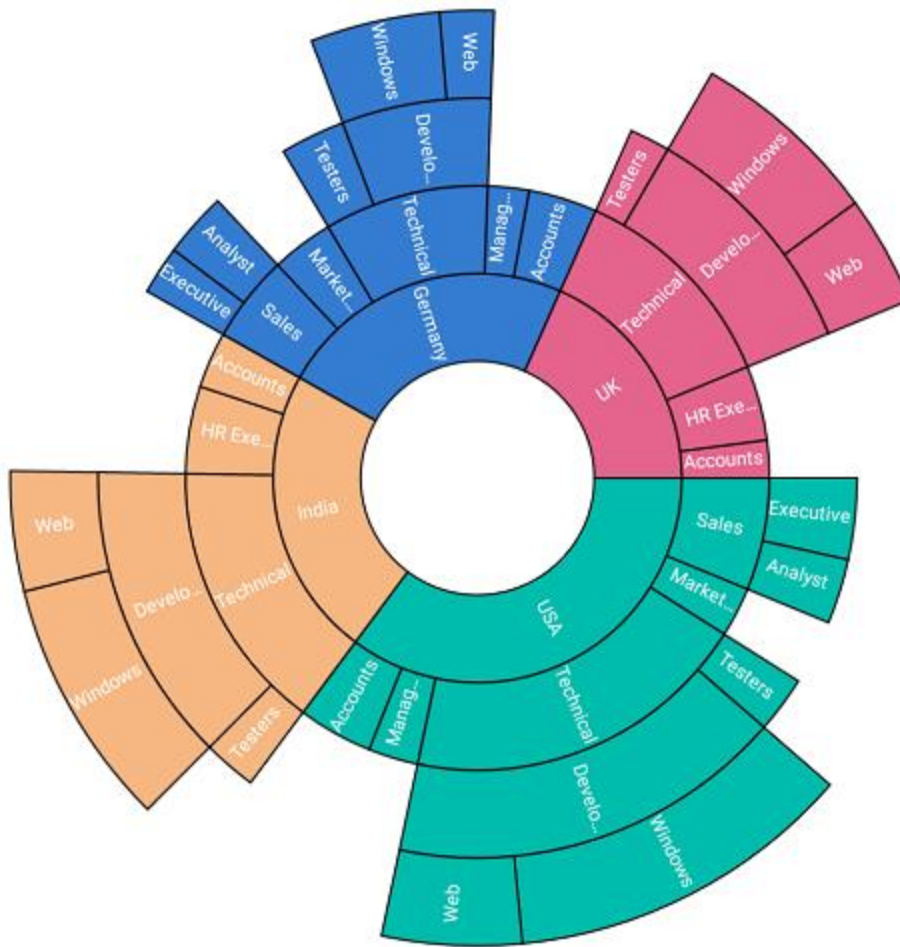
```

**C#**

```

SfSunburstChart sunburstChart = new SfSunburstChart();
sunburstChart.SetBinding(SfSunburstChart.ItemsSourceProperty, "DataSource");
sunburstChart.ValueMemberPath = "EmployeesCount";
sunburstChart.StrokeColor = Color.Black;
sunburstChart.StrokeWidth = 2;
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburstChart.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
SunburstChartDataLabel label = new SunburstChartDataLabel();
label.ShowLabel = true;
sunburstChart.DataLabel = label;
this.Content = sunburstChart;

```



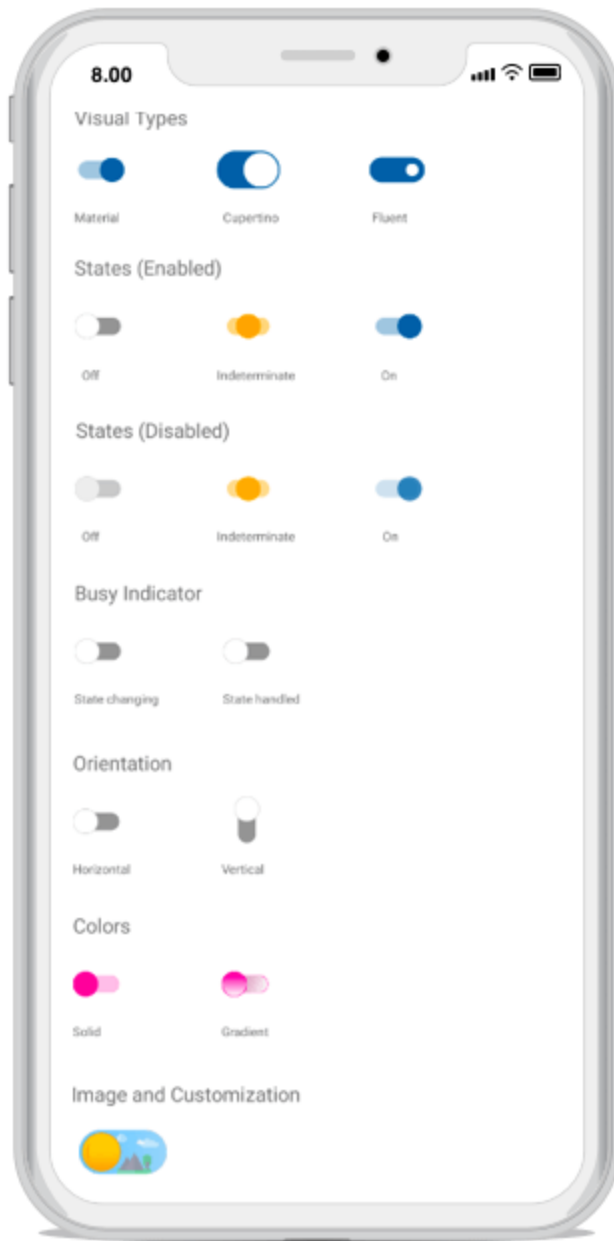
## SfSwitch

### SfSwitch

#### Overview

Essential SfSwitch for Xamarin.Forms provides an efficient way to select between the states based on the toggled value.





### Key features

- Provides support for inbuilt visual types like Material, Cupertino or fluent type.
- Customizable visual based on device platforms.
- Provides options to visualize indeterminate state.
- Performs async action with busy indicator.
- Supports horizontal and vertical orientation.
- Allows you to add the images to enhance the look and feel.



## Getting Started

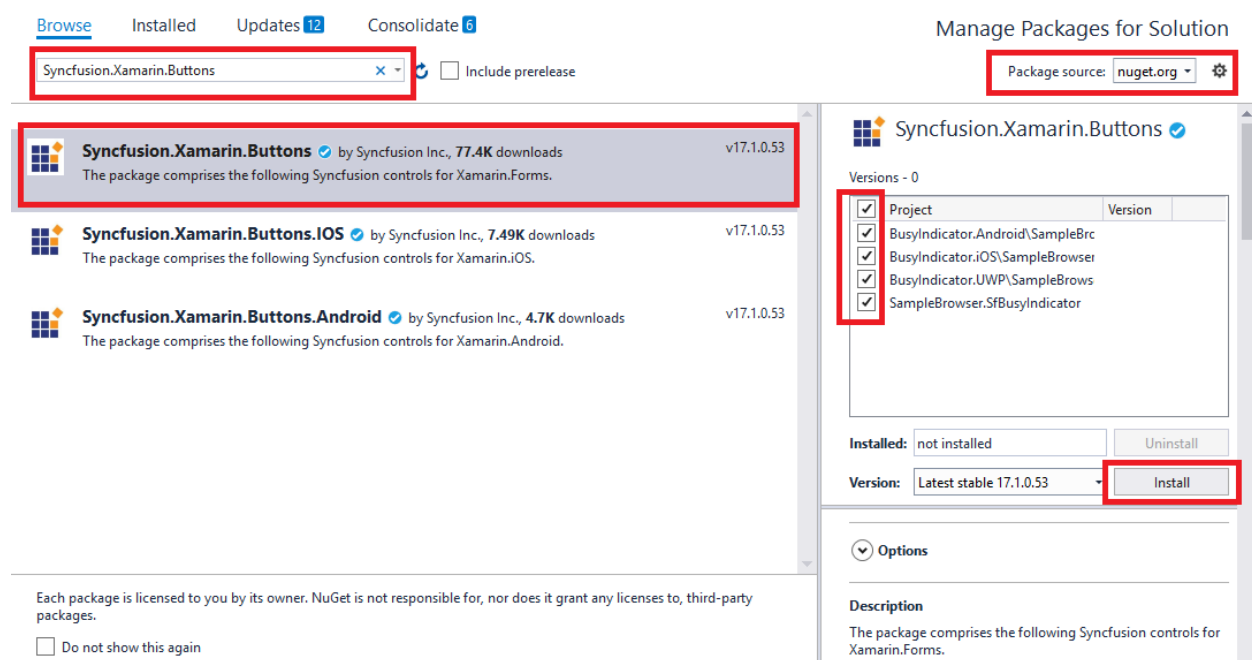
This section explains you the steps to add the SfSwitch control with basic functionalities in Xamarin.Forms.

### Adding SfSwitch reference

You can add SfSwitch reference using one of the below methods.

#### Method 1: Adding SfSwitch reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfSwitch to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Core](#), and then install it.



#### Method 2: Adding SfSwitch reference from toolbox

Syncfusion provides Xamarin Toolbox. Using this toolbox, you can drag the SfSwitch control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfSwitch assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Buttons.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.Android.dll Syncfusion.Core.XForms.dll

	Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Buttons.XForms.dll Syncfusion.Buttons.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

#### Launching an application on each platform with SfSwitch

To use the SfSwitch inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and is discussed in the following sections:

**Note:** If you are adding the references from toolbox, below steps are not needed.

#### iOS

To launch the switch in iOS, call the `SfSwitchRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms framework initialization and before the `LoadApplication` method is called as demonstrated in the following code sample:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    Syncfusion.XForms.iOS.Buttons.SfSwitchRenderer.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

**Note:** If you are adding the references from toolbox, this step is not needed.

#### Universal Windows Platform (UWP)

To deploy the switch in Release mode, you need to initialize the button assemblies in App.xaml.cs in UWP project as shown in the below code snippets.

### **C#**

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    .....
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Add using System.Reflection;
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies that your app uses.
    assembliesToInclude.Add(typeof(SfBorderRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    .....
}
```

### **Android**

The Android platform does not require any additional configuration to render the chart.

#### Initializing SfSwitch

Import the Button namespace as shown below in your respective Page,

### **XML**

```
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
```

### **C#**

```
using Syncfusion.XForms.Buttons;
```

Then initialize the switch control as shown below using the code example.

### **XML**

```
<syncfusion:SfSwitch />
```

### **C#**

```
SfSwitch sfSwitch = new SfSwitch();
```



ANDROID



iOS



UWP

### Performing an action based on state

You can switch between the states. When the state is changed the `StateChanging` and `StateChanged` event will be triggered where you can perform an action based on the current state. The `StateChanging` event allows you to cancel moving to a new state.

The following code example displays a message box when switched to off state when work is completed.

#### XML

```
<syncfusion:SfSwitch StateChanged="SfSwitch_StateChanged" />
```

#### C#

```
sfSwitch.StateChanged += SfSwitch_StateChanged;
```

#### C#

```
private void SfSwitch_StateChanged(object sender,
SwitchStateChangedEventArgs e)
{
    DisplayAlert("Message", "SUCCESS", "OK");
}
```

You can find the complete getting started sample from this [link](#).

### Visual Types

SfSwitch supports customization using built-in visual types. The visual types based on device platform are listed as follows.

- Android – Material
- iOS – Cupertino
- Windows – Fluent

### Default

This is the default value set for visual type.



ANDROID



iOS



UWP

---

**Note:** If you set default, the visual type will be internally changed based on the device platform.

---

### Material

Material visual type brings the appearance based on material guidelines. The following code example demonstrates how to define material visual type.

#### XML

```
<syncfusion:SfSwitch VisualType="Material" />
```

#### C#

```
SfSwitch sfSwitch=new SfSwitch();  
sfSwitch.VisualType = VisualType.Material;
```



### Cupertino

Cupertino visual type brings the appearance based on Cupertino guidelines. The following code example demonstrates how to define Cupertino visual type.

#### XML

```
<syncfusion:SfSwitch VisualType="Cupertino" />
```

#### C#

```
SfSwitch sfSwitch=new SfSwitch();  
sfSwitch.VisualType = VisualType.Cupertino;
```



### Fluent

Fluent visual type brings the appearance based on Fluent guidelines. The following code example demonstrates how to define Fluent visual type.

#### XML

```
<syncfusion:SfSwitch VisualType="Fluent" />
```

#### C#

```
SfSwitch sfSwitch=new SfSwitch();  
sfSwitch.VisualType = VisualType.Fluent;
```



### Custom

Custom type will allow you to customize the control, where you can handle the size, colors, images etc. of the control. Refer to this [documentation](#).

### States

Switch allows you to configure the states as explained in the following sections.

#### On

You can switch to on state by tapping the switch button or by setting a value as demonstrated in the following code example.

#### XML

```
<syncfusion:SfSwitch IsOn="True" />
```

#### C#

```
SfSwitch sfSwitch=new SfSwitch();
```

```
sfSwitch.IsOn=true;
```



ANDROID



iOS



UWP

### Off

This is the default state. You can switch to off state by tapping the switch button or by defining as demonstrated in the following code example.

#### XML

```
<syncfusion:SfSwitch IsOn="False" />
```

#### C#

```
SfSwitch sfSwitch = new SfSwitch();  
sfSwitch.IsOn = false;
```



ANDROID



iOS



UWP

### Indeterminate

The indeterminate state can be enabled when you need to display the work progress. The following code example demonstrates how to load the switch in indeterminate state by setting the IsOn property to null.

#### XML

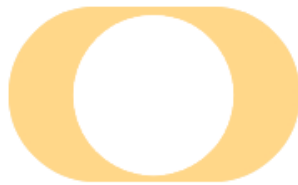
```
<syncfusion:SfSwitch IsOn="{x:Null}" AllowIndeterminateState="True" />
```

#### C#

```
SfSwitch sfSwitch = new SfSwitch();  
sfSwitch.IsOn = null;  
sfSwitch.AllowIndeterminateState = true;
```



ANDROID



iOS



UWP

---

**Note:** By default, the switch control has only two states: on and off.

---

#### Disabled On

You can switch to disabled on state by setting the `IsOn` property as true and `IsEnabled` property as false.

#### XML

```
<syncfusion:SfSwitch IsOn="True" IsEnabled="False" />
```

#### C#

```
SfSwitch sfSwitch = new SfSwitch();  
sfSwitch.IsOn = true;  
sfSwitch.IsEnabled = false;
```



ANDROID



iOS



UWP

#### Disabled Off

You can switch to disabled off state by setting the `IsOn` property as false and `IsEnabled` property as false.

#### XML

```
<syncfusion:SfSwitch IsOn="False" IsEnabled="False" />
```

#### C#

```
SfSwitch sfSwitch = new SfSwitch();  
sfSwitch.IsOn = false;
```



```
sfSwitch.IsEnabled = false;
```



ANDROID



iOS



UWP

### Disabled Indeterminate

The disabled indeterminate state can be enabled when you need to display the work progress. The below code example demonstrates loading the switch in disabled indeterminate state by setting `IsOn` property value as null and `IsEnabled` property as false.

#### XML

```
<syncfusion:SfSwitch AllowIndeterminateState="True" IsOn="{x:Null}"  
IsEnabled="False"/>
```

#### C#

```
SfSwitch sfSwitch = new SfSwitch();  
sfSwitch.AllowIndeterminateState = true;  
sfSwitch.IsOn = null;  
sfSwitch.IsEnabled = false;
```



ANDROID



iOS



UWP

### Orientation

The switch control provides options to change the default orientation.

#### Horizontal

By default, it is displayed horizontally. You can also define the orientation as demonstrated in the following code example.

#### XML

```
<syncfusion:SfSwitch Orientation="Horizontal" />
```

**C#**

```
SfSwitch sfSwitch = new SfSwitch();  
sfSwitch.Orientation = SwitchOrientation.Horizontal;
```

**ANDROID****iOS****UWP****Vertical**

To view the switch control vertically, you can define the vertical orientation as demonstrated in the following code example.

**XML**

```
<syncfusion:SfSwitch Orientation="Vertical" />
```

**C#**

```
SfSwitch sfSwitch = new SfSwitch();  
sfSwitch.Orientation = SwitchOrientation.Vertical;
```

**ANDROID****iOS****UWP****Customization with visual states**

The switch control provides options to customize the color based on states. The following code example demonstrates how to customize the switch control.

**Solid colors**

The following properties are used to apply solid colors to the thumb, track, border, and busy indicator, respectively:

- ThumbColor: Represents the color for the thumb.
- ThumbBorderColor: Represents the border color for the thumb.
- TrackBorderColor: Represents the color for the border of the track.
- TrackColor: Represents the color for the track.

### XML

```
<syncfusion:SfSwitch VisualType="Custom">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="On">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OnState"
                ThumbBorderColor="DarkCyan" ThumbColor="Aqua"
                TrackBorderColor="Green" TrackColor="GreenYellow"
                BusyIndicatorColor="Coral" />
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
      <VisualState x:Name="Off">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OffState"
                ThumbBorderColor="DarkCyan" ThumbColor="Aqua"
                TrackBorderColor="Green" TrackColor="GreenYellow" />
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
      <VisualState x:Name="Indeterminate">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings
                x:TypeArguments="syncfusion:IndeterminateState"
                ThumbBorderColor="DarkCyan" TrackBorderColor="Green"
                ThumbColor="Aqua" TrackColor="GreenYellow" />
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</syncfusion:SfSwitch>
```

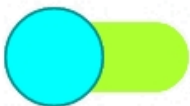
### C#

```
SfSwitch sfSwitch = new SfSwitch();
sfSwitch.VisualType = VisualType.Custom;
DefaultSwitchSettings<OnState> defaultSwitch = new
DefaultSwitchSettings<OnState>();
```

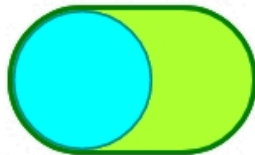
```

defaultSwitch.ThumbBorderColor = Color.DarkCyan;
defaultSwitch.ThumbColor = Color.Aqua;
defaultSwitch.TrackBorderColor = Color.Green;
defaultSwitch.TrackColor = Color.GreenYellow;
defaultSwitch.BusyIndicatorColor = Color.Coral;
DefaultSwitchSettings<OffState> defaultSwitch1 = new
DefaultSwitchSettings<OffState>();
defaultSwitch1.ThumbBorderColor = Color.DarkCyan;
defaultSwitch1.ThumbColor = Color.Aqua;
defaultSwitch1.TrackBorderColor = Color.Green;
defaultSwitch1.TrackColor = Color.GreenYellow;
DefaultSwitchSettings<IndeterminateState> defaultSwitch2 = new
DefaultSwitchSettings<IndeterminateState>();
defaultSwitch2.ThumbBorderColor = Color.DarkCyan;
defaultSwitch2.ThumbColor = Color.Aqua;
defaultSwitch2.TrackBorderColor = Color.Green;
defaultSwitch2.TrackColor = Color.GreenYellow;
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState onState = new VisualState
{
    Name = "On"
};
onState.Setters.Add(new Setter { Property = SfSwitch.SwitchSettingsProperty,
Value= defaultSwitch});
VisualState offState = new VisualState
{
    Name = "Off"
};
offState.Setters.Add(new Setter { Property =
SfSwitch.SwitchSettingsProperty, Value = defaultSwitch1 });
VisualState indeterminate = new VisualState
{
    Name = "Indeterminate"
};
indeterminate.Setters.Add(new Setter { Property =
SfSwitch.SwitchSettingsProperty, Value = defaultSwitch2 });
commonStateGroup.States.Add(onState);
commonStateGroup.States.Add(offState);
commonStateGroup.States.Add(indeterminate);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(sfSwitch, visualStateGroupList);
this.Content = sfSwitch;

```



ANDROID



iOS



UWP

## Gradients

You can also specify a range of colors in thumb and track using ThumbGradient and TrackGradient as demonstrates in the following code example.

### XML

```
<syncfusion:SfSwitch VisualType="Custom" >
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="On">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OnState">
                <syncfusion:DefaultSwitchSettings.ThumbGradient>
                  <graphics:SfLinearGradientBrush>
                    <graphics:SfLinearGradientBrush.GradientStops>
                      <graphics:GradientStopCollection>
                        <graphics:SfGradientStop Color="White" Offset="0"></graphics:SfGradientStop>
                        <graphics:SfGradientStop Color="LightSkyBlue"
                          Offset="0.5"></graphics:SfGradientStop>
                        <graphics:SfGradientStop Color="White" Offset="1"></graphics:SfGradientStop>
                      </graphics:GradientStopCollection>
                    </graphics:SfLinearGradientBrush.GradientStops>
                  </graphics:SfLinearGradientBrush>
                </syncfusion:DefaultSwitchSettings.ThumbGradient>
                <syncfusion:DefaultSwitchSettings.TrackGradient>
                  <graphics:SfLinearGradientBrush>
                    <graphics:SfLinearGradientBrush.GradientStops>
                      <graphics:GradientStopCollection>
                        <graphics:SfGradientStop Color="White" Offset="0"></graphics:SfGradientStop>
                        <graphics:SfGradientStop Color="LightCoral"
                          Offset="0.25"></graphics:SfGradientStop>
                        <graphics:SfGradientStop Color="Red" Offset="0.5"></graphics:SfGradientStop>
                        <graphics:SfGradientStop Color="LightSalmon"
                          Offset="0.75"></graphics:SfGradientStop>
                      </graphics:GradientStopCollection>
                    </graphics:SfLinearGradientBrush.GradientStops>
                  </graphics:SfLinearGradientBrush>
                </syncfusion:DefaultSwitchSettings.TrackGradient>
              </syncfusion:DefaultSwitchSettings>
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</syncfusion:SfSwitch>
```

### C#

```
SfSwitch sfSwitch = new SfSwitch();
sfSwitch.VisualType = VisualType.Custom;
DefaultSwitchSettings<OnState> defaultSwitch = new
DefaultSwitchSettings<OnState>();
SfLinearGradientBrush sfLinearGradientBrush = new SfLinearGradientBrush();
```

```
GradientStopCollection sfGradientStops = new GradientStopCollection();
SfGradientStop sfGradientStop = new SfGradientStop();
sfGradientStop.Color = Color.White;
sfGradientStop.Offset = 0;
sfGradientStops.Add(sfGradientStop);
SfGradientStop sfGradientStop1 = new SfGradientStop();
sfGradientStop1.Color = Color.LightSkyBlue;
sfGradientStop1.Offset = 0.5;
sfGradientStops.Add(sfGradientStop1);
SfGradientStop sfGradientStop2 = new SfGradientStop();
sfGradientStop2.Color = Color.White;
sfGradientStop2.Offset = 1;
sfGradientStops.Add(sfGradientStop2);
sfLinearGradientBrush.GradientStops = sfGradientStops;
defaultSwitch.ThumbGradient = sfLinearGradientBrush;
SfLinearGradientBrush sfLinearGradientBrush1 = new SfLinearGradientBrush();
GradientStopCollection sfGradientStops1 = new GradientStopCollection();
SfGradientStop sfGradientStop3 = new SfGradientStop();
sfGradientStop3.Color = Color.White;
sfGradientStop3.Offset = 0;
sfGradientStops1.Add(sfGradientStop3);
SfGradientStop sfGradientStop4 = new SfGradientStop();
sfGradientStop4.Color = Color.LightCoral;
sfGradientStop4.Offset = 0.25;
sfGradientStops1.Add(sfGradientStop4);
SfGradientStop sfGradientStop5 = new SfGradientStop();
sfGradientStop5.Color = Color.Red;
sfGradientStop5.Offset = 0.5;
sfGradientStops1.Add(sfGradientStop5);
SfGradientStop sfGradientStop6 = new SfGradientStop();
sfGradientStop6.Color = Color.LightSalmon;
sfGradientStop6.Offset = 0.75;
sfGradientStops1.Add(sfGradientStop6);
sfLinearGradientBrush1.GradientStops = sfGradientStops1;
defaultSwitch.TrackGradient = sfLinearGradientBrush1;
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState onState = new VisualState
{
    Name = "On"
};
onState.Setters.Add(new Setter { Property = SfSwitch.SwitchSettingsProperty,
    Value = defaultSwitch });
commonStateGroup.States.Add(onState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(sfSwitch, visualStateGroupList);
this.Content = sfSwitch;
```



**Note:** Here ThumbGradient and TrackGradient are type of SfLinearGradientBrush, So You can apply SfLinearGradientBrush or SfRadialGradientBrush on it. This SfLinearGradientBrush is available in [Syncfusion.Xamarin.Core](https://nuget.org/packages/Syncfusion.Xamarin.Core) from [nuget.org](https://nuget.org). To know more about gradient view refer this [link](#).

### Sizing

In the switch control, sizing of the thumb and the track can be controlled using the following properties:

- TrackCornerRadius: Represents a double value to create curved corner.
- TrackBorderWidth: Represents a double value for defining the track border width.
- TrackWidthRequest: Represents a double value for defining the track width.
- TrackHeightRequest: Represents double value for defining the track height.
- ThumbBorderWidth: Represents a double value for defining the thumb border width.
- ThumbCornerRadius: Represents a double value to create curved corner.
- ThumbHeightRequest: Represents double value for defining the thumb height.
- ThumbWidthRequest: Represents double value for defining the thumb width.

The following code example demonstrates how to customize the size of the track and thumb.

### XML

```
<syncfusion:SfSwitch VisualType="Custom" >
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="On">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OnState"
                ThumbBorderWidth="2" TrackBorderWidth="2"
                TrackHeightRequest="25" TrackWidthRequest="75" TrackCornerRadius="4"
                ThumbCornerRadius="4" ThumbHeightRequest="10" ThumbWidthRequest="10">
            </syncfusion:DefaultSwitchSettings>
          </Setter.Value>
        </Setter>
      </VisualState.Setters>
    </VisualState>
    <VisualState x:Name="Off">
      <VisualState.Setters>
        <Setter Property="SwitchSettings">
          <Setter.Value>
            <syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OffState"
              ThumbBorderWidth="2" TrackBorderWidth="2">
          </syncfusion:DefaultSwitchSettings>
        </Setter.Value>
      </Setter>
    </VisualState.Setters>
  </VisualStateGroup>
</SfSwitch>
```

```

TrackHeightRequest="25" TrackWidthRequest="75" TrackCornerRadius="4"
ThumbCornerRadius="4" ThumbHeightRequest="10" ThumbWidthRequest="10"/>
</Setter.Value>
</Setter>
</VisualState.Setters>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</syncfusion:SfSwitch>

```

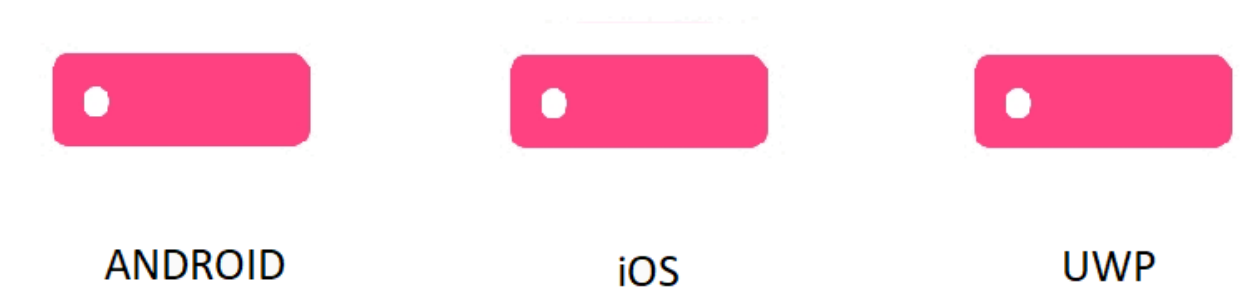
**C#**

```

SfSwitch sfSwitch = new SfSwitch();
sfSwitch.VisualType = VisualType.Custom;
DefaultSwitchSettings<OnState> defaultSwitch = new
DefaultSwitchSettings<OnState>();
defaultSwitch.ThumbBorderWidth = 2;
defaultSwitch.TrackBorderWidth = 2;
defaultSwitch.TrackHeightRequest = 25;
defaultSwitch.TrackWidthRequest = 75;
defaultSwitch.TrackCornerRadius = 4;
defaultSwitch.ThumbCornerRadius = 4;
defaultSwitch.ThumbHeightRequest = 10;
defaultSwitch.ThumbWidthRequest = 10;
DefaultSwitchSettings<OffState> defaultSwitch1 = new
DefaultSwitchSettings<OffState>();
defaultSwitch1.ThumbBorderWidth = 2;
defaultSwitch1.TrackBorderWidth = 2;
defaultSwitch1.TrackHeightRequest = 25;
defaultSwitch1.TrackWidthRequest = 75;
defaultSwitch1.TrackCornerRadius = 4;
defaultSwitch1.ThumbCornerRadius = 4;
defaultSwitch1.ThumbHeightRequest = 10;
defaultSwitch1.ThumbWidthRequest = 10;
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState onState = new VisualState
{
    Name = "On"
};
onState.Setters.Add(new Setter { Property = SfSwitch.SwitchSettingsProperty,
Value = defaultSwitch });
VisualState offState = new VisualState
{
    Name = "Off"
};
offState.Setters.Add(new Setter { Property =
SfSwitch.SwitchSettingsProperty, Value = defaultSwitch1 });
commonStateGroup.States.Add(onState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(sfSwitch, visualStateGroupList);
this.Content = sfSwitch;

```





### Images

Images can also be added to enhance the visual appearance. The following properties are used to add the images:

- `TrackImageSource`: Represents the image source for the track.
- `ThumbImageSource`: Represents the image source for thumb.

The following code example demonstrates how to customize the image.

### XML

```
<syncfusion:SfSwitch IsOn="False" HorizontalOptions="Center"
VisualType="Custom">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="On">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OnState"
TrackImageSource="switchbg2.png"
ThumbImageSource="switchmoon.png"/>
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
      <VisualState x:Name="Off">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OffState"
TrackImageSource="switchbg.png"
ThumbImageSource="switchsun.png"/>
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</syncfusion:SfSwitch>
```

### C#

```

SfSwitch sfSwitch = new SfSwitch();
sfSwitch.IsOn = false;
sfSwitch.VisualType = VisualType.Custom;
DefaultSwitchSettings<OnState> defaultSwitch = new
DefaultSwitchSettings<OnState>();
defaultSwitch.TrackImageSource = "switchbg2.png";
defaultSwitch.ThumbImageSource = "switchmoon.png";
DefaultSwitchSettings<OffState> defaultSwitch1 = new
DefaultSwitchSettings<OffState>();
defaultSwitch1.TrackImageSource = "switchbg.png";
defaultSwitch1.ThumbImageSource = "switchsun.png";
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState onState = new VisualState
{
    Name = "On"
};
onState.Setters.Add(new Setter { Property = SfSwitch.SwitchSettingsProperty,
Value = defaultSwitch });
VisualState offState = new VisualState
{
    Name = "Off"
};
offState.Setters.Add(new Setter { Property =
SfSwitch.SwitchSettingsProperty, Value = defaultSwitch1 });
commonStateGroup.States.Add(onState);
commonStateGroup.States.Add(offState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(sfSwitch, visualStateGroupList);
this.Content = sfSwitch;

```



ANDROID



iOS



UWP

### Limitation

Since the Visual State Manger (internally) is handled in our control, applying Visual State Manger thorough style (as dynamic resource) does not work.

### How to

#### Show busy indicator to perform async action

The busy indicator indicates users that something is on progress in the background. For instance, some data is being fetched from the back end. Here, when users about to switch the state, the StateChanging event occurs, and users can set the `IsBusy` property to true to show busy indicator and perform fetching

the data from the server. After fetching the data, the `IsOn` property will be set to true or false based on validation. After validation, the `IsBusy` property is set to false.

### XML

```
<syncfusion:SfSwitch x:Name="sfSwitch" StateChanging="State_StateChanging"/>
```

### C#

```
SfSwitch sfSwitch = new SfSwitch();
sfSwitch.StateChanged += State_StateChanging;
```

### C#

```
private async void State_StateChanging(object sender,
SwitchStateChangingEventArgs e)
{
    this.sfSwitch.IsBusy = true;
    await Task.Delay(2500);
    this.sfSwitch.IsOn = ValidateInternetConnection();
    this.sfSwitch.IsBusy = false;
}
private bool ValidateInternetConnection()
{
    Random randomValue = new Random();
    if (randomValue.Next() % 2 == 0)
    {
        return false;
    }
    return true;
}
```



ANDROID



iOS



UWP

Change thumb color alone based on its state and devices

You can customize the thumb color using the `ThumbColor` property based on its visual state and devices.

### XML

```
<syncfusion:SfSwitch VisualType="Custom">
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="On">
```

```

<VisualState.Setters>
  <Setter Property="SwitchSettings">
    <Setter.Value>
      <syncfusion:FluentSwitchSettings x:TypeArguments="syncfusion:OnState"
      ThumbColor="Green"/>
    </Setter.Value>
  </Setter>
</VisualState.Setters>
</VisualState>
<VisualState x:Name="Off">
  <VisualState.Setters>
    <Setter Property="SwitchSettings">
      <Setter.Value>
        <syncfusion:FluentSwitchSettings x:TypeArguments="syncfusion:OffState"
        ThumbColor="Green"/>
      </Setter.Value>
    </Setter>
  </VisualState.Setters>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</syncfusion:SfSwitch>

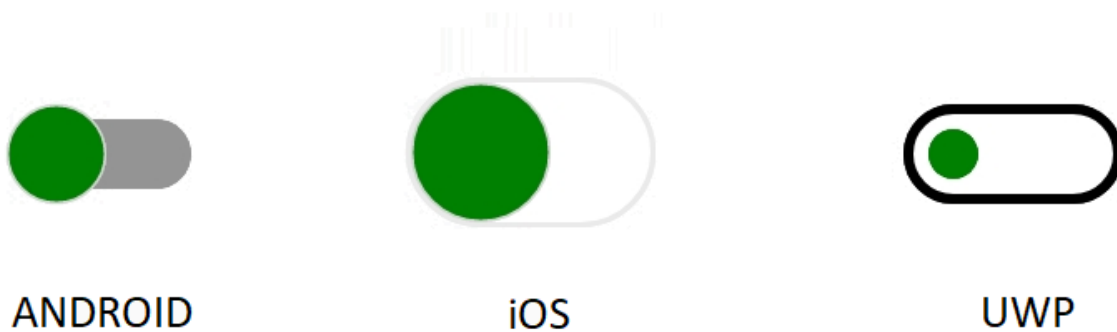
```

## C#

```

SfSwitch sfSwitch = new SfSwitch();
sfSwitch.VisualType = VisualType.Custom;
DefaultSwitchSettings<OnState> defaultSwitch = new
DefaultSwitchSettings<OnState>();
defaultSwitch.ThumbColor = Color.Green;
DefaultSwitchSettings<OffState> defaultSwitch1 = new
DefaultSwitchSettings<OffState>();
defaultSwitch1.ThumbColor = Color.Green;
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState onState = new VisualState
{
    Name = "On"
};
onState.Setters.Add(new Setter { Property = SfSwitch.SwitchSettingsProperty,
Value = defaultSwitch });
VisualState offState = new VisualState
{
    Name = "Off"
};
offState.Setters.Add(new Setter { Property =
SfSwitch.SwitchSettingsProperty, Value = defaultSwitch1 });
commonStateGroup.States.Add(onState);
commonStateGroup.States.Add(offState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(sfSwitch, visualStateGroupList);
this.Content = sfSwitch;

```



Change thumb color alone based on its state with Material theme for all devices

By using the `MaterialSwitchSettings`, `CupertinoSwitchSettings`, and `FluentSwitchSettings` properties, you can change the thumb color based on its state for all devices.

### XML

```
<syncfusion:SfSwitch VisualType="Custom">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="On">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:MaterialSwitchSettings x:TypeArguments="syncfusion:OnState"
                ThumbColor="Brown"/>
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
      <VisualState x:Name="Off">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:MaterialSwitchSettings x:TypeArguments="syncfusion:OffState"
                ThumbColor="Brown"/>
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</syncfusion:SfSwitch>
```

### C#

```
SfSwitch sfSwitch = new SfSwitch();
sfSwitch.VisualType = VisualType.Custom;
MaterialSwitchSettings<OnState> defaultSwitch = new
MaterialSwitchSettings<OnState>();
defaultSwitch.ThumbColor = Color.Brown;
MaterialSwitchSettings<OffState> defaultSwitch1 = new
MaterialSwitchSettings<OffState>();
defaultSwitch1.ThumbColor = Color.Brown;
```

```

VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState onState = new VisualState
{
    Name = "On"
};
onState.Setters.Add(new Setter { Property = SfSwitch.SwitchSettingsProperty,
    Value = defaultSwitch });
VisualState offState = new VisualState
{
    Name = "Off"
};
offState.Setters.Add(new Setter { Property =
    SfSwitch.SwitchSettingsProperty, Value = defaultSwitch1 });
commonStateGroup.States.Add(onState);
commonStateGroup.States.Add(offState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(sfSwitch, visualStateGroupList);
this.Content = sfSwitch;

```



ANDROID



iOS



UWP

### Set color for disabled state

The Switch control provides options to customize the color based on the disabled states. The below code example illustrates this customization.

### XML

```

<syncfusion:SfSwitch VisualType="Custom" IsEnabled="False" IsOn="True">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="DisabledOn">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings
                x:TypeArguments="syncfusion:DisabledOnState"
                ThumbBorderColor="Red" ThumbColor="Green"
                TrackBorderColor="LightGreen" TrackColor="OrangeRed"
                BusyIndicatorColor="Pink">
            </syncfusion:DefaultSwitchSettings>
          </Setter.Value>
        </Setter>
      </VisualState.Setters>
    </VisualState>
  </VisualStateGroup>

```

```

<VisualState x:Name="DisabledOff">
<VisualState.Setters>
<Setter Property="SwitchSettings">
<Setter.Value>
<syncfusion:DefaultSwitchSettings
x:TypeArguments="syncfusion:DisabledOffState"
ThumbBorderColor="Red" ThumbColor="Green"
TrackBorderColor="LightGreen" TrackColor="OrangeRed"
/>
</Setter.Value>
</Setter>
</VisualState.Setters>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</syncfusion:SfSwitch>

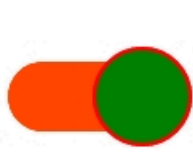
```

## C#

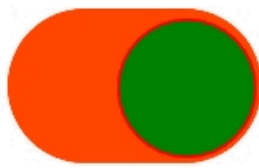
```

SfSwitch sfSwitch = new SfSwitch();
sfSwitch.VisualType = VisualType.Custom;
sfSwitch.IsOn = true;
sfSwitch.IsEnabled = false;
DefaultSwitchSettings<DisabledOnState> defaultSwitch = new
DefaultSwitchSettings<DisabledOnState>();
defaultSwitch.ThumbBorderColor = Color.Red;
defaultSwitch.ThumbColor = Color.Green;
defaultSwitch.TrackBorderColor = Color.LightGreen;
defaultSwitch.TrackColor = Color.OrangeRed;
defaultSwitch.BusyIndicatorColor = Color.Pink;
DefaultSwitchSettings<DisabledOffState> defaultSwitch1 = new
DefaultSwitchSettings<DisabledOffState>();
defaultSwitch1.ThumbBorderColor = Color.Red;
defaultSwitch1.ThumbColor = Color.Green;
defaultSwitch1.TrackBorderColor = Color.LightGreen;
defaultSwitch1.TrackColor = Color.OrangeRed;
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState onState = new VisualState
{
    Name = "DisabledOn"
};
onState.Setters.Add(new Setter { Property = SfSwitch.SwitchSettingsProperty,
Value = defaultSwitch });
VisualState offState = new VisualState
{
    Name = "DisabledOff"
};
offState.Setters.Add(new Setter { Property =
SfSwitch.SwitchSettingsProperty, Value = defaultSwitch1 });
commonStateGroup.States.Add(onState);
commonStateGroup.States.Add(offState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(sfSwitch, visualStateGroupList);
this.Content = sfSwitch;

```



ANDROID



iOS



UWP

### Change busy indicator color

You can customize the busy indicator color using the `BusyIndicatorColor` property based on its visual state and devices.

#### XML

```
<syncfusion:SfSwitch VisualType="Custom" IsBusy="True">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="On">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OnState"
                BusyIndicatorColor="Red" />
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
      <VisualState x:Name="Off">
        <VisualState.Setters>
          <Setter Property="SwitchSettings">
            <Setter.Value>
              <syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OffState"
                BusyIndicatorColor="Red" />
            </Setter.Value>
          </Setter>
        </VisualState.Setters>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</syncfusion:SfSwitch>
```

#### C#

```
SfSwitch sfSwitch = new SfSwitch();
sfSwitch.VisualType = VisualType.Custom;
sfSwitch.IsBusy = true;
DefaultSwitchSettings<OnState> defaultSwitch = new
DefaultSwitchSettings<OnState>();
defaultSwitch.BusyIndicatorColor = Color.Red;
DefaultSwitchSettings<OffState> defaultSwitch1 = new
DefaultSwitchSettings<OffState>();
```



```

defaultSwitch1.BusyIndicatorColor = Color.Red;
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState onState = new VisualState
{
    Name = "On"
};
onState.Setters.Add(new Setter { Property = SfSwitch.SwitchSettingsProperty,
    Value = defaultSwitch1 });
VisualState offState = new VisualState
{
    Name = "Off"
};
offState.Setters.Add(new Setter { Property =
    SfSwitch.SwitchSettingsProperty, Value = defaultSwitch1 });
commonStateGroup.States.Add(onState);
commonStateGroup.States.Add(offState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(sfSwitch, visualStateGroupList);
this.Content = sfSwitch;

```



ANDROID



iOS



UWP

Set all state in same look

All state can be customized in same look by setting the same state for all switch settings.

#### XML

```

<syncfusion:SfSwitch VisualType="Custom" >
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="CommonStates">
<VisualState x:Name="On">
<VisualState.Setters>
<Setter Property="SwitchSettings">
<Setter.Value>
<syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OnState" />
</Setter.Value>
</Setter>
</VisualState.Setters>
</VisualState>
<VisualState x:Name="Off">
<VisualState.Setters>
<Setter Property="SwitchSettings">
<Setter.Value>
<syncfusion:DefaultSwitchSettings x:TypeArguments="syncfusion:OnState" />
</Setter.Value>

```

```

</Setter>
</VisualState.Setters>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
</syncfusion:SfSwitch>

```

## C#

```

SfSwitch sfSwitch = new SfSwitch();
sfSwitch.VisualType = VisualType.Custom;
DefaultSwitchSettings<OnState> defaultSwitch = new
DefaultSwitchSettings<OnState>();
DefaultSwitchSettings<OnState> defaultSwitch1 = new
DefaultSwitchSettings<OnState>();
VisualStateGroupList visualStateGroupList = new VisualStateGroupList();
VisualStateGroup commonStateGroup = new VisualStateGroup();
VisualState onState = new VisualState
{
    Name = "On"
};
onState.Setters.Add(new Setter { Property = SfSwitch.SwitchSettingsProperty,
    Value = defaultSwitch });
VisualState offState = new VisualState
{
    Name = "Off"
};
offState.Setters.Add(new Setter { Property =
    SfSwitch.SwitchSettingsProperty, Value = defaultSwitch1 });
commonStateGroup.States.Add(onState);
commonStateGroup.States.Add(offState);
visualStateGroupList.Add(commonStateGroup);
VisualStateManager.SetVisualStateGroups(sfSwitch, visualStateGroupList);
this.Content = sfSwitch;

```



ANDROID



iOS



UWP

### Set RTL to switch control

Switch supports to change the layout direction of the control in the right-to-left direction by setting the `FlowDirection` to `RightToLeft`.

The device direction takes when we set the `FlowDirection` as `MatchParent`.

*Setting right to left*

#### XML

```
<syncfusion:SfSwitch FlowDirection="RightToLeft" >  
</syncfusion:SfSwitch>
```

#### C#

```
SfSwitch sfSwitch = new SfSwitch();  
sfSwitch.FlowDirection = FlowDirection.RightToLeft;  
this.Content = sfSwitch;
```



ANDROID



iOS



UWP

*Setting match parent*

#### XML

```
<Grid FlowDirection="RightToLeft">  
<syncfusion:SfSwitch FlowDirection="MatchParent" />  
</Grid>
```

#### C#

```
Grid grid = new Grid();  
grid.FlowDirection = FlowDirection.RightToLeft;  
SfSwitch sfSwitch = new SfSwitch();  
sfSwitch.FlowDirection = FlowDirection.MatchParent;  
grid.Children.Add(sfSwitch);  
this.Content = grid;
```



ANDROID



iOS



UWP

Get default color of the switches in all three state.

The following table illustrate the default color of the switches in all three state.

#### Material

Elements	State	Colors
Thumb Color	On	Accent
Thumb Color	Off	#FFFFFF
Thumb Color	Indeterminate	Accent
Thumb Color	Disabled On	Accent with 0.4 opacity
Thumb Color	Disabled Off	#FFFFFF
Thumb Color	Disabled Indeterminate	Accent with 0.4 opacity
Thumb Border Color	On	Accent
Thumb Border Color	Off	LightGray
Thumb Border Color	Indeterminate	Accent
Thumb Border Color	Disabled On	Accent with 0.4 opacity
Thumb Border Color	Disabled Off	#DADADA
Thumb Border Color	Disabled Indeterminate	Accent with 0.4 opacity
Track Color	On	Accent with 0.2 opacity
Track Color	Off	#949494
Track Color	Indeterminate	#CACACA
Track Color	Disabled On	Accent with 0.2 opacity
Track Color	Disabled Off	#E0E0E0
Track Color	Disabled Indeterminate	#E0E0E0
Track Border Color	On	Transparent
Track Border Color	Off	Transparent

Track Border Color	Indeterminate	Transparent
Track Border Color	Disabled On	Transparent
Track Border Color	Disabled Off	Transparent
Track Border Color	Disabled Indeterminate	Transparent
Busy Indicator Color	On	#00BFFF
Busy Indicator Color	Off	#00BFFF
Busy Indicator Color	Indeterminate	#00BFFF
Busy Indicator Color	Disabled On	#00BFFF
Busy Indicator Color	Disabled Off	#00BFFF
Busy Indicator Color	Disabled Indeterminate	#00BFFF

### Cupertino

Elements	State	Colors
Thumb Color	On	#FFFFFF
Thumb Color	Off	#FFFFFF
Thumb Color	Indeterminate	Accent
Thumb Color	Disabled On	#FFFFFF
Thumb Color	Disabled Off	#FFFFFF
Thumb Color	Disabled Indeterminate	Accent with 0.5 opacity
Thumb Border Color	On	0.1 Opacity
Thumb Border Color	Off	0.3 Opacity
Thumb Border Color	Indeterminate	0.1 Opacity
Thumb Border Color	Disabled On	0.5 Opacity
Thumb Border Color	Disabled Off	0.3 Opacity

Thumb Border Color	Disabled Indeterminate	0.5 Opacity
Track Color	On	Accent
Track Color	Off	Transparent
Track Color	Indeterminate	Transparent
Track Color	Disabled On	Accent with 0.5 opacity
Track Color	Disabled Off	Transparent
Track Color	Disabled Indeterminate	Transparent
Track Border Color	On	Transparent
Track Border Color	Off	#E5E5E5
Track Border Color	Indeterminate	Accent
Track Border Color	Disabled On	Accent with 0.5 opacity
Track Border Color	Disabled Off	#F2F2F2
Track Border Color	Disabled Indeterminate	Accent with 0.5 opacity
Busy Indicator Color	On	#00BFFF
Busy Indicator Color	Off	#00BFFF
Busy Indicator Color	Indeterminate	#00BFFF
Busy Indicator Color	Disabled On	#00BFFF
Busy Indicator Color	Disabled Off	#00BFFF
Busy Indicator Color	Disabled Indeterminate	#00BFFF

*Fluent*

Elements	State	Colors
Thumb Color	On	#FFFFFFFF
Thumb Color	Off	#000000

Thumb Color	Indeterminate	Accent
Thumb Color	Disabled On	0.2 Opacity
Thumb Color	Disabled Off	0.2 Opacity
Thumb Color	Disabled Indeterminate	0.2 Opacity
Thumb Border Color	On	Transparent
Thumb Border Color	Off	Transparent
Thumb Border Color	Indeterminate	Transparent
Thumb Border Color	Disabled On	Transparent
Thumb Border Color	Disabled Off	Transparent
Thumb Border Color	Disabled Indeterminate	Transparent
Track Color	On	Accent
Track Color	Off	Transparent
Track Color	Indeterminate	Transparent
Track Color	Disabled On	0.2 opacity
Track Color	Disabled Off	0.2 opacity
Track Color	Disabled Indeterminate	0.2 opacity
Track Border Color	On	Accent
Track Border Color	Off	#000000
Track Border Color	Indeterminate	Accent
Track Border Color	Disabled On	0.2 Opacity
Track Border Color	Disabled Off	0.2 Opacity
Track Border Color	Disabled Indeterminate	0.2 Opacity
Busy Indicator Color	On	#00BFFF

Busy Indicator Color	Off	#00BFFF
Busy Indicator Color	Indeterminate	#00BFFF
Busy Indicator Color	DisabledOn	#00BFFF
Busy Indicator Color	DisabledOff	#00BFFF
Busy Indicator Color	DisabledIndeterminate	#00BFFF

### AutomationId

The SfSwitch control has built-in **AutomationId** for inner elements. The **AutomationId** API allows the automation framework to find and interact with the inner elements of the SfSwitch control. To keep unique AutomationId, these inner elements' AutomationIds are updated based on the control's **AutomationId**.

For example, if you set SfSwitch's **AutomationId** as "Turn on Night mode", then the automation framework will interact the SfSwitch as "Turn on Night mode Track".

When you enable the Indeterminate state, then the Automation framework will interact the Off state as "Turn on Night mode Off Track", Indeterminate state as "Turn on Night mode Indeterminate Track" and the On state as "Turn on Night mode On Track".

**Note:** AutomationId support works on Android only.



———— Turn on Night mode Track

## SfTabView

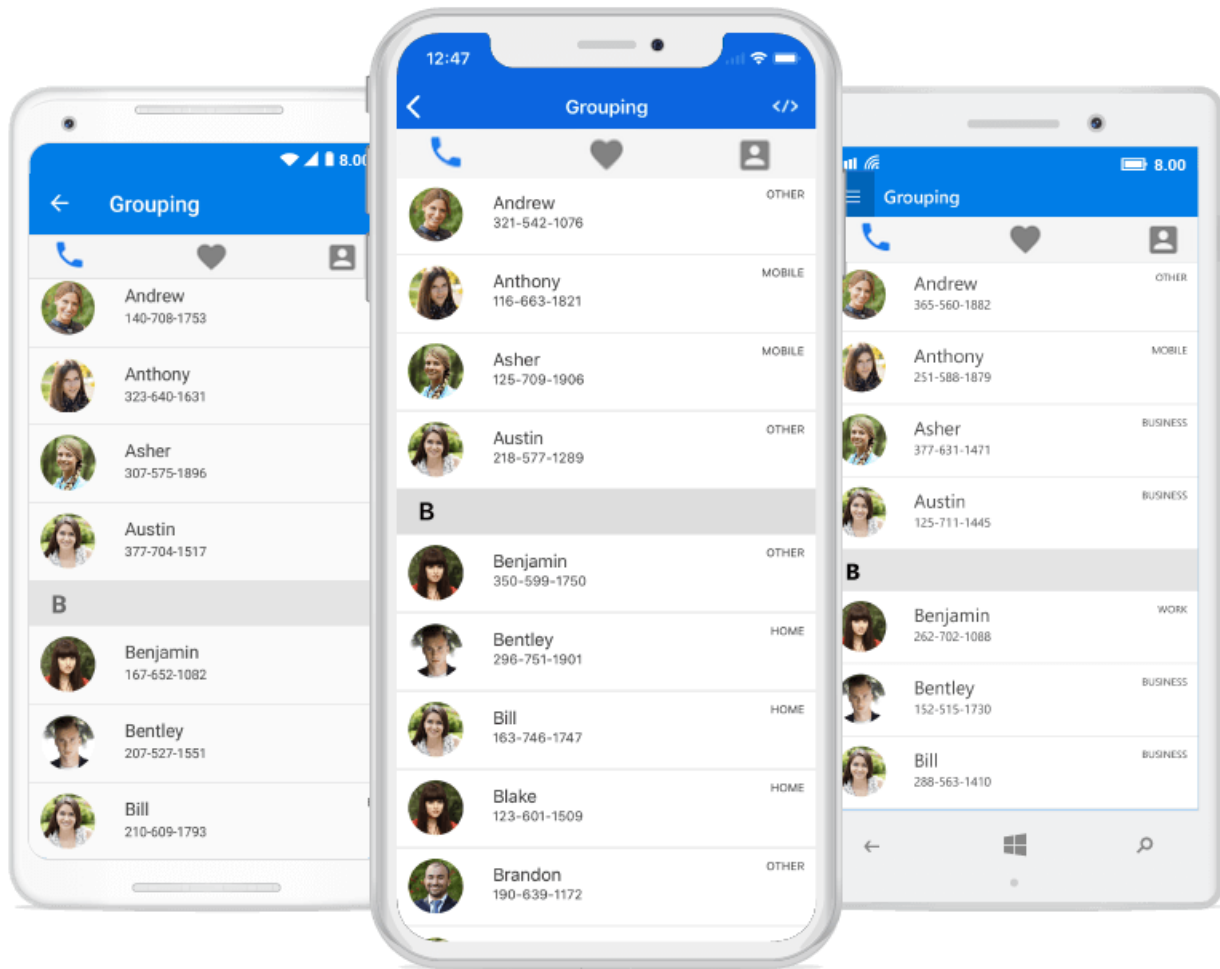
### Overview

The tab view control is available in Xamarin.Forms, Xamarin.Android, Xamarin.iOS and Xamarin.UWP. It helps you to create the customizable features that are used to explore and switch among the different views. The key features of tab view control in Xamarin.Forms, Xamarin.Android, Xamarin.iOS and Xamarin.UWP are given as follows.



## Key features

- Tab header type with text, font icons, and no header.
- Scrollable content and header.
- Top and bottom placements of header.
- Layout option for overflow tabs.



## Getting Started

This section provides an overview for working with the tab view control for Xamarin.Forms. Walk through the entire process of creating a real-world application with the tab view.

### Assembly deployment

After installing the Essential Studio for Xamarin, find all the required assemblies in installation folders, {Syncfusion Essential Studio Installed location}\Essential Studio\16.1.0.24\Xamarin\lib}.

E.g., C:\Program Files (x86)\Syncfusion\Essential Studio\16.1.0.24\Xamarin\lib

**Note:** In Mac, assemblies can be found in unzipped package location.

### Adding SfTabView reference

You can add SfTabView reference using one of the following methods:

**Method 1: Adding SfTabView reference from nuget.org**

Syncfusion Xamarin components are available in [nuget.org](https://www.syncfusion.com/nuget-packages). To add SfTabView to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfTabView](https://www.syncfusion.com/nuget-packages), and then install it.

![[Adding SfTabView reference from NuGet]](images/Getting-Started/Adding SfTabView reference.png)

**Note:** Install the same version of SfTabView NuGet in all the projects.

**Method 2: Adding SfTabView reference from toolbox**

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfTabView control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

**Method 3: Adding SfTabView assemblies manually from the installed location**

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfTabView.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfTabView.XForms.dll Syncfusion.SfTabView.XForms.Android.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfTabView.XForms.dll Syncfusion.SfTabView.XForms.iOS.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfTabView.XForms.dll Syncfusion.SfTabView.XForms.UWP.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

### Launching the tab view on each platform

To use the tab view inside an application, each platform application must initialize the tab view renderer. This initialization steps vary from platform to platform, and it is discussed in the following sections:

#### *Android and UWP*

Android launches the tab view without any initialization, and it is enough to only initialize the Xamarin.Forms Framework to launch the application.

---

**Note:** If you are adding the references from toolbox, this step is not needed.

---

#### *iOS*

To launch the tab view in iOS, call the `SfTabViewRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework has been initialized and before the `LoadApplication` is called, as demonstrated in the following code example.

#### **C#**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    Syncfusion.XForms.iOS.TabView.SfTabViewRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

#### *ReleaseMode issue in UWP platform*

There is a known Framework issue in UWP platform. The custom controls will not render when deployed the application in **Release Mode**.

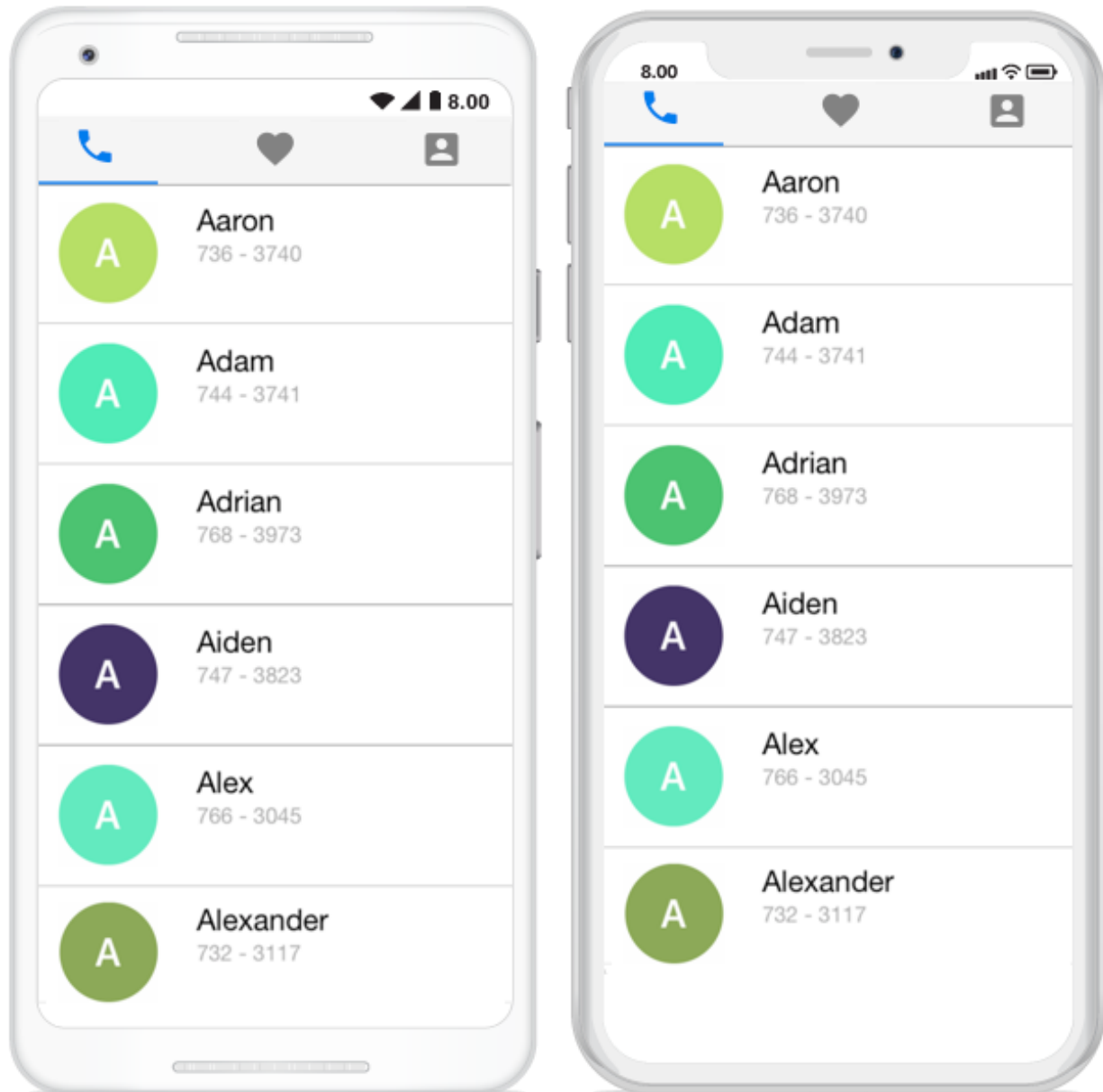
The above problem can be resolved by initializing the SfTabView assemblies in `App.xaml.cs` in UWP project as like in below code snippet.

#### **C#**

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(Syncfusion.XForms.UWP.TabView.SfTabViewRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

### Create a simple tab view

This section explains how to create a tab view and configure it. The control can be configured entirely in C# code or by using XAML markup. The following screenshot illustrates the output of tab view on iOS, Android and UWP devices.



### Creating the project

Create a new BlankApp (Xamarin.Forms.Portable) application in Xamarin Studio or Visual Studio for Xamarin.Forms.

### Adding SfTabView in Xamarin.Forms

Add the required assembly references to the PCL and renderer projects as discussed in the Assembly deployment section.

Import the control namespace as shown in the following code.

#### XML

```
xmlns:tabView="clr-  
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
```

#### C#

```
using Syncfusion.XForms.TabView;
```

Set the control to content in `ContentPage`.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:tabView="clr-  
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"  
x:Class="TabViewAutomationSample.TabViewAutomationSample">  
<ContentPage.Content>  
<tabView:SfTabView/>  
</ContentPage.Content>  
</ContentPage>
```

#### C#

```
using Syncfusion.XForms.TabView;  
using Xamarin.Forms;  
namespace GettingStarted  
{  
    public partial class MainPage : ContentPage  
    {  
        private SfTabView tabView;  
        public MainPage()  
        {  
            InitializeComponent();  
            tabView = new SfTabView();  
            this.Content = tabView;  
        }  
    }  
}
```

### Populating tab items

Tab items can be configured in tab view through the `Items` property of `SfTabView`, which holds the collection of `SfTabItem` through `TabItemsCollection`.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:tabView="clr-  
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"  
x:Class="TabViewAutomationSample.TabViewAutomationSample">
```

```

<ContentPage.Content>
<tabView:SfTabView BackgroundColor="Aqua">
<tabView:SfTabItem Title="Call">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Red" x:Name="AllContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Favorites">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="FavoritesGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabViewAutomationSample
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class TabView : ContentPage
{
SfTabView tabView;
public TabView ()
{
InitializeComponent ();
var tabView = new SfTabView();
Grid allContactsGrid = new Grid {BackgroundColor = Color.Red};
Grid favoritesGrid = new Grid {BackgroundColor = Color.Green};
Grid contactsGrid = new Grid {BackgroundColor = Color.Blue};
var tabItems = new TabItemCollection
{
new SfTabItem()
{
Title = "Calls",
Content = allContactsGrid
},
new SfTabItem()
{
Title = "Favorites",
Content = favoritesGrid
},
new SfTabItem()
{
Title = "Contacts",
Content = contactsGrid
}
}
}
}

```

```
};
tabView.Items = tabItems;
this.Content = tabView;
}
}
}
```

### Adding ListView in tab view

UseCase Sample with Contacts Information stored as a ListView in TabView Control

Create a view model class with the **ContactsInfo** collection property, which is initialized with required number of data objects.

### C#

```
public class ContactInfo
{
    public string Name { get; set; }
    public long Number { get; set; }
}

public class ContactsViewModel : INotifyPropertyChanged
{
    private ObservableCollection<ContactInfo> contactList;
    public event PropertyChangedEventHandler PropertyChanged;
    public ObservableCollection<ContactInfo> ContactList
    {
        get { return contactList; }
        set { contactList = value; }
    }
    public ContactsViewModel()
    {
        ContactList = new ObservableCollection<ContactInfo>();
        ContactList.Add(new ContactInfo { Name = "Aaron", Number = 7363750 });
        ContactList.Add(new ContactInfo { Name = "Adam", Number = 723250 });
        ContactList.Add(new ContactInfo { Name = "Adrian", Number = 7239121 });
        ContactList.Add(new ContactInfo { Name = "Alwin", Number = 2329823 });
        ContactList.Add(new ContactInfo { Name = "Alex", Number = 8013481 });
        ContactList.Add(new ContactInfo { Name = "Alexander", Number = 7872329 });
        ContactList.Add(new ContactInfo { Name = "Barry", Number = 7317750 });
    }
}
```

### Binding data to list view

Bind the items source of the **ListView**, and set the required appearance in its **ItemTemplate** property in which the list view can be hosted within the content region of tab item.

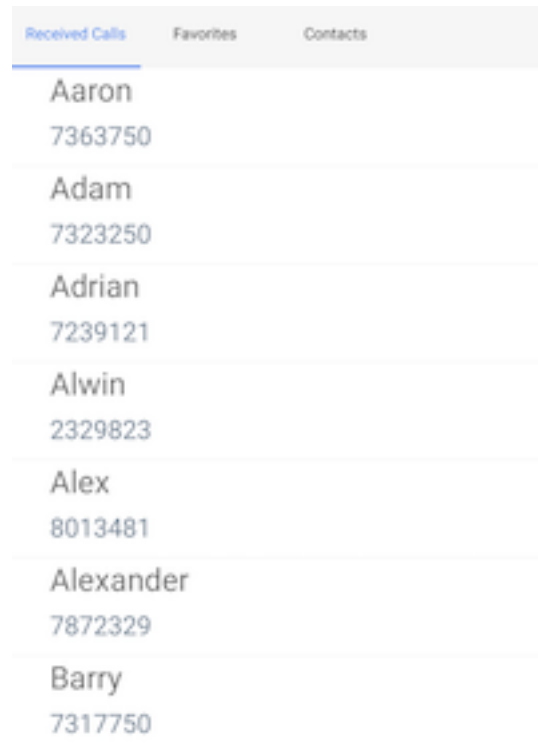
### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:TabViewAutomationSample"
xmlns:tabView="clr-namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabViewAutomationSample.XAMARIN_22272">
<ContentPage.Content>
<tabView:SfTabView BackgroundColor="#f6f6f6" x:Name="tabView">
```

```
<tabView:SfTabItem Title="{Binding Title1}">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="White" x:Name="AllContactsGrid" >
<ListView x:Name="ContactListView"
ItemsSource="{Binding ContactList}"
RowHeight="75">
<ListView.BindingContext>
<local:ContactsViewModel />
</ListView.BindingContext>
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout Orientation="Vertical" Margin="30,0,0,0">
<Label
Text="{Binding Name}"
FontSize="24" />
<Label
Text="{Binding Number}"
FontSize="20"
TextColor="LightSlateGray" />
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Favorites">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="FavouritesGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>
```

Similarly, content region for other tabs also can be configured.





### Swiping

By default, both the vertical swiping for list view and horizontal swiping for tab view will work. If it is not required, it can be customized by using the `EnableSwiping` property of `SfTabView`.

Note: Getting started sample can be downloaded from [this link](#).

### Tab Items

Tab items can be configured in tab view through the `Items` property of `SfTabView`, which holds the collection of `SfTabItem` through `TabItemsCollection`.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabViewAutomationSample.TabViewAutomationSample">
  <ContentPage.Content>
    <tabView:SfTabView BackgroundColor="White">
      <tabView:SfTabItem Title="Call">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="Gray" x:Name="AllContactsGrid" />
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
      <tabView:SfTabItem Title="Favorites">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="Green" x:Name="FavoritesGrid" />
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
      <tabView:SfTabItem Title="Contacts">
```

```
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabViewAutomationSample
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class TabView : ContentPage
    {
        SfTabView tabView;
        public TabView ()
        {
            InitializeComponent ();
            var tabView = new SfTabView();
            var allContactsGrid = new Grid {BackgroundColor = Color.Gray};
            var favoritesGrid = new Grid {BackgroundColor = Color.Green};
            var contactsGrid = new Grid {BackgroundColor = Color.Blue};
            var tabItems = new TabItemCollection
            {
                new SfTabItem()
                {
                    Title = "Calls",
                    Content = allContactsGrid
                },
                new SfTabItem()
                {
                    Title = "Favorites",
                    Content = favoritesGrid
                },
                new SfTabItem()
                {
                    Title = "Contacts",
                    Content = contactsGrid
                }
            };
            tabView.Items = tabItems;
            this.Content = tabView;
        }
    }
}
```



### Share the header space equally

To share the header space to the tabs equally, set the number of tabs that can be distributed in the available space though the `VisibleHeaderCount` of `SfTabView`.

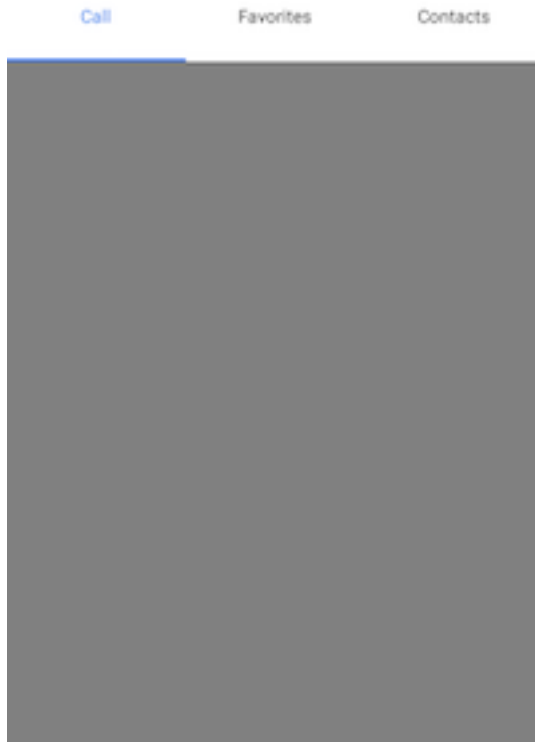
### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabViewAutomationSample.TabViewAutomationSample">
  <ContentPage.Content>
    <tabView:SfTabView BackgroundColor="White" VisibleHeaderCount="3">
      <tabView:SfTabItem Title="Call">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="Gray" x:Name="AllContactsGrid" />
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
      <tabView:SfTabItem Title="Favorites">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="Green" x:Name="FavoritesGrid" />
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
      <tabView:SfTabItem Title="Contacts">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
      <tabView:SfTabItem Title="Email">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="Olive" x:Name="EmailGrid" />
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
    </tabView:SfTabView>
  </ContentPage.Content>
</ContentPage>
```

```
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabViewAutomationSample
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class TabView : ContentPage
    {
        SfTabView tabView;
        public TabView ()
        {
            InitializeComponent ();
            var tabView = new SfTabView();
            var allContactsGrid = new Grid {BackgroundColor = Color.Red};
            var favoritesGrid = new Grid {BackgroundColor = Color.Green};
            var contactsGrid = new Grid {BackgroundColor = Color.Blue};
            var emailGrid = new Grid {BackgroundColor = Color.Olive};
            tabView.VisibleHeaderCount = 3;
            var tabItems = new TabItemCollection
            {
                new SfTabItem()
                {
                    Title = "Calls",
                    Content = allContactsGrid
                },
                new SfTabItem()
                {
                    Title = "Favorites",
                    Content = favoritesGrid
                },
                new SfTabItem()
                {
                    Title = "Contacts",
                    Content = contactsGrid
                },
                new SfTabItem()
                {
                    Title = "Email",
                    Content = emailGrid
                }
            };
            tabView.Items = tabItems;
            this.Content = tabView;
        }
    }
}
```



### Selection Changed

The `SelectionChanged` event is used to notify when the selection is changed by swiping or dynamically setting the `SelectedIndex` property of `SfTabView`.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Syncfusion.XForms.TabView"
xmlns:tabView="clr-namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
  <ContentPage.Content>
    <tabView:SfTabView VisibleHeaderCount="3"
SelectionChanged="Handle_SelectionChanged"
BackgroundColor="Aqua">
      <tabView:SfTabItem Title="Call">
        <tabView:SfTabItem.Content>
          <StackLayout>
            <Grid BackgroundColor="Green" />
            <Button Text="Contacts" WidthRequest="300" />
            <Button Text="Location" WidthRequest="300" />
            <Button Text="Email" WidthRequest="300" />
          </StackLayout>
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
      <tabView:SfTabItem Title="Favorites">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="Green" x:Name="FavoritesGrid"/>
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
    </tabView:SfTabView>
  </ContentPage.Content>
</ContentPage>
```

```

</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Location">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Pink" x:Name="LocationGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Email">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Navy" x:Name="EmailGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Alternative">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="AlternativeGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabView
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class TabView : ContentPage
{
public TabView ()
{
InitializeComponent ();
}
// Occurred when the selected index is changed
void Handle_SelectionChanged(object sender,
Syncfusion.XForms.TabView.SelectionChangedEventArgs e)
{
var selectedIndex = e.Index;
}
}
}

```

## Enable swiping

To restrict the user interaction, the `EnableSwiping` property of `SfTabView` can be set to `false`.

## XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

```

```

xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
<ContentPage.Content>
<tabView:SfTabView OverflowMode="DropDown"
EnableSwiping="false"
VisibleHeaderCount="3"
BackgroundColor="Aqua">
<tabView:SfTabItem Title="Call">
<tabView:SfTabItem.Content>
<StackLayout>
<Grid BackgroundColor="Green" />
<Button Text="Contacts" WidthRequest="300" />
<Button Text="Location" WidthRequest="300" />
<Button Text="Email" WidthRequest="300" />
</StackLayout>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Favorites">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="FavoritesGrid"/>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Location">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Pink" x:Name="LocationGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Email">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Navy" x:Name="EmailGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Alternative">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="AlternativeGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabView
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class TabView : ContentPage

```

```
{
SfTabView tabView;
public TabView ()
{
InitializeComponent ();
tabView = new SfTabView();
var allContactsGrid = new Grid { BackgroundColor = Color.Red };
var favoritesGrid = new Grid { BackgroundColor = Color.Green };
var contactsGrid = new Grid { BackgroundColor = Color.Blue };
var overflowButtonSettings = new OverflowButtonSettings();
overflowButtonSettings.BackgroundColor = Color.Yellow;
overflowButtonSettings.DisplayMode = OverflowButtonDisplayMode.Text;
overflowButtonSettings.Title = "OverFlow";
overflowButtonSettings.TitleFontSize = 10;
overflowButtonSettings.TitleFontColor = Color.Blue;
tabView.OverflowButtonSettings = overflowButtonSettings;
tabView.EnableSwiping = false;
var tabItems = new TabItemCollection
{
new SfTabItem()
{
Title = "Calls",
Content = allContactsGrid
},
new SfTabItem()
{
Title = "Favorites",
Content = favoritesGrid
},
new SfTabItem()
{
Title = "Contacts",
Content = contactsGrid
},
new SfTabItem()
{
Title = "Location",
Content = contactsGrid
},
new SfTabItem()
{
Title = "Email",
Content = contactsGrid
},
new SfTabItem()
{
Title = "Alternative",
Content = contactsGrid
}
};
tabView.Items = tabItems;
this.Content = tabView;
tabView.BackgroundColor = Color.Aqua;
tabView.OverflowMode = OverflowMode.DropDown;
this.Content = tabView;
}
}
```

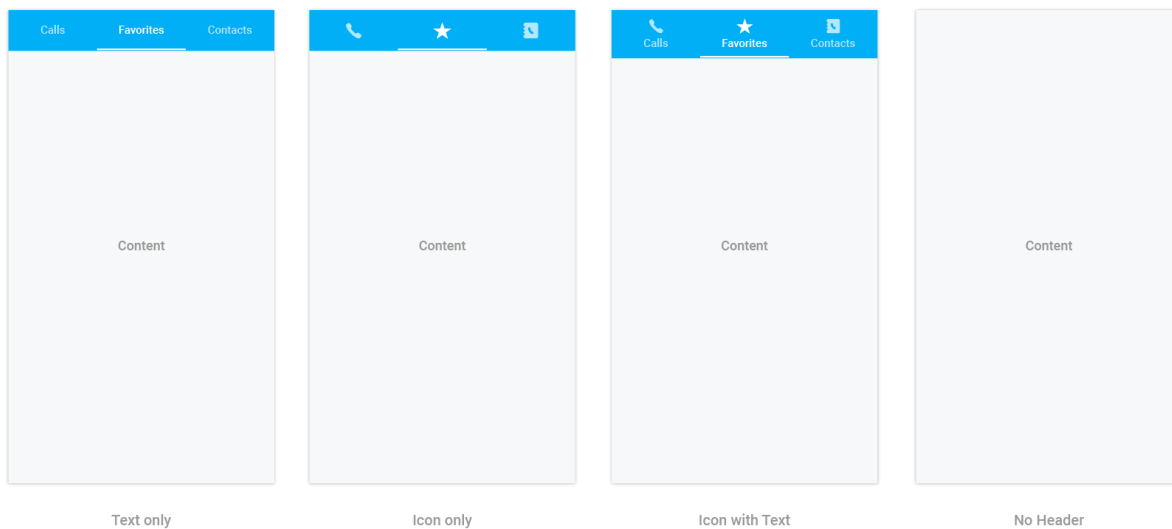


```
}
```

## Display Type

By default, the tab view control displays the title of each tab item. It can be changed to any of the following types:

- Text only
- Image only
- Image with text
- No header



The tab view can be changed by setting the `DisplayMode` property of `SfTabView`.

## XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
<ContentPage.Content>
<tabView:SfTabView OverflowMode="DropDown"
EnableSwiping="false"
VisibleHeaderCount="3"
DisplayMode="ImageWithText">
<tabView:SfTabItem Title="Call"
SelectionColor="Aqua">
<tabView:SfTabItem.Content>
<StackLayout>
<Grid BackgroundColor="Green"/>
<Button Text="Contacts" WidthRequest="300" />
<Button Text="Location" WidthRequest="300" />
<Button Text="Email" WidthRequest="300" />
</StackLayout>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>
```

```

</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Favorites">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="FavoritesGrid"/>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Location">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Pink" x:Name="LocationGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Email">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Navy" x:Name="EmailGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Alternative">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="AlternativeGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabView
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class TabView : ContentPage
    {
        SfTabView tabView;
        public TabView ()
        {
            InitializeComponent ();
            tabView = new SfTabView();
            var allContactsGrid = new Grid { BackgroundColor = Color.Red };
            var favoritesGrid = new Grid { BackgroundColor = Color.Green };
            var contactsGrid = new Grid { BackgroundColor = Color.Blue };
            var overflowButtonSettings = new OverflowButtonSettings();
            overflowButtonSettings.BackgroundColor = Color.Yellow;
            overflowButtonSettings.DisplayMode = OverflowButtonDisplayMode.Text;
            overflowButtonSettings.Title = "OverFlow";
            overflowButtonSettings.TitleFontSize = 10;
            overflowButtonSettings.TitleFontColor = Color.Blue;
            tabView.OverflowButtonSettings = overflowButtonSettings;
        }
    }
}

```

```
tabView.EnableSwiping = false;
tabView.DisplayMode = TabDisplayMode.ImageWithText;
var tabItems = new TabItemCollection
{
    new SfTabItem()
    {
        Title = "Calls",
        Content = allContactsGrid,
        SelectionColor = Color.Aqua
    },
    new SfTabItem()
    {
        Title = "Favorites",
        Content = favoritesGrid
    },
    new SfTabItem()
    {
        Title = "Contacts",
        Content = contactsGrid
    },
    new SfTabItem()
    {
        Title = "Location",
        Content = contactsGrid
    },
    new SfTabItem()
    {
        Title = "Email",
        Content = contactsGrid
    },
    new SfTabItem()
    {
        Title = "Alternative",
        Content = contactsGrid
    }
};
tabView.Items = tabItems;
tabView.BackgroundColor = Color.Aqua;
tabView.OverflowMode = OverflowMode.DropDown;
this.Content = tabView;
}
```

The "no header" type can be used when the header is not needed for the tab view control. So, the content space will be occupied to the entire available height.

---

**Note:** Image appearance in the header can be achieved through font icons.

---

How to change the selection color for text and font icons?

The selected index can be differentiated by setting the `SelectionColor` property of `SfTabItem`.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```

xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
<ContentPage.Content>
<tabView:SfTabView OverflowMode="DropDown"
EnableSwiping="false"
VisibleHeaderCount="3">
<tabView:SfTabItem Title="Call"
SelectionColor="Aqua">
<tabView:SfTabItem.Content>
<StackLayout>
<Grid BackgroundColor="Green"/>
<Button Text="Contacts" WidthRequest="300" />
<Button Text="Location" WidthRequest="300" />
<Button Text="Email" WidthRequest="300" />
</StackLayout>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Favorites">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="FavoritesGrid"/>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Location">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Pink" x:Name="LocationGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Email">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Navy" x:Name="EmailGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Alternative">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="AlternativeGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

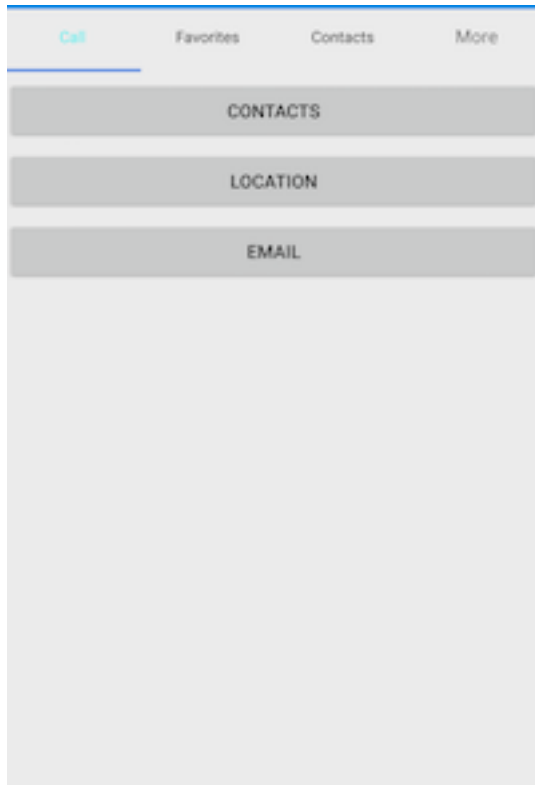
## C#

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabView
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class TabView : ContentPage

```

```
{
SfTabView tabView;
public TabView ()
{
InitializeComponent ();
tabView = new SfTabView();
var allContactsGrid = new Grid { BackgroundColor = Color.Red };
var favoritesGrid = new Grid { BackgroundColor = Color.Green };
var contactsGrid = new Grid { BackgroundColor = Color.Blue };
var overflowButtonSettings = new OverflowButtonSettings();
overflowButtonSettings.DisplayMode = OverflowButtonDisplayMode.Text;
overflowButtonSettings.Title = "OverFlow";
overflowButtonSettings.TitleFontSize = 10;
tabView.OverflowButtonSettings = overflowButtonSettings;
tabView.EnableSwiping = false;
var tabItems = new TabItemCollection
{
new SfTabItem()
{
Title = "Calls",
Content = allContactsGrid,
SelectionColor = Color.Aqua
},
new SfTabItem()
{
Title = "Favorites",
Content = favoritesGrid
},
new SfTabItem()
{
Title = "Contacts",
Content = contactsGrid
},
new SfTabItem()
{
Title = "Location",
Content = contactsGrid
},
new SfTabItem()
{
Title = "Email",
Content = contactsGrid
},
new SfTabItem()
{
Title = "Alternative",
Content = contactsGrid
}
};
tabView.Items = tabItems;
tabView.OverflowMode = OverflowMode.DropDown;
this.Content = tabView;
}
}
```



The further customizations of header are discussed in the following sections:

[How to customize text appearance of the header title?](#)

### **XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
  <ContentPage.Content>
    <tabView:SfTabView OverflowMode="DropDown"
    EnableSwiping="false"
    VisibleHeaderCount="3">
      <tabView:SfTabItem Title="Call"
      SelectionColor="Aqua">
        <tabView:SfTabItem.Content>
          <StackLayout>
            <Grid BackgroundColor="Green"/>
            <Button Text="Contacts" WidthRequest="300" />
            <Button Text="Location" WidthRequest="300" />
            <Button Text="Email" WidthRequest="300" />
          </StackLayout>
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
      <tabView:SfTabItem Title="Favorites">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="Green" x:Name="FavoritesGrid"/>
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
    </tabView:SfTabView>
  </ContentPage.Content>
</ContentPage>
```

```

<tabView:SfTabItem Title="Contacts"
TitleFontAttributes="Bold"
TitleFontColor="Red"
TitleFontSize="22">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Location"
TitleFontAttributes="Bold"
TitleFontColor="Red"
TitleFontSize="22">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Pink" x:Name="LocationGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Email"
TitleFontAttributes="Bold"
TitleFontColor="Red"
TitleFontSize="22">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Navy" x:Name="EmailGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Alternative"
TitleFontAttributes="Bold"
TitleFontColor="Red"
TitleFontSize="22">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="AlternativeGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabView
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class TabView : ContentPage
{
SfTabView tabView;
public TabView ()
{
InitializeComponent ();
tabView = new SfTabView();
var allContactsGrid = new Grid { BackgroundColor = Color.Red };
var favoritesGrid = new Grid { BackgroundColor = Color.Green };
var contactsGrid = new Grid { BackgroundColor = Color.Blue };
var overflowButtonSettings = new OverflowButtonSettings();
overflowButtonSettings.BackgroundColor = Color.Yellow;
}
}

```

```
overflowButtonSettings.DisplayMode = OverflowButtonDisplayMode.Text;
overflowButtonSettings.Title = "OverFlow";
overflowButtonSettings.TitleFontSize = 10;
overflowButtonSettings.TitleFontColor = Color.Blue;
tabView.OverflowButtonSettings = overflowButtonSettings;
tabView.EnableSwiping = false;
var tabItems = new TabItemCollection
{
    new SfTabItem()
    {
        Title = "Calls",
        Content = allContactsGrid,
        SelectionColor = Color.Aqua
    },
    new SfTabItem()
    {
        Title = "Favorites",
        Content = favoritesGrid,
        TitleFontAttributes = FontAttributes.Bold,
        TitleFontColor = Color.Red,
        TitleFontSize = 22
    },
    new SfTabItem()
    {
        Title = "Contacts",
        Content = contactsGrid,
        TitleFontAttributes = FontAttributes.Bold,
        TitleFontColor = Color.Red,
        TitleFontSize = 22
    },
    new SfTabItem()
    {
        Title = "Location",
        Content = contactsGrid,
        TitleFontAttributes = FontAttributes.Bold,
        TitleFontColor = Color.Red,
        TitleFontSize = 22
    },
    new SfTabItem()
    {
        Title = "Email",
        Content = contactsGrid,
        TitleFontAttributes = FontAttributes.Bold,
        TitleFontColor = Color.Red,
        TitleFontSize = 22
    },
    new SfTabItem()
    {
        Title = "Alternative",
        Content = contactsGrid,
        TitleFontAttributes = FontAttributes.Bold,
        TitleFontColor = Color.Red,
        TitleFontSize = 22
    }
};
tabView.Items = tabItems;
tabView.BackgroundColor = Color.Aqua;
```



```
tabView.OverflowMode = OverflowMode.DropDown;
this.Content = tabView;
}
}
}
```

How to set and customize the font icons' appearance in the header?

You can refer this [link](#) for getting the font icons. Add the font file to your application by using the following steps for each platform:

#### Adding font file for iOS

1. Add the font family inside **Resource** folder iOS project.
2. Add the font file with the following build action: **BundleResource**.
3. Update the **Info.plist** file (fonts that are provided by application, UIAppFonts, or key).

#### Adding font file for Android

Add the font file to the **Assets** folder in the application project, and set the following build action: **AndroidAsset**.

#### Adding font file for UWP

Add the font family inside the application project of UWP.

**Note:** For iOS alone, FontFamily property is declared without succeeding with .ttf and for android and UWP platform font family name is defined followed by .ttf.

#### Setting font file for font icons

##### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:TabViewFontSample"
xmlns:tabview="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabViewFontSample.TabViewFontSamplePage">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String"
x:Key="fonts"
iOS="OpenTypeFont"
Android="Fonts/OpenTypeFont.ttf" />
<OnPlatform x:TypeArguments="x:String"
x:Key="fonts"
iOS="Fonts/fa-regular-400"
Android="Fonts/fa-regular-400.ttf" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<Grid BackgroundColor="White" Padding="0,20,0,0">
<Grid.RowDefinitions>
<RowDefinition Height="30" />
<RowDefinition Height="*" />
```

```

</Grid.RowDefinitions>
<Label Text="Welcome to the Xamarin forms" Grid.Row="0" />
<tabview:SfTabView Margin="0,0,0,2"
x:Name="SimTab"
VisibleHeaderCount="4"
TabHeaderPosition="Bottom"
DisplayMode="ImageWithText"
EnableSwiping="true"
Grid.Row="1" >
<tabview:SfTabView.Items>
<tabview:SfTabItem Title="Chat"
TitleFontSize="14"
IconFont="A"
FontIconFontFamily="{StaticResource fonts}"
SelectionColor="#FF00AFF0"
FontIconFontColor="#FF00AFF0"
TitleFontColor="#FF00AFF0">
<tabview:SfTabItem.Content>
<Label Text="Hai" />
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="Chat2"
TitleFontSize="14"
IconFont="&#xf000;"
FontIconFontFamily="{StaticResource fonts}"
SelectionColor="#FF00AFF0"
FontIconFontColor="#FF00AFF0"
TitleFontColor="#FF00AFF0">
<tabview:SfTabItem.Content>
<Label Text="Hai" />
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="like"
TitleFontSize="14"
IconFont="&#0041;"
FontIconFontFamily="{StaticResource fonts}"
SelectionColor="#FF00AFF0"
FontIconFontColor="#FF00AFF0"
TitleFontColor="#FF00AFF0">
<tabview:SfTabItem.Content>
<Label Text="Hello"/>
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="dislike"
TitleFontSize="14"
IconFont="&#0041;"
FontIconFontFamily="Fonts/OpenTypeFont.ttf"
SelectionColor="#FF00AFF0"
FontIconFontColor="#FF00AFF0"
TitleFontColor="#FF00AFF0">
<tabview:SfTabItem.Content>
<Label Text="How are"/>
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="status"
TitleFontSize="14"
IconFont="C"

```

```

FontIconFontFamily="{StaticResource fonts}"
SelectionColor="#FF00AFF0"
FontIconFontColor="#FF00AFF0"
TitleFontColor="#FF00AFF0">
<tabview:SfTabItem.Content>
<Label Text="You"/>
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
</tabview:SfTabView.Items>
</tabview:SfTabView>
</Grid>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabViewFontSample
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class TabView : ContentPage
    {
        SfTabView tabView;
        public TabView ()
        {
            InitializeComponent ();
            tabView = new SfTabView();
            var allContactsGrid = new Grid { BackgroundColor = Color.Red };
            var favoritesGrid = new Grid { BackgroundColor = Color.Green };
            var contactsGrid = new Grid { BackgroundColor = Color.Blue };
            var overflowButtonSettings = new OverflowButtonSettings();
            overflowButtonSettings.BackgroundColor = Color.Yellow;
            overflowButtonSettings.DisplayMode = OverflowButtonDisplayMode.Text;
            overflowButtonSettings.Title = "OverFlow";
            overflowButtonSettings.TitleFontSize = 10;
            overflowButtonSettings.TitleFontColor = Color.Blue;
            tabView.OverflowButtonSettings = overflowButtonSettings;
            tabView.EnableSwiping = false;
            var tabItems = new TabItemCollection
            {
                new SfTabItem()
                {
                    Title = "Calls",
                    Content = allContactsGrid,
                    IconFont = "A", // setting value for font icons as mentioned in *.ttf.
                    FontIconFontFamily = Device.RuntimePlatform == "iOS" ? "TabIcons" :
                    Device.RuntimePlatform == "Android" ? "TabIcons.ttf" :
                    "TabIcons.ttf#TabIcons",
                    FontIconFontColor = Color.LightBlue,
                    FontIconFontSize = 20
                },
                new SfTabItem()
                {
                    Title = "Favorites",

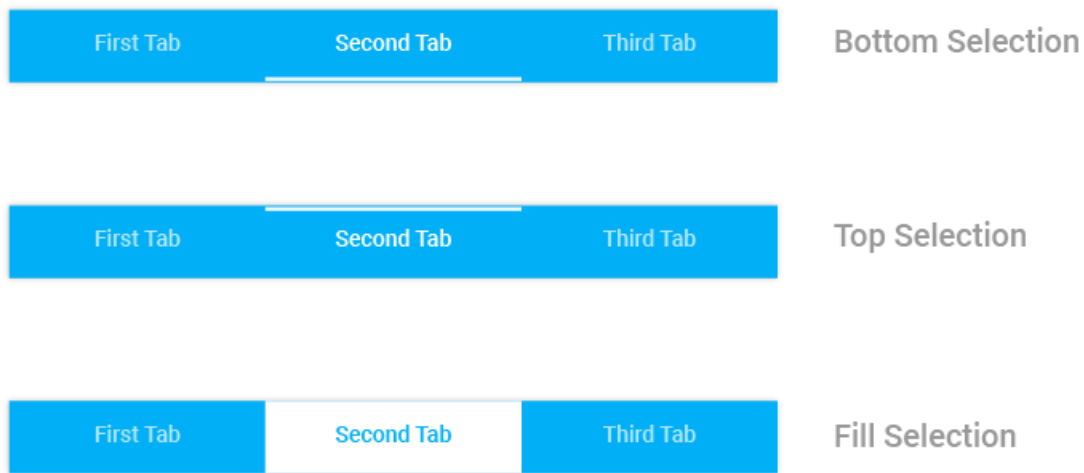
```

```
Content = favoritesGrid,
IconFont = "B", // setting value for font icons as mentioned in *.ttf.
FontIconFontFamily = Device.RuntimePlatform == "iOS" ? "TabIcons" :
Device.RuntimePlatform == "Android" ? "TabIcons.ttf" :
"TabIcons.ttf#TabIcons",
FontIconFontColor = Color.LightBlue,
FontIconFontSize = 20
},
new SfTabItem()
{
Title = "Contacts",
Content = contactsGrid,
IconFont = "C", // setting value for font icons as mentioned in *.ttf.
FontIconFontFamily = Device.RuntimePlatform == "iOS" ? "TabIcons" :
Device.RuntimePlatform == "Android" ? "TabIcons.ttf" :
"TabIcons.ttf#TabIcons",
FontIconFontColor = Color.LightBlue,
FontIconFontSize = 20
},
new SfTabItem()
{
Title = "Alternative",
Content = contactsGrid,
IconFont = "D", // setting value for font icons as mentioned in *.ttf.
FontIconFontFamily = Device.RuntimePlatform == "iOS" ? "TabIcons" :
Device.RuntimePlatform == "Android" ? "TabIcons.ttf" :
"TabIcons.ttf#TabIcons",
FontIconFontColor = Color.LightBlue,
FontIconFontSize = 20
}
};
tabView.Items = tabItems;
tabView.BackgroundColor = Color.Aqua;
tabView.OverflowMode = OverflowMode.DropDown;
this.Content = tabView;
}
}
```

You can refer this [link](#) for the simple sample defining the tab view items with font icons.

### Selection Indicator Strip

The selection indicator strip can be used to indicate the selected index of the tab view control. It can be customized with the built-in APIs that are available in the `SelectionIndicatorSettings` property of `SfTabView`.



The selection indicator can be positioned below the title or above the title, or else it can be filled in the entire selected header space.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
  <ContentPage.Content>
    <tabView:SfTabView VisibleHeaderCount="3"
    TabHeaderPosition="Bottom"
    OverflowMode="DropDown">
      <tabView:SfTabView.SelectionIndicatorSettings>
        <tabView:SelectionIndicatorSettings
        Color="Aqua"
        Position="Top"
        StrokeThickness="10"/>
      </tabView:SfTabView.SelectionIndicatorSettings>
      <tabView:SfTabItem Title="CEO">
        <tabView:SfTabItem.Content>
          <StackLayout>
            <Grid BackgroundColor="Green"/>
            <Button Text="Contacts" WidthRequest="300" />
            <Button Text="Location" WidthRequest="300" />
            <Button Text="Email" WidthRequest="300" />
          </StackLayout>
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
      <tabView:SfTabItem Title="Patients">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="Blue" x:Name="FavoritesGrid"/>
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
    </tabView:SfTabView>
  </ContentPage.Content>
</ContentPage>
```

```

<tabView:SfTabItem Title="Staff">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Alternative">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Olive" x:Name="AlternativeGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

## C#

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabView
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class TabView : ContentPage
{
SfTabView tabView;
public TabView ()
{
InitializeComponent ();
tabView = new SfTabView();
var allContactsGrid = new Grid { BackgroundColor = Color.Red };
var favoritesGrid = new Grid { BackgroundColor = Color.Blue };
var contactsGrid = new Grid { BackgroundColor = Color.Green };
var alternativeGrid = new Grid { BackgroundColor = Color.Olive };
var selectionIndicatorSettings = new SelectionIndicatorSettings();
selectionIndicatorSettings.Color = Color.Red;
selectionIndicatorSettings.Position = SelectionIndicatorPosition.Top;
selectionIndicatorSettings.StrokeThickness = 10;
tabView.SelectionIndicatorSettings = selectionIndicatorSettings;
tabView.TabHeaderPosition = TabHeaderPosition.Bottom;
tabView.OverflowMode = OverflowMode.DropDown;
var tabItems = new TabItemCollection
{
new SfTabItem()
{
Title = "CEO",
Content = allContactsGrid
},
new SfTabItem()
{
Title = "Patients",
Content = favoritesGrid
},
new SfTabItem()
{
Title = "Staff",
Content = contactsGrid
}
}
}
}

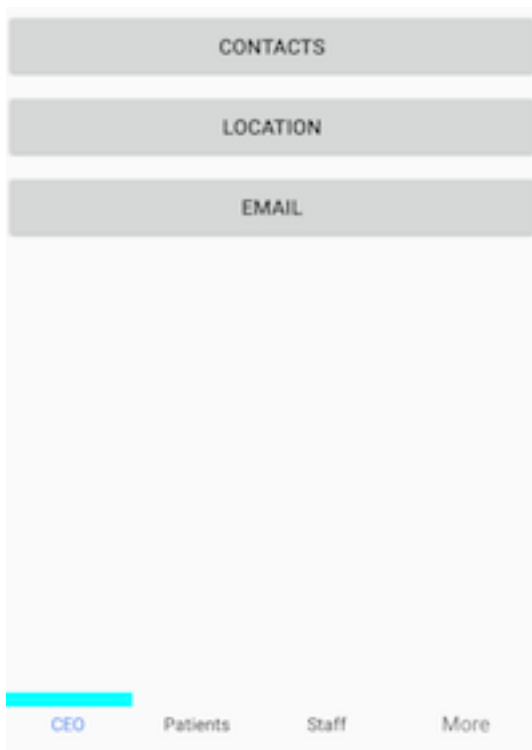
```

```

    }
    new SfTabItem()
    {
        Title = "Alternative",
        Content = alternativeGrid
    }
    };
    tabView.Items = tabItems;
    this.Content = tabView;
    }
    }
    }

```

**Note:** Stroke thickness will not be applicable when the selection indicator's position is set to "Fill".



### Positioning of header

Tab headers can be positioned either above the content or below the content. This can be done by setting the `TabHeaderPosition` property of `SfTabView`.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
<ContentPage.Content>
<tabView:SfTabView VisibleHeaderCount="3"
TabHeaderPosition="Bottom"
OverflowMode="DropDown">
<tabView:SfTabItem Title="CEO">

```

```

<tabView:SfTabItem.Content>
<StackLayout>
<Grid BackgroundColor="Green"/>
<Button Text="Contacts" WidthRequest="300" />
<Button Text="Location" WidthRequest="300" />
<Button Text="Email" WidthRequest="300" />
</StackLayout>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Patients">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="FavoritesGrid"/>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Staff">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Alternative">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Olive" x:Name="AlternativeGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

## C#

```

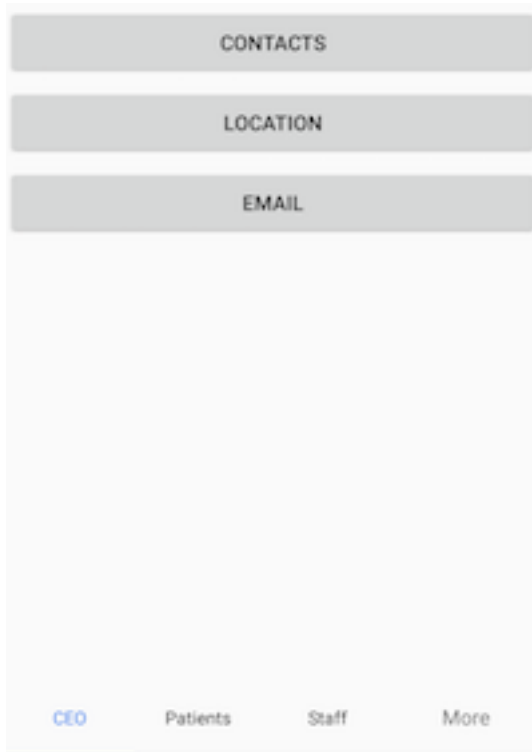
using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabView
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class TabView : ContentPage
{
SfTabView tabView;
public TabView ()
{
InitializeComponent ();
tabView = new SfTabView();
var allContactsGrid = new Grid { BackgroundColor = Color.Red };
var favoritesGrid = new Grid { BackgroundColor = Color.Blue };
var contactsGrid = new Grid { BackgroundColor = Color.Green };
var alternativeGrid = new Grid { BackgroundColor = Color.Olive };
tabView.TabHeaderPosition = TabHeaderPosition.Bottom;
tabView.OverflowMode = OverflowMode.DropDown;
var tabItems = new TabItemCollection
{
new SfTabItem()
{
Title = "CEO",
Content = allContactsGrid
},

```



```
new SfTabItem()
{
    Title = "Patients",
    Content = favoritesGrid
},
new SfTabItem()
{
    Title = "Staff",
    Content = contactsGrid
}
new SfTabItem()
{
    Title = "Alternative",
    Content = alternativeGrid
}
};
tabView.Items = tabItems;
this.Content = tabView;
}
}
```

When the header is not needed, set the `DisplayMode` property of `SfTabView` to `NoHeader`.



### Handling overflow tabs

When you have large number of tabs, by default, the scroller will be enabled to view the overflow of headers, if needed. It can be selected from the pop-up by setting the `OverflowMode` property of `SfTabView` to `DropDown`.

### XML

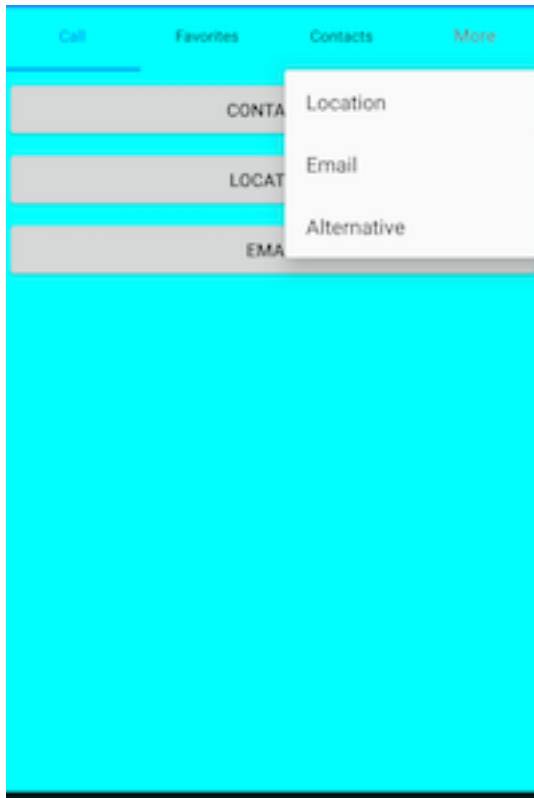
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
<ContentPage.Content>
<tabView:SfTabView OverflowMode="DropDown" VisibleHeaderCount="3"
BackgroundColor="Aqua">
<tabView:SfTabItem Title="Call">
<tabView:SfTabItem.Content>
<StackLayout>
<Grid BackgroundColor="Green" />
<Button Text="Contacts" WidthRequest="300" />
<Button Text="Location" WidthRequest="300" />
<Button Text="Email" WidthRequest="300" />
</StackLayout>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Favorites">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="FavoritesGrid"/>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Location">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Pink" x:Name="LocationGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Email">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Navy" x:Name="EmailGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Alternative">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="AlternativeGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>
```

## C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Syncfusion.XForms.TabView;
using Xamarin.Forms;
```

```
using Xamarin.Forms.Xaml;
namespace TabView
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class TabView : ContentPage
    {
        SfTabView tabView;
        public TabView ()
        {
            InitializeComponent ();
            tabView = new SfTabView();
            var allContactsGrid = new Grid { BackgroundColor = Color.Red };
            var favoritesGrid = new Grid { BackgroundColor = Color.Green };
            var contactsGrid = new Grid { BackgroundColor = Color.Blue };
            var tabItems = new TabItemCollection
            {
                new SfTabItem()
                {
                    Title = "Calls",
                    Content = allContactsGrid
                },
                new SfTabItem()
                {
                    Title = "Favorites",
                    Content = favoritesGrid
                },
                new SfTabItem()
                {
                    Title = "Contacts",
                    Content = contactsGrid
                },
                new SfTabItem()
                {
                    Title = "Location",
                    Content = contactsGrid
                },
                new SfTabItem()
                {
                    Title = "Email",
                    Content = contactsGrid
                },
                new SfTabItem()
                {
                    Title = "Alternative",
                    Content = contactsGrid
                }
            };
            tabView.Items = tabItems;
            tabView.BackgroundColor = Color.Aqua;
            tabView.OverflowMode = OverflowMode.DropDown;
            this.Content = tabView;
        }
    }
}
```

By selecting the drop-down option for tab view control, The “Overflow button” (or “More button”) will be added to the header. When you click this button, a pop-up will be displayed to navigate the other indices.



**Note:** The pop-up will display the text value and title value of the respective tab item.

How to customize the more button?

Appearance of the text can be customized through the APIs that are available on the `OverflowButtonSettings` property of `SfTabView`. This property has APIs to customize the both text and font icons available in the more button.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
<ContentPage.Content>
<tabView:SfTabView OverflowMode="DropDown" VisibleHeaderCount="3"
BackgroundColor="Aqua">
<tabView:SfTabView.OverflowButtonSettings>
<tabView:OverflowButtonSettings
BackgroundColor="Yellow"
DisplayMode="Text"
Title="OverFlow"
TitleFontSize="10"
TitleFontColor="Blue"/>
</tabView:SfTabView.OverflowButtonSettings>
<tabView:SfTabItem Title="Call">
```

```

<tabView:SfTabItem.Content>
<StackLayout>
<Grid BackgroundColor="Green" x:Name="CotactsGrid" />
<Button Text="Contacts" WidthRequest="300" />
<Button Text="Location" WidthRequest="300" />
<Button Text="Email" WidthRequest="300" />
</StackLayout>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Favorites">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="FavoritesGrid"/>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Location">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Pink" x:Name="ConttsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Email">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Navy" x:Name="Contactrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Alternative">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

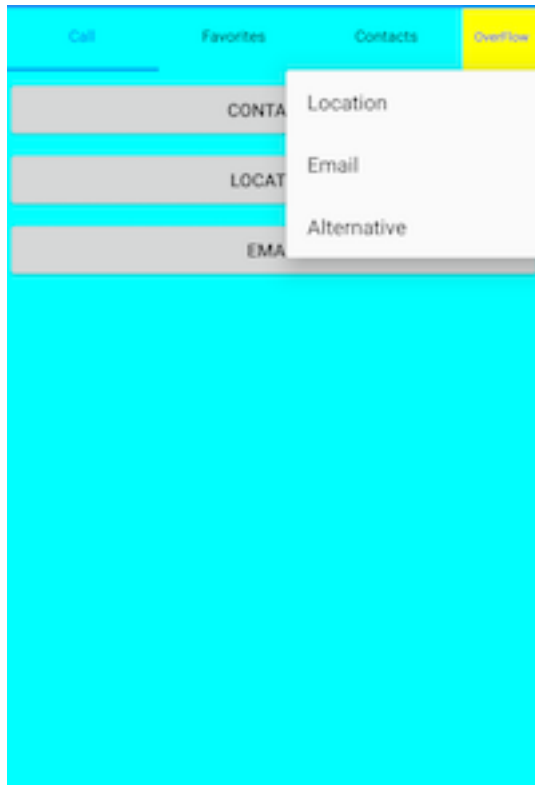
## C#

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabView
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class TabView : ContentPage
{
SfTabView tabView;
public TabView ()
{
InitializeComponent ();
tabView = new SfTabView();
var allContactsGrid = new Grid { BackgroundColor = Color.Red };
var favoritesGrid = new Grid { BackgroundColor = Color.Green };
var contactsGrid = new Grid { BackgroundColor = Color.Blue };

```

```
var overflowButtonSettings = new OverflowButtonSettings();
overflowButtonSettings.BackgroundColor = Color.Yellow;
overflowButtonSettings.DisplayMode = OverflowButtonDisplayMode.Text;
overflowButtonSettings.Title = "OverFlow";
overflowButtonSettings.TitleFontSize = 10;
overflowButtonSettings.TitleFontColor = Color.Blue;
tabView.OverflowButtonSettings = overflowButtonSettings;
var tabItems = new TabItemCollection
{
    new SfTabItem()
    {
        Title = "Calls",
        Content = allContactsGrid
    },
    new SfTabItem()
    {
        Title = "Favorites",
        Content = favoritesGrid
    },
    new SfTabItem()
    {
        Title = "Contacts",
        Content = contactsGrid
    },
    new SfTabItem()
    {
        Title = "Location",
        Content = contactsGrid
    },
    new SfTabItem()
    {
        Title = "Email",
        Content = contactsGrid
    },
    new SfTabItem()
    {
        Title = "Alternative",
        Content = contactsGrid
    }
};
tabView.Items = tabItems;
tabView.BackgroundColor = Color.Aqua;
tabView.OverflowMode = OverflowMode.DropDown;
this.Content = tabView;
}
```



### Custom Header

When built-in view is not needed, it can be overridden by adding custom views to the header in tabs. The tab view header can be customized by adding different views such as image, button, and label inside the header content. The following code sample demonstrates how to customize the header content as needed.

**Note:** The selection indicator setting properties will not work when using custom header.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
  <ContentPage.Content>
    <tabView:SfTabView BackgroundColor="Aqua">
      <tabView:SfTabItem Title="Call"
        SelectionColor="Aqua">
        <tabView:SfTabItem.HeaderContent>
          <Button
            Text="All Calls"
            BackgroundColor="Yellow"
            FontSize="10"
            Clicked="Handle_Clicked"/>
        </tabView:SfTabItem.HeaderContent>
        <tabView:SfTabItem.Content>
          <StackLayout>
            <Grid BackgroundColor="Green"/>
            <Button Text="Contacts" WidthRequest="300" />
          </StackLayout>
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
    </tabView:SfTabView>
  </ContentPage.Content>
</ContentPage>
```

```

<Button Text="Location" WidthRequest="300" />
<Button Text="Email" WidthRequest="300" />
</StackLayout>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Favorites">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="FavoritesGrid"/>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

**C#**

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabView
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class TabView : ContentPage
    {
        SfTabView tabView;
        public TabView ()
        {
            InitializeComponent ();
            tabView = new SfTabView();
            var allContactsGrid = new Grid { BackgroundColor = Color.Red };
            var favoritesGrid = new Grid { BackgroundColor = Color.Green };
            var contactsGrid = new Grid { BackgroundColor = Color.Blue };
            var overflowButtonSettings = new OverflowButtonSettings();
            overflowButtonSettings.BackgroundColor = Color.Yellow;
            overflowButtonSettings.DisplayMode = OverflowButtonDisplayMode.Text;
            overflowButtonSettings.TitleFontSize = 10;
            overflowButtonSettings.TitleFontColor = Color.Blue;
            tabView.OverflowButtonSettings = overflowButtonSettings;
            tabView.EnableSwiping = false;
            var allCallsButton = new Button();
            Xamarin.Forms.Button button = new Xamarin.Forms.Button()
            {
                Text = "All Calls",
                BackgroundColor = Color.Yellow,
                FontSize = 10
            };
            var tabItems = new TabItemCollection
            {
                new SfTabItem()
                {
                    HeaderContent = button,

```



```

Content = allContactsGrid,
SelectionColor = Color.Red
},
new SfTabItem()
{
    Title = "Favorites",
    Content = favoritesGrid
},
new SfTabItem()
{
    Title = "Contacts",
    Content = contactsGrid
}
};
tabView.Items = tabItems;
tabView.BackgroundColor = Color.Aqua;
this.Content = tabView;
}
void Handle_Clicked(object sender, System.EventArgs e)
{
}
}
}
}

```

The following code sample demonstrates customizing the header by adding image and label as the header content of the tab view.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabView.TabView">
<ContentPage.BindingContext>
<local:ContactsViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfTabView x:Name="tabView"
TabHeight="80"
BackgroundColor="#BEBEBE"
EnableSwiping="False"
VisibleHeaderCount="3"
Margin="0,40,0,0">
<syncfusion:SfTabView.SelectionIndicatorSettings>
<syncfusion:SelectionIndicatorSettings Color="Green" Position="Fill"
StrokeThickness="4"/>
</syncfusion:SfTabView.SelectionIndicatorSettings>
<syncfusion:SfTabItem>
<syncfusion:SfTabItem.HeaderContent>
<Grid VerticalOptions="Center"
BackgroundColor="#eea782"
HeightRequest="400"
WidthRequest="500"
x:Name="ChatsHeader"
StyleId="ChatsHeader"

```

```

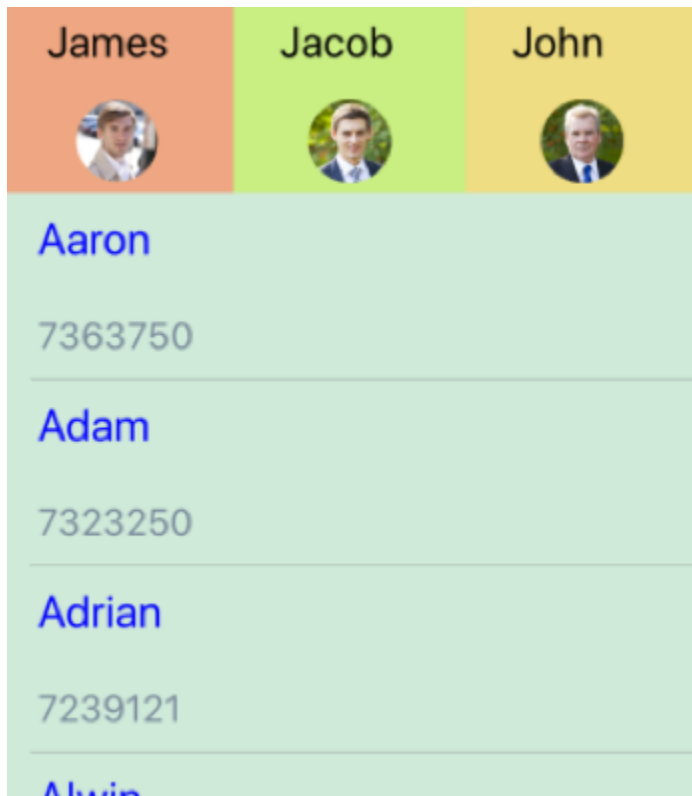
HorizontalOptions="Center">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Label Text="James"
TextColor="Black"
FontSize="23"
Grid.Row="0"
WidthRequest="80"
HeightRequest="50"
VerticalOptions="Center"
HorizontalOptions="Center"/>
<Grid Grid.Row="1">
<Image HeightRequest="100"
WidthRequest="70"
Source="a0.png"/>
</Grid>
</Grid>
</syncfusion:SfTabItem.HeaderContent>
<syncfusion:SfTabItem.Content>
<Grid BackgroundColor="Yellow" x:Name="FavoritesGrid" />
</syncfusion:SfTabItem.Content>
</syncfusion:SfTabItem>
<syncfusion:SfTabItem>
<syncfusion:SfTabItem.HeaderContent>
<Grid VerticalOptions="Center"
BackgroundColor="#C9EE82"
HeightRequest="400"
WidthRequest="500"
x:Name="ChatsHeader2"
StyleId="ChatsHeader"
HorizontalOptions="Center">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Label Text="Jacob"
TextColor="Black"
FontSize="23"
Grid.Row="0"
WidthRequest="80"
HeightRequest="50"
VerticalOptions="Center"
HorizontalOptions="Center"/>
<Grid Grid.Row="1">
<Image HeightRequest="100"
WidthRequest="70"
Source="a2.png"/>
</Grid>
</Grid>
</syncfusion:SfTabItem.HeaderContent>
<syncfusion:SfTabItem.Content>
<Grid BackgroundColor="Blue"
x:Name="ContactsGrid" />
</syncfusion:SfTabItem.Content>
</syncfusion:SfTabItem>

```

```

<syncfusion:SfTabItem>
<syncfusion:SfTabItem.HeaderContent>
<Grid VerticalOptions="Center"
BackgroundColor="#eedd82"
HeightRequest="400"
WidthRequest="500"
x:Name="ChatsHeader3"
StyleId="ChatsHeader"
HorizontalOptions="Center">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Label Text="John"
TextColor="Black"
FontSize="23"
Grid.Row="0"
WidthRequest="80"
HeightRequest="50"
VerticalOptions="Center"
HorizontalOptions="Center"/>
<Grid Grid.Row="1">
<Image HeightRequest="100"
WidthRequest="70"
Source="a1.png"/>
</Grid>
</Grid>
</syncfusion:SfTabItem.HeaderContent>
<syncfusion:SfTabItem.Content>
<ListView x:Name="ContactListView"
ItemsSource="{Binding ContactList}"
BackgroundColor="#cfead9"
RowHeight="100">
<ListView.BindingContext>
<local:ContactsViewModel />
</ListView.BindingContext>
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<StackLayout Orientation="Vertical">
<Label Text="{Binding Name}"
FontSize="24"
TextColor="Blue" />
<Label Text="{Binding Number}"
FontSize="20"
TextColor="LightSlateGray" />
</StackLayout>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</syncfusion:SfTabItem.Content>
</syncfusion:SfTabItem>
</syncfusion:SfTabView>
</ContentPage.Content>
</ContentPage>

```



How to handle the events for custom view with tab view

When you use the button or similar control with the clicked event, it can be handled directly and set to the `SelectedIndex` property to navigate the clicked view.

#### C#

```
private void Button_Clicked(object sender, System.EventArgs e)
{
    tabView.SelectedIndex = 0;
}
```

**Note:** If the click event is not available, it can be achieved by setting the `TapGestureRecognizer` to the custom view.

#### Image Source

The `ImageSource` property customizes the icon image of `SfTabView` by adding a custom image.

#### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:tabView="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
x:Class="TabViewAutomationSample.TabViewAutomationSample">
<ContentPage.Content>
<tabView:SfTabView BackgroundColor="Aqua" VisibleHeaderCount="3"
DisplayMode="ImageWithText">
```

```

<tabView:SfTabItem Title="Call" ImageSource="phone.png">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Red" x:Name="AllContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Favorites" ImageSource="home.png">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="FavoritesGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts" ImageSource="review.png">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Email" ImageSource="notifications.png">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="Olive" x:Name="EmailGrid" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</ContentPage.Content>
</ContentPage>

```

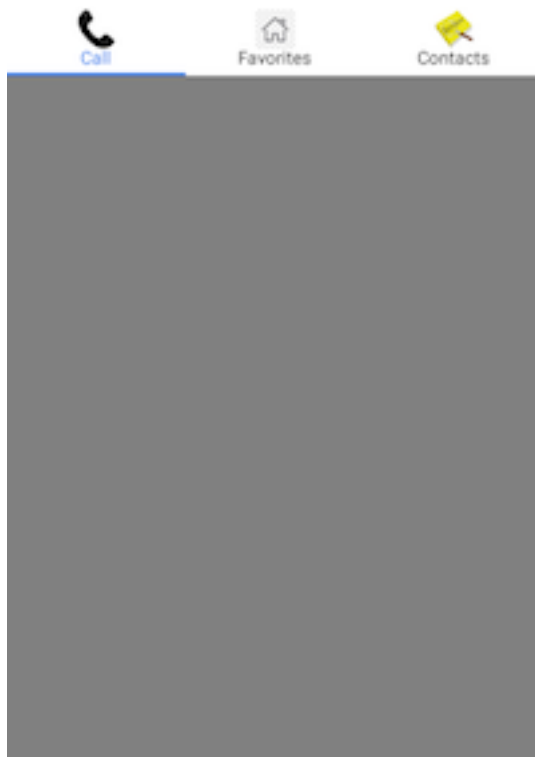
**C#**

```

using Syncfusion.XForms.TabView;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace TabViewAutomationSample
{
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class TabView : ContentPage
{
SfTabView tabView;
public TabView ()
{
InitializeComponent ();
var tabView = new SfTabView();
var allContactsGrid = new Grid { BackgroundColor = Color.Red };
var favoritesGrid = new Grid { BackgroundColor = Color.Green };
var contactsGrid = new Grid { BackgroundColor = Color.Blue };
var emailGrid = new Grid { BackgroundColor = Color.Blue };
var tabItems = new TabItemCollection
{
new SfTabItem()
{
Title = "Calls",
Content = allContactsGrid,
ImageSource = "phone.png"
},
new SfTabItem()
{
Title = "Favorites",
Content = favoritesGrid,
ImageSource = "home.png"
}
}
}
}

```

```
},  
new SfTabItem()  
{  
    Title = "Contacts",  
    Content = contactsGrid,  
    ImageSource = "review.png"  
},  
new SfTabItem()  
{  
    Title = "Email",  
    Content = EmailGrid,  
    ImageSource = "notification.png"  
}  
};  
tabView.Items = tabItems;  
this.Content = tabView;  
}  
}
```



## Nested Tab Items

Nested Tab items can be configured in tab view through the `Items` property of `SfTabView`, which holds the nested collection of `SfTabItem` through `TabItemsCollection`.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
    xmlns:local="clr-  
        namespace:TabViewAutomationSample;assembly=TabViewAutomationSample"
```

```

xmlns:tabview="clr-
namespace:Syncfusion.XForms.TabView;assembly=Syncfusion.SfTabView.XForms"
BackgroundColor="#2196F3" x:Name="root"
xmlns:data="clr-namespace:TabViewAutomationSample"
x:Class="TabViewAutomationSample.SBNestedTab">
<ContentPage.Resources>
<ResourceDictionary>
<OnPlatform x:TypeArguments="x:String" x:Key="fontfamily" iOS="NestedTab"
Android="NestedTab.ttf" />
<OnPlatform x:TypeArguments="x:String" x:Key="controlfontfamily"
iOS="NestedTab" Android="NestedTab.ttf#NestedTab" />
<data:CustomViewConverter x:Key="nestedtabfont"/>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<Grid BackgroundColor="White">
<tabview:SfTabView EnableSwiping="False"
Margin="0,0,0,2"
x:Name="SimTab"
VisibleHeaderCount="2"
BackgroundColor="#f1f1f1" >
<tabview:SfTabView.Items>
<tabview:SfTabItem Title="SIM 1"
TitleFontSize="14"
SelectionColor="White"
FontIconFontColor="White"
TitleFontColor="Black">
<tabview:SfTabItem.Content>
<tabview:SfTabView EnableSwiping="False"
VisibleHeaderCount="3"
BackgroundColor="#FF00FF0"
DisplayMode="Image">
<tabview:SfTabView.SelectionIndicatorSettings>
<tabview:SelectionIndicatorSettings Color="White"/>
</tabview:SfTabView.SelectionIndicatorSettings>
<tabview:SfTabView.Items>
<tabview:SfTabItem Title="Recent"
IconFont="H"
FontIconFontColor="White"
TitleFontColor="White"
SelectionColor="White"
FontIconFontFamily="{Binding Converter={StaticResource NestedTabFont}}">
<tabview:SfTabItem.Content>
<tabview:SfTabView EnableSwiping="False"
x:Name="CallsTab1"
VisibleHeaderCount="3"
BackgroundColor="#FFF1F1F1"
TabHeight="72"
DisplayMode="ImageWithText"
TabHeaderPosition="Bottom">
<tabview:SfTabView.Items>
<tabview:SfTabItem Title="Dialled Calls"
IconFont="E"
FontIconFontSize="22"
FontIconFontFamily="{Binding Converter={StaticResource NestedTabFont}}">
<tabview:SfTabItem.Content>
<Grid BackgroundColor="Gray" x:Name="FavouritesGrid" />

```

```

</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="Received Calls"
IconFont="F"
FontIconFontSize="22"
FontIconFontFamily="{Binding Converter={StaticResource NestedTabFont}}">
<tabview:SfTabItem.Content>
<Grid BackgroundColor="Blue" x:Name="ContactsGrid" />
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="Missed Calls"
IconFont="O"
FontIconFontSize="22"
FontIconFontFamily="{Binding Converter={StaticResource NestedTabFont}}">
<tabview:SfTabItem.Content>
<Grid BackgroundColor="Olive" x:Name="EmailGrid" />
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
</tabview:SfTabView.Items>
</tabview:SfTabView>
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="Favorites"
IconFont="Z"
FontIconFontColor="White"
TitleFontColor="White"
SelectionColor="White"
FontIconFontFamily="{Binding Converter={StaticResource NestedTabFont}}">
<tabview:SfTabItem.Content>
<Grid BackgroundColor="Green" x:Name="NativeGrid" />
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="Contacts"
IconFont="N"
FontIconFontColor="White"
TitleFontColor="White"
SelectionColor="White"
FontIconFontFamily="{Binding Converter={StaticResource NestedTabFont}}">
<tabview:SfTabItem.Content>
<tabview:SfTabView EnableSwiping="False"
VisibleHeaderCount="2"
x:Name="ContactTab1"
DisplayMode="ImageWithText"
TabHeight="72"
BackgroundColor="White"
TabHeaderPosition="Bottom">
<tabview:SfTabView.Items>
<tabview:SfTabItem Title="All Contacts"
IconFont="f"
FontIconFontSize="22"
FontIconFontFamily="{Binding Converter={StaticResource NestedTabFont}}">
<tabview:SfTabItem.Content>
<Grid BackgroundColor="Green" />
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="Speed Dial"
IconFont="g"

```



```

FontIconFontSize="22"
FontIconFontFamily="{Binding Converter={StaticResource NestedTabFont}}">
<tabview:SfTabItem.Content>
<Grid BackgroundColor="Olive" />
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
</tabview:SfTabView.Items>
</tabview:SfTabView>
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
</tabview:SfTabView.Items>
</tabview:SfTabView>
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
<tabview:SfTabItem Title="SIM 2"
TitleFontSize="14"
SelectionColor="White"
FontIconFontColor="White"
TitleFontColor="Black">
<tabview:SfTabItem.Content>
<Grid BackgroundColor="Olive" />
</tabview:SfTabItem.Content>
</tabview:SfTabItem>
</tabview:SfTabView.Items>
</tabview:SfTabView>
</Grid>
</ContentPage.Content>
</ContentPage>

```

## C#

```

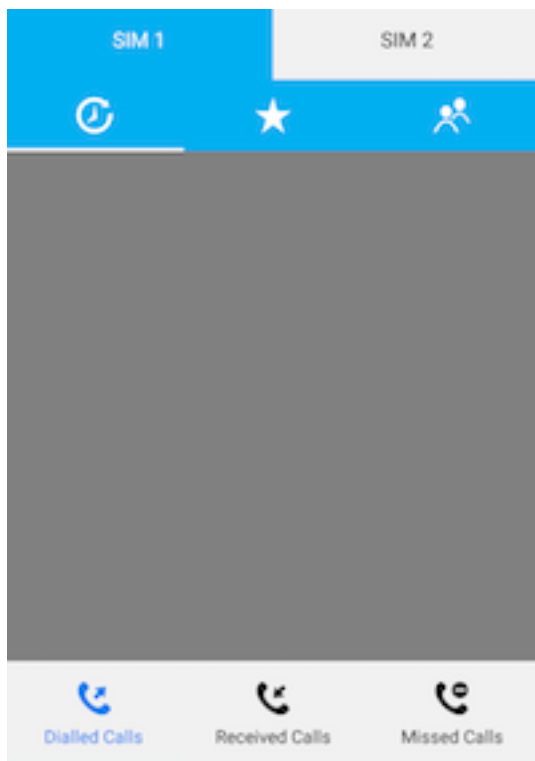
using System;
using System.Collections.Generic;
using System.Globalization;
using Syncfusion.XForms.TabView;
using Xamarin.Forms;
namespace TabViewAutomationSample
{
    public partial class SBNestedTab : ContentPage
    {
        public SBNestedTab()
        {
            InitializeComponent();
        }
    }
    public class CustomViewConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter,
            CultureInfo culture)
        {
            if (Device.RuntimePlatform == "Android")
            {
                if (parameter != null && parameter is string)
                {
                    return "NestedTab.ttf#" + parameter.ToString();
                }
            }
        }
    }
}

```

```

else
{
    return "NestedTab.ttf";
}
}
else if (Device.RuntimePlatform == "iOS")
{
    return "NestedTab";
}
else
{
    return "/Assets/Fonts/NestedTab.ttf#NestedTab";
}
}
public object ConvertBack(object value, Type targetType, object parameter,
    CultureInfo culture)
{
    throw new NotImplementedException();
}
}
public class CustomFrame : Frame
{
}
}

```



### CenterButtonSettings

This section explains how to create and customize The Xamarin.Forms SfTabView CenterButton. To enable CenterButton, set the **OverflowMode** of Xamarin.Forms SfTabView to **CenterButton**.

### XML

```
<tabView:SfTabView OverflowMode="CenterButton"
x:Name="tabView">
</tabView:SfTabView>
```

**C#**

```
public MainPage()
{
    InitializeComponent();
    SfTabView tabView = new SfTabView();
    tabView.OverflowMode = OverflowMode.CenterButton;
    this.Content = tabView;
}
```

## Customize CenterButtonSettings

We can customize the CenterButton using the properties of CenterButtonSetting. The following properties are used to customize the view of CenterButton BackgroundColor, BorderColor, BorderThickness, Height, Title, TitleFontAttributes, TitleFontColor, TitleFontSize, Width.

**XML**

```
<tabView:SfTabView.CenterButtonSettings>
<tabView:CenterButtonSettings Height="80" Width="100"
Title="Center Button" TitleFontColor="Green"
TitleFontAttributes="Bold">
</tabView:CenterButtonSettings>
</tabView:SfTabView.CenterButtonSettings>
```

**C#**

```
public MainPage()
{
    InitializeComponent();
    var centerButton = tabView.CenterButtonSettings;
    centerButton.Height = 80;
    centerButton.Width = 100;
    centerButton.Title = "Center Button";
    centerButton.TitleFontAttributes = FontAttributes.Bold;
    centerButton.TitleFontColor = Color.Green;
}
```

## CenterButtonTapped event

When CenterButton is tapped, the CenterButtonTapped event occurs. Using this event we can set alert message.

**XML**

```
<tabView:SfTabView CenterButtonTapped="TabView_CenterButtonTapped">
</tabView:SfTabView>
```

**C#**

```
public MainPage()
```

```
{
InitializeComponent();
tabView.CenterButtonTapped += TabView_CenterButtonTapped;
}
private void TabView_CenterButtonTapped(object sender, EventArgs e)
{
DisplayAlert("Message", "CenterButton Clicked", "Ok");
}
```

### Custom CenterButton

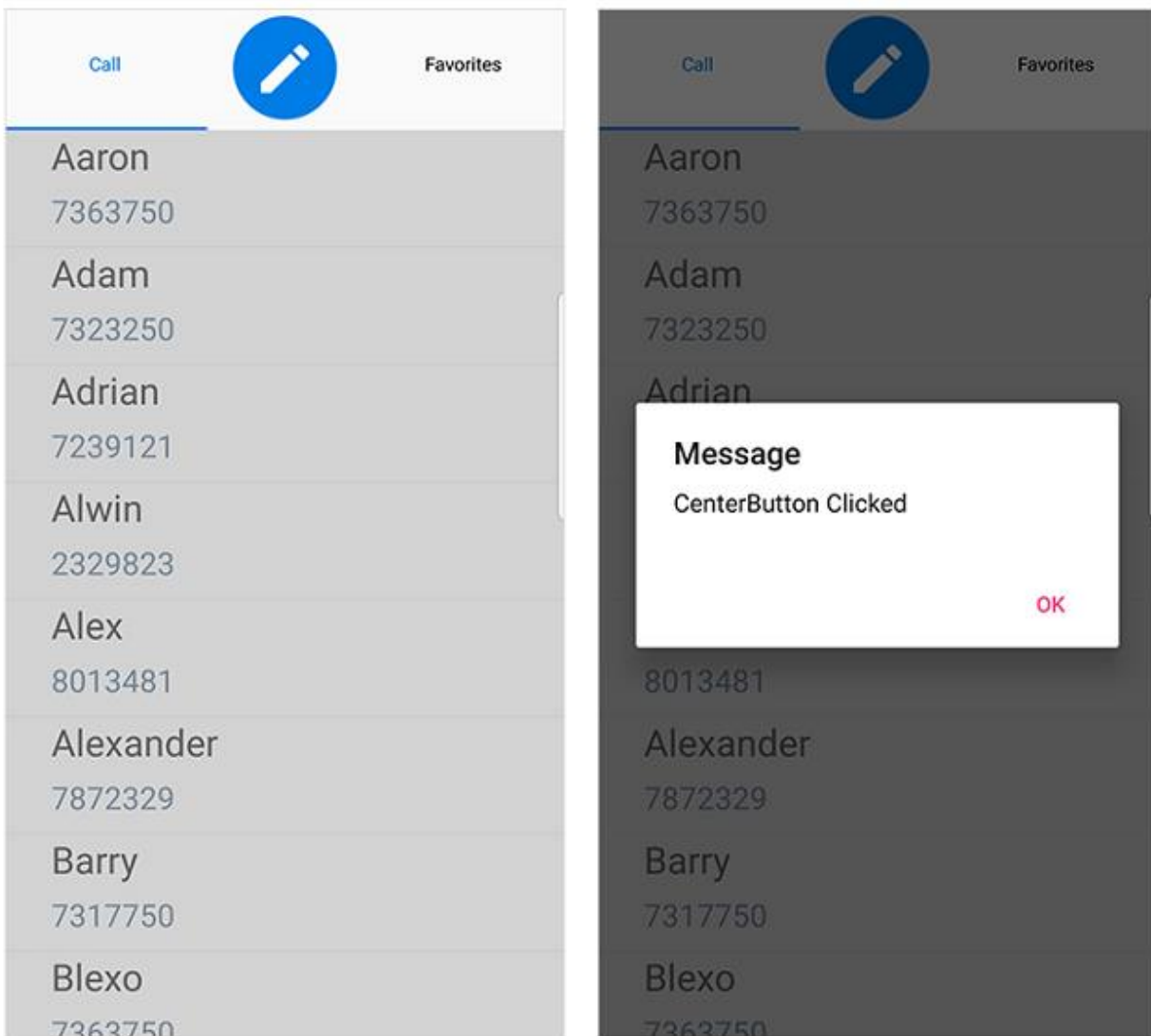
When built-in view is not needed, it can be overridden by adding custom views to the CenterButtonView. The CenterButton view can be customized by adding images, labels, buttons inside the CenterButtonView. Refer the following code sample to know about customizing the view of CenterButton.

### XML

```
<tabView:SfTabView.CenterButtonView>
<Grid>
<Image Source="Compose.png"
Aspect="AspectFill"
VerticalOptions="CenterAndExpand"
HorizontalOptions="CenterAndExpand">
<Image.GestureRecognizers>
<TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" />
</Image.GestureRecognizers>
</Image>
</Grid>
</tabView:SfTabView.CenterButtonView>
```

### C#

```
public MainPage()
{
InitializeComponent();
-----
Grid customCenterButtonGrid = new Grid();
Image image = new Image();
TapGestureRecognizer tapGestureRecognizer = new TapGestureRecognizer();
image.Source = ImageSource.FromFile("Compose.png");
image.Aspect = Aspect.AspectFill;
image.VerticalOptions = LayoutOptions.CenterAndExpand;
image.HorizontalOptions = LayoutOptions.CenterAndExpand;
tapGestureRecognizer.Tapped += TapGestureRecognizer_Tapped; ;
image.GestureRecognizers.Add(tapGestureRecognizer);
customCenterButtonGrid.Children.Add(image);
tabView.CenterButtonView = customCenterButtonGrid;
-----
}
private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
{
DisplayAlert("Message", "CenterButton Clicked", "Ok");
}
```



Please find the sample from the following [Sample](#).

### TabItemTapped

Whenever the TabItem is tapped, the **TabItemTapped** event will occur. Using this event, you can Modify the selected Tab Item properties.

### XML

```
<ContentPage.Content>
  <StackLayout>
    <tabView:SfTabView x:Name="tabView" VerticalOptions="FillAndExpand"
      TabItemTapped="TabView_TabItemTapped"
      VisibleHeaderCount="3">
      <tabView:SfTabItem Title="Call">
        <tabView:SfTabItem.Content>
          <Grid BackgroundColor="LightGreen" />
        </tabView:SfTabItem.Content>
      </tabView:SfTabItem>
    </tabView:SfTabView>
  </StackLayout>
</ContentPage.Content>
```

```
<tabView:SfTabItem Title="Favorites">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="LightBlue"/>
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
<tabView:SfTabItem Title="Contacts">
<tabView:SfTabItem.Content>
<Grid BackgroundColor="LightGreen" />
</tabView:SfTabItem.Content>
</tabView:SfTabItem>
</tabView:SfTabView>
</StackLayout>
</ContentPage.Content>
```

## C#

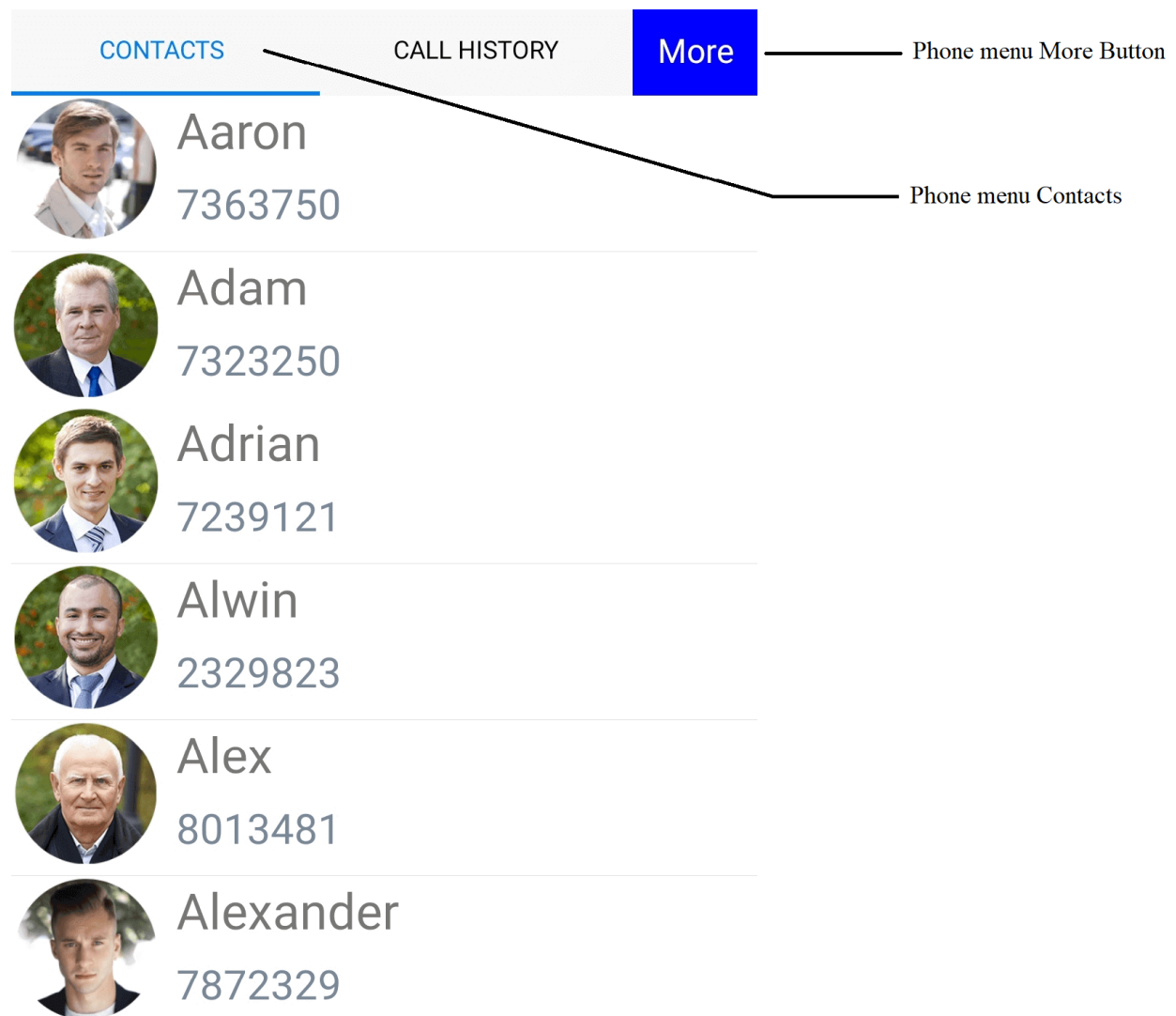
```
public MainPage ()
{
    InitializeComponent();
    tabView.TabItemTapped += TabView_TabItemTapped;
}

private void TabView_TabItemTapped(object sender,
Syncfusion.XForms.TabView.TabItemTappedEventArgs e)
{
    DisplayAlert("TabViewItemTapped", e.TabItem.Title + " Item Tapped", "Ok");
}
```

## AutomationId

The SfTabView control has built-in **AutomationId** for inner elements. The **AutomationId** API allows the automation framework to find and interact with the inner elements of the SfTabView control. To keep unique AutomationId, these inner elements' AutomationIds are updated based on the control's **AutomationId**.

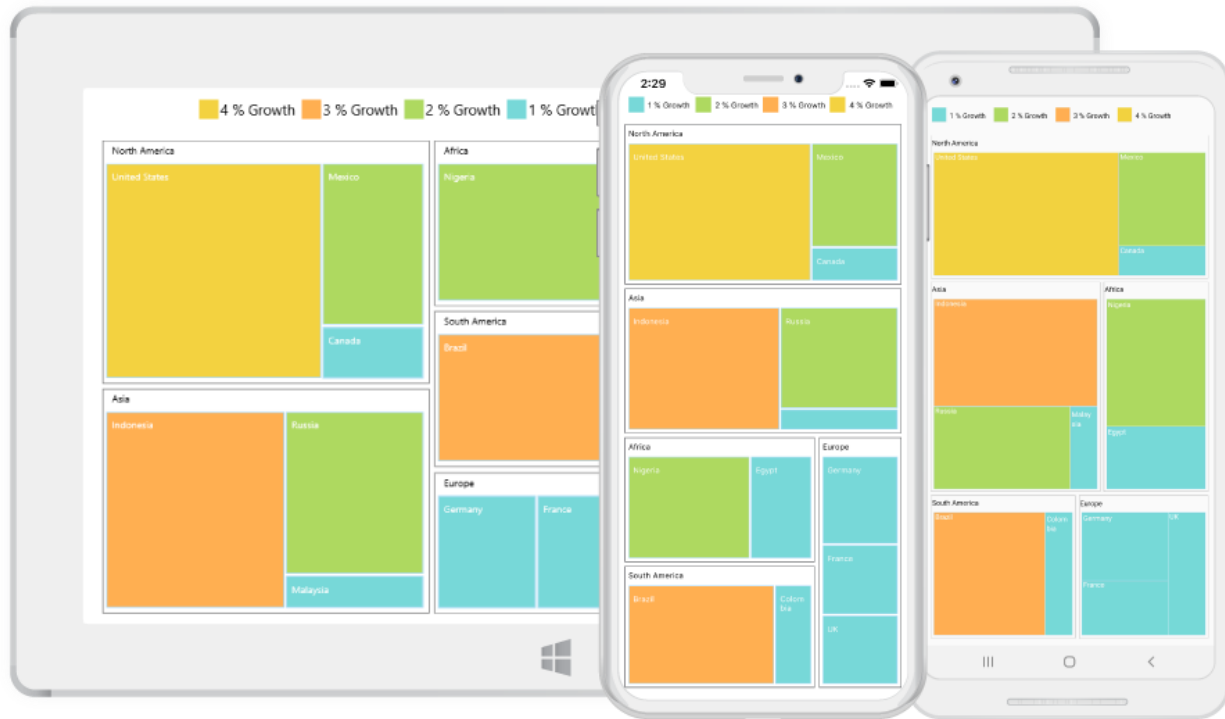
For example, if you set SfTabView's **AutomationId** as "Phone menu", then the Automation framework will interact with the More button as "Phone menu More Button". The following screenshot illustrates the AutomationIds of inner elements. The Automation framework will also interact with the Center Button and the element's inside the tab item content with the element's AutomationId.



## SfTreeMap

### Overview

The TreeMap control for Xamarin.Forms provides a simple yet effective way to visualize flat or hierarchical data as clustered rectangles with a specific weighted attribute determining the size of the rectangle.



### Key features

- **Levels** - Define the levels of various flat data and hierarchical data collections.
- **Layouts** - Determine the visual representation of nodes belonging to all the levels in TreeMap using the layouts such as Squarified, SliceAndDiceAuto, SliceAndDiceHorizontal, and SliceAndDiceVertical.
- **Visualization for colors** - Customize the colors of the leaf nodes in TreeMap.
- **Tooltip** - Provides additional information about the leaf nodes.
- **Selection** - Allows you select or highlight specific leaf nodes.
- **Legends** - Help you relate data with the leaf nodes.
- **DataLabels** - Show group path information on the leaf node with Trim, Wrap, and Hide options.

### Getting Started

This section explains the steps required to configure the TreeMap control in a real-time scenario and provides a walk-through on some of the customization features available in the TreeMap control.

#### Adding SfTreeMap reference

You can add SfTreeMap reference using one of the following methods:

##### Method 1: Adding SfTreeMap reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfTreeMap). To add SfTreeMap to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfTreeMap](https://www.nuget.org/packages/Syncfusion.Xamarin.SfTreeMap), and then install it.

![Adding SfTreeMap reference from NuGet](Getting-Started\_images/Adding SfTreeMap reference.png)

**Note:** Install the same version of SfTreeMap NuGet in all the projects.

##### Method 2: Adding SfTreeMap reference from toolbox



Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfTreeMap control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

### Method 3: Adding SfTreeMap assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfTreeMap.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.SfTreeMap.Android.dll Syncfusion.SfTreeMap.XForms.Android.dll Syncfusion.SfTreeMap.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.SfTreeMap.iOS.dll Syncfusion.SfTreeMap.XForms.iOS.dll Syncfusion.SfTreeMap.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.SfTreeMap.UWP.dll Syncfusion.SfTreeMap.XForms.UWP.dll Syncfusion.SfTreeMap.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

**Note:** Currently an additional step is required for UWP project. You need to create an instance of the TreeMap custom renderer. If you are adding the references from toolbox, this step is not needed.

Create an instance of SfTreeMapRenderer in the MainPage constructor of the UWP project as demonstrated in the following code sample.

### C#

```
public MainPage ()
{
```

```
new SfTreeMapRenderer ();  
...  
}
```

Create an instance of `SfTreeMapRenderer` in the `FinishedLaunching` overridden method of `AppDelegate` class in the iOS Project as demonstrated in the following code sample.

### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary  
options)  
{  
...  
new SfTreeMapRenderer ();  
...  
}
```

### Initializing TreeMap

The Treemap control can be configured entirely in C# or using XAML markup.

The first step is to create a `TreeMap` object.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
xmlns:treemap="clr-  
namespace:Syncfusion.SfTreeMap.XForms;assembly=Syncfusion.SfTreeMap.XForms"  
x:Class="TreeMap.Page3">  
  <ContentPage.Content>  
    <treemap:SfTreeMap></treemap:SfTreeMap>  
  </ContentPage.Content>  
</ContentPage>
```

### C#

```
public Page3 ()  
{  
  InitializeComponent ();  
  SfTreeMap treeMap = new SfTreeMap();  
  this.Content = treeMap;  
}
```

### Populating TreeMap items

The Treemap items can be populated in two ways using the following properties:

- `DataSource`
- `Items`

Using the `DataSource` property, you can bind the tree map data collection to it. To render leaf nodes for underlying data, levels and leaf item settings have to be specified; it is explained in detail in the [Levels](#)

section.

### XML

```
<ContentPage.BindingContext>
<local:PopulationViewModel></local:PopulationViewModel>
</ContentPage.BindingContext>
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}" WeightValuePath="Population">
<treemap:SfTreeMap.LeafItemSettings>
<treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
>
</treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
<treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
GroupGap =" 5" ShowHeader = "true">
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
</treemap:SfTreeMap>
```

### 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; }
}

```

The [Items](#) property accepts a collection of TreeMapItems as input. You can bind tree map items to the Items collection as demonstrated in the following code sample.

### XML

```

<ContentPage.BindingContext>
<local:DataModel></local:DataModel>
</ContentPage.BindingContext>
<ContentPage.Content>
<treemap:SfTreeMap x:Name="treeMap" Items="{Binding
TreeMapItems}"></treemap:SfTreeMap>
</ContentPage.Content>

```

### C#

```

SfTreeMap treeMap = new SfTreeMap();
DataModel model = new DataModel();
treeMap.Items = model.TreeMapItems;
public class DataModel : BindableObject
{
    public static readonly BindableProperty TreeMapItemsProperty =
        BindableProperty.Create<DataModel, ObservableCollection<TreeMapItem>>(p =>
            p.TreeMapItems, null, BindingMode.TwoWay, null, null, null, null);
    public ObservableCollection<TreeMapItem> TreeMapItems
    {
        get { return
            (ObservableCollection<TreeMapItem>)GetValue(TreeMapItemsProperty); }
        set { SetValue(TreeMapItemsProperty, value); }
    }
    public DataModel()
    {
        this.TreeMapItems = new ObservableCollection<TreeMapItem>();
        TreeMapItems.Add(new TreeMapItem() { Label = "Indonesia", ColorWeight = 3,
            Weight = 237641326 });
        TreeMapItems.Add(new TreeMapItem() { Label = "Russia", ColorWeight = 2,
            Weight = 152518015 });
        TreeMapItems.Add(new TreeMapItem() { Label = "United States", ColorWeight =
            4, Weight = 315645000 });
        TreeMapItems.Add(new TreeMapItem() { Label = "Mexico", ColorWeight = 2,
            Weight = 112336538 });
        TreeMapItems.Add(new TreeMapItem() { Label = "Nigeria", ColorWeight = 2,
            Weight = 170901000 });
        TreeMapItems.Add(new TreeMapItem() { Label = "Egypt", ColorWeight = 1,
            Weight = 83661000 });
    }
}

```

```

TreeMapItems.Add(new TreeMapItem() { Label = "Germany", ColorWeight = 1,
Weight = 81993000 });
TreeMapItems.Add(new TreeMapItem() { Label = "France", ColorWeight = 1,
Weight = 65605000 });
TreeMapItems.Add(new TreeMapItem() { Label = "UK", ColorWeight = 1, Weight =
63181775 });
}
}

```

### Grouping TreeMap items using levels

You can group TreeMapItems using the following two types of levels:

- TreeMap Flat Level
- TreeMap Hierarchical Level

The [Levels](#) are explained in detail in the [TreeMapLevels](#) section.

### Customizing the appearance of TreeMap by range

You can differentiate the nodes based on their values and colors using [RangeColorMapping](#). You can define the color value range using the [From](#) and [To](#) properties in [Range](#). The values of [From](#) and [To](#) properties depend on underlying data bound to the [ColorValuePath](#) property.

#### XML

```

<treemap:SfTreeMap.LeafItemColorMapping>
<treemap:RangeColorMapping>
<treemap:RangeColorMapping.Ranges>
<treemap:Range LegendLabel = "1 % Growth" From = "0" To = "1" Color =
"#77D8D8" />
<treemap:Range LegendLabel = "2 % Growth" From = "0" To = "2" Color =
"#AED960" />
<treemap:Range LegendLabel = "3 % Growth" From = "0" To = "3" Color =
"#FFAF51" />
<treemap:Range LegendLabel = "4 % Growth" From = "0" To = "4" Color =
"#F3D240" />
</treemap:RangeColorMapping.Ranges>
</treemap:RangeColorMapping>
</treemap:SfTreeMap.LeafItemColorMapping>

```

#### C#

```

...
ObservableCollection<Range> ranges = new ObservableCollection<Range>();
ranges.Add(new Range() { LegendLabel="1 % Growth", From = 0, To = 1, Color =
Color.FromHex("#77D8D8") });
ranges.Add(new Range() { LegendLabel = "2 % Growth", From = 0, To = 2, Color
= Color.FromHex("#AED960") });
ranges.Add(new Range() { LegendLabel = "3 % Growth", From = 0, To = 3, Color
= Color.FromHex("#FFAF51") });
ranges.Add(new Range() { LegendLabel = "4 % Growth", From = 0, To = 4, Color
= Color.FromHex("#F3D240") });
treeMap.LeafItemColorMapping = new RangeColorMapping () { Ranges = ranges };

```

### LeafItemSetting

You can customize the tree map leaf nodes using [LeafItemSettings](#).

#### XML

```
<treemap:SfTreeMap.LeafItemSettings>
<treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
>
</treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
```

#### C#

```
...
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
```

### Enabling legends

The color value of leaf nodes can be tracked using tree map legend. The legend support is applicable only for the TreeMap whose leaf nodes are colored using RangeColorMapping. Set the value of [ShowLegend](#) property to "True" to make legends visible.

### Labels for legends

You can customize the labels of legend items using the [LegendLabel](#) property in RangeColorMapping.

#### XML

```
<treemap:SfTreeMap.LegendSettings>
<treemap:LegendSettings ShowLegend="True" Size="700,45">
</treemap:LegendSettings>
</treemap:SfTreeMap.LegendSettings>
```

#### C#

```
...
LegendSettings legendSettings = new LegendSettings();
legendSettings.ShowLegend = true;
legendSettings.Size = new Size(700, 45);
treeMap.LegendSettings = legendSettings;
```

The following code sample helps you reproduce the output.

#### XML

```
<ContentPage.BindingContext>
<local:PopulationViewModel></local:PopulationViewModel>
</ContentPage.BindingContext>
<ContentPage.Content>
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth" LayoutType="Squarified"
ShowTooltip="True">
```

```

<treemap:SfTreeMap.LeafItemSettings>
<treemap:LeafItemSettings OverflowMode="Wrap" Gap="2" BorderColor="#A9D9F7"
LabelPath="Country" >
</treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
<treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
GroupGap = " 5" ShowHeader = "true">
<treemap:TreeMapFlatLevel.HeaderStyle>
<treemap:Style Color= "Black"/>
</treemap:TreeMapFlatLevel.HeaderStyle>
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
<treemap:SfTreeMap.LegendSettings>
<treemap:LegendSettings ShowLegend="True" Size="700,45">
<treemap:LegendSettings.LabelStyle>
<treemap:Style Color="Black"></treemap:Style>
</treemap:LegendSettings.LabelStyle>
</treemap:LegendSettings>
</treemap:SfTreeMap.LegendSettings>
<treemap:SfTreeMap.LeafItemColorMapping>
<treemap:RangeColorMapping>
<treemap:RangeColorMapping.Ranges>
<treemap:Range LegendLabel = "1 % Growth" From = "0" To = "1" Color =
"#77D8D8" />
<treemap:Range LegendLabel = "2 % Growth" From = "0" To = "2" Color =
"#AED960" />
<treemap:Range LegendLabel = "3 % Growth" From = "0" To = "3" Color =
"#FFAF51" />
<treemap:Range LegendLabel = "4 % Growth" From = "0" To = "4" Color =
"#F3D240" />
</treemap:RangeColorMapping.Ranges>
</treemap:RangeColorMapping>
</treemap:SfTreeMap.LeafItemColorMapping>
</treemap:SfTreeMap>
</ContentPage.Content>

```

## C#

```

PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
treeMap.LayoutType = LayoutTypes.Squarified;
treeMap.ShowTooltip = true;
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.OverflowMode = LabelOverflowMode.Wrap;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;

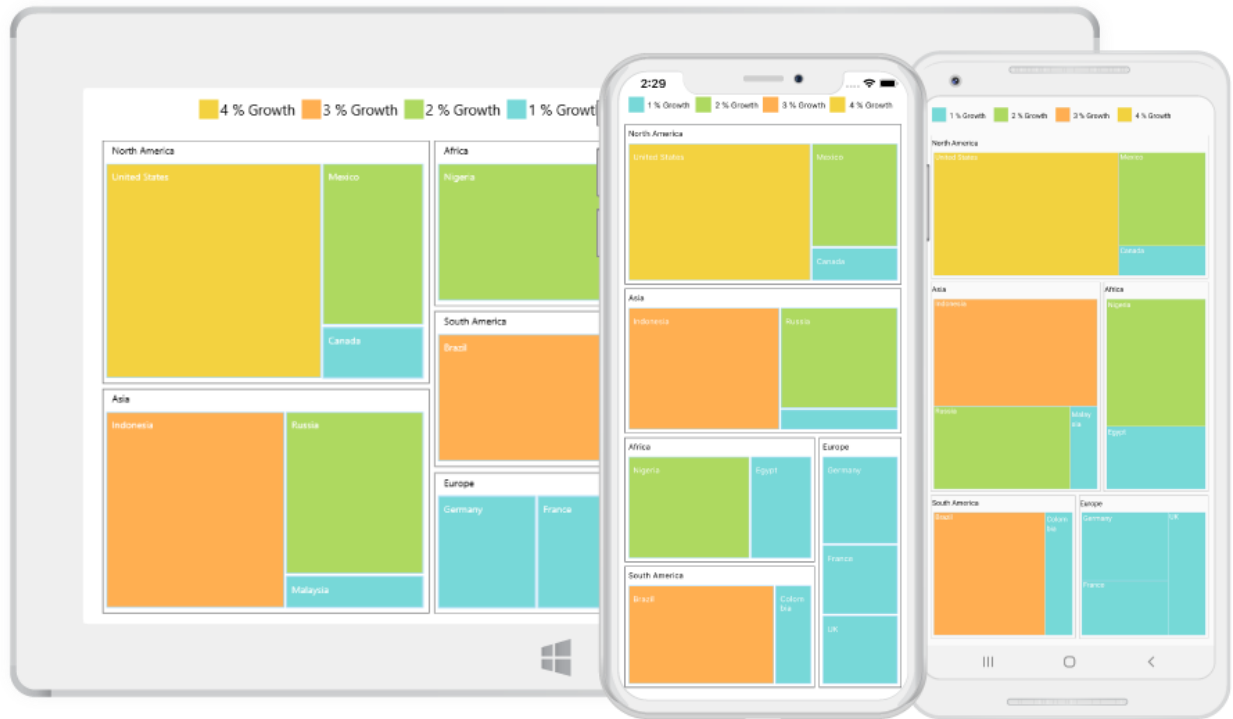
```

```
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
LegendSettings legendSettings = new LegendSettings();
legendSettings.ShowLegend = true;
legendSettings.Size = new Size(700, 45);
legendSettings.LabelStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color
= Color.Black };
treeMap.LegendSettings = legendSettings;
RangeColorMapping rangeColorMapping = new RangeColorMapping();
Range range1 = new Range();
range1.From = 0;
range1.To = 1;
range1.Color = Color.FromHex("#77D8D8");
range1.LegendLabel = "1 % Growth";
Range range2 = new Range();
range2.From = 0;
range2.To = 2;
range2.Color = Color.FromHex("#AED960");
range2.LegendLabel = "2 % Growth";
Range range3 = new Range();
range3.From = 0;
range3.To = 3;
range3.Color = Color.FromHex("#FFAF51");
range3.LegendLabel = "3 % Growth";
Range range4 = new Range();
range4.From = 0;
range4.To = 4;
range4.Color = Color.FromHex("#F3D240");
range4.LegendLabel = "4 % Growth";
rangeColorMapping.Ranges.Add(range1);
rangeColorMapping.Ranges.Add(range2);
rangeColorMapping.Ranges.Add(range3);
rangeColorMapping.Ranges.Add(range4);
treeMap.LeafItemColorMapping = rangeColorMapping;
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;
```

You can find the complete getting-started sample in the following link: [Getting-started sample](#).

The following screenshot illustrates the output of SfTreeMap.





## DataBinding

The TreeMap control supports data binding, and it can be achieved using the [DataSource](#) property.

The [DataSource](#) property accepts a collection of values as input. The following code sample demonstrates how to bind a flat collection as data source to TreeMap.

### XML

```
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}" WeightValuePath="Population" ColorValuePath="Growth">
  <treemap:SfTreeMap.LeafItemSettings>
    <treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
    >
    </treemap:LeafItemSettings>
  </treemap:SfTreeMap.LeafItemSettings>
  <treemap:SfTreeMap.Levels>
    <treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
    GroupGap =" 5" ShowHeader = "true">
    </treemap:TreeMapFlatLevel>
  </treemap:SfTreeMap.Levels>
</treemap:SfTreeMap>
```

### 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; }
}

```

## TreeMap Levels

The levels of tree map can be categorized into the following two types:

- Flat level
- Hierarchical level

### Flat Level

#### GroupPath

You can use the [GroupPath](#) property for every flat level in the TreeMap control. It is a path to a field on the source object that serves as “Group” for the level specified. You can group the data based on the [GroupPath](#) property. When [GroupPath](#) is not specified, the items will not be grouped, and the data will be displayed in the order specified in [DataSource](#).

### GroupGap

You can use the [GroupGap](#) property to separate items from every flat level and differentiate the levels mentioned in the TreeMap control.

### XML

```
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}" WeightValuePath="Population" ColorValuePath="Growth">
  <treemap:SfTreeMap.LeafItemSettings>
    <treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
    >
  </treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
  <treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
  GroupGap =" 5" ShowHeader = "true">
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
<treemap:SfTreeMap.LegendSettings>
  <treemap:LegendSettings ShowLegend="True" Size="700,45">
  <treemap:LegendSettings.LabelStyle>
  <treemap:Style Color="Black"></treemap:Style>
  </treemap:LegendSettings.LabelStyle>
  </treemap:LegendSettings>
</treemap:SfTreeMap.LegendSettings>
<treemap:SfTreeMap.LeafItemColorMapping>
  <treemap:RangeColorMapping>
  <treemap:RangeColorMapping.Ranges>
  <treemap:Range LegendLabel = "1 % Growth" From = "0" To = "1" Color =
  "#77D8D8" />
  <treemap:Range LegendLabel = "2 % Growth" From = "0" To = "2" Color =
  "#AED960" />
  <treemap:Range LegendLabel = "3 % Growth" From = "0" To = "3" Color =
  "#FFAF51" />
  <treemap:Range LegendLabel = "4 % Growth" From = "0" To = "4" Color =
  "#F3D240" />
  </treemap:RangeColorMapping.Ranges>
  </treemap:RangeColorMapping>
  </treemap:SfTreeMap.LeafItemColorMapping>
</treemap:SfTreeMap>
```

### C#

```
PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.DataSource = viewModel.PopulationDetails;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
```

```
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
treeMap.Levels.Add(flatLevel);
LegendSettings legendSettings = new LegendSettings();
legendSettings.ShowLegend = true;
legendSettings.Size = new Size(700, 45);
legendSettings.LabelStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color
= Color.Black };
treeMap.LegendSettings = legendSettings;
RangeColorMapping rangeColorMapping = new RangeColorMapping();
Range range1 = new Range();
range1.From = 0;
range1.To = 1;
range1.Color = Color.FromHex("#77D8D8");
range1.LegendLabel = "1 % Growth";
Range range2 = new Range();
range2.From = 0;
range2.To = 2;
range2.Color = Color.FromHex("#AED960");
range2.LegendLabel = "2 % Growth";
Range range3 = new Range();
range3.From = 0;
range3.To = 3;
range3.Color = Color.FromHex("#FFAF51");
range3.LegendLabel = "3 % Growth";
Range range4 = new Range();
range4.From = 0;
range4.To = 4;
range4.Color = Color.FromHex("#F3D240");
range4.LegendLabel = "4 % Growth";
rangeColorMapping.Ranges.Add(range1);
rangeColorMapping.Ranges.Add(range2);
rangeColorMapping.Ranges.Add(range3);
rangeColorMapping.Ranges.Add(range4);
treeMap.LeafItemColorMapping = rangeColorMapping;
this.Content = treeMap;
```



### Hierarchical Level

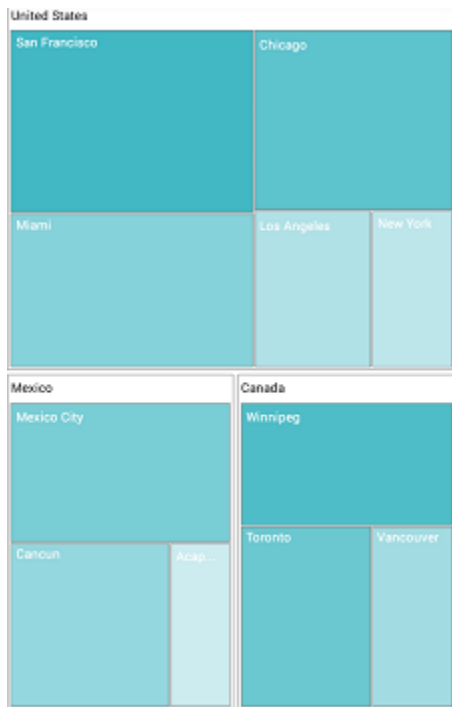
Hierarchical level is used to define levels for hierarchical data collection that contains tree-structured data.

### XML

```
<ContentPage.BindingContext>
<local:CountrySalesCollection></local:CountrySalesCollection>
</ContentPage.BindingContext>
<treemap:SfTreeMap x:Name="TreeMap" HorizontalOptions="FillAndExpand"
DataSource="{Binding CountrySales}"
VerticalOptions="FillAndExpand" WeightValuePath="Sales"
ColorValuePath="Expense" BackgroundColor="White" >
<treemap:SfTreeMap.LeafItemSettings>
<treemap:LeafItemSettings LabelPath="Name"/>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.LeafItemColorMapping>
<treemap:DesaturationColorMapping Color="#41B8C4" From="1" To=" 0.2"/>
</treemap:SfTreeMap.LeafItemColorMapping>
<treemap:SfTreeMap.Levels>
<treemap:TreeMapHierarchicalLevel ChildPadding="4" ChildBackground="White"
ShowHeader = "true" HeaderHeight = "20"
HeaderPath = "Name" ChildPath = "RegionalSales" >
<treemap:TreeMapHierarchicalLevel.HeaderStyle>
<treemap:Style Color= "Black"/>
</treemap:TreeMapHierarchicalLevel.HeaderStyle>
</treemap:TreeMapHierarchicalLevel>
</treemap:SfTreeMap.Levels>
</treemap:SfTreeMap>
```

### C#

```
SfTreeMap tree = new SfTreeMap ();
tree.WeightValuePath = "Sales";
tree.DataSource = new CountrySalesCollection ();
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].RegionalSales.Add(new RegionSale() { Country = "United States", Name = "New York", Sales = 2353, Expense = 2000 });
    this[0].RegionalSales.Add(new RegionSale() { Country = "United States", Name = "Los Angeles", Sales = 3453, Expense = 3000 });
    this[0].RegionalSales.Add(new RegionSale() { Country = "United States", Name = "San Francisco", Sales = 8456, Expense = 8000 });
    this[0].RegionalSales.Add(new RegionSale() { Country = "United States", Name = "Chicago", Sales = 6785, Expense = 7000 });
    this[0].RegionalSales.Add(new RegionSale() { Country = "United States", Name = "Miami", Sales = 7045, Expense = 6000 });
    this[1].RegionalSales.Add(new RegionSale() { Country = "Canada", Name = "Toronto", Sales = 7045, Expense = 7000 });
    this[1].RegionalSales.Add(new RegionSale() { Country = "Canada", Name = "Vancouver", Sales = 4352, Expense = 4000 });
    this[1].RegionalSales.Add(new RegionSale() { Country = "Canada", Name = "Winnipeg", Sales = 7843, Expense = 7500 });
    this[2].RegionalSales.Add(new RegionSale() { Country = "Mexico", Name = "Mexico City", Sales = 7843, Expense = 6500 });
    this[2].RegionalSales.Add(new RegionSale() { Country = "Mexico", Name = "Cancun", Sales = 6683, Expense = 6000 });
    this[2].RegionalSales.Add(new RegionSale() { Country = "Mexico", Name = "Acapulco", Sales = 2454, Expense = 2000 });
}
```



## Layout

You can decide the visual representation of nodes belonging to all the TreeMap levels using the [LayoutType](#) property of TreeMap.

The following four different types of layouts available in TreeMap:

- Squarified
- SliceAndDiceAuto
- SliceAndDiceHorizontal
- SliceAndDiceVertical

## Squarified

The **Squarified** layout creates rectangles with best aspect ratio.

## XML

```
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth"
LayoutType="Squarified">
  <treemap:SfTreeMap.LeafItemSettings>
    <treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
    >
  </treemap:LeafItemSettings>
  <treemap:SfTreeMap.Levels>
    <treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
    GroupGap = " 5" ShowHeader = "true">
    <treemap:TreeMapFlatLevel.HeaderStyle>
    <treemap:Style Color= "Black"/>
    </treemap:TreeMapFlatLevel.HeaderStyle>
```

```

</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
</treemap:SfTreeMap>

```

## C#

```

PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
treeMap.LayoutType = LayoutTypes.Squarified;
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;

```



## SliceAndDiceAuto

The **SliceAndDiceAuto** layout creates rectangles with high aspect ratio and displays them sorted both horizontally and vertically.



**XML**

```

<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth"
LayoutType="SliceAndDiceAuto">
  <treemap:SfTreeMap.LeafItemSettings>
  <treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
  >
  </treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
<treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
GroupGap =" 5" ShowHeader = "true">
  <treemap:TreeMapFlatLevel.HeaderStyle>
  <treemap:Style Color= "Black"/>
  </treemap:TreeMapFlatLevel.HeaderStyle>
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
</treemap:SfTreeMap>

```

**C#**

```

PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
treeMap.LayoutType = LayoutTypes.SliceAndDiceAuto;
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;

```



### SliceAndDiceHorizontal

The **SliceAndDiceHorizontal** layout creates rectangles with high aspect ratio and displays them sorted horizontally.

### XML

```
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth"
LayoutType="SliceAndDiceHorizontal">
  <treemap:SfTreeMap.LeafItemSettings>
    <treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
    >
  </treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
  <treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
  GroupGap = " 5" ShowHeader = "true">
    <treemap:TreeMapFlatLevel.HeaderStyle>
      <treemap:Style Color= "Black"/>
    </treemap:TreeMapFlatLevel.HeaderStyle>
  </treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
</treemap:SfTreeMap>
```

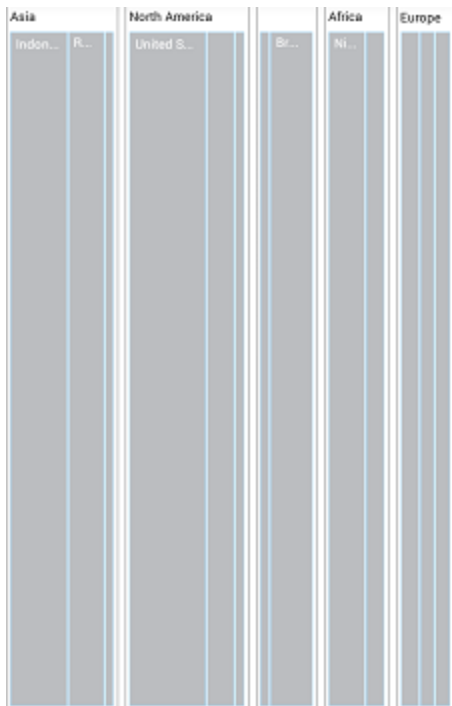
### C#

```
PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
```

```

treeMap.LayoutType = LayoutTypes.SliceAndDiceHorizontal;
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;

```



### SliceAndDiceVertical

The **SliceAndDiceVertical** layout creates rectangles with high aspect ratio and displays them sorted vertically.

### XML

```

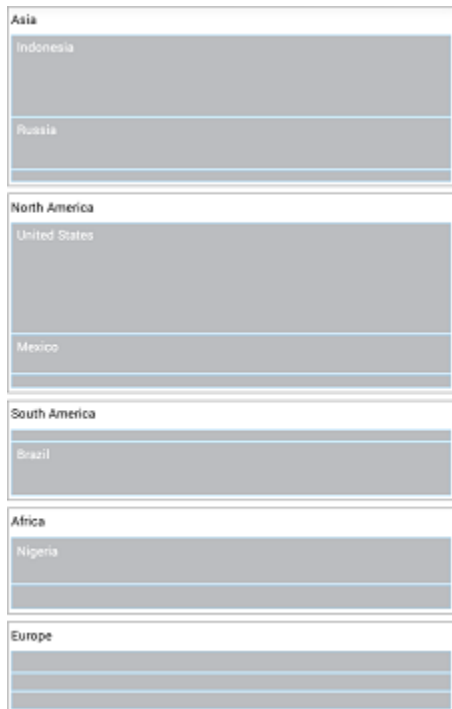
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth"
LayoutType="SliceAndDiceVertical">
<treemap:SfTreeMap.LeafItemSettings>
<treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
>
</treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>

```

```
<treemap:SfTreeMap.Levels>
<treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
GroupGap =" 5" ShowHeader = "true">
<treemap:TreeMapFlatLevel.HeaderStyle>
<treemap:Style Color= "Black"/>
</treemap:TreeMapFlatLevel.HeaderStyle>
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
</treemap:SfTreeMap>
```

## C#

```
PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
treeMap.LayoutType = LayoutTypes.SliceAndDiceVertical;
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;
```



## Customization

The TreeMap control supports color customization to determine the exact combination of colors for the tree nodes displayed and provides tooltip support to display additional information of TreeMap data.

### Color

You can customize the colors of leaf nodes using the [ColorMapping](#) support.

The [ColorMapping](#) is categorized into the following three different types:

- UniColorMapping
- RangeColorMapping
- DesaturationColorMapping

### UniColorMapping

You can color all the leaf nodes with the same color by setting value to the [Color](#) property of [UniColorMapping](#).

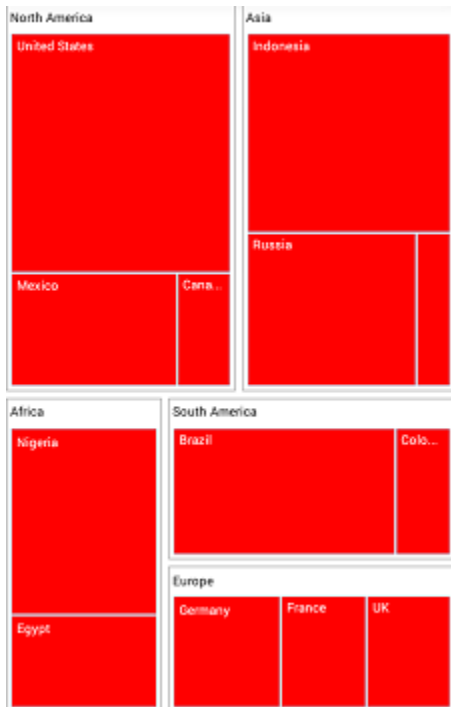
### XML

```
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth"
LayoutType="Squarified">
  <treemap:SfTreeMap.LeafItemSettings>
    <treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
    >
  </treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
  <treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
  GroupGap =" 5" ShowHeader = "true">
```

```
<treemap:TreeMapFlatLevel.HeaderStyle>
<treemap:Style Color= "Black"/>
</treemap:TreeMapFlatLevel.HeaderStyle>
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
<treemap:SfTreeMap.LeafItemColorMapping>
<treemap:UniColorMapping Color="Red"></treemap:UniColorMapping>
</treemap:SfTreeMap.LeafItemColorMapping>
</treemap:SfTreeMap>
```

## C#

```
PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
UniColorMapping uniColorMapping = new UniColorMapping();
uniColorMapping.Color = Color.Red;
treeMap.LeafItemColorMapping = uniColorMapping;
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;
```



### RangeColorMapping

You can group the leaf nodes based on the range of color values of data. You can set a unique color for every range. To achieve this, specify the [To](#) and [From](#) values as range bound, and specify the [Color](#) value to fill the leaf nodes of particular range using the [RangeColorMapping](#) property of [TreeMap](#). You must specify value to [ColorValuePath](#) since the ranges [From](#) and [To](#) depend on the under bound value of [ColorValuePath](#).

### XML

```
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth" LayoutType="Squarified"
ShowTooltip="True">
  <treemap:SfTreeMap.LeafItemSettings>
    <treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
    >
  </treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
  <treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
  GroupGap =" 5" ShowHeader = "true">
    <treemap:TreeMapFlatLevel.HeaderStyle>
      <treemap:Style Color= "Black"/>
    </treemap:TreeMapFlatLevel.HeaderStyle>
  </treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
<treemap:SfTreeMap.LegendSettings>
  <treemap:LegendSettings ShowLegend="True" Size="700,45">
    <treemap:LegendSettings.LabelStyle>
      <treemap:Style Color="Black"></treemap:Style>
    </treemap:LegendSettings.LabelStyle>
  </treemap:LegendSettings>
</treemap:SfTreeMap>
```

```

</treemap:SfTreeMap.LegendSettings>
<treemap:SfTreeMap.LeafItemColorMapping>
<treemap:RangeColorMapping>
<treemap:RangeColorMapping.Ranges>
<treemap:Range LegendLabel = "1 % Growth" From = "0" To = "1" Color =
"#77D8D8" />
<treemap:Range LegendLabel = "2 % Growth" From = "0" To = "2" Color =
"#AED960" />
<treemap:Range LegendLabel = "3 % Growth" From = "0" To = "3" Color =
"#FFAF51" />
<treemap:Range LegendLabel = "4 % Growth" From = "0" To = "4" Color =
"#F3D240" />
</treemap:RangeColorMapping.Ranges>
</treemap:RangeColorMapping>
</treemap:SfTreeMap.LeafItemColorMapping>
</treemap:SfTreeMap>

```

## C#

```

PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
LegendSettings legendSettings = new LegendSettings();
legendSettings.ShowLegend = true;
legendSettings.Size = new Size(700, 45);
legendSettings.LabelStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color
= Color.Black };
treeMap.LegendSettings = legendSettings;
RangeColorMapping rangeColorMapping = new RangeColorMapping();
Range range1 = new Range();
range1.From = 0;
range1.To = 1;
range1.Color = Color.FromHex("#77D8D8");
range1.LegendLabel = "1 % Growth";
Range range2 = new Range();
range2.From = 0;
range2.To = 2;
range2.Color = Color.FromHex("#AED960");
range2.LegendLabel = "2 % Growth";
Range range3 = new Range();
range3.From = 0;

```



```

range3.To = 3;
range3.Color = Color.FromHex("#FFAF51");
range3.LegendLabel = "3 % Growth";
Range range4 = new Range();
range4.From = 0;
range4.To = 4;
range4.Color = Color.FromHex("#F3D240");
range4.LegendLabel = "4 % Growth";
rangeColorMapping.Ranges.Add(range1);
rangeColorMapping.Ranges.Add(range2);
rangeColorMapping.Ranges.Add(range3);
rangeColorMapping.Ranges.Add(range4);
treeMap.LeafItemColorMapping = rangeColorMapping;
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;

```



### DesaturationColorMapping

You can differentiate all the leaf nodes using the [DesaturationColorMapping](#) property of TreeMap. Differentiation can be achieved even though the same color is applied for all the leaf nodes by varying the opacity of the leaf nodes based on the [Color](#) value specified in the color value range using the [RangeMinimum](#) and [RangeMaximum](#) values of data collection. You can also bind the opacity range by setting the [From](#) and [To](#) properties of [DesaturationColorMapping](#).

### XML

```

<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth" LayoutType="Squarified"
ShowTooltip="True">
<treemap:SfTreeMap.LeafItemSettings>
<treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
>

```

```

</treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
<treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
GroupGap =" 5" ShowHeader = "true">
<treemap:TreeMapFlatLevel.HeaderStyle>
<treemap:Style Color= "Black"/>
</treemap:TreeMapFlatLevel.HeaderStyle>
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
<treemap:SfTreeMap.LegendSettings>
<treemap:LegendSettings ShowLegend="True" Size="700,45">
<treemap:LegendSettings.LabelStyle>
<treemap:Style Color="Black"></treemap:Style>
</treemap:LegendSettings.LabelStyle>
</treemap:LegendSettings>
</treemap:SfTreeMap.LegendSettings>
<treemap:SfTreeMap.LeafItemColorMapping>
<treemap:DesaturationColorMapping From="1" To="0.1" Color="#41B8C4"/>
</treemap:SfTreeMap.LeafItemColorMapping>
</treemap:SfTreeMap>

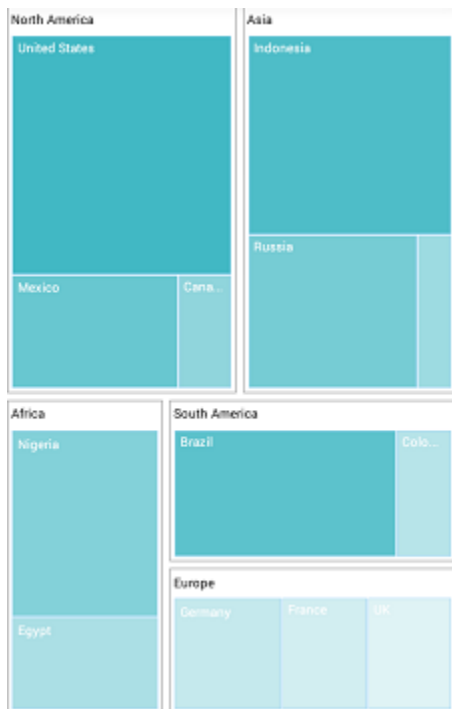
```

## C#

```

PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
LegendSettings legendSettings = new LegendSettings();
legendSettings.ShowLegend = true;
legendSettings.Size = new Size(700, 45);
legendSettings.LabelStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color
= Color.Black };
treeMap.LegendSettings = legendSettings;
DesaturationColorMapping desaturationColor = new DesaturationColorMapping();
desaturationColor.From = 1;
desaturationColor.To = 0.2;
desaturationColor.Color = Color.FromHex("#41B8C4");
treeMap.LeafItemColorMapping = desaturationColor;
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;

```



### Tooltip

You can enable the tooltip support for TreeMap by setting the [ShowTooltip](#) property to true. By default, it takes the property of bound object that is referenced in [GroupPath](#) and displays its content when the corresponding node is tapped.



### Tooltip customization

You can customize the tooltip to show more details by specifying [TooltipTemplate](#) to tooltip.

### XML

```

<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}"
WeightValuePath="Population" ColorValuePath="Growth" LayoutType="Squarified"
ShowTooltip="True">
<treemap:SfTreeMap.LeafItemSettings>
<treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
>
</treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
<treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
GroupGap =" 5" ShowHeader = "true">
<treemap:TreeMapFlatLevel.HeaderStyle>
<treemap:Style Color= "Black"/>
</treemap:TreeMapFlatLevel.HeaderStyle>
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
<treemap:SfTreeMap.LegendSettings>
<treemap:LegendSettings ShowLegend="True" Size="700,45">
<treemap:LegendSettings.LabelStyle>
<treemap:Style Color="Black"></treemap:Style>
</treemap:LegendSettings.LabelStyle>
</treemap:LegendSettings>
</treemap:SfTreeMap.LegendSettings>
<treemap:SfTreeMap.LeafItemColorMapping>
<treemap:RangeColorMapping>
<treemap:RangeColorMapping.Ranges>
<treemap:Range LegendLabel = "1 % Growth" From = "0" To = "1" Color =
"#77D8D8" />
<treemap:Range LegendLabel = "2 % Growth" From = "0" To = "2" Color =
"#AED960" />
<treemap:Range LegendLabel = "3 % Growth" From = "0" To = "3" Color =
"#FFAF51" />
<treemap:Range LegendLabel = "4 % Growth" From = "0" To = "4" Color =
"#F3D240" />
</treemap:RangeColorMapping.Ranges>
</treemap:RangeColorMapping>
</treemap:SfTreeMap.LeafItemColorMapping>
<treemap:SfTreeMap.TooltipSettings>
<treemap:TooltipSetting>
<treemap:TooltipSetting.TooltipTemplate>
<DataTemplate>
<StackLayout>
<Label Text="{Binding Country}" HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"
TextColor="#FFFFFF" Margin="6,0,0,0" FontSize="12"/>
<BoxView Color="#888C91" HeightRequest="1" VerticalOptions="FillAndExpand"
HorizontalOptions="FillAndExpand" Margin="0,6,0,6" Opacity="0.4"/>
<StackLayout HorizontalOptions="Center" Orientation="Horizontal">
<Label Text="Population:" VerticalTextAlignment="Center"
HorizontalOptions="Center"
TextColor="#FFFFFF" FontSize="12" Margin="0,0,0,6"></Label>
<Label Text="{Binding Population}"
VerticalTextAlignment="Center" HorizontalOptions="Start" TextColor="#FFFFFF"
FontSize="12"
Margin="0,0,0,6"/>
</StackLayout>

```

```

</StackLayout>
</DataTemplate>
</treemap:TooltipSetting.TooltipTemplate>
</treemap:TooltipSetting>
</treemap:SfTreeMap.TooltipSettings>
</treemap:SfTreeMap>

```

## C#

```

PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
treeMap.LayoutType = LayoutTypes.Squarified;
treeMap.ShowTooltip = true;
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
LegendSettings legendSettings = new LegendSettings();
legendSettings.ShowLegend = true;
legendSettings.Size = new Size(700, 45);
legendSettings.LabelStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color
= Color.Black };
treeMap.LegendSettings = legendSettings;
RangeColorMapping rangeColorMapping = new RangeColorMapping();
Range range1 = new Range();
range1.From = 0;
range1.To = 1;
range1.Color = Color.FromHex("#77D8D8");
range1.LegendLabel = "1 % Growth";
Range range2 = new Range();
range2.From = 0;
range2.To = 2;
range2.Color = Color.FromHex("#AED960");
range2.LegendLabel = "2 % Growth";
Range range3 = new Range();
range3.From = 0;
range3.To = 3;
range3.Color = Color.FromHex("#FFAF51");
range3.LegendLabel = "3 % Growth";
Range range4 = new Range();
range4.From = 0;
range4.To = 4;
range4.Color = Color.FromHex("#F3D240");
range4.LegendLabel = "4 % Growth";

```

```

rangeColorMapping.Ranges.Add(range1);
rangeColorMapping.Ranges.Add(range2);
rangeColorMapping.Ranges.Add(range3);
rangeColorMapping.Ranges.Add(range4);
treeMap.LeafItemColorMapping = rangeColorMapping;
DataTemplate template = new DataTemplate(() =>
{
    StackLayout stackLayout = new StackLayout();
    Label label = new Label();
    label.HorizontalTextAlignment = TextAlignment.Center;
    label.VerticalTextAlignment = TextAlignment.Center;
    label.TextColor = Color.FromHex("#FFFFFF");
    label.Margin = new Thickness(6, 0, 0, 0);
    label.FontSize = 12;
    label.SetBinding(Label.TextProperty, "Country");
    stackLayout.Children.Add(label);
    BoxView boxView = new BoxView();
    boxView.Color = Color.FromHex("#888C91");
    boxView.HeightRequest = 1;
    boxView.VerticalOptions = LayoutOptions.FillAndExpand;
    boxView.HorizontalOptions = LayoutOptions.FillAndExpand;
    boxView.Opacity = 0.4;
    boxView.Margin = new Thickness(0, 6, 0, 6);
    stackLayout.Children.Add(boxView);
    StackLayout layout = new StackLayout();
    layout.Orientation = StackOrientation.Horizontal;
    layout.HorizontalOptions = LayoutOptions.Center;
    Label label1 = new Label();
    label1.Text = "Population:";
    label1.HorizontalOptions = LayoutOptions.Center;
    label1.VerticalTextAlignment = TextAlignment.Center;
    label1.TextColor = Color.FromHex("#FFFFFF");
    label1.Margin = new Thickness(0, 0, 0, 6);
    label1.FontSize = 12;
    layout.Children.Add(label1);
    Label label2 = new Label();
    label2.HorizontalOptions = LayoutOptions.Start;
    label2.VerticalTextAlignment = TextAlignment.Center;
    label2.TextColor = Color.FromHex("#FFFFFF");
    label2.Margin = new Thickness(0, 0, 0, 6);
    label2.FontSize = 12;
    label2.SetBinding(Label.TextProperty, "Population");
    layout.Children.Add(label2);
    stackLayout.Children.Add(layout);
    return stackLayout;
});
TooltipSetting tooltipSetting = new TooltipSetting();
tooltipSetting.TooltipTemplate = template;
treeMap.TooltipSettings = tooltipSetting;
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;

```



### Selection

The TreeMap control provides selection support, which allows you to select the tree map items. The selection can be enabled by setting the [HighlightOnSelection](#) property to true in TreeMap. You can specify the highlight color and border width using the [HighlightColor](#) and [HighlightBorderWidth](#) properties, respectively.

### XML

```
<treemap:SfTreeMap x:Name="treeMap" DataSource="{Binding
PopulationDetails}" HighlightOnSelection="True"
HighlightBorderWidth="8" HighlightColor="Red"
WeightValuePath="Population" ColorValuePath="Growth" LayoutType="Squarified"
>
<treemap:SfTreeMap.LeafItemSettings>
<treemap:LeafItemSettings Gap="2" BorderColor="#A9D9F7" LabelPath="Country"
>
</treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.Levels>
<treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
GroupGap = " 5" ShowHeader = "true">
<treemap:TreeMapFlatLevel.HeaderStyle>
<treemap:Style Color= "Black"/>
</treemap:TreeMapFlatLevel.HeaderStyle>
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
<treemap:SfTreeMap.LegendSettings>
<treemap:LegendSettings ShowLegend="True" Size="700,45">
<treemap:LegendSettings.LabelStyle>
<treemap:Style Color="Black"></treemap:Style>
</treemap:LegendSettings.LabelStyle>
</treemap:LegendSettings>
```

```

</treemap:SfTreeMap.LegendSettings>
<treemap:SfTreeMap.LeafItemColorMapping>
<treemap:RangeColorMapping>
<treemap:RangeColorMapping.Ranges>
<treemap:Range LegendLabel = "1 % Growth" From = "0" To = "1" Color =
"#77D8D8" />
<treemap:Range LegendLabel = "2 % Growth" From = "0" To = "2" Color =
"#AED960" />
<treemap:Range LegendLabel = "3 % Growth" From = "0" To = "3" Color =
"#FFAF51" />
<treemap:Range LegendLabel = "4 % Growth" From = "0" To = "4" Color =
"#F3D240" />
</treemap:RangeColorMapping.Ranges>
</treemap:RangeColorMapping>
</treemap:SfTreeMap.LeafItemColorMapping>
</treemap:SfTreeMap>

```

## C#

```

PopulationViewModel viewModel = new PopulationViewModel();
SfTreeMap treeMap = new SfTreeMap();
treeMap.BackgroundColor = Color.White;
treeMap.ColorValuePath = "Growth";
treeMap.WeightValuePath = "Population";
treeMap.LayoutType = LayoutTypes.Squarified;
treeMap.HighlightOnSelection = true;
treeMap.HightlightBorderWidth = 8;
treeMap.HightlightColor = Color.Red;
LeafItemSettings leafSetting = new LeafItemSettings();
leafSetting.Gap = 2;
leafSetting.OverflowMode = LabelOverflowMode.Wrap;
leafSetting.BorderColor = Color.FromHex("#A9D9F7");
leafSetting.LabelPath = "Country";
treeMap.LeafItemSettings = leafSetting;
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
LegendSettings legendSettings = new LegendSettings();
legendSettings.ShowLegend = true;
legendSettings.Size = new Size(700, 45);
legendSettings.LabelStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.LegendSettings = legendSettings;
RangeColorMapping rangeColorMapping = new RangeColorMapping();
Range range1 = new Range();
range1.From = 0;
range1.To = 1;
range1.Color = Color.FromHex("#77D8D8");
range1.LegendLabel = "1 % Growth";
Range range2 = new Range();
range2.From = 0;

```



```

range2.To = 2;
range2.Color = Color.FromHex("#AED960");
range2.LegendLabel = "2 % Growth";
Range range3 = new Range();
range3.From = 0;
range3.To = 3;
range3.Color = Color.FromHex("#FFAF51");
range3.LegendLabel = "3 % Growth";
Range range4 = new Range();
range4.From = 0;
range4.To = 4;
range4.Color = Color.FromHex("#F3D240");
range4.LegendLabel = "4 % Growth";
rangeColorMapping.Ranges.Add(range1);
rangeColorMapping.Ranges.Add(range2);
rangeColorMapping.Ranges.Add(range3);
rangeColorMapping.Ranges.Add(range4);
treeMap.LeafItemColorMapping = rangeColorMapping;
treeMap.DataSource = viewModel.PopulationDetails;
this.Content = treeMap;

```



### ItemTemplate

The TreeMap control provides template support to tree map items using the [ItemTemplate](#) property, which allows any type of custom template to be created with any type of view element.

### XML

```

<treemap:SfTreeMap x:Name="TreeMap" HorizontalOptions="FillAndExpand"
DataSource="{Binding OlympicMedalsDetails}"
VerticalOptions="FillAndExpand" WeightValuePath="TotalMedals"
ColorValuePath="TotalMedals" BackgroundColor="White" >
<treemap:SfTreeMap.LeafItemSettings>

```

```

<treemap:LeafItemSettings ShowLabels="false" BorderColor="#A9D9F7" Gap="2"
LabelPath="Country" >
</treemap:LeafItemSettings>
</treemap:SfTreeMap.LeafItemSettings>
<treemap:SfTreeMap.ItemTemplate>
<DataTemplate>
<Grid BackgroundColor ="#D73028" >
<Image HorizontalOptions= "Center" VerticalOptions= "Center" HeightRequest=
"50"
WidthRequest= "50" Source="{Binding Data.GameImageSource}" />
<Label Margin="5,5,0,0" FontSize="12" Text= "{Binding Data.GameName}"
TextColor = "White" HeightRequest="50" WidthRequest="100" HorizontalOptions=
"Start"
VerticalOptions= "Start"/>
</Grid>
</DataTemplate>
</treemap:SfTreeMap.ItemTemplate>
</treemap:SfTreeMap>

```

**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, ImageName = "Swimming.png" });
    this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Track and Field", GoldMedals = 9, SilverMedals = 13, BronzeMedals = 7,
TotalMedals = 29, ImageName = "TrackAndField.png" });
    this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Gymnastics", GoldMedals = 3, SilverMedals = 1, BronzeMedals = 2,
TotalMedals = 6, ImageName = "Gymnastics.png" });
    this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Boxing", GoldMedals = 1, SilverMedals = 0, BronzeMedals = 1, TotalMedals =
2, ImageName = "Boxing.png" });
    this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Cycling", GoldMedals = 1, SilverMedals = 2, BronzeMedals = 1, TotalMedals =
4, ImageName = "Cycling.png" });
    this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Shooting", GoldMedals = 3, SilverMedals = 0, BronzeMedals = 1, TotalMedals
= 4, ImageName = "Shooting.png" });
    this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Wrestling", GoldMedals = 2, SilverMedals = 0, BronzeMedals = 2, TotalMedals
= 4, ImageName = "Wrestling.png" });
    this.OlympicMedalsDetails.Add(new OlympicMedals { Country = "US", GameName =
"Diving", GoldMedals = 1, SilverMedals = 1, BronzeMedals = 2, TotalMedals =
4, ImageName = "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 GameImageSource { get; set; }
private string imageName;
public string ImageName
{
    get { return imageName; }
    set
    {
        imageName = value;
        if (Device.RuntimePlatform == Device.UWP)
        {
            GameImageSource = ImageSource.FromResource("TreeMap." + imageName,
                typeof(OlympicMedals).GetTypeInfo().Assembly);
        }
        else
        {
            GameImageSource = ImageSource.FromResource("TreeMap." + imageName);
        }
    }
}

```



## TreeMap Elements

The TreeMap contains the following elements:

- Legends

- Headers
- Labels

### Legend

You can set the color value of leaf nodes using the [LegendSettings](#) property. This legend is appropriate only for the tree map whose leaf nodes are colored using [RangeColorMapping](#).

The visibility of legend can be enabled by setting the [ShowLegend](#) property to true.

#### *TreeMap legends*

You can set the size of legend icons by setting the [IconSize](#) property of `LegendSettings` in `TreeMap`.

#### *Labels for legends*

You can customize the labels of the legend items using the [LegendLabel](#) property of [RangeColorMapping](#).

### XML

```
<treemap:SfTreeMap.LegendSettings>  
<treemap:LegendSettings ShowLegend="True" Size="700,45">  
<treemap:LegendSettings.LabelStyle>  
<treemap:Style Color="Black"></treemap:Style>  
</treemap:LegendSettings.LabelStyle>  
</treemap:LegendSettings>  
</treemap:SfTreeMap.LegendSettings>
```

### C#

```
LegendSettings legendSettings = new LegendSettings();  
legendSettings.ShowLegend = true;  
legendSettings.Size = new Size(700, 45);  
legendSettings.LabelStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color  
= Color.Black };  
treeMap.LegendSettings = legendSettings;
```



## Header

You can set headers for each level by setting the [ShowHeader](#) property of each **TreeMap** level. The [HeaderHeight](#) property helps you set the height of header, and the [GroupPath](#) value determines the header value.

## XML

```
<treemap:SfTreeMap.Levels>
<treemap:TreeMapFlatLevel HeaderHeight="20" GroupPath = "Continent"
GroupGap =" 5" ShowHeader = "true">
<treemap:TreeMapFlatLevel.HeaderStyle>
<treemap:Style Color= "Black"/>
</treemap:TreeMapFlatLevel.HeaderStyle>
</treemap:TreeMapFlatLevel>
</treemap:SfTreeMap.Levels>
```

## C#

```
TreeMapFlatLevel flatLevel = new TreeMapFlatLevel();
flatLevel.HeaderHeight = 20;
flatLevel.GroupPath = "Continent";
flatLevel.GroupGap = 5;
flatLevel.ShowHeader = true;
flatLevel.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
treeMap.Levels.Add(flatLevel);
```



### Data labels

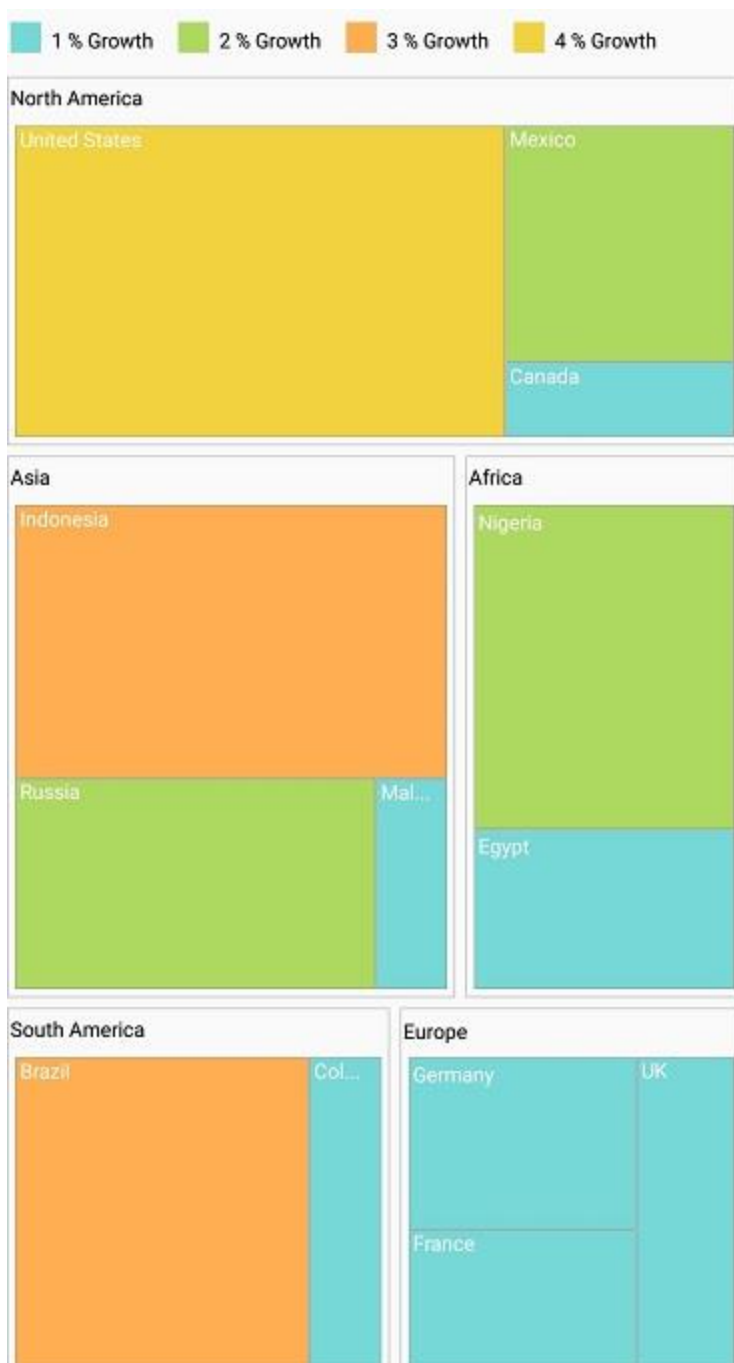
The [ShowLabels](#) property is used to enable or disable the labels in leaf nodes. The [LabelPath](#) property allows you to set values to labels.

### XML

```
<treeMap:SfTreeMap.LeafItemSettings>
  <treeMap:LeafItemSettings LabelPath="Country" ShowLabels="True">
  </treeMap:LeafItemSettings>
</treeMap:SfTreeMap.LeafItemSettings>
```

### C#

```
treeMap.LeafItemSettings.ShowLabels = true;
treeMap.LeafItemSettings.LabelPath = "Country";
```



*Avoid overlap in data labels*

The [OverflowMode](#) property aligns data labels within leaf node boundaries using the `Trim`, `Wrap`, and `Hide` options. The default value of the `OverflowMode` property is `Trim`.

`Trim`

You can trim the data labels inside the leaf node boundaries using the `Trim` option.

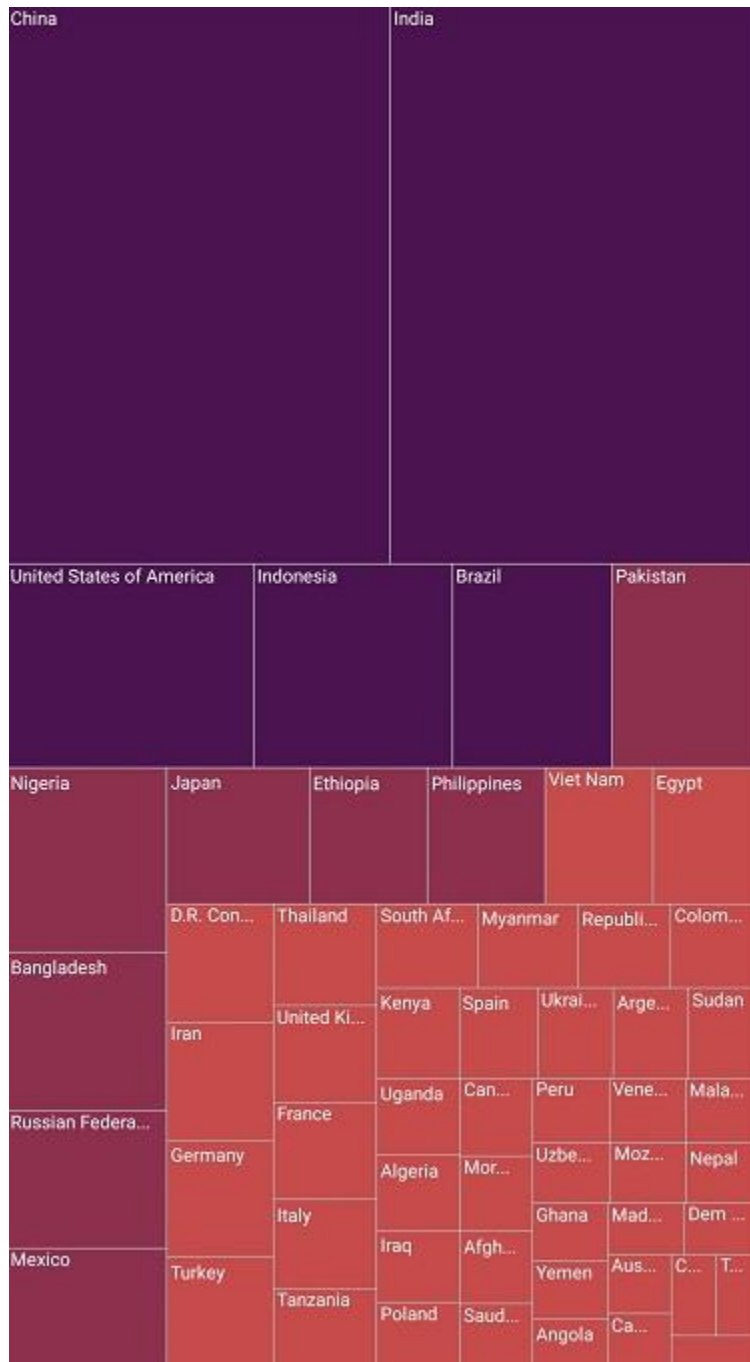
**XML**

```
<treeMap:SfTreeMap.LeafItemSettings>
<treeMap:LeafItemSettings LabelPath="Country" OverflowMode="Trim">
```

```
</treeMap:LeafItemSettings>
</treeMap:SfTreeMap.LeafItemSettings>
```

**C#**

```
treeMap.LeafItemSettings.OverflowMode = LabelOverflowMode.Trim;
```

**Wrap**

You can wrap the data labels inside the leaf node boundaries using the `Wrap` option.

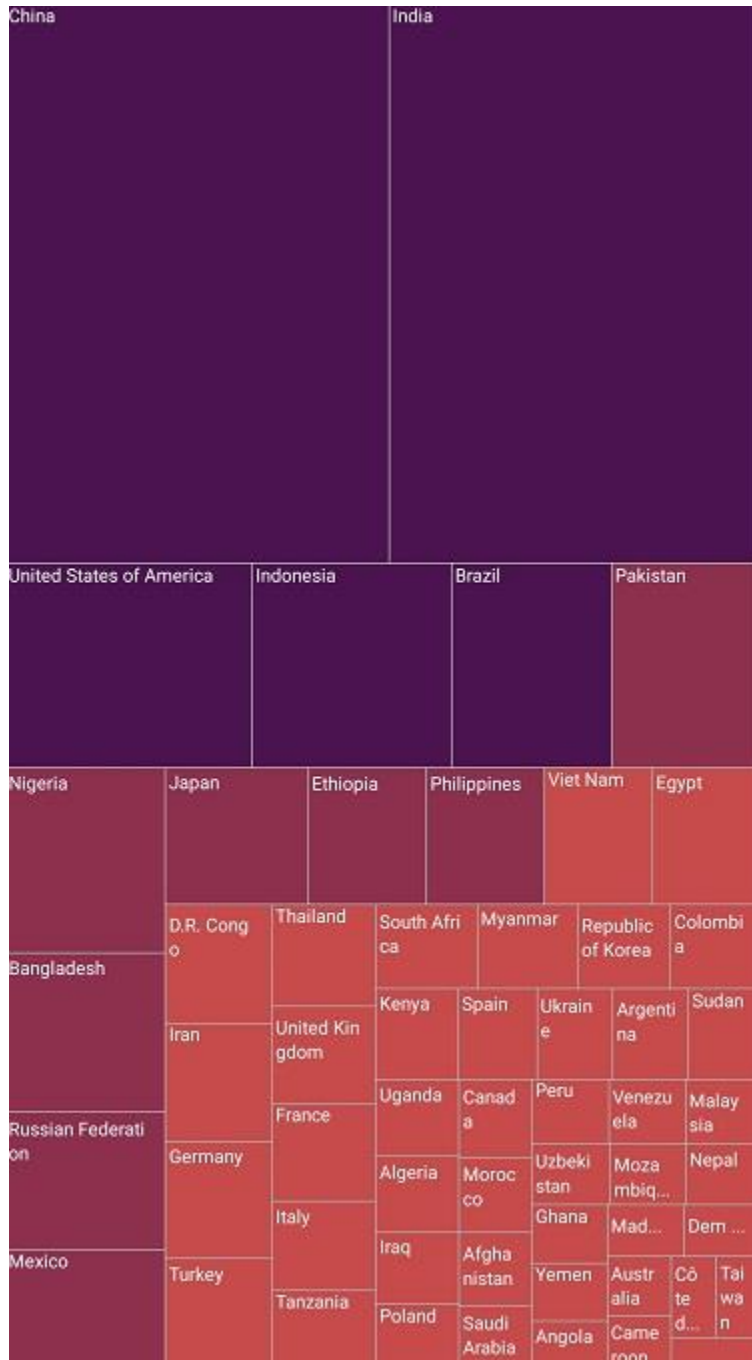


**XML**

```
<treeMap:SfTreeMap.LeafItemSettings>
<treeMap:LeafItemSettings LabelPath="Country" OverflowMode="Wrap">
</treeMap:LeafItemSettings>
</treeMap:SfTreeMap.LeafItemSettings>
```

**C#**

```
treeMap.LeafItemSettings.OverflowMode = LabelOverflowMode.Wrap;
```



### Hide

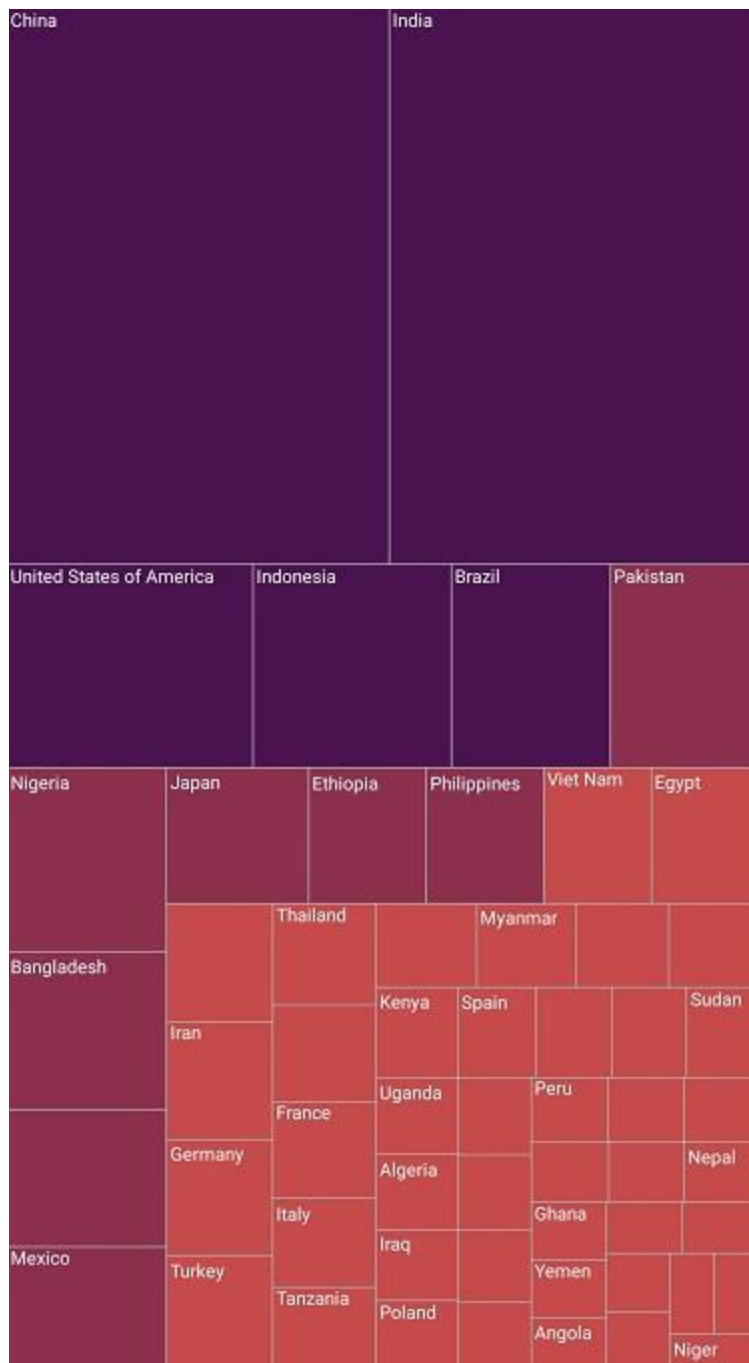
You can hide the data labels inside the leaf node boundaries using the **Hide** option.

#### **XML**

```
<treeMap:SfTreeMap.LeafItemSettings>  
<treeMap:LeafItemSettings LabelPath="Country" OverflowMode="Hide">  
</treeMap:LeafItemSettings>  
</treeMap:SfTreeMap.LeafItemSettings>
```

#### **C#**

```
treeMap.LeafItemSettings.OverflowMode = LabelOverflowMode.Hide;
```



### Customize data labels

You can customize the data labels using the [LabelStyle](#) property of `LeafItemSettings`. The font color, font size, font attribute, and font family can be customized using the `Color`, `FontSize`, `FontAttributes`, and `FontFamily` properties.

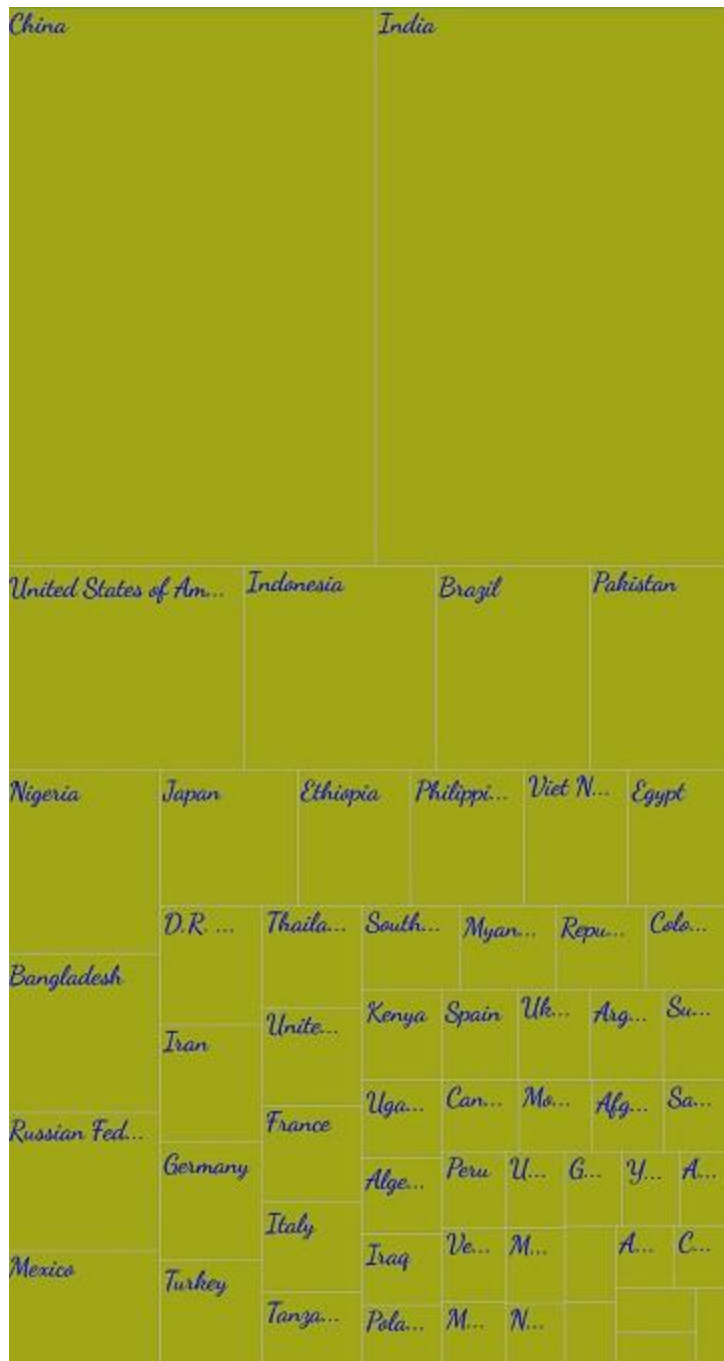
### XML

```
<treeMap:SfTreeMap.LeafItemSettings>
<treeMap:LeafItemSettings LabelPath="Country" OverflowMode="Trim">
<treeMap:LeafItemSettings.LabelStyle>
<treeMap:Style Color="Blue" FontSize="15" FontAttributes="Bold">
```

```
<treeMap:Style.FontFamily>  
<OnPlatform x:TypeArguments="x:String" iOS="Chalkduster" Android="cursive"  
WinPhone="Chiller" />  
</treeMap:Style.FontFamily>  
</treeMap:Style>  
</treeMap:LeafItemSettings.LabelStyle>  
</treeMap:LeafItemSettings>  
</treeMap:SfTreeMap.LeafItemSettings>
```

## C#

```
treeMap.LeafItemSettings.LabelStyle.FontSize = 15;  
treeMap.LeafItemSettings.LabelStyle.FontAttributes = FontAttributes.Bold;  
treeMap.LeafItemSettings.LabelStyle.FontFamily = Device.RuntimePlatform ==  
Device.iOS ? "Chalkduster" : Device.RuntimePlatform == Device.Android ?  
"cursive" : "Chiller";  
treeMap.LeafItemSettings.LabelStyle.Color = Color.Blue;
```



## TreeMap Events

### ItemSelected

The [ItemSelected](#) event occurs when an item is selected. The selected leaf node underlying data item and IsSelected boolean property will be passed as arguments to ItemSelectedEventArgs. The IsSelected property indicates whether the item has been selected.

Set the [HighlightOnSelection](#) property to true to use the `ItemSelected` event.

### XML

```
<Grid>
```

```
<treemap:SfTreeMap x:Name="treemap" DataSource="{Binding
PopulationDetails}" WeightValuePath="Population"
ColorValuePath="Growth" ItemSelected="Treemap_ItemSelected"
HighlightOnSelection="True">
</treemap:SfTreeMap>
</Grid>
```

**C#**

```
public MainPage()
{
    InitializeComponent();
    treemap.ItemSelected += Treemap_ItemSelected;
    treemap.HighlightOnSelection = true;
}
private void Treemap_ItemSelected(object sender, TreeMapSelectedEventArgs e)
{
    var selectedItem = (e.Item as PopulationViewModel);
    bool setSelection = e.IsSelected;
}
```

**Drilldown**

The drilldown feature provides better visualization of hierarchy. A large set of data can be virtualized into minimal views. Each level of items can be drilled down. TreeMap provides animation along with the drilldown support. The drilldown header will be enabled when you perform drilldown in TreeMap, and this header helps in performing zoom out operation (i.e., drill up one level).

The drilldown feature can be enabled or disabled using the **EnableDrilldown** property in TreeMap. The drilldown operation can be performed for both flat level and hierarchy level.

---

**Information:** Tap either the item or the item header to perform the drilldown operation after enabling the EnableDrilldown property.

---

Icons will be displayed at the left of the header to indicate whether the item can be further drilled or not. The plus icon indicates the drilldown operation, and the minus icon indicates the drill up operation.

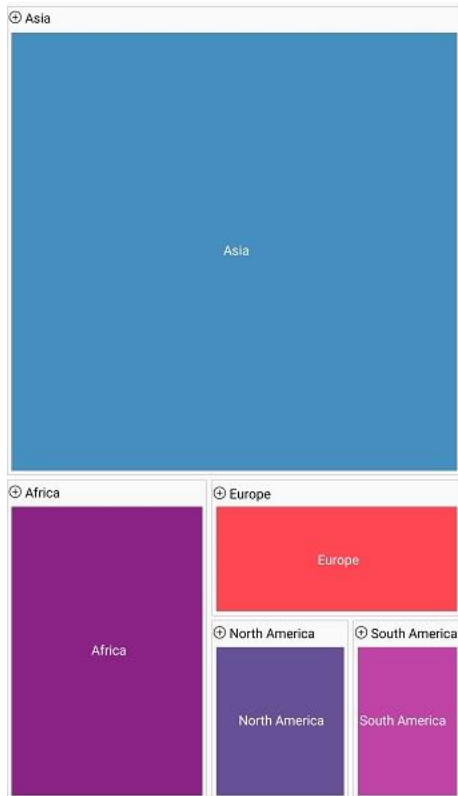
The following code snippet demonstrates how to enable the drilldown operation.

**XML**

```
<treeMap:SfTreeMap EnableDrilldown="True" >
```

**C#**

```
SfTreeMap map = new SfTreeMap();
map.EnableDrilldown = true;
```



### Header customization

The drilldown header used for zoom out (i.e., drill up one level) operation can be customized by setting style using the `DrilldownHeaderStyle` property in `TreeMap`. This property provides similar customization option as of the `TreeMap` level header.

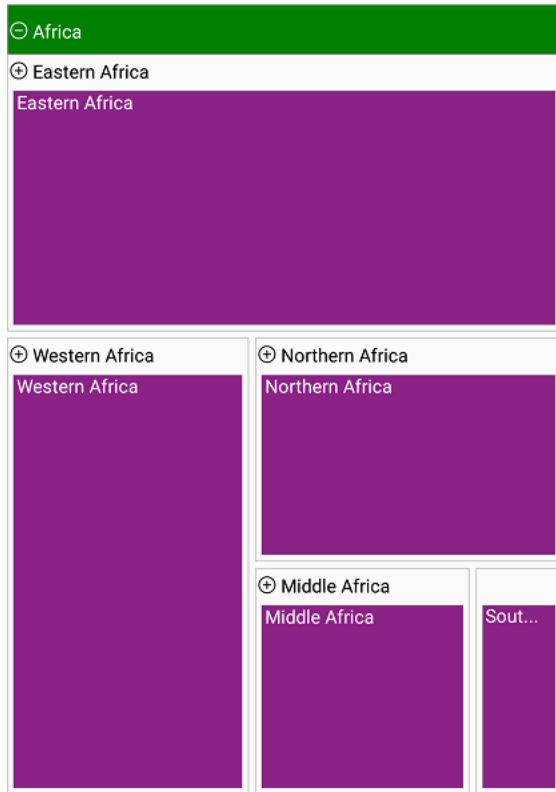
The following code snippet demonstrates how to define the drilldown header style.

### XML

```
<treeMap:SfTreeMap.DrilldownHeaderStyle>
  <treeMap:Style Color="White" BackgroundColor="Green"></treeMap:Style>
</treeMap:SfTreeMap.DrilldownHeaderStyle>
```

### C#

```
SfTreeMap map = new SfTreeMap();
map.DrilldownHeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.White, BackgroundColor = Color.Green };
```



The following code snippet is the complete code for drilldown.

#### XML

```
<treeMap:SfTreeMap Grid.Row="1" DataSource="{Binding PopulationDetails}"
EnableDrilldown="True" x:Name="treeMap"
WeightValuePath="Population" ColorValuePath="Population" ShowTooltip="True"
>
<treeMap:SfTreeMap.DrilldownHeaderStyle>
<treeMap:Style Color="Black" BackgroundColor="Green"></treeMap:Style>
</treeMap:SfTreeMap.DrilldownHeaderStyle>
<treeMap:SfTreeMap.LeafItemSettings>
<treeMap:LeafItemSettings LabelPath="Region"></treeMap:LeafItemSettings>
</treeMap:SfTreeMap.LeafItemSettings>
<treeMap:SfTreeMap.Levels>
<treeMap:TreeMapFlatLevel GroupPath="Continent">
<treeMap:TreeMapFlatLevel.HeaderStyle>
<treeMap:Style Color="Black"/>
</treeMap:TreeMapFlatLevel.HeaderStyle>
</treeMap:TreeMapFlatLevel>
<treeMap:TreeMapFlatLevel GroupPath="States">
<treeMap:TreeMapFlatLevel.HeaderStyle>
<treeMap:Style Color="Black"/>
</treeMap:TreeMapFlatLevel.HeaderStyle>
</treeMap:TreeMapFlatLevel>
</treeMap:SfTreeMap.Levels>
<treeMap:SfTreeMap.LeafItemColorMapping>
<treeMap:PaletteColorMapping>
<treeMap:PaletteColorMapping.Colors>
<Color>#C044A5</Color>
```



```
<Color>#665197</Color>
<Color>#FF4652</Color>
<Color>#8B2286</Color>
<Color>#448FC0</Color>
</treeMap:PaletteColorMapping.Colors>
</treeMap:PaletteColorMapping>
</treeMap:SfTreeMap.LeafItemColorMapping>
</treeMap:SfTreeMap>
```

## C#

```
DrilldownViewModel viewModel = new DrilldownViewModel();
SfTreeMap map = new SfTreeMap();
map.WeightValuePath = "Population";
map.ColorValuePath = "Population";
map.LeafItemSettings.LabelPath = "Region";
TreeMapFlatLevel level = new TreeMapFlatLevel();
level.GroupPath = "Continent";
level.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
map.Levels.Add(level);
TreeMapFlatLevel level1 = new TreeMapFlatLevel();
level1.GroupPath = "States";
level1.HeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
map.Levels.Add(level1);
PaletteColorMapping colorMapping = new PaletteColorMapping();
colorMapping.Colors.Add(Color.FromHex("#C044A5"));
colorMapping.Colors.Add(Color.FromHex("#665197"));
colorMapping.Colors.Add(Color.FromHex("#FF4652"));
colorMapping.Colors.Add(Color.FromHex("#8B2286"));
colorMapping.Colors.Add(Color.FromHex("#448FC0"));
map.LeafItemColorMapping = colorMapping;
map.DrilldownHeaderStyle = new Syncfusion.SfTreeMap.XForms.Style() { Color =
Color.Black };
map.DataSource = viewModel.PopulationDetails;
map.EnableDrilldown = true;
map.ShowTooltip = true;
this.Content = map;
```



The following is the model data used for the above drilldown example.

### C#

```
public class DrilldownModel
{
    public string Continent { get; set; }
    public string States { get; set; }
    public string Region { get; set; }
    public double Population { get; set; }
}
```

### C#

```
public class DrilldownViewModel
{
    public ObservableCollection<DrilldownModel> PopulationDetails
    {
        get;
        set;
    }
    public DrilldownViewModel()
    {
        this.PopulationDetails = new ObservableCollection<DrilldownModel>();
        #region Africa
```

```

PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Ethiopia", Population = 107534882 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Tanzania", Population = 59091392 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Kenya", Population = 50950879 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Uganda", Population = 44270563 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Mozambique", Population = 30528673 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Madagascar", Population = 26262810 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Malawi", Population = 19164728 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Zambia", Population = 17609178 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Zimbabwe", Population = 16913261 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Somalia", Population = 15181925 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "South Sudan", Population = 12919053 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Rwanda", Population = 12501156 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Burundi", Population = 107534882 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Eritrea", Population = 5187948 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Mauritius", Population = 1268315 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Djibouti", Population = 971408 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Réunion", Population = 883247 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Comoros", Population = 832347 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Mayotte", Population = 259682 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Eastern Africa", Region = "Seychelles", Population = 95235 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Middle Africa", Region = "Democratic Republic of the Congo", Population =
84004989 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Middle Africa", Region = "Angola", Population = 30774205 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Middle Africa", Region = "Cameroon", Population = 24678234 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Middle Africa", Region = "Chad", Population = 15353184 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Middle Africa", Region = "Congo", Population = 5399895 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Middle Africa", Region = "Central African Republic", Population = 4737423
});
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Middle Africa", Region = "Gabon", Population = 2067561 });

```

```

PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Middle Africa", Region = "Equatorial Guinea", Population = 1313894 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Middle Africa", Region = "Sao Tome and Principe", Population = 208818 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Northern Africa", Region = "Egypt", Population = 99375741 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Northern Africa", Region = "Algeria", Population = 42008054 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Northern Africa", Region = "Sudan", Population = 41511526 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Northern Africa", Region = "Morocco", Population = 36191805 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Northern Africa", Region = "Tunisia", Population = 11659174 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Northern Africa", Region = "Libya", Population = 6470956 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Northern Africa", Region = "Western Sahara", Population = 567421 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Southern Africa", Region = "South Africa", Population = 57398421 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Southern Africa", Region = "Namibia", Population = 2587801 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Southern Africa", Region = "Botswana", Population = 2333201 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Southern Africa", Region = "Lesotho", Population = 2263010 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Southern Africa", Region = "Swaziland", Population = 1391385 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Nigeria", Population = 195875237 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Ghana", Population = 29463643 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Côte d'Ivoire", Population = 24905843 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Niger", Population = 22311375 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Burkina Faso", Population = 19751651 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Mali", Population = 19107706 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Senegal", Population = 16294270 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Guinea", Population = 13052608 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Benin", Population = 11485674 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Togo", Population = 7990926 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Sierra Leone", Population = 7719729 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Liberia", Population = 4853516 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Mauritania", Population = 4540068 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Gambia", Population = 2163765 });

```

```

PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Guinea-Bissau", Population = 1907268 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Cabo Verde", Population = 553335 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Africa", States =
"Western Africa", Region = "Saint Helena", Population = 4074 });
#endregion
#region Asia
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Central Asia", Region = "Uzbekistan", Population = 32364996 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Central Asia", Region = "Kazakhstan", Population = 18403860 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Central Asia", Region = "Tajikistan", Population = 9107211 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Central Asia", Region = "Kyrgyzstan", Population = 6132932 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Central Asia", Region = "Turkmenistan", Population = 5851466 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Eastern Asia", Region = "China", Population = 1415045928 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Eastern Asia", Region = "Japan", Population = 127185332 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Eastern Asia", Region = "South Korea", Population = 51164435 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Eastern Asia", Region = "North Korea", Population = 25610672 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Eastern Asia", Region = "Taiwan", Population = 23694089 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Eastern Asia", Region = "Hong Kong", Population = 7428887 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Eastern Asia", Region = "Mongolia", Population = 3121772 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Eastern Asia", Region = "Macao", Population = 632418 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Indonesia", Population = 266794980 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Philippines", Population = 106512074 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Viet Nam", Population = 96491146 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Thailand", Population = 69183173 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Myanmar", Population = 53855735 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Malaysia", Population = 32042458 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Cambodia", Population = 16245729 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Laos", Population = 6961210 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Singapore", Population = 5791901 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Timor-Leste", Population = 1324094 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southeastern Asia", Region = "Brunei Darussalam", Population = 434076 });

```

```

PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southern Asia", Region = "India", Population = 1354051854 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southern Asia", Region = "Pakistan", Population = 200813818 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southern Asia", Region = "Bangladesh", Population = 166368149 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southern Asia", Region = "Iran", Population = 82011735 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southern Asia", Region = "Afghanistan", Population = 36373176 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southern Asia", Region = "Nepal", Population = 29624035 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southern Asia", Region = "Sri Lanka", Population = 20950041 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southern Asia", Region = "Bhutan", Population = 817054 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Southern Asia", Region = "Maldives", Population = 444259 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Turkey", Population = 81916871 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Iraq", Population = 39339753 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Saudi Arabia", Population = 33554343 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Yemen", Population = 28915284 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Syria", Population = 18284407 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Azerbaijan", Population = 9923914 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Jordan", Population = 9903802 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "United Arab Emirates", Population = 9541615 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Israel", Population = 8452841 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Lebanon", Population = 6093509 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "State of Palestine", Population = 5052776 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Oman", Population = 4829946 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Kuwait", Population = 4197128 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Georgia", Population = 3907131 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Armenia", Population = 2934152 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Qatar", Population = 2694849 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Bahrain", Population = 1566993 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Asia", States =
"Western Asia", Region = "Cyprus", Population = 1189085 });
#endregion
#region NorthAmerica

```

```

PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Central America", Region = "Mexico", Population = 130759074 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Central America", Region = "Guatemala", Population = 17245346 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Central America", Region = "Honduras", Population = 9417167 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Central America", Region = "El, Salvador", Population = 6411558
});
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Central America", Region = "Nicaragua", Population = 6284757 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Central America", Region = "Costa, Rica", Population = 4953199 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Central America", Region = "Panama", Population = 4162618 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Central America", Region = "Belize", Population = 382444 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Northern America", Region = "U.S", Population = 322179605 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Northern America", Region = "Canada", Population = 36953765 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Northern America", Region = "Bermuda", Population = 61070 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Northern America", Region = "Greenland", Population = 56565 });
PopulationDetails.Add(new DrilldownModel() { Continent = "North America",
States = "Northern America", Region = "Saint Pierre & Miquelon", Population
= 6342 });
#endregion
#region SouthAmerica
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Brazil", Population = 204519000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Colombia", Population = 48549000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Argentina", Population = 43132000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Peru", Population = 31153000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Venezuela", Population = 30620000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Chile", Population = 18006000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Ecuador", Population = 16279000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Bolivia", Population = 10520000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Paraguay", Population = 7003000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Uruguay", Population = 3310000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Guyana", Population = 747000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Suri Name", Population = 560000 });
PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "French Guiana", Population = 262000 });

```



```

PopulationDetails.Add(new DrilldownModel() { Continent = "South America",
States = "South America", Region = "Falkland Islands", Population = 3000 });
#endregion
#region Europe
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Russia", Population = 143964709 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Ukraine", Population = 44009214 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Poland", Population = 38104832 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Romania", Population = 19580634 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Czech Republic", Population = 10625250 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Hungary", Population = 9688847 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Belarus", Population = 9452113 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Bulgaria", Population = 7036848 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Slovakia", Population = 5449816 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Eastern Europe", Region = "Moldova", Population = 4041065 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "United Kingdom", Population = 66573504 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Sweden", Population = 9982709 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Denmark", Population = 5754356 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Finland", Population = 5542517 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Norway", Population = 5353363 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Ireland", Population = 4803748 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Lithuania", Population = 2876475 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Latvia", Population = 1929938 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Estonia", Population = 1306788 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Iceland", Population = 337780 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Channel Islands", Population = 166083 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Isle of Man", Population = 84831 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Northern Europe", Region = "Faeroe Islands", Population = 49489 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Italy", Population = 59290969 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Spain", Population = 46397452 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Greece", Population = 11142161 });

```



```

PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Portugal", Population = 10291196 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Serbia", Population = 8762027 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Croatia", Population = 4164783 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Bosnia and Herzegovina", Population = 3503554
});
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Albania", Population = 2934363 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Macedonia", Population = 2085051 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Slovenia", Population = 2081260 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Montenegro", Population = 629219 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Malta", Population = 432089 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Andorra", Population = 76953 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Gibraltar", Population = 34733 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "San Marino", Population = 33557 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Southern Europe", Region = "Holy, See", Population = 801 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Western Europe", Region = "Germany", Population = 82293457 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Western Europe", Region = "France", Population = 65233271 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Western Europe", Region = "Netherlands", Population = 17084459 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Western Europe", Region = "Belgium", Population = 11498519 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Western Europe", Region = "Austria", Population = 8751820 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Western Europe", Region = "Switzerland", Population = 8544034 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Western Europe", Region = "Luxembourg", Population = 590321 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Western Europe", Region = "Monaco", Population = 38897 });
PopulationDetails.Add(new DrilldownModel() { Continent = "Europe", States =
"Western Europe", Region = "Liechtenstein", Population = 38155 });
#endregion
}
}

```

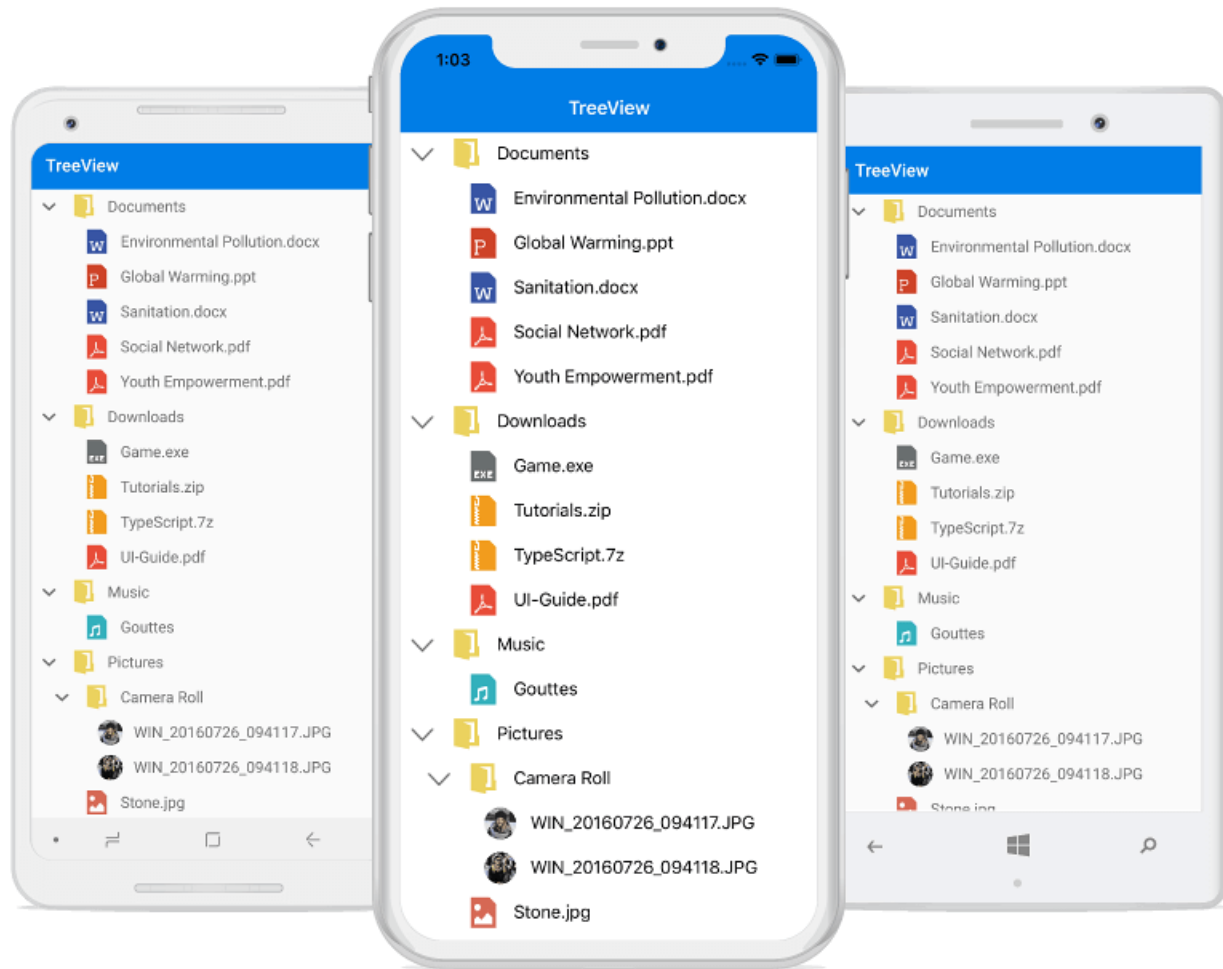
## SfTreeView

### SfTreeView

The Syncfusion TreeView for Xamarin.Forms 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

- Optimized view reusing strategy for enhanced performance.
- Binding and unbound mode - Support for binding hierarchical data or add unbound tree nodes.
- Selection - Support for selection with different selection modes and keyboard navigation.
- Templating - Provides complete UI customization through template and template selectors.



## Getting Started

This section provides a quick overview for getting started with the TreeView for Xamarin.Forms. Walk through the entire process of creating an app with TreeView.

### Assembly Deployment

After installing Essential Studio for Xamarin, you can find all the required assemblies in the {Syncfusion Essential Studio Installed location}\Essential Studio\{{ site.releaseversion }}\Xamarin\lib installation folder.

Eg: C:\Program Files (x86)\Syncfusion\Essential Studio\{{ site.releaseversion }}\Xamarin\lib

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

---

**Note:** Assemblies can be found in an unzipped package location in Mac.

---

#### Adding SfTreeView reference

You can add SfTreeView reference using one of the following methods:

##### Method 1: Adding SfTreeView reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.Xamarin.SfTreeView). To add SfTreeView to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.SfTreeView](https://www.nuget.org/packages/Syncfusion.Xamarin.SfTreeView), and then install it.

![Adding SfTreeView reference from NuGet](TreeView\_images/Adding SfTreeView reference.png)

---

**Note:** Install the same version of SfTreeView NuGet in all the projects.

---

##### Method 2: Adding SfTreeView reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfTreeView control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

##### Method 3: Adding SfTreeView assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.SfTreeView.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.GridCommon.Portable.dll Syncfusion.SfTreeView.XForms.Android.dll Syncfusion.SfTreeView.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.GridCommon.Portable.dll Syncfusion.SfTreeView.XForms.iOS.dll Syncfusion.SfTreeView.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.GridCommon.Portable.dll Syncfusion.SfTreeView.XForms.UWP.dll Syncfusion.SfTreeView.XForms.dll Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

---

---

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Launching the TreeView on Each Platform

To use this control inside an application, each platform application must initialize the TreeView renderer. This initialization step varies from platform to platform, and is discussed in the following sections:

#### Android

The Android launches the TreeView without any initialization and is enough to only initialize the Xamarin.Forms Framework to launch the application.

---

**Note:** After adding the reference, an additional step is required to initialize the renderer for iOS and UWP projects. If you are adding the references from toolbox, this step is not needed.

---

#### iOS

To launch the TreeView in iOS, call the `SfTreeViewRenderer.Init()` in the `FinishedLaunching` overridden method of the `AppDelegate` class after the Xamarin.Forms Framework initialization, and before the `LoadApplication` is called as demonstrated in the following code example:

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    ...
    global::Xamarin.Forms.Forms.Init ();
    SfTreeViewRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

#### Universal Windows Platform (UWP)

To launch the TreeView in UWP, call the `SfTreeViewRenderer.Init()` in the `MainPage` constructor before the `LoadApplication` is called as demonstrated in the following code example:

#### C#

```
public MainPage ()
{
    ...
    SfTreeViewRenderer.Init();
    LoadApplication (new App ());
    ...
}
```

#### ReleaseMode Issue in UWP Platform

The known Framework issue in UWP platform is the custom controls will not render when deployed the application in **Release Mode**.

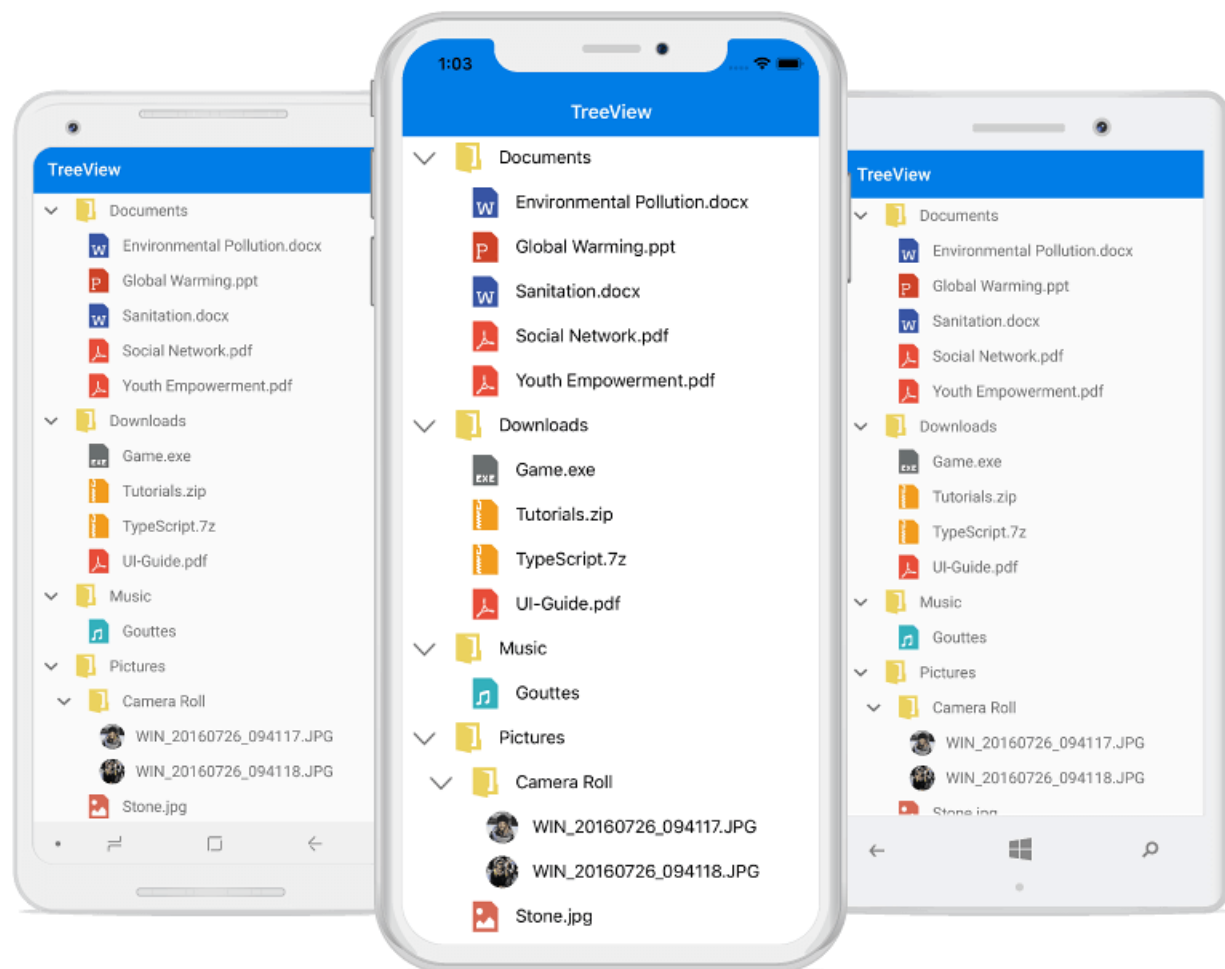
The above problem can be resolved by initializing the TreeView assemblies in `App.xaml.cs` in UWP project as in the following code snippet:

**C#**

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    rootFrame.NavigationFailed += OnNavigationFailed;
    // you'll need to add `using System.Reflection;`
    List<Assembly> assembliesToInclude = new List<Assembly>();
    //Now, add all the assemblies your app uses
    assembliesToInclude.Add(typeof(SfTreeViewRenderer).GetTypeInfo().Assembly);
    // replaces Xamarin.Forms.Forms.Init(e);
    Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    ...
}
```

**Create a tree view**

This section explains how to create a TreeView and configure it. The TreeView control can be configured entirely in C# code, or by using XAML markup. This is how the control will look like on iOS, Android, and Windows devices.



In this walk through, you will create a new application with the TreeView that includes the following topics:

- [Creating the project](#)
- [Adding tree view in Xamarin.Forms](#)
- [Populating Nodes without data source - Unbound Mode](#)
- [Creating Data Model](#)
- [Bind to a hierarchical data source - Bound Mode](#)
- [Creating hierarchical Data Model](#)
- [Bind to a Hierarchy Property Descriptors data source - Bound Mode](#)
- [Defining a template to expander and content view](#)
- [Interacting with a tree view](#)
- [Selection](#)

### Creating the Project

Create a new blank ([Xamarin.Forms.NET Standard](#)) application in Xamarin Studio or Visual Studio for Xamarin.Forms.

### Adding the tree view in Xamarin.Forms

1. Add the required assembly references to the corresponding projects as discussed in the [Assembly deployment](#) section.
2. Import the SfTreeView control namespace Syncfusion.XForms.TreeView.
3. Set the TreeView control to the ContentPage.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.MainPage">
<syncfusion:SfTreeView x:Name="treeView" />
</ContentPage>
```

### C#

```
using Syncfusion.XForms.TreeView;
using Xamarin.Forms;
namespace GettingStarted
{
    public class App : Application
    {
        SfTreeView treeView;
        public App()
        {
            treeView = new SfTreeView();
            MainPage = new ContentPage { Content = 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 [TreeView](#) control and a hierarchy of [TreeViewNode](#) objects. You create the node hierarchy by adding one or more root nodes to the TreeView control's [Nodes](#) collection. Each [TreeViewNode](#) can then have more nodes added to its Children collection. You can nest tree view nodes to whatever depth you require.

---

### Important

[ItemsSource](#) is an alternative mechanism to [Nodes](#) for putting content into the TreeView control. You cannot set both [ItemsSource](#) and [Nodes](#) at the same time. When you use [ItemsSource](#), nodes created for you internally, but you cannot access them from [Nodes](#) property.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.MainPage">
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView">
<syncfusion:SfTreeView.Nodes>
<treeviewengine:TreeViewNode Content="Australia">
<treeviewengine:TreeViewNode.ChildNodes>
<treeviewengine:TreeViewNode Content="New South Wales">
<treeviewengine:TreeViewNode.ChildNodes>
<treeviewengine:TreeViewNode Content="Sydney"/>
</treeviewengine:TreeViewNode.ChildNodes>
</treeviewengine:TreeViewNode>
</treeviewengine:TreeViewNode.ChildNodes>
</treeviewengine:TreeViewNode>
<treeviewengine:TreeViewNode Content="United States of America">
<treeviewengine:TreeViewNode.ChildNodes>
<treeviewengine:TreeViewNode Content="New York"/>
<treeviewengine:TreeViewNode Content="California">
<treeviewengine:TreeViewNode.ChildNodes>
<treeviewengine:TreeViewNode Content="San Francisco"/>
</treeviewengine:TreeViewNode.ChildNodes>
</treeviewengine:TreeViewNode>
</treeviewengine:TreeViewNode.ChildNodes>
</treeviewengine:TreeViewNode>
</syncfusion:SfTreeView.Nodes>
</syncfusion:SfTreeView>
</ContentPage.Content>
</ContentPage>
```

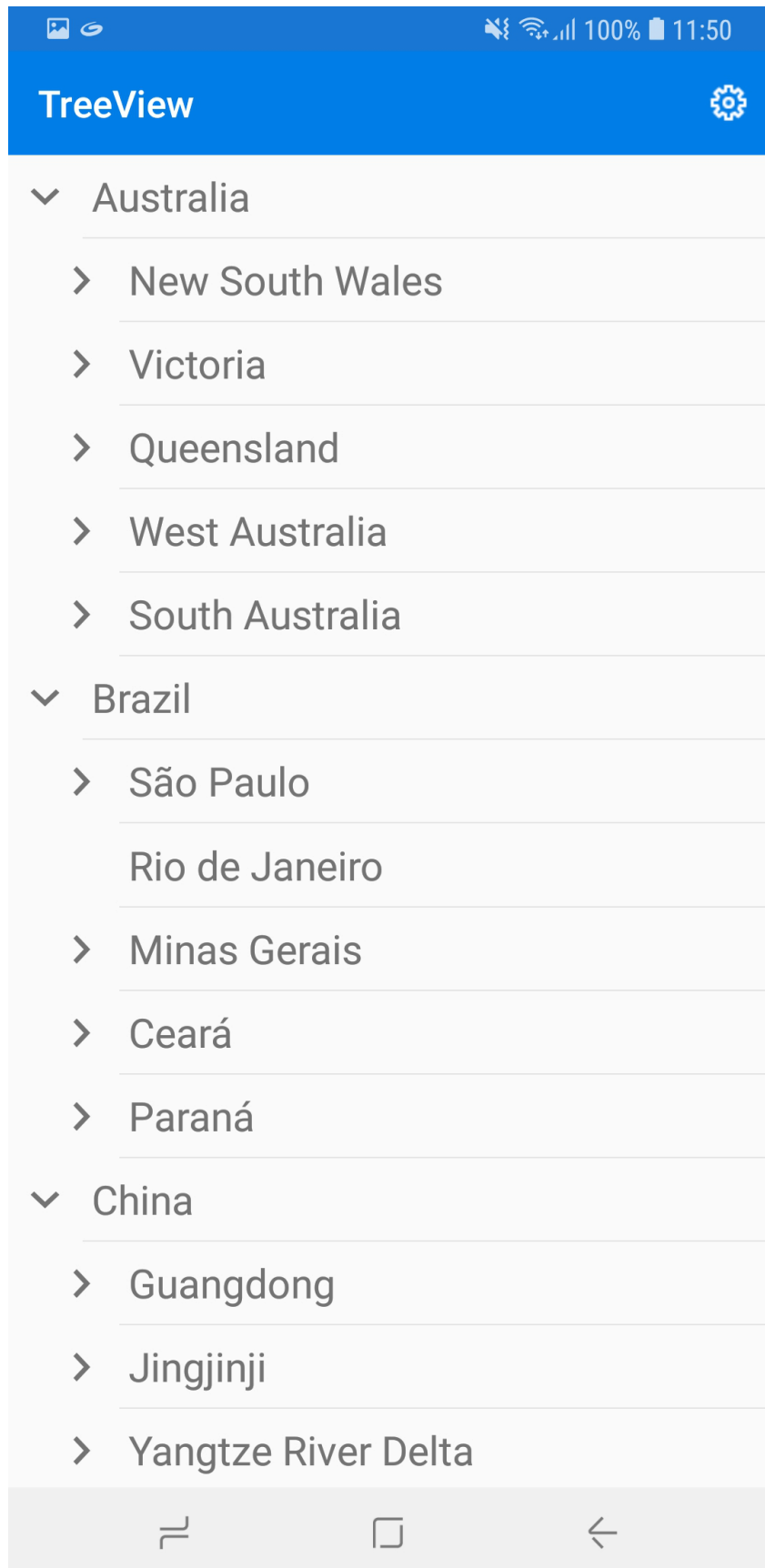
### C#

```
using Syncfusion.TreeView.Engine;
using Syncfusion.XForms.TreeView;
using Xamarin.Forms;
namespace GettingStarted
{
    public class App : Application
    {
        SfTreeView treeView;
        public App()
        {
            treeView = new SfTreeView();
            var australia = new TreeViewNode() { Content = "Australia" };
            var _NSW = new TreeViewNode() { Content = "New South Wales" };
            var Sydney = new TreeViewNode() { Content = "Sydney" };
            australia.ChildNodes.Add(_NSW);
            _NSW.ChildNodes.Add(Sydney);
            var usa = new TreeViewNode() { Content = "United States of America" };
            var newYork = new TreeViewNode() { Content = "New York," };
            var California = new TreeViewNode() { Content = "California" };
            var SanFrancisco = new TreeViewNode() { Content = "San Francisco" };
            usa.ChildNodes.Add(newYork);
            usa.ChildNodes.Add(California);
            California.ChildNodes.Add(SanFrancisco);
            treeView.Nodes.Add(australia);
            treeView.Nodes.Add(usa);
            MainPage = new ContentPage { Content = treeView };
        }
    }
}
```

Now, run the application to render the below output:

You can also download the entire source code of this demo from [here](#).





### Creating Data Model for the tree view

Create a data model to bind it to the control.

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#

```
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));
        }
    }
}
```

---

**Note:** If you want your data model to respond to property changes, then implement `INotifyPropertyChanged` interface in your model class.

---

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 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.

### Important

`ItemsSource` is an alternative mechanism to [Nodes](#) for putting content into the TreeView control. You cannot set both `ItemsSource` and `Nodes` at the same time. When you use `ItemsSource`, nodes created for you internally, but you cannot access them from `Nodes` property.

### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.MainPage">
<ContentPage.BindingContext>
<local:FileManagerViewModel x:Name="viewModel"></local:FileManagerViewModel>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView" ChildPropertyName="SubFiles"
ItemsSource="{Binding ImageNodeInfo}">
</syncfusion:SfTreeView>
</ContentPage.Content>
</ContentPage>

```

**C#**

```
SfTreeView treeView = new SfTreeView();
FileManagerViewModel viewModel = new FileManagerViewModel ();
treeView.ItemsSource = viewModel.ImageNodeInfo;
MainPage = new ContentPage { Content = treeView };
```

## Creating hierarchical Data Model for tree view

Create an hierarchical data model to bind it to the control.

Create a simple hierarchical data source as shown in the following code example in a new class file, and save it as FileManager.cs file.

**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));
    }
}
}

```

**Note:** If you need your hierarchical data model to respond to property changes, then implement the `INotifyPropertyChanged` interface in your model class.

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
{
    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.treeview_folder.png", assembly) };
        var download = new Folder() { FileName = "Downloads", ImageIcon =
            ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treeview_folder.png", assembly) };
        var mp3 = new Folder() { FileName = "Music", ImageIcon =
            ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treeview_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.treeview_video.png", assembly) };
var video2 = new File() { FileName = "Wild.mpeg", ImageIcon =
    ImageSource.FromResource("SampleBrowser.SfTreeView.Icons.NodeWithImage.treeview_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;
}
```

### 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 [HierarchyPropertyDescriptor](#)s.

---

## Important

---

**ItemsSource** is an alternative mechanism to [Nodes](#) for adding content into the TreeView control. You cannot set both **ItemsSource** and **Nodes** at the same time. When you use **ItemsSource**, nodes are created internally, but you cannot access them from the **Nodes** property.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.MainPage">
<ContentPage.BindingContext>
<local:FileManagerViewModel x:Name="viewModel"></local:FileManagerViewModel>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView" ItemsSource="{Binding
ImageNodeInfo}">
<sfTreeView:SfTreeView.HierarchyPropertyDescriptors>
<treeviewengine:HierarchyPropertyDescriptor TargetType="{x:Type
local:Folder}" ChildPropertyName="Files"/>
<treeviewengine:HierarchyPropertyDescriptor TargetType="{x:Type local:File}"
ChildPropertyName="SubFiles"/>
</sfTreeView:SfTreeView.HierarchyPropertyDescriptors>
</syncfusion:SfTreeView>
</ContentPage.Content>
</ContentPage>
```

## C#

```
SfTreeView treeView = new SfTreeView();
FileManagerViewModel viewModel = new FileManagerViewModel ();
treeView.ItemsSource = viewModel.ImageNodeInfo;
MainPage = new ContentPage { Content = treeView };
```

## Defining a template to expander and content view

By defining the [ExpanderTemplate](#) and [ItemTemplate](#) properties, a custom user interface (UI) can be created to display the data items for both expander and content view. It is applicable for both Unbound mode and Bound mode data items.

---

**Note:** By default, the binding context for each tree view item will be the data model object for Bound Mode and [TreeViewNode](#) for Unbound Mode. However, you can change the binding context for tree view items in Bound Mode as [TreeViewNode](#) by defining the [ItemTemplateContextType](#) enumeration to [Node](#), which is applicable for both [ExpanderTemplate](#) and [ItemTemplate](#) properties.

---

The following code example demonstrates how to customize your content view using the [ItemTemplate](#) and [ExpanderTemplate](#) property in both XAML and C#.

## XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.MainPage">
<ContentPage.BindingContext>
<local:FileManagerViewModel x:Name="viewModel"></local:FileManagerViewModel>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView" ItemsSource="{Binding
ImageNodeInfo}">
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid x:Name="grid" RowSpacing="0" BackgroundColor="Transparent">
<Grid Padding="5,5,5,5">
<Image Source="{Binding ImageIcon}" VerticalOptions="Center"
HorizontalOptions="Center" HeightRequest="35"
WidthRequest="35"/>
</Grid>
<Grid Grid.Column="1" RowSpacing="1" Padding="1,0,0,0"
VerticalOptions="Center">
<Label LineBreakMode="NoWrap" Text="{Binding ItemName}"
VerticalTextAlignment="Center"/>
</Grid>
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
<syncfusion:SfTreeView.ExpanderTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Image IsVisible="{Binding HasChildNodes,Converter={StaticResource
IconVisibleConverter}}"
Source="{ Binding IsExpanded,Converter={StaticResource
ExpanderIconConverter}}"
VerticalOptions="Center" HorizontalOptions="Center"/>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfTreeView.ExpanderTemplate>
</syncfusion:SfTreeView>
</ContentPage.Content>
</ContentPage>

```

## C#

```

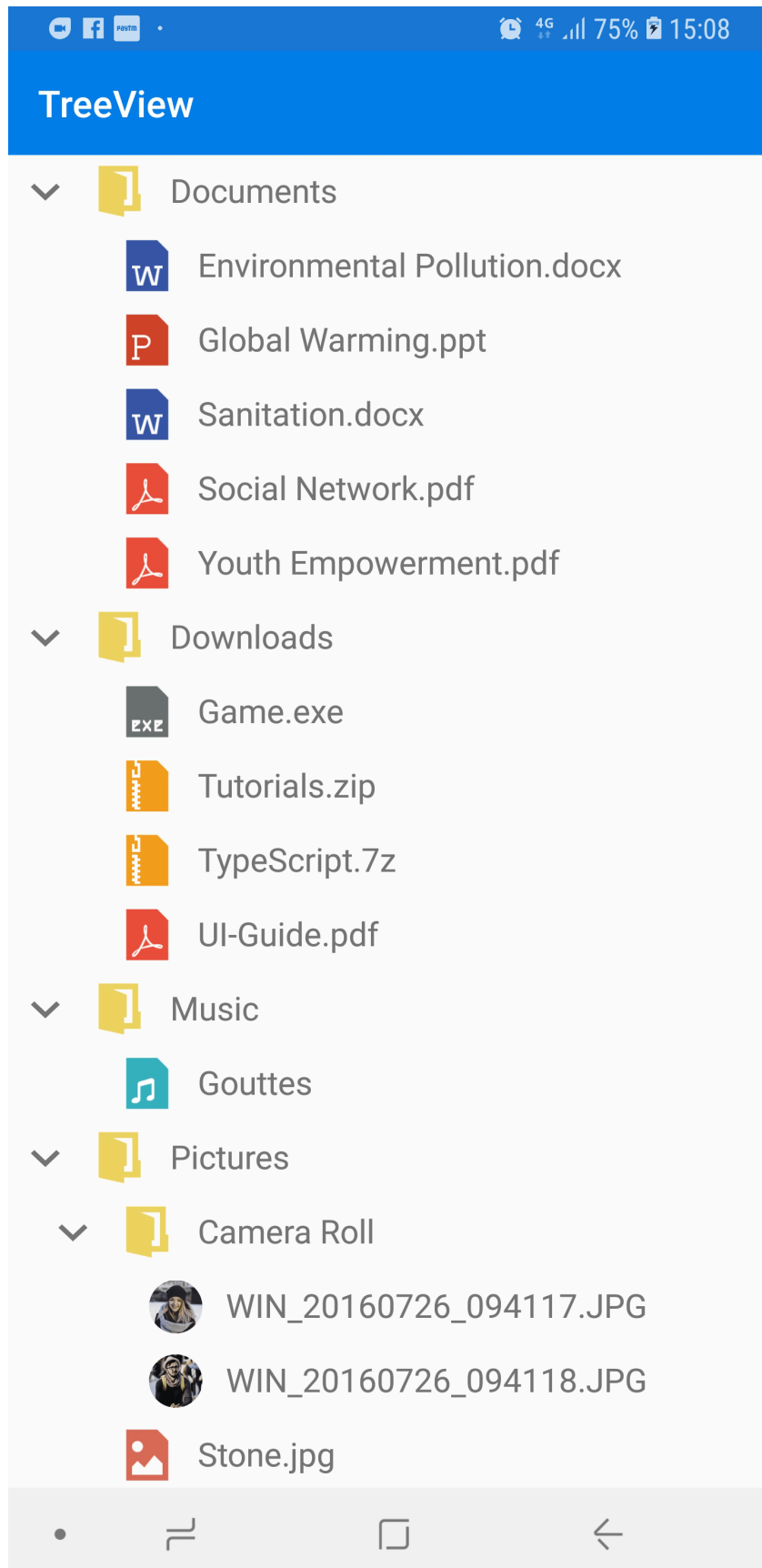
SfTreeView treeView = new SfTreeView();
FileManagerViewModel viewModel = new FileManagerViewModel ();

```

```
treeView.ItemsSource = viewModel.ImageNodeInfo;
treeView.ItemTemplateContextType = ItemTemplateContextType.Node;
treeView.ItemTemplate = new DataTemplate(() =>
{
    var grid = new Grid();
    var imageIcon = new Image();
    imageIcon.SetBinding(Image.SourceProperty, new
Binding("Content.ImageIcon"));
    var itemName = new Label { FontSize = 15 };
    itemName.SetBinding(Label.TextProperty, new Binding("Content.ItemName"));
    grid.Children.Add(imageIcon);
    grid.Children.Add(itemName, 1, 0);
    return grid;
});
treeView.ExpanderTemplate = new DataTemplate(()=>
{
    var grid = new Grid();
    var expanderIcon = new Image();
    imageIcon.SetBinding(Image.SourceProperty, new Binding("IsExpanded"));
})
MainPage = new ContentPage { Content = treeView };
```

Now, run the application to render the similar output:

You can also download the entire source code of this demo from [here](#).



### Interacting with a tree view

The **TreeView** allows you to expand and collapse the nodes either by user interaction on the nodes or by programmatically. The expanding and collapsing interactions can be handled with the help of [NodeCollapsing](#) and [NodeExpanding](#) events.

You can define how the nodes to be expanded while loading the **TreeView** by using [AutoExpandMode](#) property. Also, the **TreeView** allows you to set the restrictions whether expanding and collapsing of nodes can be performed only by tapping in expander view or in both expander view and content view by using [ExpandActionTarget](#) property.

**Note:** [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**.

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.MainPage">
<syncfusion:SfTreeView x:Name="treeView" AutoExpandMode="AllNodesExpanded"
ExpandActionTarget="Node"/>
</ContentPage>
```

### C#

```
using Syncfusion.XForms.TreeView;
using Xamarin.Forms;
namespace GettingStarted
{
    public class App : Application
    {
        SfTreeView treeView;
        public App()
        {
            treeView = new SfTreeView();
            treeView.AutoExpandMode = AutoExpandMode.AllNodesExpanded;
            treeView.ExpandActionTarget = ExpandActionTarget.Node;
            MainPage = new ContentPage { Content = treeView; }
        }
    }
}
```

### Selection

The **TreeView** allows selecting the item by setting the [SelectionMode](#) property. Set the [SelectionMode](#) property to single, single-deselect, multiple, extended and none based on the requirements. Informations about the selected item can be tracked using the [SelectedItem](#), [CurrentItem](#) and [SelectedItems](#) properties. Also, **TreeView** provides key board navigation support in UWP platform.

It also allows changing the selection highlight color by using the [SelectionBackgroundColor](#) property. Additionally, for unbound mode, you can change the selection foreground color of the text by using the [SelectionForegroundColor](#) property.

The selection operations can be handled with the help of [SelectionChanging](#) and [SelectionChanged](#).

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted;assembly=GettingStarted"
x:Class="GettingStarted.MainPage">
<syncfusion:SfTreeView x:Name="treeView" SelectionMode="Single"
SelectionBackgroundColor="#E4E4E4" SelectionForegroundColor="#D3D3D3"/>
</ContentPage>
```

### C#

```
using Syncfusion.XForms.TreeView;
using Xamarin.Forms;
namespace GettingStarted
{
    public class App : Application
    {
        SfTreeView treeView;
        public App()
        {
            treeView = new SfTreeView();
            treeView.SelectionMode = SelectionMode.Single;
            treeView.SelectionBackgroundColor = Color.FromHex("#E4E4E4");
            treeView.SelectionForegroundColor = Color.FromHex("#D3D3D3");
            MainPage = new ContentPage { Content = treeView; }
        }
    }
}
```

## Data Population

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**

```

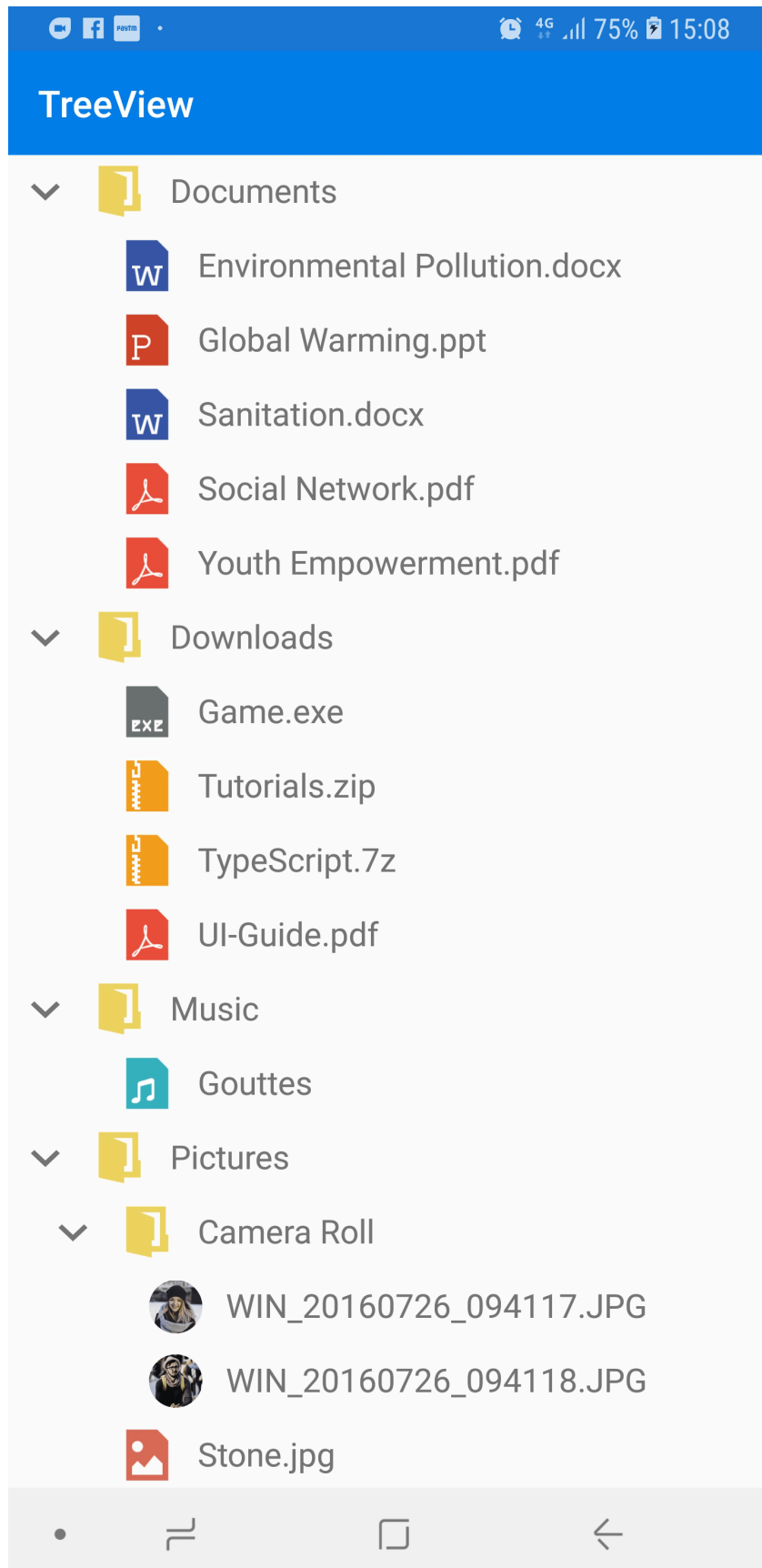
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms"
x:Class="GettingStarted.MainPage">
<ContentPage.BindingContext>
<local:FileManagerViewModel x:Name="viewModel"></local:FileManagerViewModel>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView"
ChildPropertyName="SubFiles"
ItemsSource="{Binding ImageNodeInfo}"/>
</syncfusion:SfTreeView>
</ContentPage.Content>
</ContentPage>

```

#### **C#**

```
SfTreeView treeView = new SfTreeView();  
FileManagerViewModel viewModel = new FileManagerViewModel ();  
treeView.ChildPropertyName = "SubFiles";  
treeView.ItemsSource = viewModel.ImageNodeInfo;  
MainPage = new ContentPage { Content = treeView };
```

You can also download the entire source code of this demo from [here](#).



### Populating Nodes without data binding - Unbound Mode

You can create and manage the [TreeViewNode](#) objects by yourself to display the data in a hierarchical view. Create the node hierarchy by adding one or more root nodes to the [Nodes](#) collection. Each [TreeViewNode](#) can then have more nodes added to its Children collection. You can nest tree view nodes to whatever depth you require.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms">
  <ContentPage.Content>
    <syncfusion:SfTreeView x:Name="treeView">
      <syncfusion:SfTreeView.Nodes>
        <treeviewengine:TreeViewNode Content="Australia">
          <treeviewengine:TreeViewNode.ChildNodes>
            <treeviewengine:TreeViewNode Content="New South Wales">
              <treeviewengine:TreeViewNode.ChildNodes>
                <treeviewengine:TreeViewNode Content="Sydney"/>
              </treeviewengine:TreeViewNode.ChildNodes>
            </treeviewengine:TreeViewNode>
          </treeviewengine:TreeViewNode.ChildNodes>
        </treeviewengine:TreeViewNode>
        <treeviewengine:TreeViewNode Content="United States of America">
          <treeviewengine:TreeViewNode.ChildNodes>
            <treeviewengine:TreeViewNode Content="New York"/>
            <treeviewengine:TreeViewNode Content="California">
              <treeviewengine:TreeViewNode.ChildNodes>
                <treeviewengine:TreeViewNode Content="San Francisco"/>
              </treeviewengine:TreeViewNode.ChildNodes>
            </treeviewengine:TreeViewNode>
          </treeviewengine:TreeViewNode.ChildNodes>
        </treeviewengine:TreeViewNode>
      </syncfusion:SfTreeView.Nodes>
    </syncfusion:SfTreeView>
  </ContentPage.Content>
</ContentPage>
```

#### C#

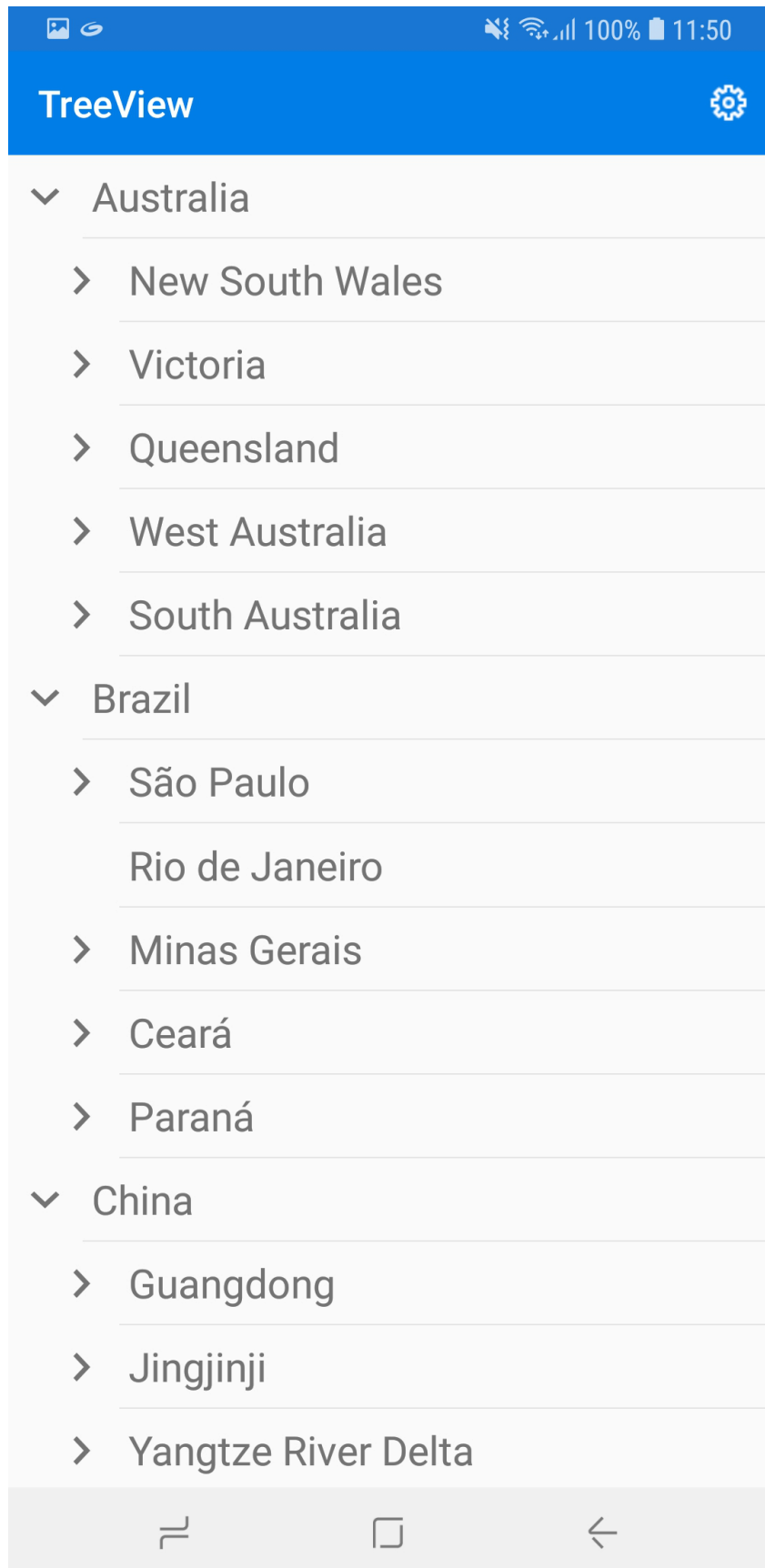
```
public class App : Application
{
    SfTreeView treeView;
    public App()
    {
        treeView = new SfTreeView();
        var australia = new TreeViewNode() { Content = "Australia" };
        var _NSW = new TreeViewNode() { Content = "New South Wales" };
        var _Sydney = new TreeViewNode() { Content = "Sydney" };
        australia.Children.Add(_NSW);
        _NSW.Children.Add(_Sydney);
        var usa = new TreeViewNode() { Content = "United States of America" };
        var newYork = new TreeViewNode() { Content = "New York," };
        var california = new TreeViewNode() { Content = "California" };
    }
}
```

```
var SanFrancisco = new TreeViewNode() { Content = "San Francisco" };  
usa.ChildNodes.Add(newYork);  
usa.ChildNodes.Add(_California);  
_California.ChildNodes.Add(SanFrancisco);  
treeView.Nodes.Add(australia);  
treeView.Nodes.Add(usa);  
MainPage = new ContentPage { Content = treeView};  
}  
}
```

Now, run the application to render the below output:

You can also download the entire source code of this demo from [here](#).





## Appearance

The TreeView allows customizing appearance of the underlying data, and provides different functionalities to the end-user.

## ItemTemplate

A template can be used to present the data in a way that makes sense for the application by using different controls.

The TreeView allows you to customize the appearance of content view and expander view by setting the [ItemTemplate](#) and [ExpanderTemplate](#) properties.

## XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage">
  <ContentPage.BindingContext>
    <local:FileManagerViewModel x:Name="viewModel"></local:FileManagerViewModel>
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <syncfusion:SfTreeView x:Name="treeView"
      ChildPropertyName="SubFiles"
      ItemsSource="{Binding ImageNodeInfo}"/>
    <syncfusion:SfTreeView.ItemTemplate>
      <DataTemplate>
        <Grid Padding="5,0,0,0">
          <Label Text="{Binding ItemName}"
            VerticalTextAlignment="Center"/>
        </Grid>
      </DataTemplate>
    </syncfusion:SfTreeView.ItemTemplate>
  </syncfusion:SfTreeView>
</ContentPage.Content>
</ContentPage>
```

## C#

```
public class MainPage : ContentPage
{
    SfTreeView treeView;
    public MainPage()
    {
        treeView = new SfTreeView();
        FileManagerViewModel viewModel = new FileManagerViewModel ();
        treeView.ChildPropertyName = "SubFiles";
        treeView.ItemsSource = viewModel.ImageNodeInfo;
        treeView.ItemTemplate = new DataTemplate(() => {
            var grid = new Grid ();
            var itemName = new Label;
            itemName.SetBinding(Label.TextProperty, new Binding("ItemName"));
            grid.Children.Add(itemName);
            return grid;
        });
    }
}
```

```

    });
}
}

```

### 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

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStartedBound"
x:Class="GettingStarted.MainPage">
  <ContentPage.BindingContext>
  <local:FileManagerViewModel x:Name="viewModel"></local:FileManagerViewModel>
</ContentPage.BindingContext>
  <ContentPage.Content>
  <syncfusion:SfTreeView x:Name="treeView"
  ItemTemplateContextType="Node"
  ChildPropertyName="SubFiles"
  ItemsSource="{Binding ImageNodeInfo}"/>
  <syncfusion:SfTreeView.ItemTemplate>
  <DataTemplate>
  <Grid Padding="5,0,0,0">
  <Label Text="{Binding Content.ItemName}"
  VerticalTextAlignment="Center"/>
  </Grid>
  </DataTemplate>
  </syncfusion:SfTreeView.ItemTemplate>
  </syncfusion:SfTreeView>
  </ContentPage.Content>
</ContentPage>

```

### C#

```

public class MainPage : ContentPage
{
    SfTreeView treeView;
    public MainPage()
    {
        treeView = new SfTreeView();
        FileManagerViewModel viewModel = new FileManagerViewModel ();
        treeView.ChildPropertyName = "SubFiles";
        treeView.ItemTemplateContextType = ItemTemplateContextType.Node;
        treeView.ItemsSource = viewModel.ImageNodeInfo;
        treeView.ItemTemplate = new DataTemplate(() => {
            var grid = new Grid ();
            var itemName = new Label;

```

```

itemName.SetBinding(Label.TextProperty, new Binding("Content.ItemName"));
grid.Children.Add(itemName);
return grid;
});
}
}

```

Similarly, you can customize the expander view by using `ExpanderTemplate` property like above example.

### ItemTemplate Selector

The TreeView allows you to customize the appearance of each item with different templates based on specific constraints by using the [DataTemplateSelector](#). You can choose a [DataTemplate](#) for each item at runtime based on the value of data-bound property using `DataTemplateSelector`.

#### Create a data template selector

Create custom class that inherits from `DataTemplateSelector`, and override the `OnSelectTemplate` method to return the [DataTemplate](#) for that item. At runtime, the TreeView invokes the `OnSelectTemplate` method for each item and passes the data object as parameter.

Create different templates and by using `DataTemplateSelector`, load those templates using `OnSelectTemplate` based on requirements.

### C#

```

public class ItemTemplateSelector : DataTemplateSelector
{
    public DataTemplate Template1 { get; set; }
    public DataTemplate Template2 { get; set; }
    public ItemTemplateSelector()
    {
        this.Template1 = new DataTemplate(typeof(Template1));
        this.Template2 = new DataTemplate(typeof(Template2));
    }
    protected override DataTemplate OnSelectTemplate(object item, BindableObject container)
    {
        var treeviewNode = item as TreeViewNode;
        if (treeviewNode == null)
            return null;
        if (treeviewNode.Level == 0)
            return Template1;
        else
            return Template2;
    }
}

```

#### Applying a data template selector

Assign custom [DataTemplateSelector](#) to the [ItemTemplate](#) either in XAML or C#.

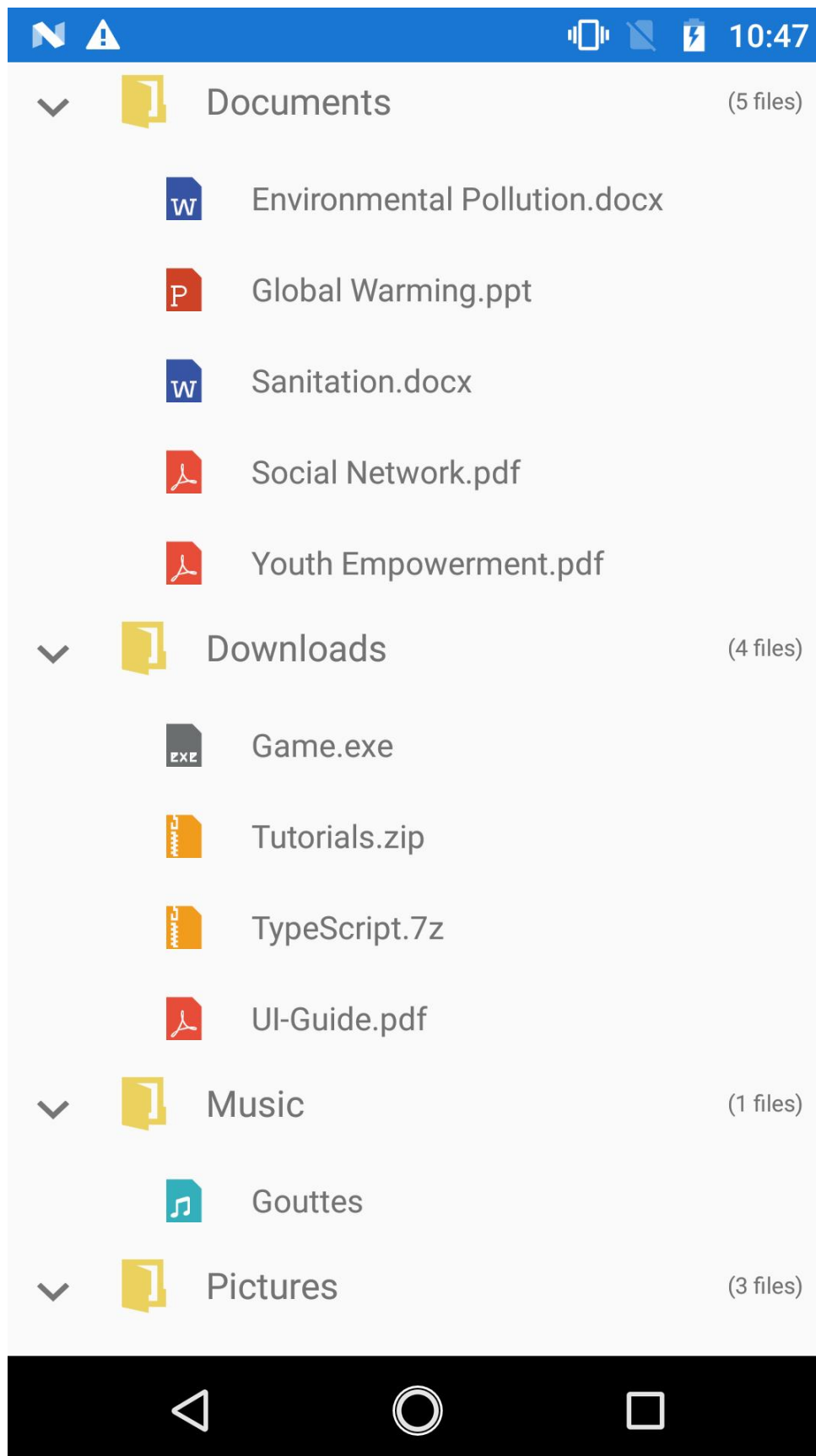
Following code example illustrates to load the different templates for treeview items using `DataTemplateSelector` based on different levels.

### XML

```
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms"
x:Class="TemplateSelector.MainPage"
xmlns:local="clr-namespace:TemplateSelector;assembly=TemplateSelector">
<ContentPage.Resources>
<ResourceDictionary>
<local:ItemTemplateSelector x:Key="ItemTemplateSelector" />
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView"
ItemTemplate="{StaticResource ItemTemplateSelector}"/>
</ContentPage.Content>
</ContentPage>
```

## C#

```
SfTreeView treeView = new SfTreeView();
treeView.ItemTemplate = new ItemTemplateSelector()
```



You can also download the entire source code of this demo from [here](#)

Similarly, you can provide `DataTemplateSelector` for [ExpanderTemplate](#) property.

### Indentation

The TreeView allows customizing the indent spacing of items by setting the [Indentation](#) property. The default value of this property is `40`. This property can be customized at runtime.

#### XML

```
<syncfusion:SfTreeView x:Name="treeView" Indentation="40">
```

#### C#

```
SfTreeView treeView = new SfTreeView();  
treeView.Indentation = 40;
```

### ExpanderWidth

The TreeView allows customizing the width of expander view by setting the [ExpanderWidth](#) property. The default value of this property is `40`. This property can be customized at runtime.

#### XML

```
<syncfusion:SfTreeView x:Name="treeView" ExpanderWidth="40">
```

#### C#

```
SfTreeView treeView = new SfTreeView();  
treeView.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="treeView" ExpanderPosition="End">
```

#### C#

```
SfTreeView treeView = new SfTreeView();  
treeView.ExpanderPosition = ExpanderPosition.End;
```

### Level based styling

The TreeView allows you to customize the style of `TreeViewItem` based on different levels by using [IValueConverter](#).

#### XML

```
<ContentPage.Resources>
<ResourceDictionary>
<local:FontAttributeConverter x:Key="FontAttributeConverter"/>
</ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView"
ChildPropertyName="SubFolder"
AutoExpandMode="AllNodesExpanded"
ItemTemplateContextType="Node"
ExpanderWidth="0"
ItemsSource="{Binding Folders}">
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Label LineBreakMode="NoWrap"
Text="{Binding Content.FolderName}"
FontSize="Medium"
FontAttributes="{Binding Level, Converter={x:StaticResource
FontAttributeConverter}}"/>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>
</ContentPage.Content>
```

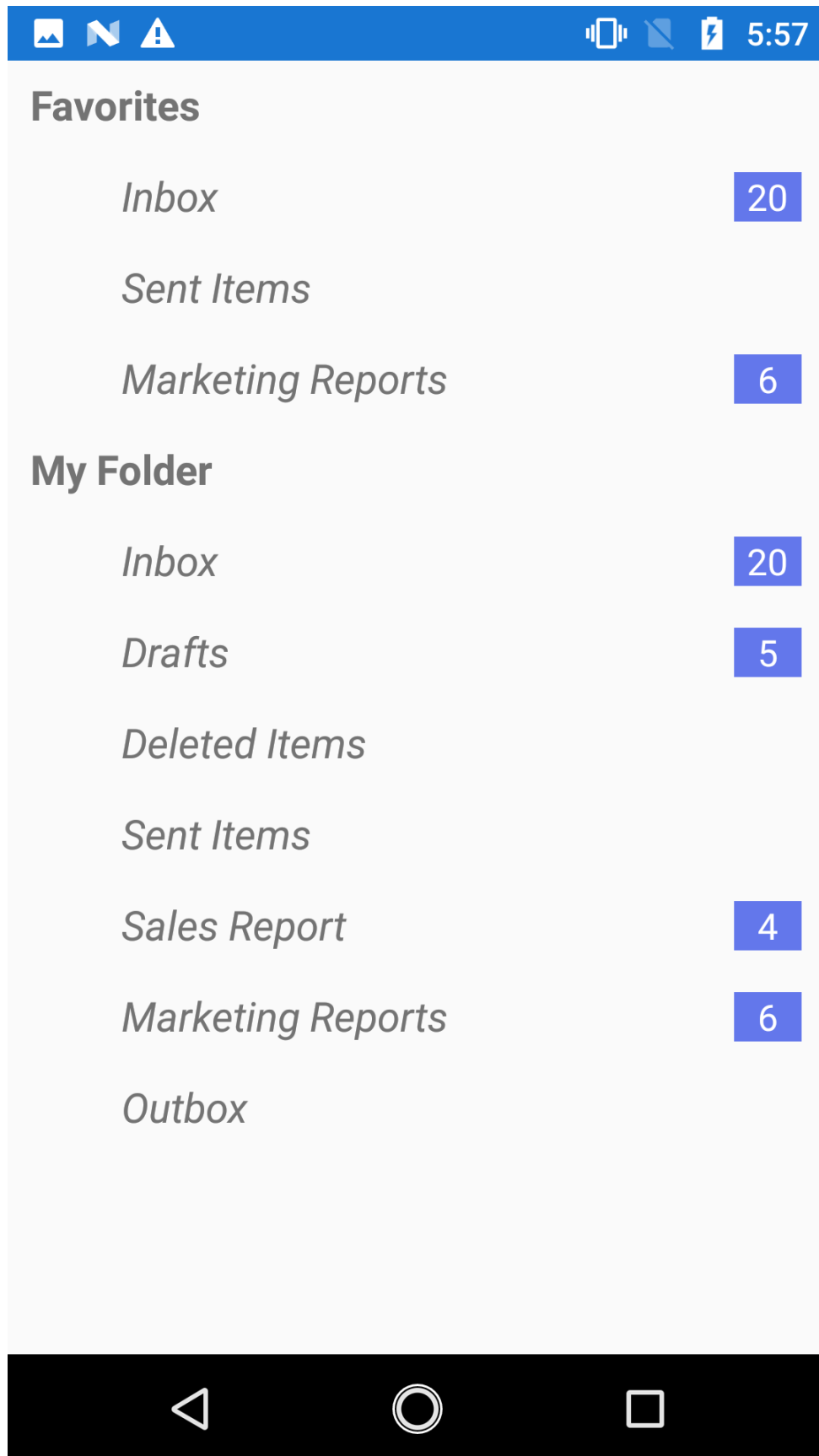
## C#

```
public class FontAttributeConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var level = (int)value;
        return level == 0 ? FontAttributes.Bold : FontAttributes.Italic;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Now, run the application to render the below output:

You can also download the entire source code of this demo from [here](#).





### Animation

The SfTreeView supports to animate expanding or collapsing the [TreeViewNode](#). To enable/disable the animation use [IsAnimationEnabled](#) property of SfTreeView.

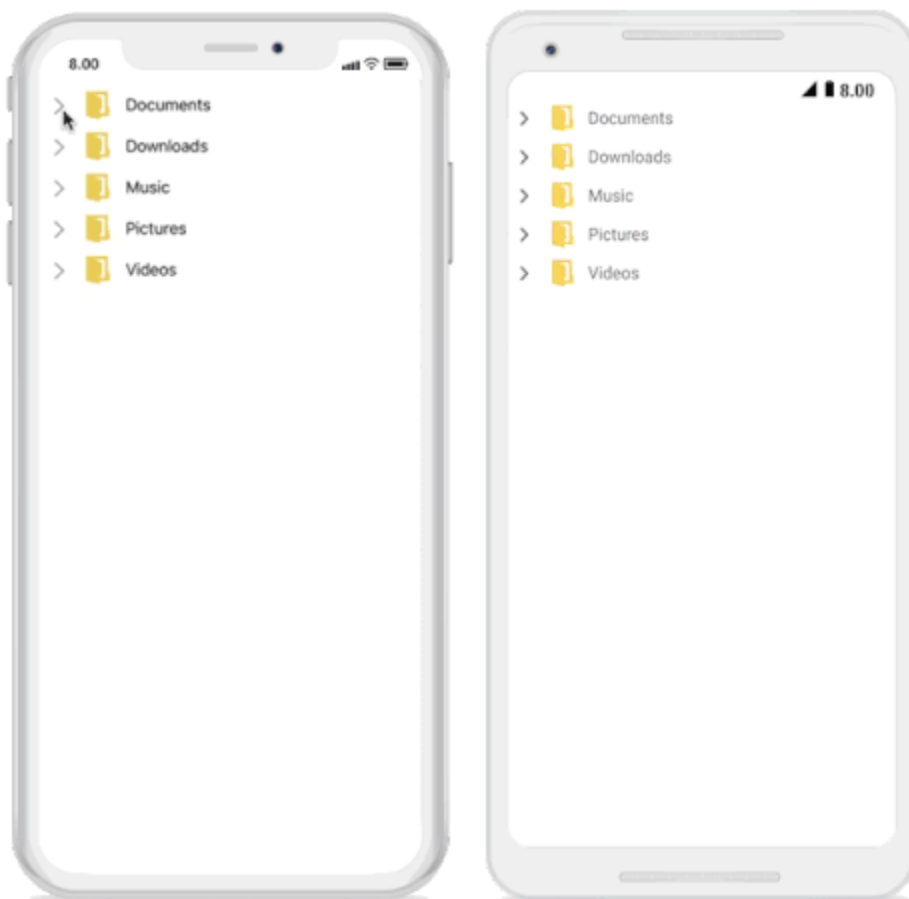
**Note:** The default value of the IsAnimationEnabled property is false.

### XML

```
<syncfusion:SfTreeView x:Name="treeView" IsAnimationEnabled="true">
```

### C#

```
SfTreeView treeView = new SfTreeView();  
treeView.IsAnimationEnabled = true;
```



### Expand and Collapse

The TreeView allows you to expand and collapse the nodes either by user interaction on the nodes or by programmatically.

#### Expand Action Target

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 [ExpandActionTarget](#) property.

## XML

```
<syncfusion:SfTreeView x:Name="TreeView" ExpandActionTarget="Node"/>
```

## C#

```
// Expands by tapping both expander view and content view.  
treeView.ExpandActionTarget = ExpandActionTarget.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.
- RootNodesExpanded : Expands only the root item when loaded.
- AllNodesExpanded : Expands all the items when loaded.

### 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 expand the all items of level passed to it.

## C#

```
// Expands all the nodes of root level '0'  
treeView.ExpandNodes(0);  
// Collapses all the nodes of root level '0'  
treeView.CollapseNodes(0);  
// Expand a particular node.  
treeView.ExpandNode(node);  
// Expand a particular node.  
treeView.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  
treeView.ExpandAll();
```

```
//Collapses all the nodes  
treeView.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.

You can also achieve handle expand and collapse operation using `ExpandCommand` and `CollapseCommand`.

### Interactivity

This section explains about how to interact with `TreeView` and its items.

#### Interacting with TreeView items

##### Loaded event

The [Loaded](#) event is raised when the TreeView is loading in view for the first time.

### C#

```
treeView.Loaded += TreeView_Loaded;  
private void TreeView_Loaded(object sender, TreeViewLoadedEventArgs e)  
{  
    DisplayAlert("Message", "TreeView is Loaded", "Done");  
}
```

The `Loaded` event is used for the following use case:

- To scroll the desired item by using the [BringIntoView](#).

##### Tapped 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#**

```
treeView.ItemTapped += TreeView_ItemTapped;
private void TreeView_ItemTapped(object sender, ItemTappedEventArgs e)
{
    DisplayAlert("Item Tapped", "TreeView item tapped", "Close");
}
```

*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#**

```
treeView.ItemDoubleTapped += TreeView_ItemDoubleTapped;
private void TreeView_ItemDoubleTapped(object sender,
ItemDoubleTappedEventArgs e)
{
    DisplayAlert("Item DoubleTapped", "TreeView item double tapped", "Close");
}
```

*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.
- **Handled**: Gets or sets whether the event is handled or not.

**C#**

```
treeView.ItemHolding += TreeView_ItemHolding;
private void TreeView_ItemHolding(object sender, ItemHoldingEventArgs e)
{
    DisplayAlert("Item Hold", "TreeView item is holding", "Close");
}
```

## Scrolling

The TreeView provides various options to achieve programmatic scrolling. Please walkthrough the below section in detail to achieve the same.

### Bring Into View

The TreeView allows programmatic scrolling based on the data model and [TreeNode](#) by using the [BringIntoView](#) method.

#### C#

```
private void BringIntoView_Clicked(object sender, EventArgs e)
{
    var count= viewModel.ImageNodeInfo.Count;
    var data = viewModel.ImageNodeInfo[count-1];
    TreeView.BringIntoView(data);
}
```

You can also download the entire source code of this demo from [here](#).

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_Clicked(object sender, EventArgs e)
{
    var count= viewModel.ImageNodeInfo.Count;
    var data = viewModel.ImageNodeInfo[count-1];
    // Here, the second optional parameter has been passed as true hence it will
    // disable the animation
    TreeView.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_Clicked(object sender, EventArgs e)
{
    var count= viewModel.ImageNodeInfo.Count;
```

```
var data = viewModel.ImageNodeInfo[count-1];
TreeView.BringIntoView(data, false, true);
}
```

**Note:** We need to set the [NodePopulationMode](#) API value as `PopulationMode.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_Clicked(object sender, EventArgs e)
{
    var count= viewModel.ImageNodeInfo.Count;
    var data = viewModel.ImageNodeInfo[count-1];
    // Scrolls to the data item to make visible in the view.
    treeView.BringIntoView(data, false, false, ScrollToPosition.MakeVisible);
}
```

#### Scrollbar Visibility

The TreeView provides an option to enable or disable the `Scrollbar` visibility by using the [IsScrollBarVisible](#) property. By default, the value will be true.

**Note:** Due to some restrictions in native ScrollView renderer in Xamarin.Forms, you cannot change the `IsScrollBarVisible` value at runtime. It can be defined only when initializing the TreeView.

#### **XML**

```
<syncfusion:SfTreeView x:Name="treeView" IsScrollBarVisible="False" />
```

#### **C#**

```
SfTreeView treeView = new SfTreeView();
treeView.IsScrollBarVisible=false;
```

#### Selection

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 not 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 in **UWP** platform and for other platforms, it works as **Single** selection.

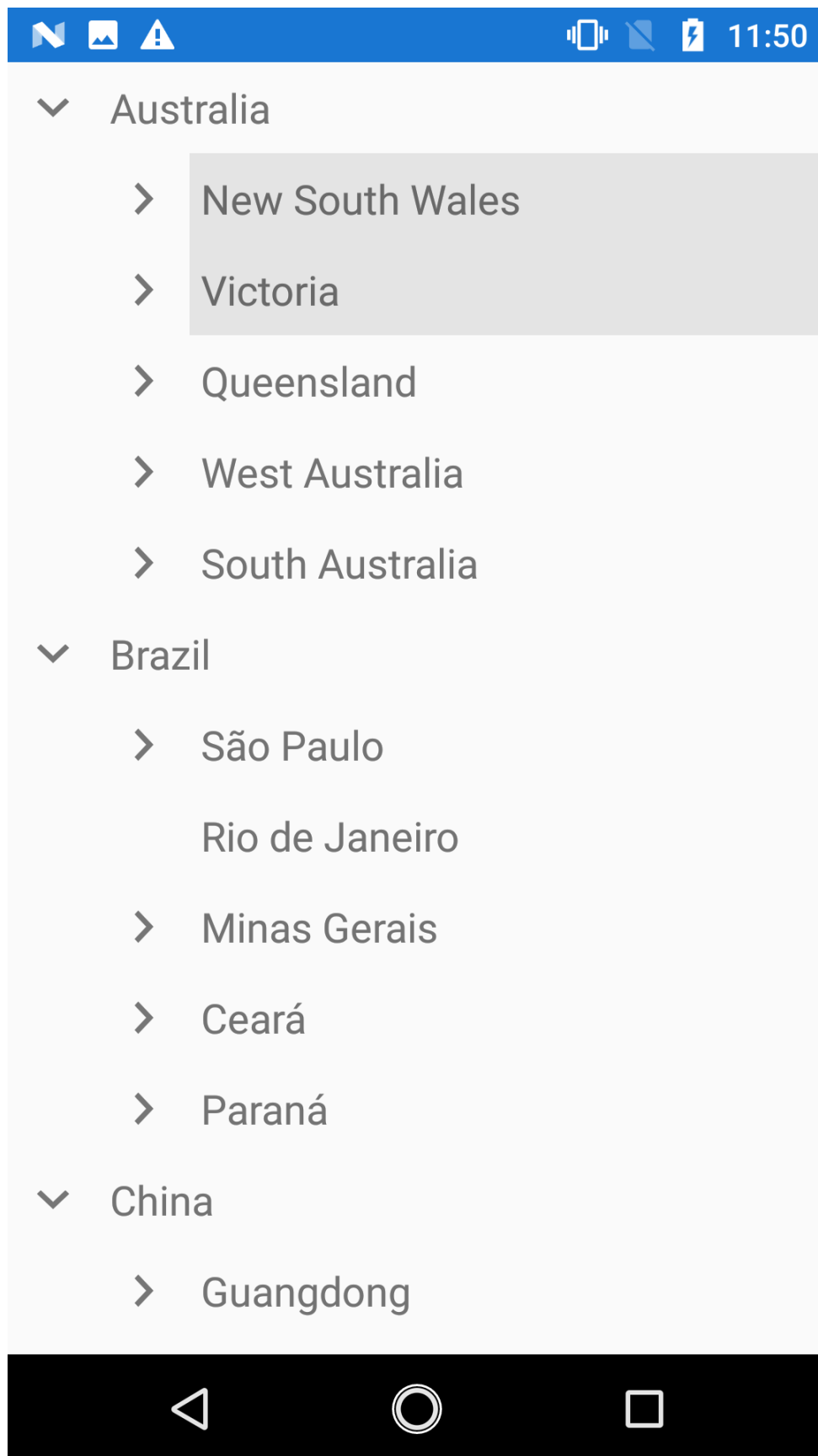
## XML

```
<syncfusion:SfTreeView x:Name="TreeView" SelectionMode="Multiple"/>
```

## C#

```
TreeView.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#

```
treeView.SelectedItem = viewModel.CountriesInfo[2];
```

When the selection mode is **Multiple**, programmatically select more than one item by adding the underlying object to the [SelectedItems](#) property.

#### C#

```
treeView.SelectedItems.Add(viewModel.CountriesInfo[2]);  
treeView.SelectedItems.Add(viewModel.CountriesInfo[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.

---

### 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#

```
treeView.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#**

```
treeView.SelectionChanging += TreeView_SelectionChanging;
private void TreeView_SelectionChanging(object sender,
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#**

```
treeView.SelectionChanged += TreeView_SelectionChanged;
private void TreeView_SelectionChanged(object sender,
ItemSelectionChangedEventArgs e)
{
    treeView.SelectedItems.Clear();
}
```

---

**Note:** [SelectionChanging](#) and [SelectionChanged](#) events will be triggered only on UI interactions.

---

## Key Navigation

The TreeView allows to select the items through keyboard interactions. Behavior of key navigation in UWP platform is explained as follows:

- When the [SelectionMode](#) is `Single` or `SingleDeselect`, the selected item is highlighted with [FocusBorderColor](#) around the item while key navigation.
- 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. For Android and iOS platform, the default color is `Color.Transparent` and for UWP platform, the default color is `Color.FromRgb(213, 213, 213)`.

#### FocusBorderThickness

The [FocusBorderThickness](#) property is used to set the border thickness for the current focused item. For Android and iOS platform, the default thickness is `0` and for UWP platform, the default thickness is `1`.

#### 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.

#### Checkbox

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 [ItemTemplateContextType](#) as `Node` to bind the [TreeViewNode.IsChecked](#) property to [CheckBox](#) in [ItemTemplate](#). To know more about [ItemTemplateContextType](#) click [here](#).

**Note:** TreeView process and sets [TreeViewNode.IsChecked](#) based on [CheckedItems](#) only when you are binding [ItemsSource](#).

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:BoundMode_CheckBox"
  x:Class="BoundMode_CheckBox.MainPage"
  xmlns:TreeView="clr-namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms">
```

```

xmlns:SfButtons="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:TreeViewEngine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms">
<ContentPage.BindingContext>
<local:FileManagerViewModel />
</ContentPage.BindingContext>
<TreeView:SfTreeView x:Name="SfTreeView"
ItemsSource="{Binding Folders}"
AutoExpandMode="RootNodesExpanded"
ItemTemplateContextType="Node"
CheckBoxMode="Recursive"
CheckedItems="{Binding CheckedItems}">
<TreeView:SfTreeView.HierarchyPropertyDescriptors>
<TreeViewEngine.HierarchyPropertyDescriptor TargetType="{x:Type
local:Folder}" ChildPropertyName="Files"/>
<TreeViewEngine.HierarchyPropertyDescriptor TargetType="{x:Type local:File}"
ChildPropertyName="SubFiles"/>
</TreeView:SfTreeView.HierarchyPropertyDescriptors>
<TreeView:SfTreeView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid Padding="5">
<SfButtons:SfCheckBox
x:Name="CheckBox"
Text="{Binding Content.FileName}"
IsChecked="{Binding IsChecked, Mode=TwoWay}"/>
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</TreeView:SfTreeView.ItemTemplate>
</TreeView:SfTreeView>
</ContentPage>

```

**C#**

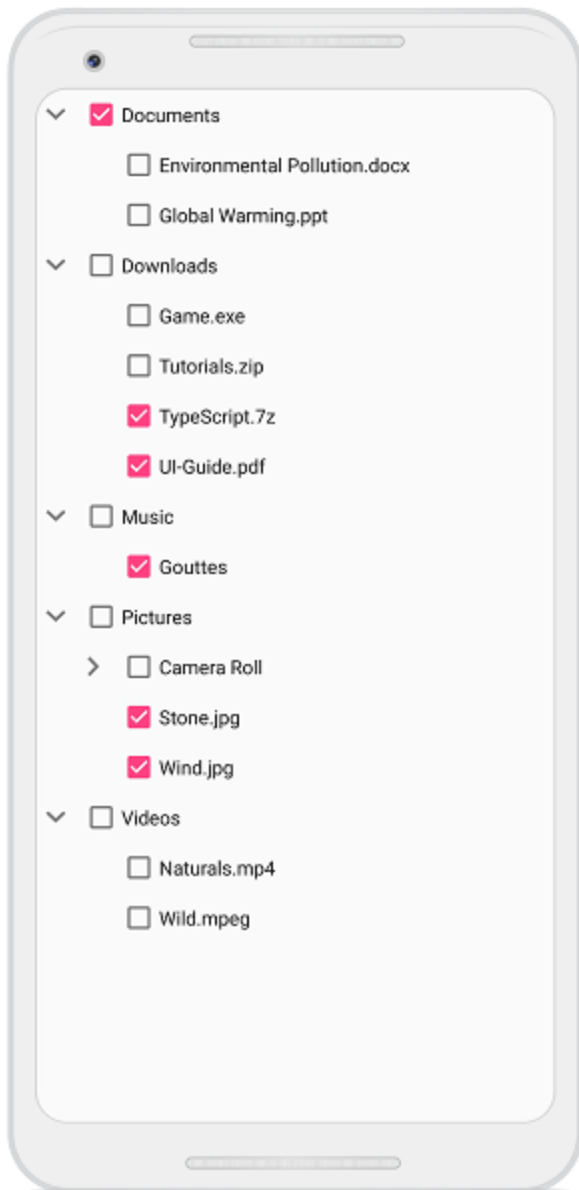
```

public class FileManagerViewModel
{
    private ObservableCollection<object> checkedItems;
    public ObservableCollection<object> CheckedItems
    {
        get { return checkedItems; }
        set { this.checkedItems = value; }
    }
    public ObservableCollection<Folder> Folders { get; set; }
    public FileManagerViewModel()
    {
        this.Folders = GetFiles();
    }
    private ObservableCollection<Folder> GetFiles()
    {
        var nodeImageInfo = new ObservableCollection<Folder>();
        Assembly assembly = typeof(MainPage).GetTypeInfo().Assembly;
        var doc = new Folder() { FileName = "Documents"};
    }
}

```

```
var download = new Folder() { FileName = "Downloads", };
var mp3 = new Folder() { FileName = "Music", };
var pictures = new Folder() { FileName = "Pictures", };
var video = new Folder() { FileName = "Videos", };
var pollution = new File() { FileName = "Environmental Pollution.docx", };
var globalWarming = new File() { FileName = "Global Warming.ppt", };
var games = new File() { FileName = "Game.exe", };
var tutorials = new File() { FileName = "Tutorials.zip", };
var TypeScript = new File() { FileName = "TypeScript.7z", };
var uiGuide = new File() { FileName = "UI-Guide.pdf", };
var song = new File() { FileName = "Gouts", };
var camera = new File() { FileName = "Camera Roll", };
var stone = new File() { FileName = "Stone.jpg", };
var wind = new File() { FileName = "Wind.jpg", };
var img0 = new SubFile() { FileName = "WIN_20160726_094117.JPG", };
var img1 = new SubFile() { FileName = "WIN_20160726_094118.JPG", };
var video1 = new File() { FileName = "Naturals.mp4", };
var video2 = new File() { FileName = "Wild.mpg", };
doc.Files = new ObservableCollection<File>
{
    pollution,
    globalWarming
};
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);
checkedItems = new ObservableCollection<object>();
checkedItems.Add(doc);
```

```
checkedItems.Add(TypeScript);  
checkedItems.Add(uiGuide);  
checkedItems.Add(stone);  
checkedItems.Add(wind);  
checkedItems.Add(song);  
return nodeImageInfo;  
}  
}
```



You can download the entire source of this demo from [here](#)

#### Working with Checkbox in UnboundMode

You can directly set the checkbox state by setting the [TreeViewNode.IsChecked](#) property value while creating nodes.

**XML**

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:UnboundMode_Checkbox"
x:Class="UnboundMode_Checkbox.MainPage"
xmlns:treeviewengine="clr-
namespace:Syncfusion.TreeView.Engine;assembly=Syncfusion.SfTreeView.XForms"
xmlns:SfCheckBox="clr-
namespace:Syncfusion.XForms.Buttons;assembly=Syncfusion.Buttons.XForms"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms">
<ContentPage.Content>
<syncfusion:SfTreeView CheckBoxMode="Recursive">
<syncfusion:SfTreeView.Nodes>
<treeviewengine:TreeNode Content="Australia" IsChecked="True"
IsExpanded="True">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode x:Name="newSouthWales" Content="New South
Wales">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="Sydney"/>
<treeviewengine:TreeNode Content="New York">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="San Francisco"/>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
<treeviewengine:TreeNode Content="United States of America">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="New York">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="San Francisco"/>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
<treeviewengine:TreeNode Content="California">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="San Francisco"/>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
<treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
<treeviewengine:TreeNode Content="New South Wales" IsChecked="True">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="Sydney"/>
<treeviewengine:TreeNode Content="New York">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="San Francisco"/>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
<treeviewengine:TreeNode Content="United States of America"
IsChecked="True">
<treeviewengine:TreeNode.ChildNodes>

```

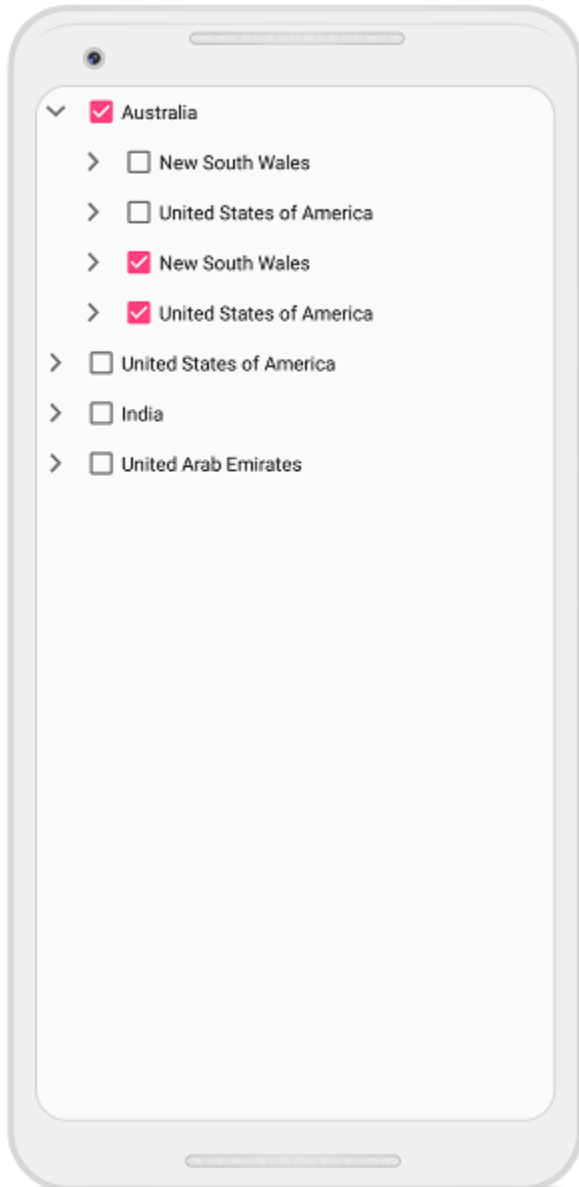


```

<treeviewengine:TreeNode Content="New York">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="San Francisco"/>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
<treeviewengine:TreeNode Content="California">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="San Francisco"/>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
<treeviewengine:TreeNode x:Name="America" Content="United States of
America">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="New York"/>
<treeviewengine:TreeNode Content="California">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="San Francisco"/>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
<treeviewengine:TreeNode x:Name="India" Content="India">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="TamilNadu">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="Chennai"/>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
<treeviewengine:TreeNode Content="United Arab Emirates">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="Abu Dhabi"/>
<treeviewengine:TreeNode Content="Dubai">
<treeviewengine:TreeNode.ChildNodes>
<treeviewengine:TreeNode Content="Burj Khalifa"/>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
</treeviewengine:TreeNode.ChildNodes>
</treeviewengine:TreeNode>
</syncfusion:SfTreeView.Nodes>
<syncfusion:SfTreeView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<Grid Padding="5">
<SfCheckBox:SfCheckBox
Text="{Binding Content}"
IsChecked="{Binding IsChecked, Mode=TwoWay}" />
</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>

```

```
</syncfusion:SfTreeView.ItemTemplate>  
</syncfusion:SfTreeView>  
</ContentPage.Content>  
</ContentPage>
```



You can download the entire source of this demo [here](#)

### 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="TreeView" CheckBoxMode="Recursive"/>
```

### C#

```
TreeView.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 **TreeNode** or collection of **TreeNode** 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#

```
treeView.CheckedItems.Add(viewModel.Folders[2]);
treeView.CheckedItems.Add(viewModel.Folders[3]);
```

#### Get or Set Checked Nodes in Unbound Mode

You can get the list of checked nodes by using [GetCheckedNodes](#) method. You can set checkbox state by setting [TreeNode.IsChecked](#) property.

### C#

```
treeView.GetCheckedNodes();
```

### 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 **TreeNode** and data associated with the checked item as its arguments.

**C#**

```
treeView.NodeChecked += TreeView_NodeChecked;
private void TreeView_NodeChecked(object sender,
Syncfusion.XForms.TreeView.NodeCheckedEventArgs e)
{
}
```

**Note:** **NodeChecked** event occurs only in UI interactions.

**MVVM**

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="treeView"
SelectedItem="{Binding SelectedPlace}"
ChildPropertyName="States"
ItemsSource="{Binding CountriesInfo}"/>
```

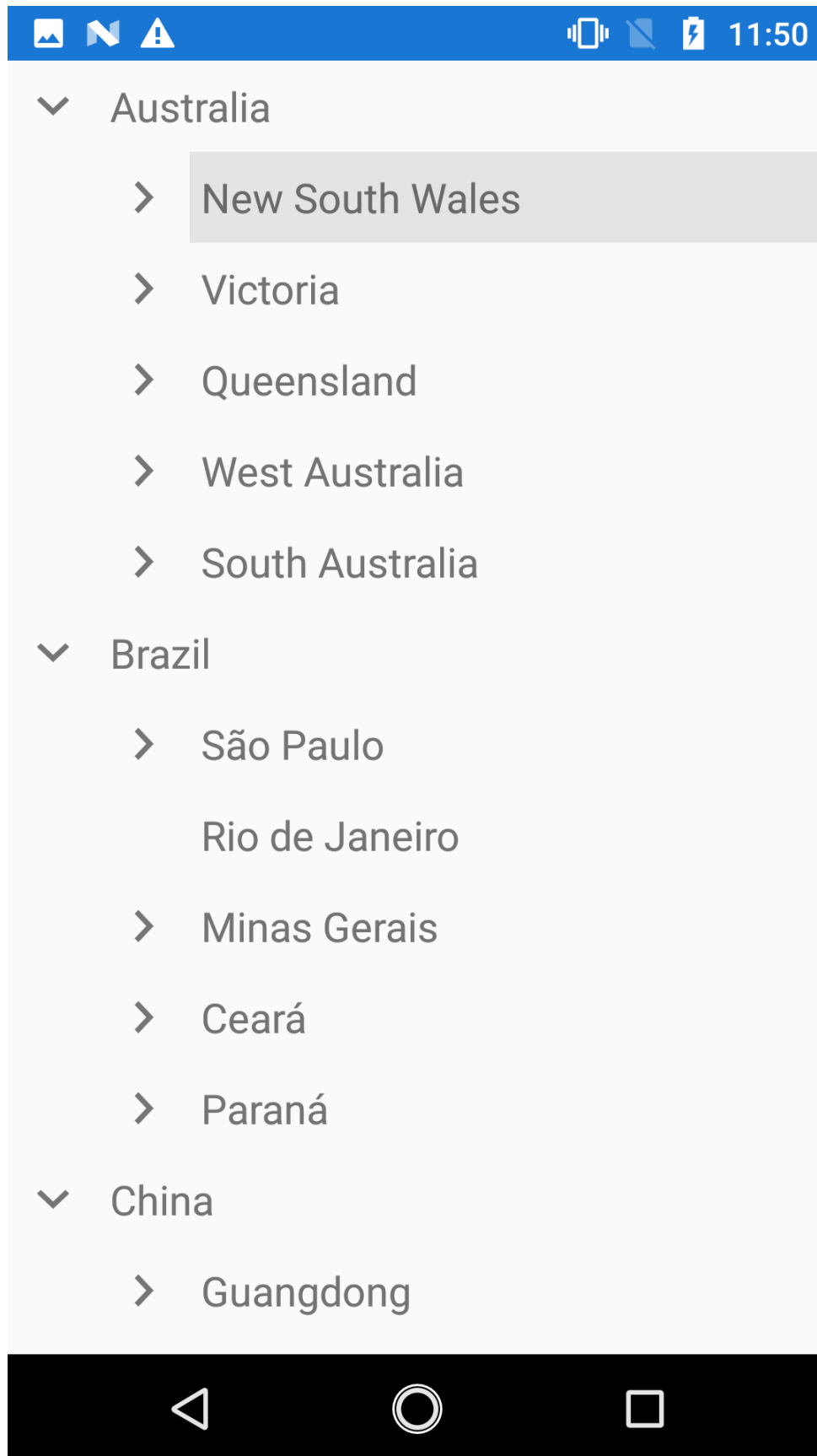
**C#**

```
treeView.SetBinding(SfTreeView.SelectedItemProperty, new
Binding("SelectedPlace", BindingMode.TwoWay));
```

**C#**

```
// CountriesViewModel.cs
public class CountriesViewModel
{
    public CountriesViewModel()
    {
        GenerateCountriesInfo();
    }
    public ObservableCollection<Countries> CountriesInfo { get; set; }
    public object SelectedPlace { get; set; }
    private void GenerateCountriesInfo()
    {
        var australia = new Countries() { Name = "Australia" };
        var _NSW = new Countries() { Name = "New South Wales" };
        var _Sydney = new Countries() { Name = "Sydney" };
        australia.States = new ObservableCollection<Countries>();
        australia.States.Add(_NSW);
        _NSW.States = new ObservableCollection<Countries>();
        _NSW.States.Add(_Sydney);
        var usa = new Countries() { Name = "United States of America" };
        var _California = new Countries() { Name = "California" };
```

```
var _LosAngeles = new Countries() { Name = "Los Angeles" };
usa.States = new ObservableCollection<Countries>();
usa.States.Add(_California);
_California.States = new ObservableCollection<Countries>();
_California.States.Add(_LosAngeles);
this.CountriesInfo = new ObservableCollection<Countries>();
CountriesInfo.Add(australia);
CountriesInfo.Add(usa);
SelectedPlace=_NSW
}
}
```



*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="treeView"
SelectionMode="Multiple"
SelectedItems="{Binding SelectedCountries}"
ChildPropertyName="States"
ItemsSource="{Binding CountriesInfo}"/>
```

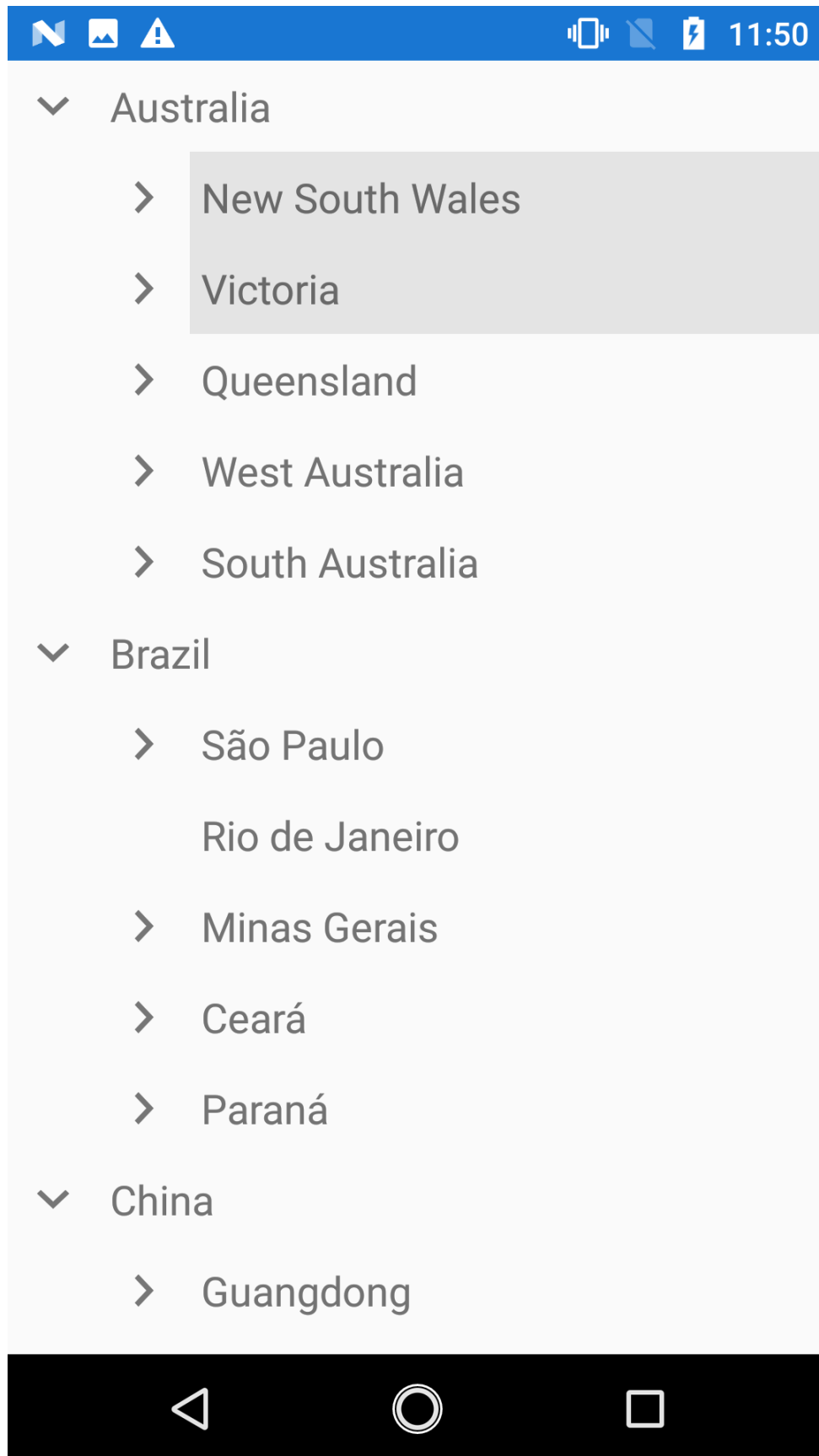
**C#**

```
treeView.SelectionMode = SelectionMode.Multiple;
treeView.SetBinding(SfTreeView.SelectedItemsProperty, new
Binding("SelectedCountries", BindingMode.TwoWay));
```

**C#**

```
// CountriesViewModel.cs
// CountriesViewModel.cs
public class CountriesViewModel
{
    public CountriesViewModel()
    {
        GenerateCountriesInfo();
    }
    public ObservableCollection<Countries> CountriesInfo { get; set; }
    public ObservableCollection<object> SelectedCountries { get; set; }
    private void GenerateCountriesInfo()
    {
        var australia = new Countries() { Name = "Australia" };
        var _NSW = new Countries() { Name = "New South Wales" };
        var _Sydney = new Countries() { Name = "Sydney" };
        var _Victoria = new Countries() { Name = "Victoria" };
        australia.States = new ObservableCollection<Countries>();
        australia.States.Add(_NSW);
        australia.States.Add(_Victoria);
        _NSW.States = new ObservableCollection<Countries>();
        _NSW.States.Add(_Sydney);
        var usa = new Countries() { Name = "United States of America" };
        var _California = new Countries() { Name = "California" };
        usa.States = new ObservableCollection<Countries>();
        usa.States.Add(_California);
        this.CountriesInfo = new ObservableCollection<Countries>();
        CountriesInfo.Add(australia);
        CountriesInfo.Add(usa);
        SelectedCountries = new ObservableCollection<object>();
        SelectedCountries.Add(_NSW);
        SelectedCountries.Add(_Victoria);
    }
}
```

You can download the entire sample [here](#).





## Commands

### *Tap command*

The [TapCommand](#) will be triggered whenever tapping the item and passing the [ItemTappedEventArgs](#) as parameter.

### **C#**

```
treeView.TapCommand = viewModel.TappedCommand;
public class CommandViewModel
{
    private Command<Object> tappedCommand;
    public Command<object> TappedCommand
    {
        get { return tappedCommand; }
        set { tappedCommand = value; }
    }
    public CommandViewModel()
    {
        TappedCommand = new Command<object>(TappedCommandMethod);
    }
    private void TappedCommandMethod(object obj)
    {
        App.Current.MainPage.DisplayAlert("Alert", (obj.AddedItems[0] as
        Countries).Name + " is Tapped", "OK");
    }
}
```

### *Hold command*

The [HoldCommand](#) will be triggered whenever an item is long pressed and passing the [ItemHoldingEventArgs](#) as parameter.

### **C#**

```
treeView.HoldCommand = viewModel.HoldingCommand;
public class CommandViewModel
{
    private Command<Object> holdingCommand;
    public Command<object> HoldingCommand
    {
        get { return holdingCommand; }
        set { holdingCommand = value; }
    }
    public CommandViewModel()
    {
        HoldingCommand = new Command<object>(HoldingCommandMethod);
    }
    private void HoldingCommandMethod(object obj)
    {
        App.Current.MainPage.DisplayAlert("Alert", (obj.AddedItems[0] as
        Countries).Name + " is Holding", "OK");
    }
}
```

### Expand command

The **ExpandCommand** will be triggered while expanding the node and passing the [TreeViewNode](#) as command parameter. TreeView expands the node based on the return value of **CanExecute** method implementation of **ExpandCommand**. If you return false, then expand action will be canceled. **Execute** method implementation of **ExpandCommand** will get called after expanding of node.

### XML

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Selection"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
x:Class="Selection.MainPage">
<ContentPage.BindingContext>
<local:CountriesViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView"
ExpandCommand="ExpandingCommand"/>
</ContentPage.Content>
</ContentPage>
```

### C#

```
/// <summary>
/// CommandViewModel class that implements
[Command] (https://docs.microsoft.com/en-
us/dotnet/api/xamarin.forms.command?view=xamarin-forms).
/// </summary>
public class CommandViewModel
{
    private Command<Object> expandingCommand;
    public Command<object> ExpandingCommand
    {
        get { return expandingCommand; }
        set { expandingCommand = value; }
    }
    public CommandViewModel()
    {
        ExpandingCommand = new Command<object>(ExpandCommandAction, CanExecute);
    }
    /// <summary>
    /// CanExecute method is called before expanding of node.
    /// </summary>
    /// <returns>Handle expand action by returning true or false. </returns>
    /// <param name="obj">TreeViewNode is passed as command parameter. </param>
    public bool CanExecute(object obj)
    {
        //You can also return false to cancel the expand action.
        return true;
    }
    /// <summary>
    /// Method gets called after expanding action performed.
    /// </summary>
```

```

/// <param name="obj">TreeNode is passed as command parameter. </param>
private void ExpandCommandAction(object obj)
{
    App.Current.MainPage.DisplayAlert("Alert", "TreeView node is expanded",
    "OK");
}
}

```

### Collapse command

The **CollapseCommand** will be triggered while collapsing the node and passing the **TreeNode** as command parameter. TreeView collapses the node based on the return value of **CanExecute** method implementation of **CollapseCommand**. If you return false, then collapse action will be canceled. **Execute** method implementation of **CollapseCommand** will get called after collapsing of node.

### XML

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:Selection"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
x:Class="Selection.MainPage">
<ContentPage.BindingContext>
<local:CountriesViewModel x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView"
CollapseCommand="CollapsingCommand"/>
</ContentPage.Content>
</ContentPage>

```

### C#

```

/// <summary>
/// CommandViewModel class that implements
[Command] (https://docs.microsoft.com/en-
us/dotnet/api/xamarin.forms.command?view=xamarin-forms).
/// </summary>
public class CommandViewModel
{
    private Command<object> collapsingCommand;
    public Command<object> CollapsingCommand
    {
        get { return collapsingCommand; }
        set { collapsingCommand = value; }
    }
    public CommandViewModel()
    {
        CollapsingCommand = new Command<object>(CollapseCommandAction, CanExecute);
    }
    /// <summary>
    /// CanExecute method is called before collapsing of node.
    /// </summary>
    /// <returns>Handle collapse action by returning true or false. </returns>
    /// <param name="obj">TreeNode is passed as command parameter. </param>

```

```

public bool CanExecute(object obj)
{
    //You can also return false to cancel the collapse action.
    return true;
}
/// <summary>
/// Method gets called after collapsing action performed.
/// </summary>
/// <param name="obj">TreeNode is passed as command parameter. </param>
private void CollapseCommandAction(object obj)
{
    App.Current.MainPage.DisplayAlert("Alert", "TreeView node is collapsed",
    "OK");
}
}

```

### 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="treeView"
SelectionMode="Multiple"
SelectedItems="{Binding SelectedCountries}"
ChildPropertyName="States"
ItemsSource="{Binding CountriesInfo}">
<syncfusion:SfTreeView.Behaviors>
<local:EventToCommandBehavior EventName="SelectionChanged" Command="{Binding
SelectionChangedCommand}"/>
</syncfusion:SfTreeView.Behaviors>
</syncfusion:SfTreeView>

```

### C#

```

public class CountriesViewModel : INotifyPropertyChanged
{
    public Command<ItemSelectedEventArgs> selectionChangedCommand;
    public CountriesViewModel()
    {
        SelectionChangedCommand = new
        Command<Syncfusion.XForms.TreeView.ItemSelectionChangedEventArgs>(OnSelectionChanged);
        GenerateCountriesInfo();
    }
    public ObservableCollection<Countries> CountriesInfo { get; set; }
    public ObservableCollection<object> SelectedCountries { get; set; }
    public Command<ItemSelectedEventArgs> SelectionChangedCommand
    {
        get { return selectionChangedCommand; }
        protected set { selectionChangedCommand = value; }
    }
    private void OnSelectionChanged(ItemSelectionChangedEventArgs obj)
    {

```

```

App.Current.MainPage.DisplayAlert("Alert", (obj.AddedItems[0] as
Countries).Name + " is selected", "OK");
}
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string name)
{
    if (this.PropertyChanged != null)
        this.PropertyChanged(this, new PropertyChangedEventArgs(name));
}
private void GenerateCountriesInfo()
{
    var australia = new Countries() { Name = "Australia" };
    var _NSW = new Countries() { Name = "New South Wales" };
    var _Victoria = new Countries() { Name = "Victoria" };
    australia.States = new ObservableCollection<Countries>();
    australia.States.Add(_NSW);
    australia.States.Add(_Victoria);
    var usa = new Countries() { Name = "United States of America" };
    var _California = new Countries() { Name = "California" };
    usa.States = new ObservableCollection<Countries>();
    usa.States.Add(_California);
    this.CountriesInfo = new ObservableCollection<Countries>();
    CountriesInfo.Add(australia);
    CountriesInfo.Add(usa);
}
}

```

You can download the example sample [here](#).

For more information regarding the event to command behavior in Xamarin.Forms, you can refer [this](#) link.

### Checkbox items binding in MVVM

SfTreeView support to check multiple items through binding the [CheckedItems](#) property from view model with `ObservableCollection<object>` type.

---

**Note:** TreeView process and sets [TreeViewNode.IsChecked](#) based on `CheckedItems` only when you are binding `ItemsSource`.

---

### XML

```

<syncfusion:SfTreeView
x:Name="TreeView"
CheckBoxMode="Recursive"
ItemsSource="{Binding Folders}"
CheckedItems="{Binding CheckedNodeInfo}"
ItemTemplateContextType="Node">
    <syncfusion:SfTreeView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <ViewCell.View>
                    <Grid Padding="5">
                        <SfCheckBox:SfCheckBox
x:Name="CheckBox"
Text="{Binding Content.FileName}"
IsChecked="{Binding IsChecked, Mode=TwoWay}"/>

```

```

</Grid>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</syncfusion:SfTreeView.ItemTemplate>
</syncfusion:SfTreeView>

```

## C#

```

public class ViewModel
{
    private ObservableCollection<object> checkedNodeInfo;
    public ObservableCollection<object> CheckedNodeInfo
    {
        get
        {
            return checkedNodeInfo;
        }
        set
        {
            this.checkedNodeInfo = value;
        }
    }
    public ViewModel()
    {
        var doc = new Folder() { FileName = "Documents" };
        checkedNodeInfo = new ObservableCollection<object>();
        checkedNodeInfo.Add(doc);
    }
}

```

You can download the entire source of this demo from [here](#). To know more about usage of checkbox, you can refer the documentation from [here](#).

## Load on demand

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

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:TreeView"
xmlns:sfTreeView="clr-namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
x:Class="TreeView.MainPage">
<ContentPage.BindingContext>
<local:MusicInfoRepository x:Name="viewModel"/>
</ContentPage.BindingContext>
<ContentPage.Content>
<sfTreeView:SfTreeView x:Name="treeView"
LoadOnDemandCommand="{Binding TreeViewOnDemandCommand}"
ItemsSource="{Binding Menu}" />

```

```
</ContentPage.Content>
</ContentPage>
```

## C#

```
/// <summary>
/// ViewModel class with <see cref="Command"/> for load on demand.
/// </summary>
public class MusicInfoRepository
{
    private ObservableCollection<MusicInfo> menu;
    public ObservableCollection<MusicInfo> Menu
    {
        get { return menu; }
        set { menu = value; }
    }
    public ICommand TreeViewOnDemandCommand
    {
        get; set;
    }
    public MusicInfoRepository()
    {
        this.Menu = this.GetMenuItems();
        TreeViewOnDemandCommand = new Command(ExecuteOnDemandLoading,
        CanExecuteOnDemandLoading);
    }
    /// <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 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;
MusicInfo musicInfo = node.Content as MusicInfo;
Device.BeginInvokeOnMainThread(async () =>
{
    await Task.Delay(2000);
    //Fetching child items to add
    var items = GetSubMenu(musicInfo.ID);
    // Populating child items for the node in on-demand
    node.PopulateChildNodes(items);
    if (items.Count() > 0)
        //Expand the node after child items are added.
        node.IsExpanded = true;
        //Stop the animation after load on demand is executed, if animation not
        stopped, it remains still after execution of load on demand.
        node.ShowExpanderAnimation = false;
    });
}
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 : INotifyPropertyChanged
{
    public string itemName;
    public int id;
    public bool hasChildNodes;
    public string ItemName
    {
        get { return itemName; }
        set
        {
            itemName = value;
            OnPropertyChanged("ItemName");
        }
    }
    public int ID
    {
        get { return id; }
        set
        {
            id = value;
            OnPropertyChanged("ID");
        }
    }
    public bool HasChildNodes
    {
        get { return hasChildNodes; }
        set
        {
            hasChildNodes = value;
            OnPropertyChanged("HasChildNodes");
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    private void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}

```

---

**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 of expander visibility of the node is handled.
/// </summary>

```

```

/// <param name="sender">TreeNode is passed as default parameter
</param>
/// <returns>Returns true, if the specified node has child items to load 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 TreeNode).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 [TreeNode.ShowExpanderAnimation](#) until the data fetched.
- Once data fetched, you can populate the child nodes by calling [TreeNode.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 [TreeNode.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 [TreeNode.ChildNodes](#) count.

### C#

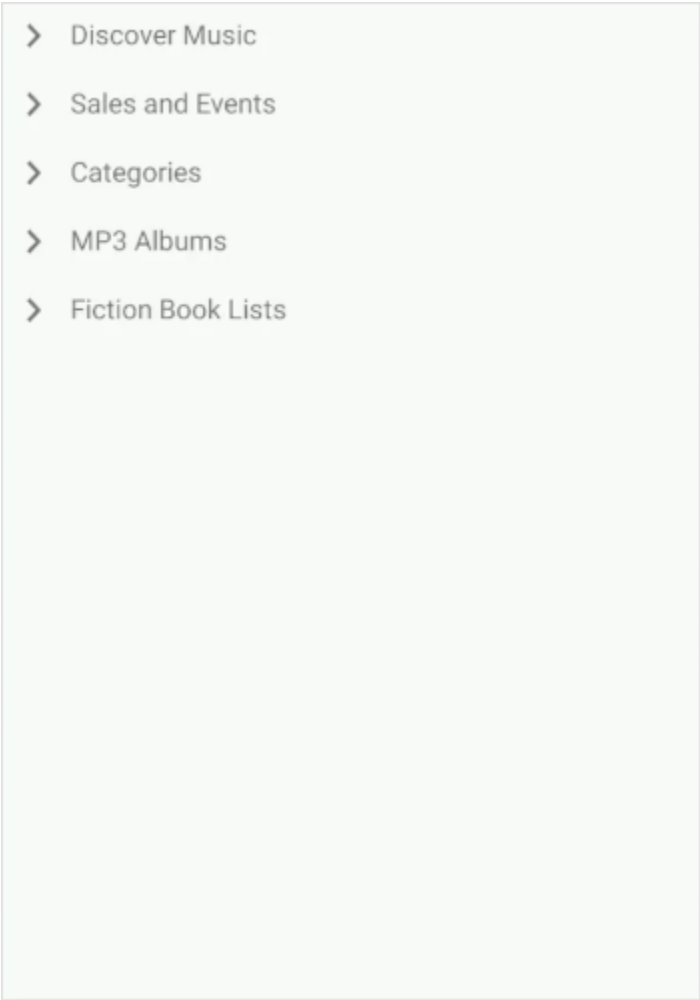
```

/// <summary>
/// Execute method is called when any item is requested for load-on-demand
items.
/// </summary>
/// <param name="obj">TreeNode is passed as default parameter </param>
private void ExecuteOnDemandLoading(object obj)
{
    var node = obj as TreeNode;
    // 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;
    MusicInfo musicInfo = node.Content as MusicInfo;
    Device.BeginInvokeOnMainThread(async () =>
    {
        await Task.Delay(2000);
    }
    )
}

```

```
//Fetching child items to add  
var items = GetSubMenu(musicInfo.ID);  
// Populating child items for the node in on-demand  
node.PopulateChildNodes(items);  
if (items.Count() > 0)  
//Expand the node after child items are added.  
node.IsExpanded = true;  
//Stop the animation after load on demand is executed, if animation not  
stopped, it remains still after execution of load on demand.  
node.ShowExpanderAnimation = false;  
});  
}
```

You can download the entire [source code](#) here.

- 
- > Discover Music
  - > Sales and Events
  - > Categories
  - > MP3 Albums
  - > Fiction Book Lists

### Item Height Customization

The TreeView provides various options to customize the height of items. To achieve this customization, please walkthrough 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 40. This property can be customized at runtime.

#### XML

```
<syncfusion:SfTreeView x:Name="treeView" ItemHeight="40">
```

#### C#

```
SfTreeView treeView = new SfTreeView();
treeView.ItemHeight = 40;
```

### 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.
- [GetActualNodeHeight](#): This method will return the measured height of the node item based on content loaded in it.

### 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

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage">
  <ContentPage.BindingContext>
    <local:FileManagerViewModel x:Name="viewModel"></local:FileManagerViewModel>
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <syncfusion:SfTreeView x:Name="treeView"
      QueryNodeSize="TreeView_QueryNodeSize"
      ChildPropertyName="SubFiles"
      ItemsSource="{Binding ImageNodeInfo}"/>
    </syncfusion:SfTreeView>
  </ContentPage.Content>
</ContentPage>
```

#### C#

```
public class MainPage : ContentPage
```

```

{
public MainPage()
{
InitializeComponent();
this.treeView.QueryNodeSize += TreeView_QueryNodeSize;
}
private void TreeView_QueryNodeSize(object sender, QueryNodeSizeEventArgs e)
{
if (e.Node.Level == 0)
{
//Returns speified item height for items.
e.Height = 200;
e.Handled = true;
}
}
}

```

#### Customize specific item height based on the content size

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.GetActualNodeHeight](#) method.

#### XML

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage">
<ContentPage.BindingContext>
<local:FileManagerViewModel x:Name="viewModel"></local:FileManagerViewModel>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView"
QueryNodeSize="TreeView_QueryNodeSize"
ChildPropertyName="SubFiles"
ItemsSource="{Binding ImageNodeInfo}"/>
</syncfusion:SfTreeView>
</ContentPage.Content>
</ContentPage>

```

#### C#

```

public class MainPage : ContentPage
{
public MainPage()
{
InitializeComponent();
this.treeView.QueryNodeSize += TreeView_QueryNodeSize;
}
private void TreeView_QueryNodeSize(object sender, QueryNodeSizeEventArgs e)
{
if (e.Node.Level != 0)
{
// Returns item height based on the content loaded.

```

```
e.Height = e.GetActualNodeHeight();
e.Handled = true;
}
}
}
```

### Autofit item height on dynamic changes

The Treeview supports autofit the item based on dynamic change in item size. It is enabled by setting `NodeSizeMode` property to `NodeSizeMode.Dynamic`. The default value is `NodeSizeMode.None`.

- **None:** The item height does not autofit for dynamic changes.
- **Dynamic:** The item height responds for dynamic changes and the size is recalculated.

### XML

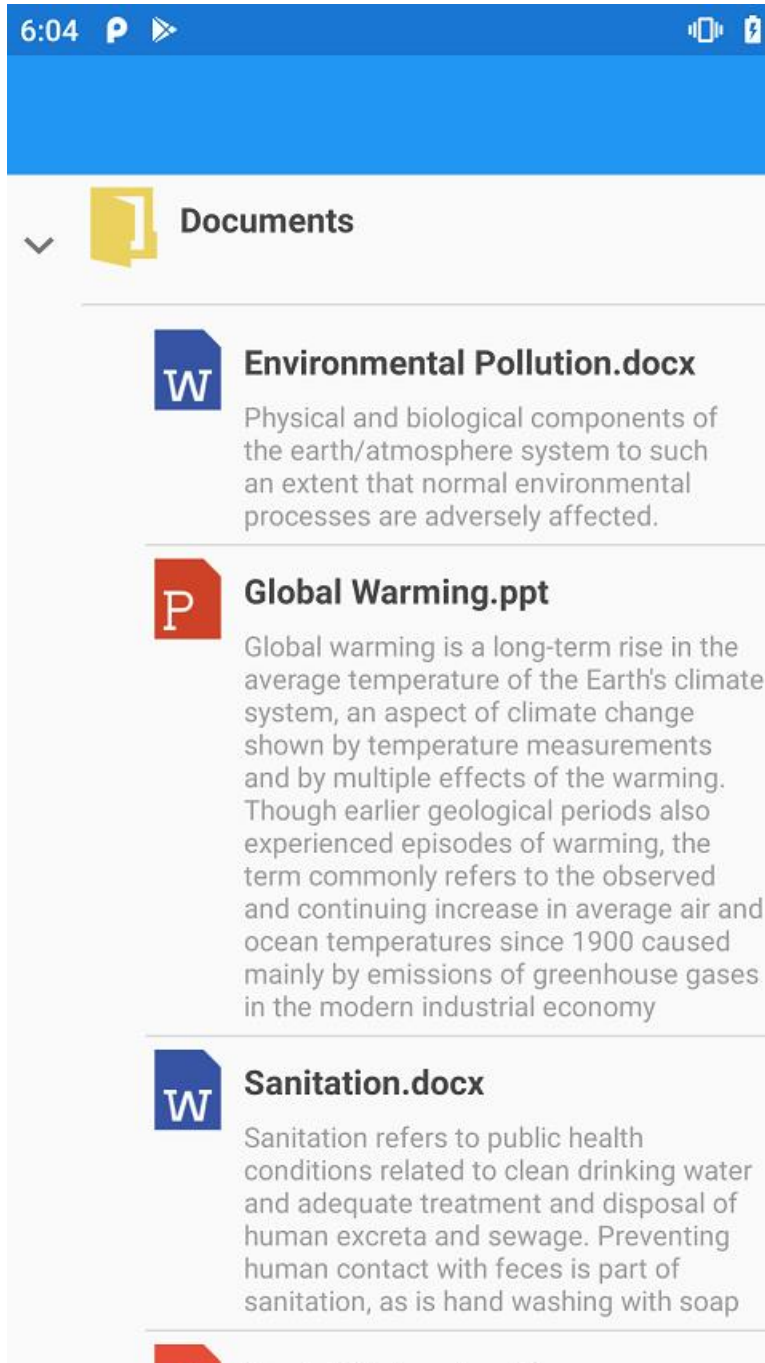
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms"
xmlns:local="clr-namespace:GettingStarted"
x:Class="GettingStarted.MainPage">
<ContentPage.BindingContext>
<local:FileManagerViewModel x:Name="viewModel"></local:FileManagerViewModel>
</ContentPage.BindingContext>
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView"
QueryNodeSize="TreeView_QueryNodeSize"
NodeSizeMode="Dynamic"
ChildPropertyName="SubFiles"
ItemsSource="{Binding ImageNodeInfo}"/>
</syncfusion:SfTreeView>
</ContentPage.Content>
</ContentPage>
```

### C#

```
public class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        this.treeView.QueryNodeSize += TreeView_QueryNodeSize;
        this.treeView.NodeSizeMode = NodeSizeMode.Dynamic;
    }
    private void TreeView_QueryNodeSize(object sender, QueryNodeSizeEventArgs e)
    {
        if (e.Node.Level == 0)
        {
            //Returns speified item height for items.
            e.Height = 200;
            e.Handled = true;
        }
        else
        {
            // Returns item height based on the content loaded.
        }
    }
}
```

```
e.Height = e.GetActualNodeHeight();  
e.Handled = true;  
}  
}  
}
```

You can download the entire source code of this demo [here](#).





## 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.

## Right to left(RTL)

TreeView supports to change the flow of text to the right-to-left direction by setting the [FlowDirection](#) to [RightToLeft](#). TreeView supports RTL in Xamarin.Forms version 3.0 and above.

**Note:** Specific platform setup is required to enable right-to-left localization. For platform settings you can refer [here](#).

### XML

```
<ContentPage xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms">
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView" FlowDirection="RightToLeft"/>
</ContentPage.Content>
</ContentPage>
```

### C#

```
treeView.FlowDirection = FlowDirection.RightToLeft;
```

TreeView also supports RTL when device's flow direction is changed, it is achieved by setting the [FlowDirection](#) to [Device.FlowDirection](#).

### XML

```
<ContentPage xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:syncfusion="clr-
namespace:Syncfusion.XForms.TreeView;assembly=Syncfusion.SfTreeView.XForms">
<ContentPage.Content>
<syncfusion:SfTreeView x:Name="treeView" FlowDirection="{x:Static
Device.FlowDirection}"/>
</ContentPage.Content>
</ContentPage>
```

### C#

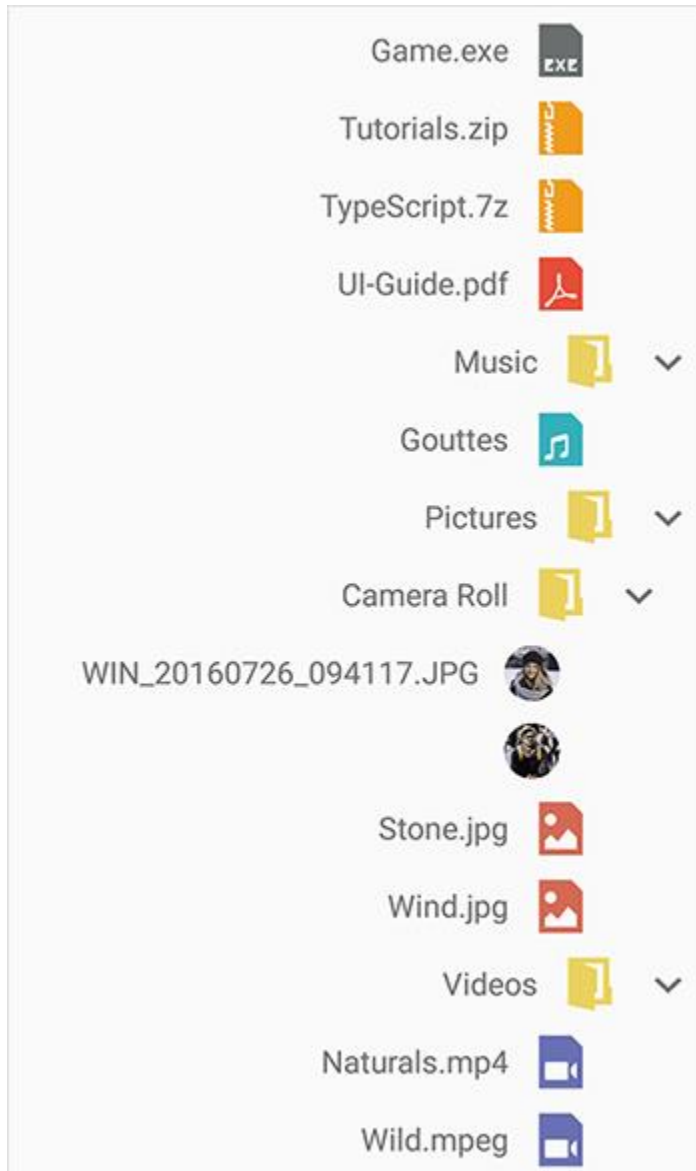
```
treeView.FlowDirection = Device.FlowDirection;
```

In UWP platform, the ScrollView is not changed when RTL is enabled (framework issue). To overcome this issue, set the [FlowDirection](#) property in constructor of [MainPage](#) in UWP renderer as demonstrated in the following code example.

### C#

```
public MainPage ()
{
...
}
```

```
SfTreeViewRenderer.Init();  
this.FlowDirection = FlowDirection.RightToLeft;  
LoadApplication (new App ());  
...  
}
```



You can download the entire [source code](#) here.

**Note:** When a label is loaded in the `ItemTemplate`, the right-to-left direction is not applied due to the framework issue. It has been reported to the Xamarin team; for more details about this, refer to this [link](#). To overcome this issue, set the `HorizontalOptions` to `StartAndExpand` in Label.

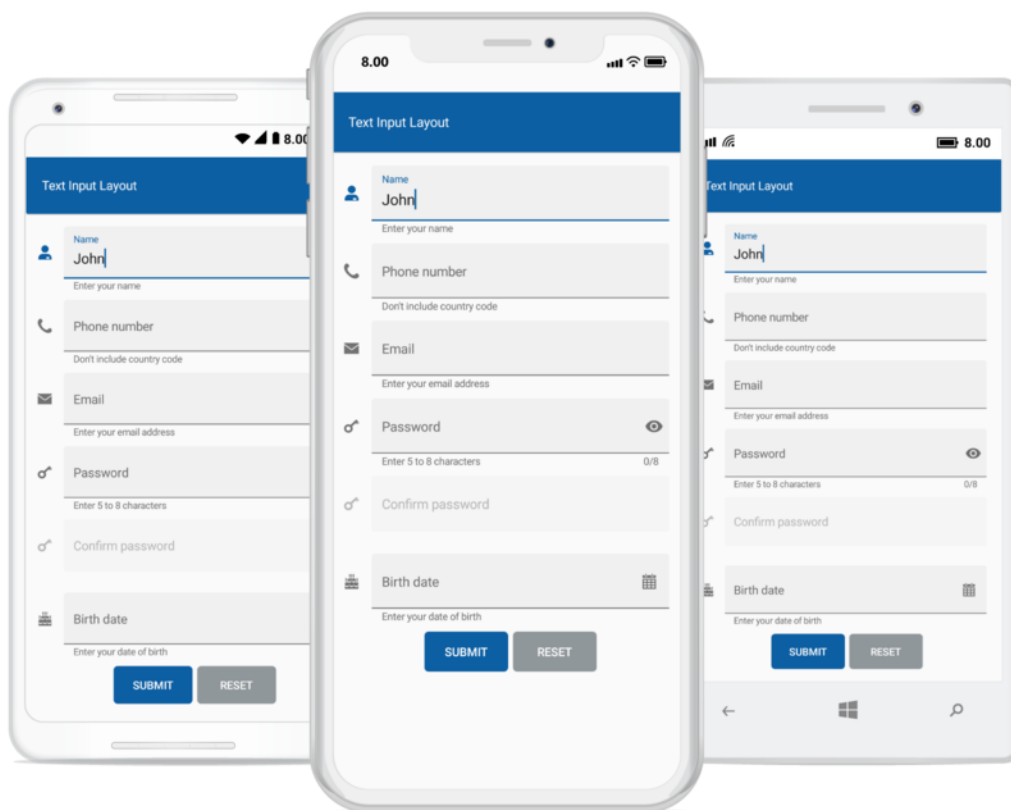
## Limitations

- When you use custom [ItemTemplate](#) in [TreeView](#), it does not respond to [FlowDirection](#) due to framework issue. To respond to [FlowDirection](#) changes, you need to set [FlowDirection](#) of [TreeView](#) to parent view of your custom [ItemTemplate](#).

## SfTextInputLayout

### Overview

The text input layout control for Xamarin.Forms adds decorative elements such as floating labels, icons, and assistive labels on the top of input views such as [SfMaskedEdit](#), [SfNumericTextBox](#), [Entry](#), and [Editor](#) controls.



### Key features

- Displays floating labels when the input view is focused.
- Displays error labels.
- Supports filled, outlined, and none container types.
- Provides RTL support.
- Provides options to reserve space for assistive labels.
- Provides options to customize the thickness of stroke width and corner radius.
- Provides options to customize padding around the input view.
- Provides options to customize the font family, font size, and font attributes for hint, error, helper text, and counter label.

- Displays leading and trailing icons.
- Displays help labels.
- Displays maximum character count.
- Toggles password visibility.

## Getting Started

This section explains the steps required to configure the text input layout control with floating label.

### Adding SfTextInputLayout reference

You can add SfTextInputLayout reference using one of the following methods:

#### Method 1: Adding SfTextInputLayout reference from nuget.org

Syncfusion Xamarin components are available in [nuget.org](https://nuget.org). To add SfTextInputLayout to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.Xamarin.Core](https://nuget.org/packages/Syncfusion.Xamarin.Core), and then install it.

![Add Packages](Getting-Started\_images/Adding SfTextInputLayout reference.png)

---

**Note:** Install the same version of SfTextInputLayout NuGet in all the projects.

---

#### Method 2: Adding SfTextInputLayout reference from toolbox

Syncfusion also provides Xamarin Toolbox. Using this toolbox, you can drag the SfTextInputLayout control to the XAML page. It will automatically install the required NuGet packages and add the namespace to the page. To install Syncfusion Xamarin Toolbox, refer to [Toolbox](#).

#### Method 3: Adding SfTextInputLayout assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location : {Installed location}/{version}/Xamarin/lib

PCL	Syncfusion.Core.XForms.dll Syncfusion.Licensing.dll
Android	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.Android.dll Syncfusion.Licensing.dll
iOS	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.iOS.dll Syncfusion.Licensing.dll
UWP	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.UWP.dll Syncfusion.Licensing.dll
WPF	Syncfusion.Core.XForms.dll Syncfusion.Core.XForms.WPF.dll Syncfusion.Licensing.dll

---

**Note:** To know more about obtaining our components, refer to these links for [Mac](#) and [Windows](#).

---

---

**Information:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from the trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to [Syncfusion license key](#) to know about registering Syncfusion license key in your Xamarin application to use our components.

---

### Launching an application on each platform with text input layout

To use the text input layout inside an application, each platform application requires some additional configurations. The configurations vary from platform to platform and are discussed in the following sections:

---

**Note:** If you are adding the references from toolbox, below steps are not needed.

---

#### iOS

To launch the text input layout in iOS, call the `SfTextInputLayoutRenderer.Init()` method in the `FinishedLaunching` overridden method of the `AppDelegate` class after the `Xamarin.Forms` framework has been initialized and before the `LoadApplication` method is called as demonstrated in the following code sample.

#### C#

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();
    SfTextInputLayoutRenderer.Init();
    LoadApplication(new App());
    return base.FinishedLaunching(app, options);
}
```

#### Universal Windows Platform (UWP)

To deploy the text input layout in `Release` mode, initialize the core assemblies in the `App.xaml.cs` file in the UWP project as demonstrated in the following code samples.

#### C#

```
// In App.xaml.cs
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    if (rootFrame == null)
    {
        List<Assembly> assembliesToInclude = new List<Assembly>();
        assembliesToInclude.Add(typeof(SfTextInputLayoutRenderer).GetTypeInfo().Assembly);
        Xamarin.Forms.Forms.Init(e, assembliesToInclude);
    }
    ...
}
```

#### Android

Android platform does not require any additional configuration to render the text input layout control.

### Initializing text input layout

Import the [SfTextInputLayout](#) control namespace in respective page as demonstrated in the following code sample.

#### XML

```
xmlns:inputLayout="clr-namespace:Syncfusion.XForms.TextInputLayout;assembly=Syncfusion.Core.XForms"
```

#### C#

```
using Syncfusion.XForms.TextInputLayout;
```

Add any input view control such as [Entry](#), [Editor](#), [SfNumericTextBox](#), or [SfMaskedEdit](#), and add hint label (floating label).

#### XML

```
<inputLayout:SfTextInputLayout>  
<Entry />  
</inputLayout:SfTextInputLayout>
```

#### C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.InputView = new Entry();
```

### Adding hint

Floating label for the text input layout can be added by setting the [Hint](#) property. Visibility of the hint can be collapsed by setting the [ShowHint](#) property to `false`. By default, this property is set to `true`.

#### XML

```
<inputLayout:SfTextInputLayout  
Hint="Name">  
<Entry />  
</inputLayout:SfTextInputLayout>
```

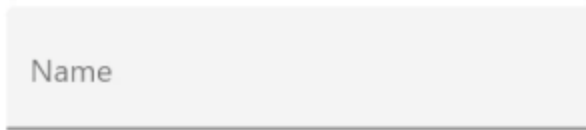
#### C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Name";  
inputLayout.InputView = new Entry();
```

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.

The default translate animation for the hint can be disabled by setting the [EnableHintAnimation](#) property to `false`. Instead translate animation in hint, you can use the alpha animation by setting the [EnableFloating](#) property to `false`.

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.



### Enabling password visibility toggle

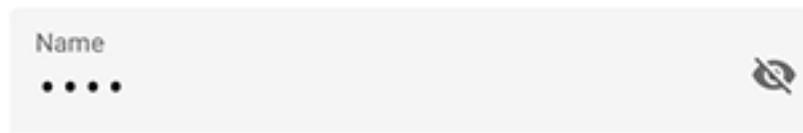
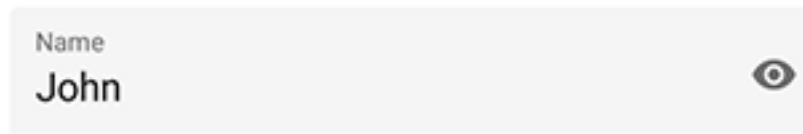
The password visibility toggle is used to show or hide the visibility of characters in the input view added to the control. You can enable this toggle by setting the [EnablePasswordVisibilityToggle](#) property to true.

#### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
EnablePasswordVisibilityToggle="true">
<Entry Text="John" />
</inputLayout:SfTextInputLayout>
```

#### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.EnablePasswordVisibilityToggle = true;
inputLayout.InputView = new Entry() { Text = "John" };
```



---

**Note:** Password visibility toggle can be enabled only for [Entry](#) control.

---

You can find the complete getting started sample from this [link](#).

### Supported input views

Input views can be added to the text input layout control by setting the [InputView](#) property. To reduce the XAML syntax, [InputView](#) property is applied with [ContentPropertyAttribute](#).

## Entry

To enter a single line text input, add [Entry](#).

### XML

```
<inputLayout:SfTextInputLayout  
Hint="Name"  
HelperText="Enter your name">  
<Entry />  
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Name";  
inputLayout.HelperText = "Enter your name";  
inputLayout.InputView = new Entry();
```



## Editor

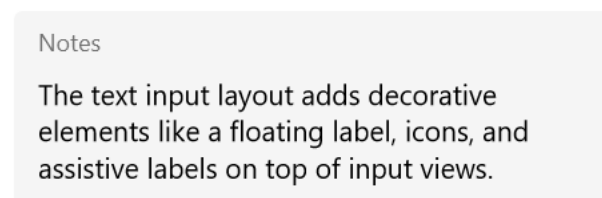
To enter multi-line text input, add [Editor](#), and then set the `AutoSize` property to `TextChanges`.

### XML

```
<inputLayout:SfTextInputLayout  
Hint="Notes">  
<Editor AutoSize="TextChanges" />  
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Notes";  
inputLayout.InputView = new Editor();
```



## Masked edit

To initialize the masked edit control and launch it in each platform, refer to the [getting started with masked edit](#) documentation.

### XML



```
<inputLayout:SfTextInputLayout
Hint="Card number"
HelperText="Required *">
<maskededit:SfMaskedEdit
Keyboard="Numeric"
Mask="0000 0000 0000 0000" />
</inputLayout:SfTextInputLayout>
```

## C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Card number";
inputLayout.HelperText = "Required *"
inputLayout.InputView = new SfMaskedEdit() { Keyboard = Keyboard.Numeric,
Mask = "0000 0000 0000 0000" };
```

Card number

6246 56\_\_ \_\_ \_\_

Required \*

## Numeric text box

To initialize the numeric text box control and launch it in each platform, refer to the [getting started with numeric text box](#) documentation.

## XML

```
<inputLayout:SfTextInputLayout
Hint="Amount"
HelperText="Required *">
<numericBox:SfNumericTextBox
Value="123.45"
FormatString="c" />
</inputLayout:SfTextInputLayout>
```

## C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Amount";
inputLayout.HelperText = "Required *"
inputLayout.InputView = new SfNumericTextBox() { Value = 123.45,
FormatString="c" };
```

Amount

\$2,904.50

Required \*

## Autocomplete

To initialize the Autocomplete control and launch it in each platform, refer to the [getting started with auto complete](#) documentation.

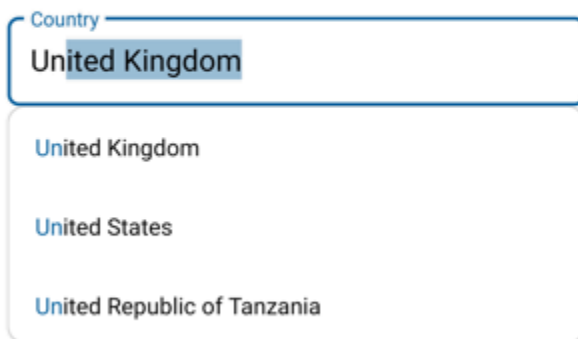
### XML

```
<inputLayout:SfTextInputLayout Hint="Country">
  <autocomplete:SfAutoComplete AutoCompleteMode="SuggestAppend">
    <autocomplete:SfAutoComplete.AutoCompleteSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> United Kingdom </x:String>
        <x:String> United States </x:String>
        <x:String> United Republic of Tanzania </x:String>
      </ListCollection:List>
    </autocomplete:SfAutoComplete.AutoCompleteSource>
  </autocomplete:SfAutoComplete>
</inputLayout:SfTextInputLayout>
```

### C#

```
var autoComplete = new SfAutoComplete();
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Country";
List<String> countryNames = new List<String>();
countryNames.Add("United Kingdom");
countryNames.Add("United States");
countryNames.Add("United Republic of Tanzania");
autoComplete.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
autoComplete.DataSource = countryNames;
inputLayout.InputView = autoComplete;
```

Autocomplete



## Combo box

To initialize the ComboBox control and launch it in each platform, refer to the [getting started with combo box](#) documentation.

## XML

```
<inputLayout:SfTextInputLayout Hint="Country">
  <combobox:SfComboBox>
    <combobox:SfComboBox.ComboBoxSource>
      <ListCollection:List x:TypeArguments="x:String">
        <x:String> Afghanistan </x:String>
        <x:String> Akrotiri </x:String>
        <x:String> Albania </x:String>
      </ListCollection:List>
    </combobox:SfComboBox.ComboBoxSource>
  </combobox:SfComboBox>
</inputLayout:SfTextInputLayout>
```

## C#

```
var combobox = new SfComboBox();
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Country";
List<String> countryNames = new List<String>();
countryNames.Add("Afghanistan");
countryNames.Add("Akrotiri");
countryNames.Add("Albania");
combobox.DataSource = countryNames;
inputLayout.InputView = combobox;
```



**Note:** Entry and Editor are the only input views supported by [SfTextInputLayout](#) in WPF platform.

## Container type

Containers improve the discoverability of input view by creating a contrast between the input view and assistive elements.

### Filled

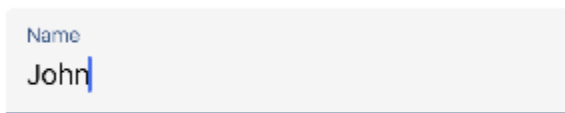
The background of the input view will be filled with container color, and its stroke (at the bottom edge) color and thickness will be changed based on its states. It can be enabled by setting the [ContainerType](#) property to [Filled](#).

#### XML

```
<inputLayout:SfTextInputLayout  
Hint="Name"  
ContainerType="Filled">  
<Entry Text="John" />  
</inputLayout:SfTextInputLayout>
```

#### C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Name";  
inputLayout.ContainerType = ContainerType.Filled;  
inputLayout.InputView = new Entry() { Text = "John" };
```



### Outlined

When setting the [ContainerType](#) property to [Outlined](#), the container will be covered with a rounded-corner border.

#### XML

```
<inputLayout:SfTextInputLayout  
Hint="Name"  
ContainerType="Outlined">  
<Entry Text="John" />  
</inputLayout:SfTextInputLayout>
```

#### C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Name";  
inputLayout.ContainerType = ContainerType.Outlined;  
inputLayout.InputView = new Entry() { Text = "John" };
```



#### *Customize the corner radius of the outline border*

When setting the [OutlineCornerRadius](#) property to double value, the corner radius of the container will be changed.

**XML**

```
<inputLayout:SfTextInputLayout
Hint="Name"
ContainerType="Outlined"
OutlineCornerRadius="8">
<Entry />
</inputLayout:SfTextInputLayout>
```

**C#**

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.OutlineCornerRadius = 8;
inputLayout.InputView = new Entry();
```

**NOTE**

It is applicable for the outline border when setting the container type to outlined.

*Custom Padding*

Spaces around the input view can be customized by setting the InputViewPadding property to [thickness](#) value.

**XML**

```
<inputLayout:SfTextInputLayout
Hint="Name"
InputViewPadding="5"
ContainerType="Outlined"
HelperText="Enter your name">
<Entry />
</inputLayout:SfTextInputLayout>
```

**C#**

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.InputViewPadding = new Thickness(5);
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.HelperText = "Enter your name";
inputLayout.InputView = new Entry();
```



## None

When setting the [ContainerType](#) property to [None](#), the container will have empty background and enough spacing.

### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
ContainerType="None">
<Entry Text="John" />
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.None;
inputLayout.InputView = new Entry() { Text = "John" };
```



## Adding custom icons

Any custom icons can be added to the leading edge or the trailing edge of input view in the text input layout control. The events and commands related to the custom icons should be handled at the application level.

Unicode or font icons for the labels can be displayed as icons.

**Note:** Refer to the following links to learn more about font icons:

- [How to create font icons using our metro studio and export as ttf?](#)
- [How to set font family for the custom fonts in labels?](#)

## Leading view

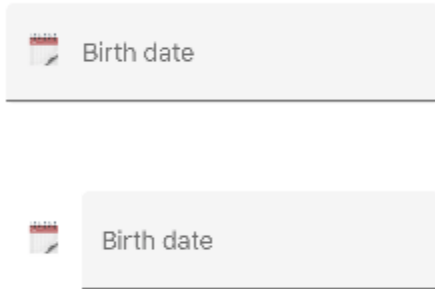
A label can be added as a leading icon for the input view by setting the [LeadingView](#) property. It can be positioned either inside or outside the container by setting the [LeadingViewPosition](#) property. By default, it is positioned [Outside](#).

### XML

```
<inputLayout:SfTextInputLayout
Hint="Birth date"
LeadingViewPosition="Inside" >
<Entry />
<inputLayout:SfTextInputLayout.LleadingView>
<Label
Text="&#x1F5D3;">
</Label>
</inputLayout:SfTextInputLayout.LleadingView>
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Birth date";
inputLayout.LeadingViewPosition = ViewPosition.Inside;
inputLayout.LeadingView = new Label() { Text = "\U0001F5D3" };
inputLayout.InputView = new Entry();
```



### Trailing view

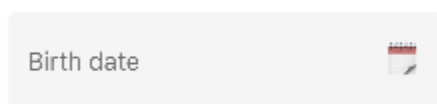
A label can be added as a trailing icon for the input view by setting the [TrailingView](#) property. It can be positioned either inside or outside the container of input view by setting the [TrailingViewPosition](#) property. By default, it is positioned [Inside](#).

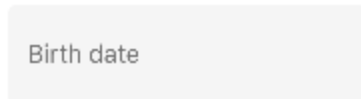
### XML

```
<inputLayout:SfTextInputLayout
Hint="Birth date"
TrailingViewPosition="Outside">
<Entry />
<inputLayout:SfTextInputLayout.TrailingView>
<Label
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 Entry();
```





## Font Customization

You can customize the appearance (size, attributes, and family) of font by setting the `FontFamily`, `FontSize`, and `FontAttributes` properties of the `LabelStyle` property.

Refer to this [documentation](#) to configure the custom fonts in `Xamarin.Forms`.

## Hint

You can customize the font of [hint](#) label by setting the `FontFamily`, `FontSize`, and `FontAttributes` properties of `HintLabelStyle` in `SfTextInputLayout`.

## XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
ContainerType="Outlined"
CharMaxLength="3"
ShowCharCount="True"
HelperText="Enter your name">
<Entry />
<inputLayout:SfTextInputLayout.HintLabelStyle>
<inputLayout:LabelStyle FontSize="16">
<inputLayout:LabelStyle.FontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf#Lobster-Regular"
WinPhone="Assets/Fonts/Lobster-Regular.ttf#Lobster" />
</inputLayout:LabelStyle.FontFamily>
</inputLayout:LabelStyle>
</inputLayout:SfTextInputLayout.HintLabelStyle>
</inputLayout:SfTextInputLayout>
```

## C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.HelperText = "Enter your name";
inputLayout.CharMaxLength = 3;
inputLayout.ShowCharCount = true;
inputLayout.HintLabelStyle = new LabelStyle() { FontFamily =
Device.OnPlatform("Lobster-Regular", "Lobster-Regular.ttf#Lobster-Regular",
"Assets/Fonts/Lobster-Regular.ttf#Lobster"), FontSize = 16};
inputLayout.InputView = new Entry();
```





### Helper text

You can customize the font of [helper text](#) label by setting the FontFamily, FontSize, and FontAttributes properties of HelperLabelStyle in SfTextInputLayout.

#### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
ContainerType="Outlined"
CharMaxLength="3"
ShowCharCount="True"
HelperText="Enter your name">
<Entry />
<inputLayout:SfTextInputLayout.HelperLabelStyle>
<inputLayout:LabelStyle FontSize="12">
<inputLayout:LabelStyle.FontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf#Lobster-Regular"
WinPhone="Assets/Fonts/Lobster-Regular.ttf#Lobster" />
</inputLayout:LabelStyle.FontFamily>
</inputLayout:LabelStyle>
</inputLayout:SfTextInputLayout.HelperLabelStyle>
</inputLayout:SfTextInputLayout>
```

#### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.HelperText = "Enter your name";
inputLayout.CharMaxLength = 3;
inputLayout.ShowCharCount = true;
inputLayout.HelperLabelStyle = new LabelStyle() { FontFamily =
Device.OnPlatform("Lobster-Regular", "Lobster-Regular.ttf#Lobster-Regular",
"Assets/Fonts/Lobster-Regular.ttf#Lobster"), FontSize = 12};
inputLayout.InputView = new Entry();
```



### Error text

You can customize the font of [error text](#) label by setting the FontFamily, FontSize, and FontAttributes properties of ErrorLabelStyle in SfTextInputLayout.

#### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
ContainerType="Outlined"
CharMaxLength="3"
ShowCharCount="True"
HelperText="Enter your name"
HasError="True">
```

```

ErrorText="Enter valid name">
<Entry />
<inputLayout:SfTextInputLayout.ErrorLabelStyle>
<inputLayout:LabelStyle FontSize="12">
<inputLayout:LabelStyle.FontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf#Lobster-Regular"
WinPhone="Assets/Fonts/Lobster-Regular.ttf#Lobster" />
</inputLayout:LabelStyle.FontFamily>
</inputLayout:LabelStyle>
</inputLayout:SfTextInputLayout.ErrorLabelStyle>
</inputLayout:SfTextInputLayout>

```

## C#

```

var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.HelperText = "Enter your name";
inputLayout.CharMaxLength = 3;
inputLayout.ShowCharCount = true;
inputLayout.HasError = true;
inputLayout.ErrorText = "Enter valid name";
inputLayout.ErrorLabelStyle = new LabelStyle() { FontFamily =
Device.OnPlatform("Lobster-Regular", "Lobster-Regular.ttf#Lobster-Regular",
"Assets/Fonts/Lobster-Regular.ttf#Lobster"), FontSize = 12};
inputLayout.InputView = new Entry();

```



## Counter label

You can customize the font of [counter label](#) by setting the FontFamily, FontSize, and FontAttributes properties of CounterLabelStyle in SfTextInputLayout.

## XML

```

<inputLayout:SfTextInputLayout
Hint="Name"
ContainerType="Outlined"
CharMaxLength="3"
ShowCharCount="True"
HelperText="Enter your name">
<Entry />
<inputLayout:SfTextInputLayout.CounterLabelStyle>
<inputLayout:LabelStyle FontSize="12">
<inputLayout:LabelStyle.FontFamily>
<OnPlatform x:TypeArguments="x:String" iOS="Lobster-Regular"
Android="Lobster-Regular.ttf#Lobster-Regular"
WinPhone="Assets/Fonts/Lobster-Regular.ttf#Lobster" />
</inputLayout:LabelStyle.FontFamily>
</inputLayout:LabelStyle>
</inputLayout:SfTextInputLayout.CounterLabelStyle>

```

```
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.HelperText = "Enter your name";
inputLayout.CharMaxLength = 3;
inputLayout.ShowCharCount = true;
inputLayout.CounterLabelStyle = new LabelStyle() { FontFamily =
Device.OnPlatform("Lobster-Regular", "Lobster-Regular.ttf#Lobster-Regular",
"Assets/Fonts/Lobster-Regular.ttf#Lobster"), FontSize = 12};
inputLayout.InputView = new Entry();
```



### Adding assistive labels

Assistive labels provide additional information about text entered in the input view controls.

#### Helper text

Helper text conveys additional guidance about the input field such as how it will be used. It can be set using the [HelperText](#) property.

### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
HelperText="Enter your name">
<Entry />
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.HelperText = "Enter your name";
inputLayout.InputView = new Entry();
```

The visibility of the helper text can be disabled by setting the [ShowHelperText](#) property to false. By default, it is set to true.



### Error message

When the text input is not accepted, an error message will display instructions to fix it. Error messages will be displayed below the input line till entering the correct text. It can be set using the [ErrorText](#) property, but it will be displayed only when the [HasError](#) property is set to `true`.

#### XML

```
<inputLayout:SfTextInputLayout
Hint="Email"
HelperText="Enter your email address"
ErrorText="Invalid email"
HasError="true">
<Entry />
</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 Entry();
```



**Note:** Error validations should be done in the application level.

### Character counter

Character counter is used when you need to limit the characters. Character limit can be set using the [CharMaxLength](#) property. The character counter can be enabled by setting the [ShowCharCount](#) property to `true`.

#### XML

```
<inputLayout:SfTextInputLayout
Hint="Password"
ShowCharCount="true"
CharMaxLength="8"
HelperText="Enter 5 to 8 characters">
<Entry />
</inputLayout:SfTextInputLayout>
```

#### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Password";
inputLayout.CharMaxLength = 8;
```

```
inputLayout.ShowCharCount = true;  
inputLayout.HelperText = "Enter 5 to 8 characters";  
inputLayout.InputView = new Entry();
```



**Note:** When character count reaches the maximum character length, the error color will be applied to hint, border, and counter label.

#### Reserve spaces for assistive labels

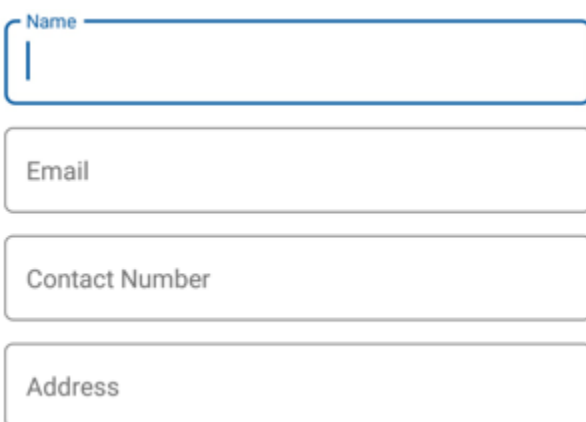
The reserved spaces for assistive labels can be removed by setting the [ReserveSpaceForAssistiveLabels](#) property to false.

#### XML

```
<inputLayout:SfTextInputLayout  
ContainerType="Outlined"  
Hint="Name"  
ReserveSpaceForAssistiveLabels="False">  
<Entry />  
</inputLayout:SfTextInputLayout>
```

#### C#

```
var inputLayout = new SfTextInputLayout();  
inputLayout.Hint = "Name";  
inputLayout.ContainerType = ContainerType.Outlined;  
inputLayout.ReserveSpaceForAssistiveLabels = false;  
inputLayout.InputView = new Entry();
```



## States and Colors

Based on the states, the colors will be applied to the hint labels and borders. So, when the input view is in focused state, the focused color will be applied; it is similar to other states also. The current hint color or active color can be obtained from the [CurrentActiveColor](#) property.

**Note:** Since error is not a state, the error color will not be set to [CurrentActiveColor](#) when [HasError](#) property is set to true.

### Focused color

When the input view is focused, the [FocusedColor](#) property value will be applied to the hint label and border.

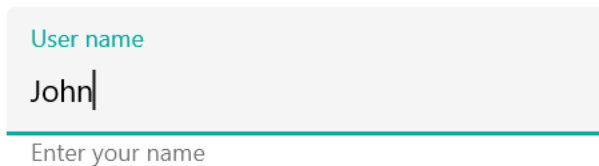
**Information:** Cursor color of the input view will be same as the [Accent](#) color of the application in each platform.

### XML

```
<inputLayout:SfTextInputLayout
Hint="User name"
FocusedColor="#00AFA0"
HelperText="Enter your name"
<Entry Text="John" />
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "User name";
inputLayout.FocusedColor = Color.FromHex("#00AFA0");
inputLayout.ErrorText = "User name available";
inputLayout.InputView = new Entry() { Text = "John" };
```



### Unfocused color

When the input view is unfocused, the [UnfocusedColor](#) property value will be applied to the hint label and border.

**Note:** Thickness of the border will also vary between the focused and unfocused states.

### XML

```
<inputLayout:SfTextInputLayout
Hint="User name"
UnfocusedColor="Silver"
HelperText="User name available">
<Entry Text="John" />
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "User name";
inputLayout.UnfocusedColor = Color.Silver;
inputLayout.ErrorText = "User name available";
inputLayout.InputView = new Entry() { Text = "John" };
```



### Error color

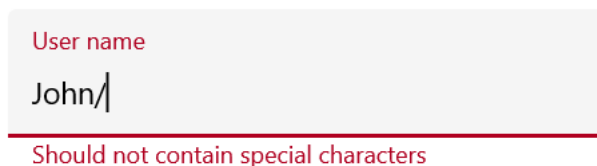
The error color can also be customized by setting the [ErrorColor](#) property.

### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
ErrorColor="#B00020"
ErrorText="Should not contain special characters"
HasError="true">
<Entry Text="John/" />
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ErrorColor = Color.FromHex("#B00020");
inputLayout.ErrorText = "Should not contain special characters";
inputLayout.HasError = true;
inputLayout.InputView = new Entry() { Text = "John/" };
```



### Container color

The color of the container can be customized by setting the [ContainerBackgroundColor](#) property. It is applicable when the [ContainerType](#) property is set to [Filled](#) and [Outlined](#).

### Filled

The color of the container is customized when the [ContainerType](#) is [Filled](#).

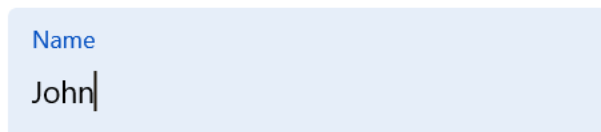
### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
FocusedColor="#0450C2"
ContainerType="Filled">
```

```
ContainerBackgroundColor="#E6EEF9">
<Entry Text="John" />
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.FocusedColor = Color.FromHex("#0450C2");
inputLayout.ContainerBackgroundColor = Color.FromHex("#E6EEF9");
inputLayout.ContainerType = ContainerType.Filled;
inputLayout.InputView = new Entry() { Text = "John" };
```



### Outlined

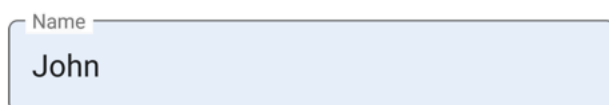
The color of the container is customized when the [ContainerType](#) is [Outlined](#).

### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
FocusedColor="#0450C2"
ContainerType="Outlined"
ContainerBackgroundColor="#E6EEF9">
<Entry Text="John" />
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.FocusedColor = Color.FromHex("#0450C2");
inputLayout.ContainerBackgroundColor = Color.FromHex("#E6EEF9");
inputLayout.InputView = new Entry() { Text = "John" };
```



### Disabled state

The text input layout is disabled by setting the [IsEnabled](#) property to `false`. The color of the container and other UI elements will also be changed to the disabled state, but its color cannot be customized.

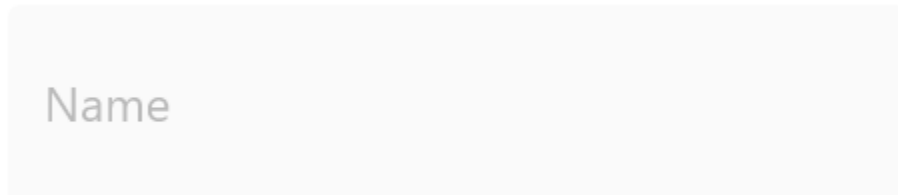
### XML



```
<inputLayout:SfTextInputLayout
Hint="Name"
IsEnabled="false">
<Entry />
</inputLayout:SfTextInputLayout>
```

**C#**

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.IsEnabled = false;
inputLayout.InputView = new Entry();
```



## Right-to-Left

The `TextInputLayout` supports to change the flow of text to the right-to-left direction by setting the [FlowDirection](#) to `RightToLeft`.

**Note:** A specific platform setup is required to enable the right-to-left localization. For platform settings, refer to this [documentation](#).

**XML**

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:inputLayout="clr-namespace:Syncfusion.XForms.TextInputLayout;assembly=Syncfusion.Core.XForms"
x:Class="TextInputLayout.MainPage">
<StackLayout>
<inputLayout:SfTextInputLayout
x:Name="textInputLayout"
FlowDirection="RightToLeft"
ContainerType="Outlined"
Hint="اسم"
HelperText="أسمك أدخل" >
<Entry />
</inputLayout:SfTextInputLayout>
</StackLayout>
</ContentPage>
```

**C#**

```
textInputLayout.FlowDirection = FlowDirection.RightToLeft;
textInputLayout.ContainerType = ContainerType.Outlined;
textInputLayout.Hint = "اسم";
textInputLayout.HelperText = "أسمك أدخل";
textInputLayout.InputView = new Entry();
```



### Fixed hint position

Hint label for the text input layout is fixed always at the top position. This helps users make the hint label floating even when the input view is not focused. It can be enabled by setting the `IsHintAlwaysFloated` property.

#### NOTE

The default value of the `IsHintAlwaysFloated` is `false`.

#### Filled

The hint label position of the input view will be set always at the top for the [Filled](#) container type.

#### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
IsHintAlwaysFloated="true"
ContainerType="Filled"
HelperText="Enter your name">
<Entry />
</inputLayout:SfTextInputLayout>
```

#### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.IsHintAlwaysFloated = true;
inputLayout.ContainerType = ContainerType.Filled;
inputLayout.HelperText = "Enter your name";
inputLayout.InputView = new Entry();
```



#### Outlined

The hint label position of the input view will be set always at the top for the [Outlined](#) container type.

#### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
IsHintAlwaysFloated="true"
ContainerType="Outlined"
HelperText="Enter your name">
<Entry />
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.IsHintAlwaysFloated = true;
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.HelperText = "Enter your name";
inputLayout.InputView = new Entry();
```



### None

The hint label position of the input view will be set always at the top for the [None](#) container type.

### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
IsHintAlwaysFloated="true"
ContainerType="None"
HelperText="Enter your name">
<Entry />
</inputLayout:SfTextInputLayout>
```

### C#

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.IsHintAlwaysFloated = true;
inputLayout.ContainerType = ContainerType.None;
inputLayout.HelperText = "Enter your name";
inputLayout.InputView = new Entry();
```



### How to

#### Customize the thickness of stroke

The border stroke width (for [Outlined](#)) and line thickness (for [Filled](#) and [None](#)) can be customized based on the focus state of the input view by setting the [FocusedStrokeWidth](#) and [UnfocusedStrokeWidth](#) properties.

### XML

```
<inputLayout:SfTextInputLayout
Hint="Name"
ContainerType="Outlined"
```

```
HelperText="Enter your name"
FocusedStrokeWidth="4"
UnfocusedStrokeWidth="2">
<Entry />
</inputLayout:SfTextInputLayout>
```

**C#**

```
var inputLayout = new SfTextInputLayout();
inputLayout.Hint = "Name";
inputLayout.ContainerType = ContainerType.Outlined;
inputLayout.HelperText = "Enter your name"
inputLayout.FocusedStrokeWidth = 4;
inputLayout.UnfocusedStrokeWidth = 2;
inputLayout.InputView = new Entry();
```



## Customize the font for input view

You can customize the font for the input view inside SfTextInputLayout using its properties.

- [Entry](#)
- [Editor](#)
- [SfMaskedEdit](#)
- [SfNumericTextBox](#)
- [SfAutoComplete](#)
- [SfComboBox](#)

## AutomationId

The text input layout control has built-in AutomationId for inner elements. To keep unique id for inner elements, AutomationId of inner elements are updated based on SfTextInputLayout's AutomationId. For example, if SfTextInputLayout's AutomationId is set as `sfTextInputLayout.AutomationId = "textInputLayout"`, then AutomationId of inner elements will be updated as follows.

Element	Element AutomationId	Qualified AutomationId
Password Toggle VisibleIcon	show password	textInputLayout show password
Password Toggle CollapsedIcon	hide password	textInputLayout hide password
DropDown Button	drop down	textInputLayout drop down

## Utilities

The Syncfusion Xamarin Extension provides quick access, so that you can easily create or configure the Syncfusion Xamarin projects. The Syncfusion Xamarin Extensions provides the following features.

- Syncfusion Project Template for Xamarin.Forms
- Syncfusion Toolbox for Xamarin.Forms

The Syncfusion Xamarin Visual Studio Extensions are installed along with the Essential Studio for Xamarin setup.

---

**Note:** \*Refer to the [installation guideline](#)\*.

---

## Project Template

Syncfusion provides the **Visual Studio Project Templates** for the Syncfusion Xamarin platform to easily create a Syncfusion Xamarin Application.

---

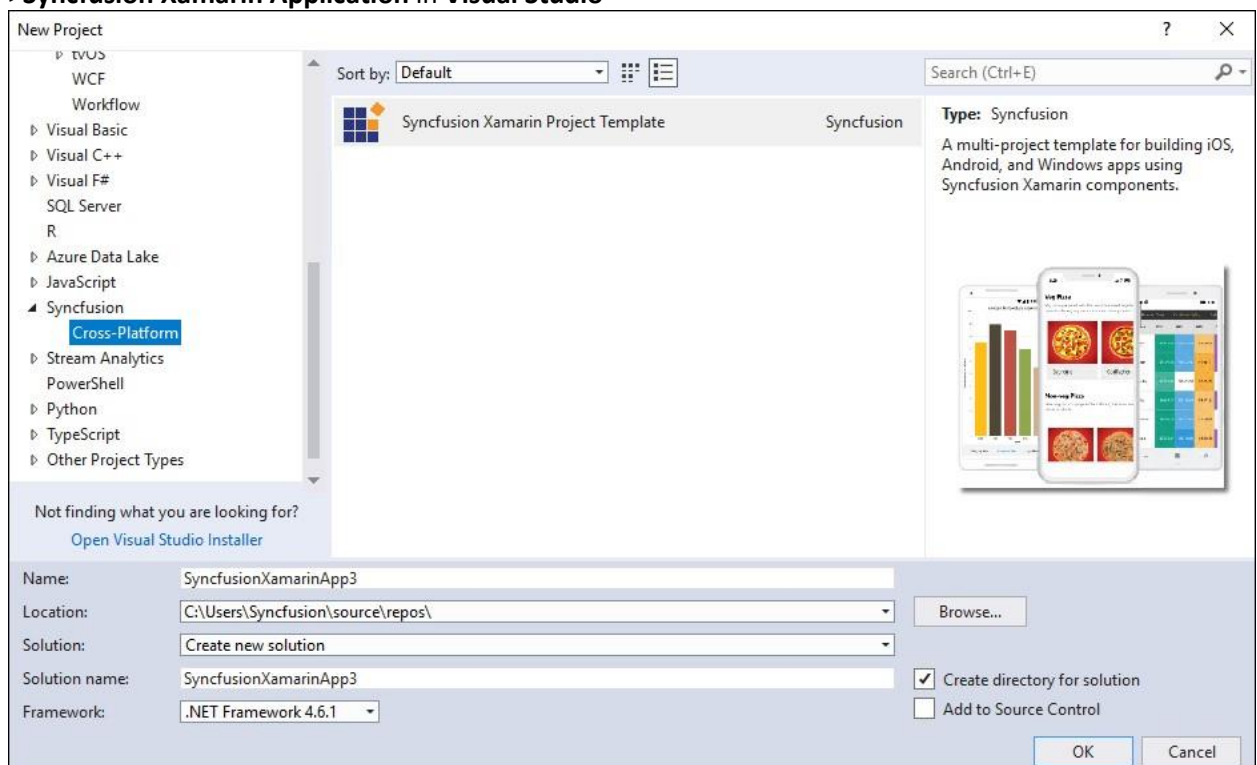
**Information:** Syncfusion Xamarin Project Templates are available from v16.2.0.41.

---

## Create Syncfusion Xamarin Application

The following steps direct you to create a **Syncfusion Xamarin Application** using **Visual Studio 2017**:

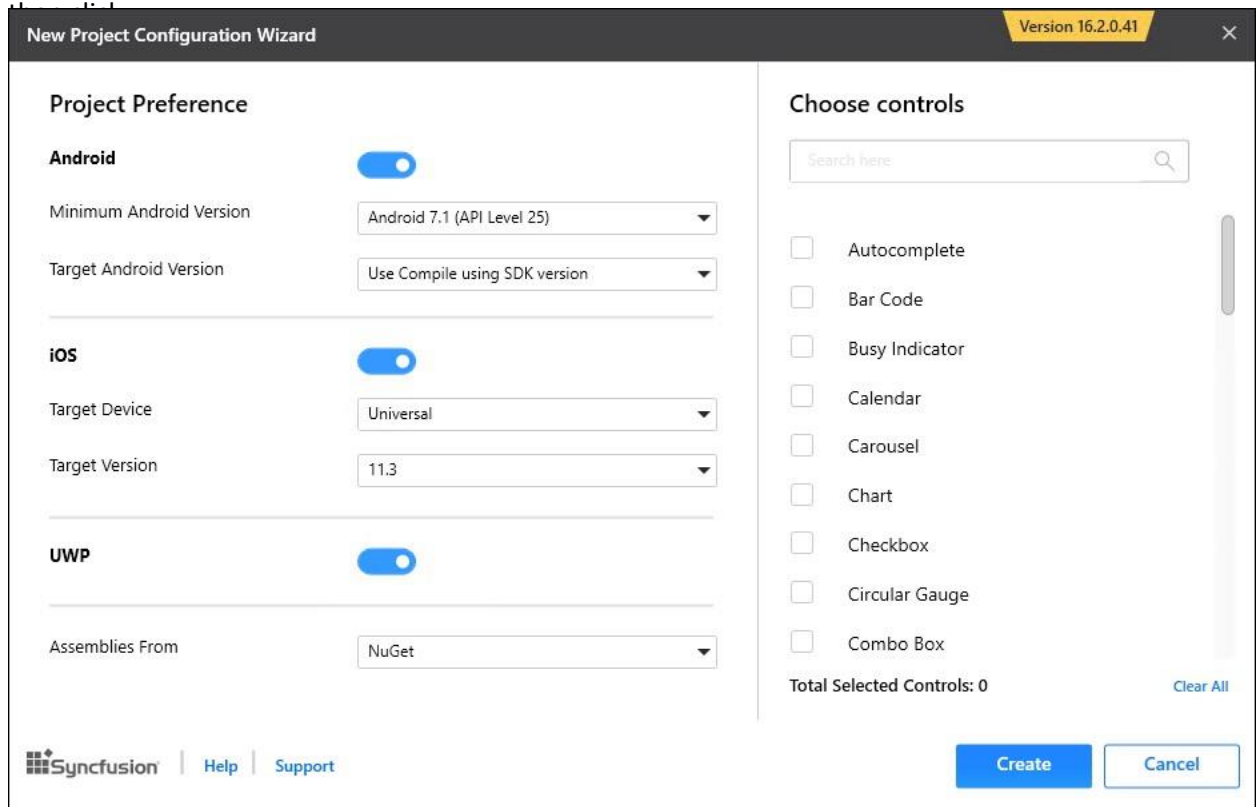
1. To create a **Syncfusion Xamarin project**, choose **New Project->Syncfusion-> Cross-Platform->Syncfusion Xamarin Application in Visual Studio**



2017.

![Add New Project Window](Syncfusion-Project-Templatesimages/Syncfusion-Project-Templatesimg1.jpeg)

2. Name the **Project**, choose the destination location, and set the Framework of the project, and



**OK.** The Project Configuration Wizard appears.

3. Choose the options to configure the Syncfusion Xamarin Application by using the following Project Configuration dialog.

#### Project Configuration:

Choose the required Project platforms: Android, iOS, and UWP.

![[New Project Configuration Wizard]](Syncfusion-Project-Templatesimages/Syncfusion-Project-Templatesimg2.jpeg)

**Assemblies From:** Specify where the Syncfusion Xamarin assemblies need to be loaded from: either NuGet or Installed Location.

---

**Note:** *Installed location option will be available only when the Syncfusion Xamarin setup has been installed.*

---

#### Android:

1. **Minimum Android Version:** Select the oldest Android version that you want to support your application. 2. **Target Android Version:** Select the version of Android to run your application.

#### iOS:

1. **Target Device:** Select the device of Xamarin.iOS Project, either Unified, iPhone/iPod, or iPad. 2. **Target Version:** Choose the version of Xamarin.iOS Project.

#### Choose controls

Choose the Syncfusion controls that need to be added to the created project.

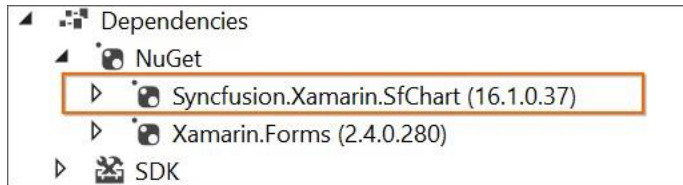
4. Click **Create** and the Syncfusion Xamarin Application will be created.

---

**Note:** Choose any one of the project types and controls from the Project Configuration Wizard.

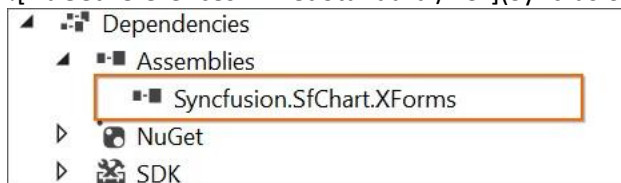
---

5. Required Syncfusion NuGet/Assemblies and configuration have been added to the project based on the control(s) chosen.



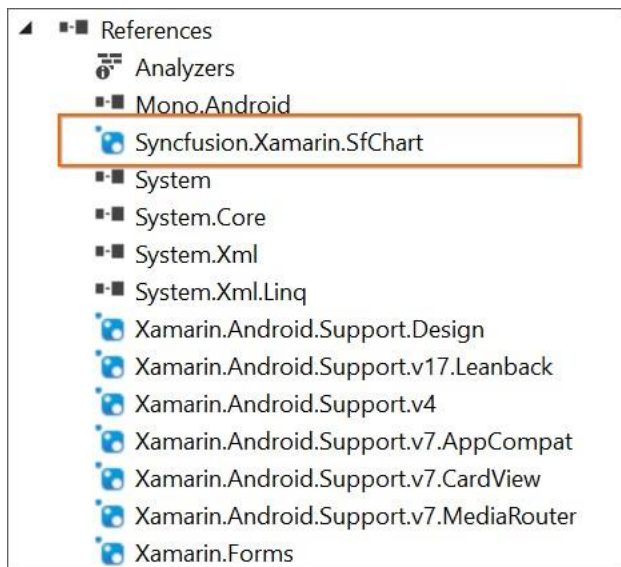
**Net Standard /PCL:**

![NuGet references in Net Standard /PCL](Syncfusion-Project-Templates



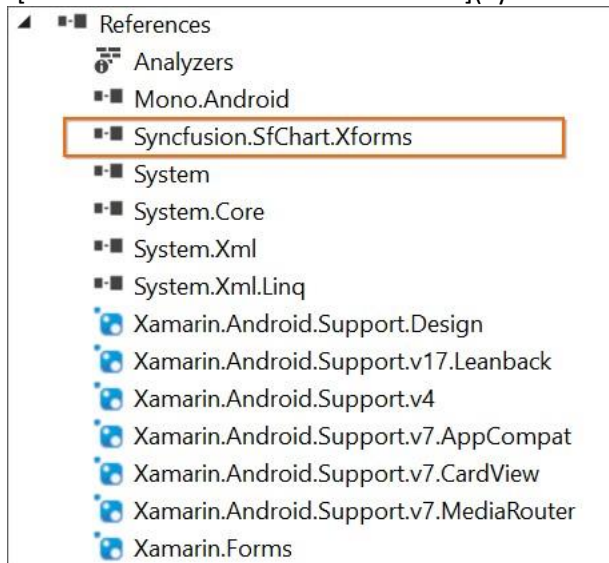
images/Syncfusion-Project-Templatesimg3.jpeg)

![Assembly references in Net Standard /PCL](Syncfusion-Project-Templatesimages/Syncfusion-Project-Templatesimg4.jpeg)



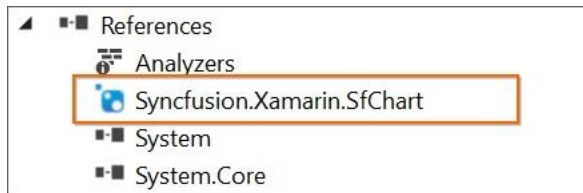
**Android:**

![NuGet references in XForms.Android](Syncfusion-Project-Templates



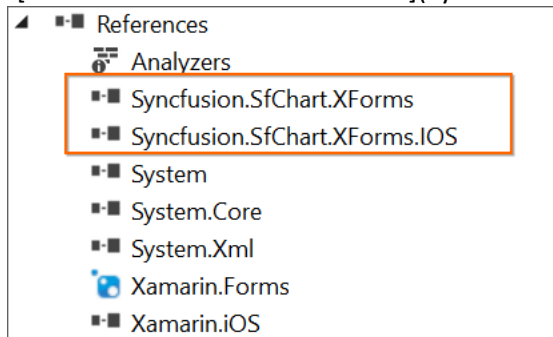
images/Syncfusion-Project-Templatesimg5.jpeg)

![Assembly references in XForms.Android](Syncfusion-Project-Templatesimages/Syncfusion-Project-Templatesimg6.jpeg)



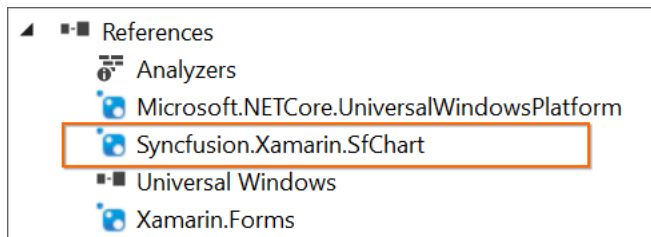
iOS:

![NuGet references in XForms.iOS](Syncfusion-Project-Templates



images/Syncfusion-Project-Templatesimg7.jpeg)

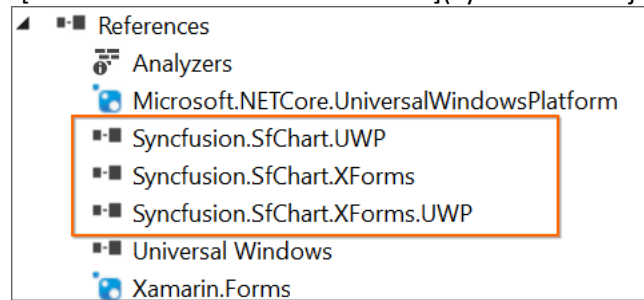
![Assembly references in XForms.iOS](Syncfusion-Project-Templatesimages/Syncfusion-Project-Templatesimg8.jpeg)



UWP:



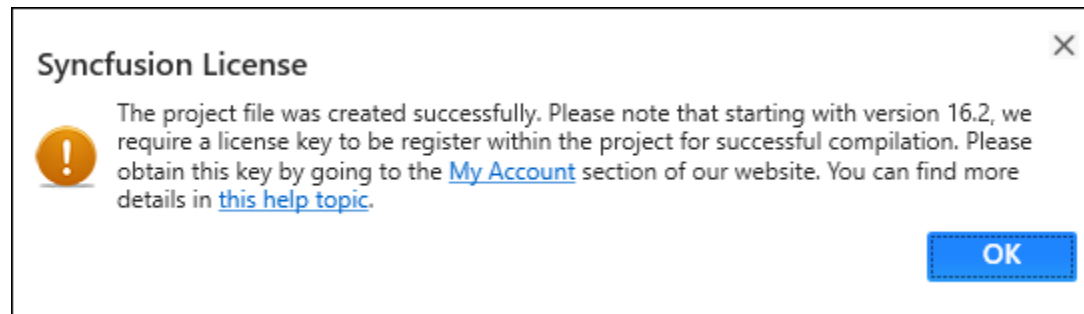
![NuGet references in XForms.UWP](Syncfusion-Project-Templates



images/Syncfusion-Project-Templatesimg9.jpeg)

![Assembly references in XForms.UWP](Syncfusion-Project-Templatesimages/Syncfusion-Project-Templatesimg10.jpeg)

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



## Toolbox

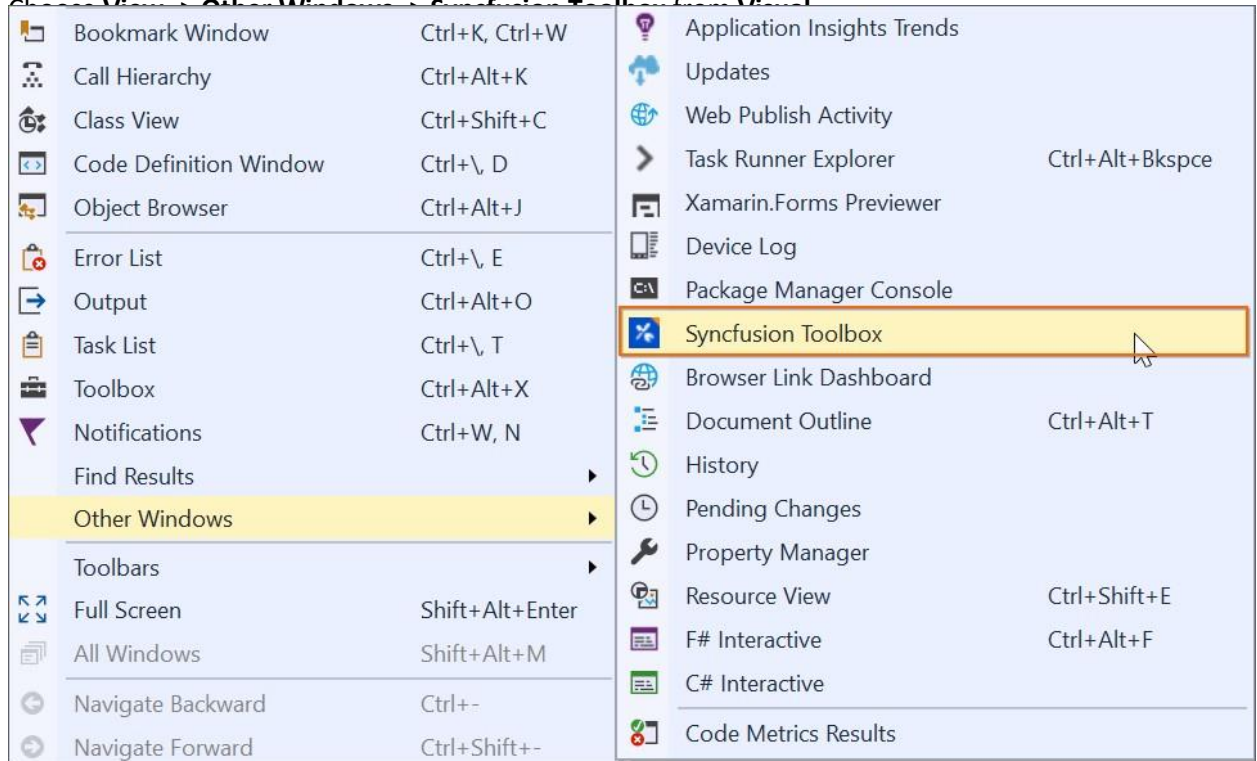
Syncfusion provides the **Visual Studio Toolbox** for the Syncfusion Xamarin platform to help add the Syncfusion Xamarin (Xamarin.Forms) controls in your project. It supports Microsoft Visual Studio 2015 and 2017. The Syncfusion Xamarin Toolbox enables easy drag and drop of Syncfusion controls without XAML coding in the Visual Studio designer.

**Information:** The Syncfusion Xamarin Toolbox is available from v16.2.0.41.

**Note:** \*Syncfusion Xamarin Toolbox will get installed when installed the Syncfusion Xamarin setup with Additional Settings (Configure Syncfusion Extensions in Visual Studio). In any case, if toolbox not installed, please run the [VSIX Installer utility](#) to configure it in Visual Studio.

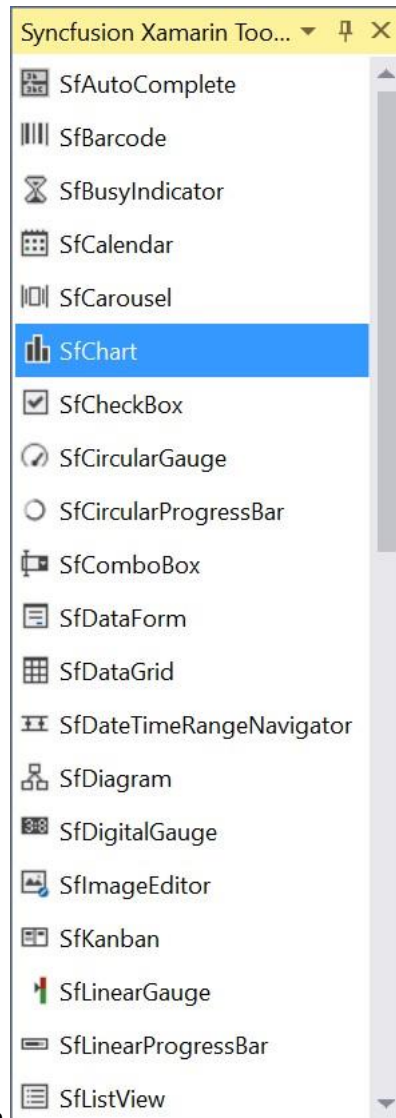
## Add Syncfusion Xamarin (Xamarin.Forms) Controls in your Project

Create the Xamarin or Syncfusion Xamarin project. The following steps direct you to add the Syncfusion controls through the Visual Studio Toolbox:

1. **View -> Other Windows -> Syncfusion Toolbox**

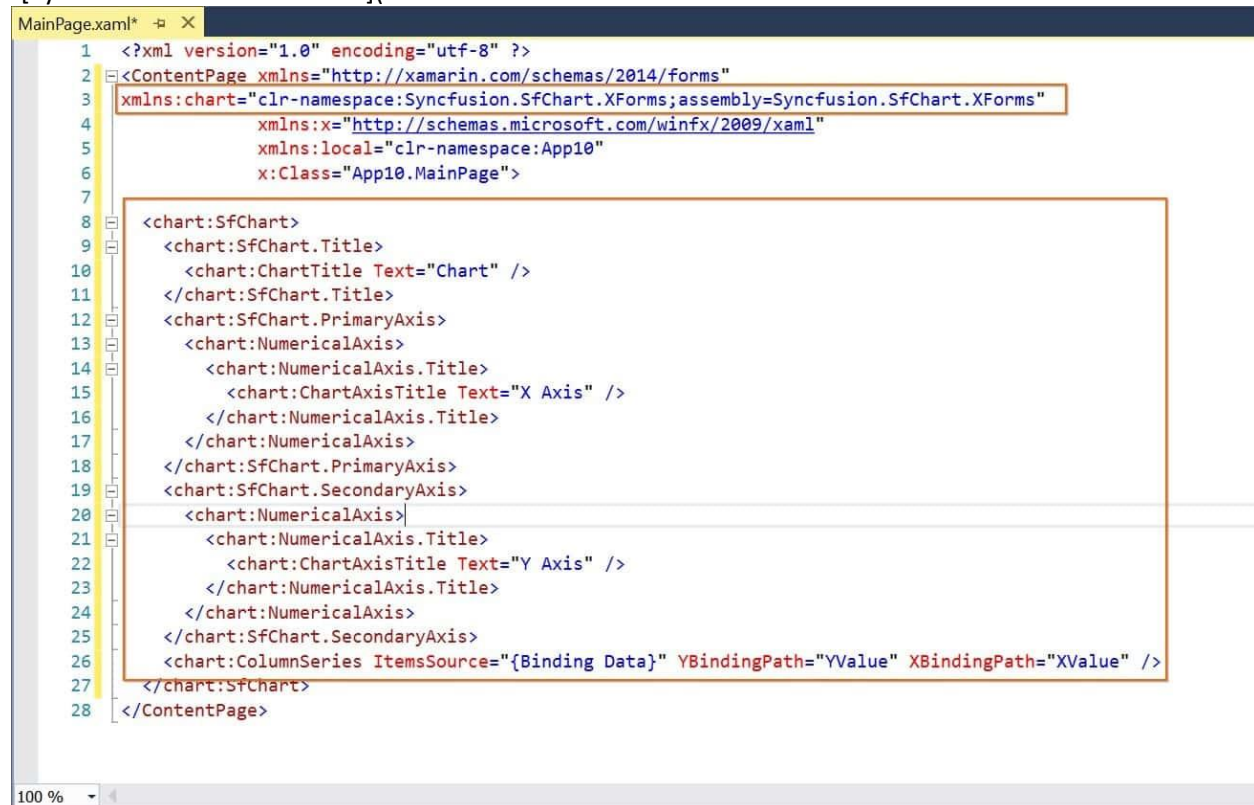
**Studio.**

![[Enable Syncfusion Xamarin Toolbox](Toolboximages/Toolboximg1.jpeg)]



2. Click **Syncfusion** **Toolbox** menu item, the Syncfusion Toolbox wizard has been appeared. The Syncfusion control will be enabled when opening the Xamarin.Forms designer page. There is no Syncfusion control appears till open the appropriate .xaml file from the Xamarin shared/.NET Standard /PCL project.

## ![Syncfusion Xamarin Toolbox](Toolbox



images/Toolboximg2.jpeg)

3. The required Syncfusion controls design (.xaml) snippet and namespace will be added by drag and drop the required control from the toolbox to the designer.

![Required Syncfusion control code snippet and namespace in design page](Toolbox\_images/Toolbox-img3.jpg)

Also, the required control Syncfusion Xamarin NuGet packages will be installed automatically when drag and drop the control to the designer to render the Syncfusion control.