ComponentOne

FlexChart for WinForms

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

Table of Contents

Overview

Modern looking, high performance FlexChart for WinForms come with powerful and flexible data binding, simple and easy-to-use API to configure charts. In addition, they provide several basic to complex chart types for your data visualization needs.

To gain insights into the controls, click the following links and access the comprehensive and useful information:

- FlexChart
- FlexPie
- Sunburst Chart
- FlexRadar
- TreeMap

Getting Started with WinForms Edition

For information on installing ComponentOne Studio WinForms Edition, licensing, technical support, namespaces, and creating a project with the FlexChart control, visit Getting Started with WinForms Edition.

FlexChart

FlexChart—a powerful data visualization control for Windows—lets you add feature-rich and visually appealing charts to your Windows Forms applications. The control empowers end-users to visualize data that resonates with their audiences.

The FlexChart control provides you with numerous 2D chart types, built-in tools for chart interactivity, and diversified formats for chart rendering.

Whether it is storytelling with data or interpreting complex data, FlexChart helps you accomplish everything seamlessly.

Below is a complete listing of the sections to get you started and acquainted with the FlexChart control:

- Getting Started with WinForms
- Key Features
- Feature Comparison
- FlexChart Quick Start
- Understanding FlexChart
- Working with FlexChart

Key Features

FlexChart is an impeccable data visualization component for .NET. In terms of performance, presentation, and overall quality, this control stands unrivalled.

1. **Automatic Legend generation:** Just specify the name of the series, and the Legend is displayed automatically.
2. **Axis labels automatic rotation:** Let long axis labels get rotated automatically, thereby rendering a clean appearance.
3. **Chart export:** You can export your chart to different image formats, such as SVG, JPG, and PNG.
4. **Direct X support:** The control supports Direct X rendering engine.
5. **Flexible data labels:** Set offset, border, and position for data labels.
6. **In-built selection support:** Click on the chart and select either a single data point or an entire data series.
7. **Interpolate nulls:** Handle null values in line and area charts effectively by using the InterpolateNulls property.
8. **Legend wrapping:** Let the Legend items appear in several rows and columns depending upon the available space.
9. **Multiple chart types:** Add as many series as you want in a single chart. Set the desired chart type at each series, and thus integrate multiple chart types in a single chart.
10. **Pre-defined and custom palettes:** Choose from a number of pre-defined palettes or apply a custom palette to the chart.
11. **Powerful and flexible data binding:** Specify data source either at the series level or the chart level, as per your requirements. You can combine multiple data sources in a single chart as well.
12. **Series toggling:** Toggle the visibility of a series in the plot as well as the legend by using the LegendToggle

property.

13. **Simple to use:** It is simple to work with the FlexChart control due to its clear object model.
14. **Stacked or 100% stacked charts:** Make a chart stacked or 100% stacked by setting just one property.
15. **Support for categorical, numerical, and data or time axis:** Bind to different data types ranging from int, float, and string to DateTime.
16. **ToolTip customization:** Leverage powerful tooltip customization features.

Feature Comparison

This topic provides you with comparison charts that help you compare features offered by FlexChart across different platforms and features of FlexChart for WinForms with those of another chart.

- Comparing FlexCharts
  Comparison of FlexChart across three platforms- WinForms, WPF, and UWP.
- Comparing WinForms Charts
  Comparison of FlexChart for WinForms with another chart.

Comparing FlexCharts

Explore all of the features offered by FlexChart in WinForms, WPF and UWP. You can download the matrix in PDF.

Chart Types

Chart Types	Win	WPF	UWP
Area	✓	✓	✓
StackedArea	✓	✓	✓
StackedArea100	✓	✓	✓
SplineArea	✓	✓	✓
StackedSplineArea	✓	✓	✓
StepArea	✓	✓	✓
StackedSplineArea100	✓	✓	✓

Bar	✓	✓	✓
StackedBar	✓	✓	✓
StackedBar✓00	✓	✓	✓
Bubble	✓	✓	✓
CandleStick	✓	✓	✓
Column	✓	✓	✓
StackedColumn	✓	✓	✓
StackedColumn100	✓	✓	✓
Histogram	✓	✓	✓
RangedHistogram	✓	✓	✓
Stock/ HighLowOpenClose	✓	✓	✓
Line	✓	✓	✓
LineStacked	✓	✓	✓
LineStacked100	✓	✓	✓
Spline	✓	✓	✓
SplineStacked	✓	✓	✓
SplineStacked100	✓	✓	✓
LineSymbols	✓	✓	✓
StackedLineSymbols	✓	✓	✓
StackedLineSymbols100	✓	✓	✓
SplineSymbols	✓	✓	✓
SplineSymbolsStacked	✓	✓	✓
StackedSplineSymbols100	✓	✓	✓
Step	✓	✓	✓
StepSymbols	✓	✓	✓
Pie	✓	✓	✓
Doughnut	✓	✓	✓
PieExploded	✓	✓	✓
DoughnutExploded	✓	✓	✓
Point/ Scatter	✓	✓	✓
Radar	✓	✓	✓
Polar	✓	✓	✓
Box-and-Whisker	✓	✓	✓
ErrorBar	✓	✓	✓

Funnel	✓	✓	✓
Sunburst	✓	✓	✓
TreeMap	✓	✓	✓
Waterfall	✓	✓	✓
2D	✓	✓	✓
Heikin-Ashi	*	*	*
LineBreak/ThreeLineBreak	*	*	*
Renko	*	*	*
Kagi	*	*	*
ColumnVolume	*	*	*
EquiVolume	*	*	*
CandleVolume	*	*	*
ArmsCandleVolume	*	*	*

* Available in FinancialChart

Data Binding

Data Binding	Win	WPF	UWP
Objects implementing IEnumerable	✓	✓	✓

Core Features

Core Features	Win	WPF	UWP
Handle Empty/ Null Data Points	✓	✓	✓
HitTest	✓	✓	✓
Annotations	✓	✓	✓
Render Modes	Native/DirectX	Native/Direct2D/Direct3D	Native/Direct3D
Trendlines	✓	✓	✓
Coordinate Conversion Methods	✓	✓	✓
Batch Updates	✓	✓	✓
Serialization Support	✓	✓	✓
Sunburst Drilldown	✓	✓	✓

TreeMap Drilldown	✓	✓	✓

Look & Feel

Look & Feel	Win	WPF	UWP
Predefined Palettes	16	16	16
Custom Palette	✓	✓	✓
Background Color	✓	✓	✓
Background Image	✓	✓	✓
Background Gradient/ HatchStyles	✓	✓	✓
Border and Border Styles	✓	✓	✓

Chart Area

Chart Area	Win	WPF	UWP
Header	✓	✓	✓
Footer	✓	✓	✓
Header/Footer Borders	✓	✓	✓
Header/Footer Alignment	✓	✓	✓
Rotate ChartArea	✓	✓	✓

Plot Area

Plot Area	Win	WPF	UWP
Plot margins	✓		
Markers for PlotElements	Supported on FlexChart with LineSymbols/SplineSymbols and Scatter chart types	Supported on FlexChart with LineSymbols/SplineSymbols and Scatter chart types	Supported on FlexChart with LineSymbols/SplineSymbols and Scatter chart types
Markers Size	✓	✓	✓
Markers: Border and Border styling	✓	✓	✓
Background Image/ Gradient/ HatchStyle	✓	✓	✓

for PlotElements			

Data Labels

DataLabels	Win	WPF	UWP
Offset	✓	✓	✓
ConnectingLines	✓	✓	✓
Borders and Border styling	✓	✓	✓
Styling	✓	✓	✓
Format String	✓	✓	✓
Custom Content	✓	✓	✓
Positions for Cartesian charts	Bottom/ Center/ Left/ None/ Right/ Top	Bottom/ Center/ Left/ None/ Right/ Top	Bottom/ Center/ Left/ None/ Right/ Top
Positions for Pie charts	Center/ Inside/ Outside/ None	Center/ Inside/ Outside/ None	Center/ Inside/ Outside/ None

Annotations

Annotations	Win	WPF	UWP
Pre-defined Shapes	✓	✓	✓
Position	✓	✓	✓
Attaching Annotations	✓	✓	✓
Offset	✓	✓	✓
Styling	✓	✓	✓
Tooltip	✓	✓	✓
Connector	✓	✓	✓
Customization	✓	✓	✓

Axes

Axes	Win	WPF	UWP
Axes: Primary X/Y	✓	✓	✓
Axes: Secondary X/Y	✓	✓	✓

Axes: Multiple Secondary X/Y	✓	✓	✓
Axis Label: Format strings	✓	✓	✓
Axis Label: Hide	✓	✓	✓
Axis Label: styling	✓	✓	✓
Axis Range (Min/Max) values	✓	✓	✓
Axis: Hide	✓	✓	✓
Axis: Logarithmic	✓	✓	✓
Axis: Reverse	✓	✓	✓
AxisLine Styling	✓	✓	✓
Labels: Alignment	✓	✓	✓
Labels: Angle	✓	✓	✓
Labels: Hide overlapping	✓	✓	✓
Major/ Minor GridLines	✓	✓	✓
Major/ Minor TickMarks	✓	✓	✓
Major/ Minor Units	✓	✓	✓
Title and Title styling	✓	✓	✓
Configure Origin	Any value	Any value	✓
TickMarks Position	Cross/ Inside/ Outside/ None	Cross/ Inside/ Outside/ None	Cross/ Inside/ Outside/ None
Position	Top/ Bottom/ Left/ Right/ Auto/ None	Top/ Bottom/ Left/ Right/ Auto/ None	Top/ Bottom/ Left/ Right/ Auto/ None

Series

Series	Win	WPF	UWP
Multiple Series	✓	✓	✓
Data binding	✓	✓	✓
Chart types at series level	✓	✓	✓
Styling	✓	✓	✓
Visibility	Plot/ Legend/ Both Plot	Plot/ Legend/ Both Plot	Plot/ Legend/ Both Plot

	and Legend/ Hidden	and Legend/ Hidden	and Legend/ Hidden
Conditional Formatting	✓	✓	✓
Stacked Groups	✓	✓	✓

Legends

Legends	Win	WPF	UWP
Title	✓	✓	✓
Title Style	✓	✓	✓
Toggle Series Visibility from legend	✓	✓	✓
Orientation	Auto/ Vertical/ Horizontal	Auto/ Vertical/ Horizontal	Auto/ Vertical/ Horizontal
Position	Left/ Top/ Right/ Bottom	Left/ Top/ Right/ Bottom	Left/ Top/ Right/ Bottom
TextWrap	✓	✓	✓

Marker Symbols

Marker Symbols	Win	WPF	UWP
Box	✓	✓	✓
Dot	✓	✓	✓

User Interactions

User Interactions	Win	WPF	UWP
Tooltips	✓	✓	✓
Series selection	✓	✓	✓
Point selection	✓	✓	✓
LineMarkers aka Crosshairs	✓	✓	✓
Range Selector	✓	✓	✓
Zooming	✓	✓	✓
Scrolling	✓	✓	✓

Tooltips

Tooltips	Win	WPF	UWP
Auto tooltips	✓	✓	✓
Custom content	✓	✓	✓

Show Delay	✓	✓	✓
Styling	✓	✓	✓
Tooltips for different chart elements			

Pie Charts

Pie Charts	Win	WPF	UWP
Exploded slices	✓	✓	✓
Inner Radius	✓	✓	✓
Starting Angle of first slice	✓	✓	✓

Exporting/Importing & Printing

Exporting/Importing & Printing	Win	WPF	UWP
Export to JPEG/ JPG	✓	✓	✓
Export to PNG	✓	✓	✓
Export to SVG	✓		
Export to BMP		✓	✓
Printing support	✓	✓	✓

Footprint

Footprint	Win	WPF	UWP
Assembly Size	229KB	183KB	218KB

Comparing WinForms Charts

Explore all of the features offered by FlexChart and MS Chart. You can download the matrix in PDF.

Chart Types

Chart Types	FlexChart	MS Chart
Area	✓	✓
StackedArea	✓	✓
StackedArea100	✓	✓
SplineArea	✓	✓
StackedSplineArea	✓	

StackedSplineArea100	✓	
StepArea	✓	
Bar	✓	✓
StackedBar	✓	✓
StackedBar100	✓	✓
Bubble	✓	✓
CandleStick	✓	✓
Column	✓	✓
Combination	✓	✓
StackedColumn	✓	✓
StackedColumn100	✓	✓
Histogram	✓	✓
Ranged Histogram	✓	✓
Stock/ HighLowOpenClose	✓	✓
Line	✓	✓
LineStacked	✓	
LineStacked100	✓	
Spline	✓	✓
SplineStacked	✓	
SplineStacked100	✓	
LineSymbols	✓	
StackedLineSymbols	✓	
StackedLineSymbols100	✓	
SplineSymbols	✓	
SplineSymbolsStacked	✓	
StackedSplineSymbols100	✓	
Step	✓	
StepSymbols	✓	
Pie	✓	✓
Doughnut	✓	✓
PieExploded	✓	✓
DoughnutExploded	✓	✓
Point/ Scatter	✓	✓
Radar	✓	✓

Polar	✓	✓
Box-and-Whisker	✓	✓
ErrorBar	✓	✓
Funnel	✓	✓
Sunburst	✓	
TreeMap	✓	
Waterfall	✓	
2D	✓	✓
Heikin-Ashi	*	
LineBreak/ThreeLineBreak	*	✓
Renko	*	✓
Kagi	*	✓
ColumnVolume	*	
EquiVolume	*	
CandleVolume	*	
ArmsCandleVolume	*	
FastLine		✓
FastPoint		✓
StepLine		✓
RangeBar		✓
RangeColumn		✓
SplineRange		✓
Range		✓
PointAndFigure		✓
Pyramid		✓
3D		✓

* Available in FinancialChart

Data Binding

Data Binding	FlexChart	MS Chart
MS SQL Server	✓	✓
OLEDB	✓	✓
ODBC	✓	✓
Object Data Source	✓	✓
Oracle	✓	✓

SharePoint Objects	✓	✓
Unbound Data		✓

Chart Features

Data Binding	FlexChart	MS Chart
Annotations	✓	✓
Axis binding	✓	✓
Customizable data labels	✓	✓
Customizable headers and footers	✓	✓

Data Manipulations

Data Manipulations	FlexChart	MS Chart
Aggregation	With custom code	✓
Sorting	With custom code	✓
TopN	With custom code	✓

Core Features

Core Features	FlexChart	MS Chart
Handle Empty/ Null Data Points	✓	✓
HitTest	✓	✓
DirectX Rendering, GDI+, SVG Rendering	✓	
Trendlines	✓	
Coordinate Conversion Methods	✓	✓
Serialization Support	✓	✓
Annotations	✓	✓
Financial Analysis using predefined Indicators and Overlays	✓	✓
DataPoint Filtering		✓
DataPoint Sorting		✓
Grouping		✓
DataPoints Searching		✓
Copy/ Merge/ Split Series Data		✓
Export to Dataset		✓
Statistical Analysis using		✓

predefined formulas		
LineMarkers	✓	✓
Multiple axes	✓	
Multiple plot areas	✓	
Sunburst Drilldown	✓	
TreeMap Drilldown	✓	

Look & Feel

Look & Feel	FlexChart	MS Chart
Predefined Palettes	16	12
Custom Palette	✓	✓
Background Color	✓	✓
Background Image	✓	✓
Background Gradient/ HatchStyles		✓
Border and Border Styles		✓
Themes	16 themes	12 themes

Chart Area

Chart Area	FlexChart	MS Chart
Header	✓	✓
Footer	✓	
Multiple Headers		✓
Header/ Footer Borders	✓	✓
Header/ Footer Alignment	✓	✓
Rotate ChartArea	✓	
Multiple Chart or Plot Areas	✓	✓
Toggle Visibility		✓

Plot Area

Plot Area	FlexChart	MS Chart
Plot position configuration		✓
Plot margins	✓	
Markers for PlotElements	Supported on FlexChart with LineSymbols/ SplineSymbols and Scatter chart types	✓
Markers Size	✓	✓

Markers as Images		✓
Markers: Border and Border styling		✓
Background Image/ Gradient/ HatchStyle for PlotElements		✓
Border and Border styling for Plot Elements		✓
Empty Points Styling		✓

Data Labels

Data Labels	FlexChart	MS Chart
Offset	✓	
ConnectingLines	✓	✓
Borders and Border styling	✓	✓
Styling	✓	✓
Format String	✓	✓
Custom Content	✓	
Positions for Cartesian charts	Bottom/ Center/ Left/ None/ Right/ Top	Only applies to bar charts. Outside/ Left/ Right/ Center
Positions for Pie charts	Center/ Inside/ Outside/ None	Inside/ Outside/ None

Axes

Axes	FlexChart	MS Chart
Axes: Primary X/Y	✓	✓
Axes: Secondary X/Y	✓	✓
Axes: Multiple Secondary X/Y	✓	
Axis Label: Format strings	✓	
Axis Label: Hide	✓	✓
Axis Label: styling	✓	✓
Axis Range (Min/Max) values	✓	
Axis: Hide	✓	✓
Axis: Logarithmic	✓	✓
Axis: Reverse	✓	✓
AxisLine Styling	✓	✓

Labels: Alignment	✓	
Labels: Angle	✓	✓
Labels: Hide overlapping	✓	
Major/ Minor GridLines	✓	✓
Major/ Minor TickMarks	✓	✓
Major/ Minor Units	✓	✓
TickMarks Length	✓	
TickMarks Styling	✓	✓
Title and Title Styling	✓	✓
Configure Origin	Any value	Auto/ Min/ Max
TickMarks Position	Cross/ Inside/ Outside/ None	Cross/ Inside/ Outside/ None
Position	Top/ Bottom/ Left/ Right/ Auto/ None	
ArrowHead styling		✓
Axis Label: Custom		✓
Axis Label: Staggered		✓
ScaleBreaks		✓

Series

Series	FlexChart	MS Chart
Multiple Series	✓	✓
Data binding	✓	✓
Chart types at series level	✓	✓
Styling	✓	✓
Visibility	Plot/ Legend/ Both Plot and Legend/ Hidden	Plot/ Plot and Legend/ Hidden
Conditional Formatting	✓	
Stacked charts	✓	

Legends

Legends	FlexChart	MS Chart
Title	✓	✓
Title Style	✓	✓
TextWrap	✓	
Toggle Series Visibility from legend	✓	

Orientation	Auto/ Vertical/ Horizontal	Vertical/ Horizontal/ Table
Position	Left/ Top/ Right/ Bottom	Left/ Top/ Right/ Bottom
Custom Legend Items		✓
Items Ordering		✓
Multiple Legends		✓

Marker Symbols

Marker Symbols	FlexChart	MS Chart
Box	✓	✓
Dot	✓	✓
Diamond		✓
Triangle		✓
Cross		✓
Star4		✓
Star5		✓
Star6		✓
Star10		✓

User Interactions

User Interactions	FlexChart	MS Chart
Tooltips	✓	✓
Series selection	✓	✓
Point selection	✓	✓
Range Selection	✓	
Zooming	✓	✓
Scrolling	✓	✓
Drilldown	With custom code	With custom code
Draggable Annotations		✓

Tooltips

Tooltips	FlexChart	MS Chart
Auto tooltips	✓	
Custom content	✓	String type only
Show Delay	✓	

Styling	✓	
String type only		✓
Tooltips for different chart elements		✓

Financial Chart Features

Financial Chart Features	FinancialChart	MS Chart
Bollinger bands overlays	*	
Envelope overlay	*	
Fibonacci tools	*	
Indicators	*	
MACD Indicator	*	
Moving averages	*	
Overlays	*	
Stochastic Oscillator Indicator	*	
Fibonacci tools	*	

Pie Charts

Pie Charts	FlexChart	MS Chart
Exploded slices	✓	✓
Inner Radius	✓	✓
Starting Angle of first slice	✓	✓

Exporting, Importing & Printing

Exporting, Importing & Printing	FlexChart	MS Chart
Export to JPEG/ JPG	✓	✓
Export to PNG	✓	✓
Export to SVG	✓	
Export to BMP		✓
Export to EMF		✓
Export to EMfDual		✓
Export to EMfPlus		✓
Export to GIF		✓
Export to TIFF		✓
Save/ Load to Binary files		✓
Save/ Load to memory streams		✓

Save/ Load to XML		✓
Printing	✓	✓
Print Support	✓	✓

Footprint

Footprint	FlexChart	MS Chart
Assembly Size	330KB	904KB

Pricing

Pricing	FlexChart	MS Chart
Pricing	$	$

Quick Start

This quick start guides you through a step-by-step process of creating a simple FlexChart application and running the same in Visual Studio.

Let us consider that you want to compare three different models of BMW, namely F30, 7 G11/G12 Sedan, and Z4 E89 Roadster in a specific year. The comparison is to be made in terms of three parameters, fuel efficiency in city, engine capacity, and number of cylinders in the engine.

- The fuel efficiency in city, engine capacity, and number of cylinders in the engine are to be plotted along one axis (Y-axis)
- The models are to be plotted along another axis (X-axis)
- There are three categories: F30, 7 G11/G12 Sedan, and Z4 E89 Roadster
- Each category has three columns

When the number of categories to be compared is small, the Column Chart is apt to use. Here we can use the Column Chart to compare the three categories.

> This tutorial assumes that the database file **C1NWind.mdb** is in the **<SystemDocumentsFolder>\ComponentOne Samples\Common** where **<SystemDocumentsFolder>** varies for each user and platform.

Go through the below-mentioned steps to walk through the FlexChart control quickly.

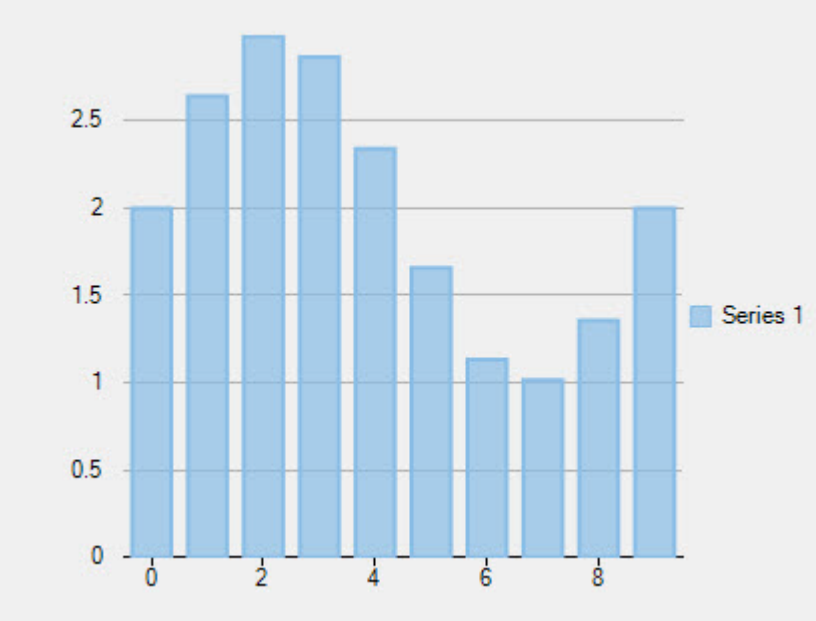
1. Adding FlexChart to the Form
2. Binding FlexChart to a Data Source

Step 1: Adding FlexChart to the Form

Perform the following steps to add the FlexChart control to the form:

1. Create a new **Windows Forms Application**.
2. In the Design view, either drag and drop or double-click the **FlexChart** control from/in the Toolbox.

The FlexChart control appears as shown below.



You have added FlexChart to the form successfully.

Step 2: Binding FlexChart to a Data Source

After you have added the FlexChart control to the form, you need to bind it to a valid data source.

The below-mentioned code illustrates how to bind FlexChart to a data source to plot the required data.

To run the given code successfully, you need to insert the **System.Data.OleDb** namespace in the code behind.

Add the following code in the **Form_Load** event:

- **Visual Basic**

```
' retrieve data from the data source
Dim conPath As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & vbCr & vbLf &
                        "Data Source=C:\\Users\\GPCTAdmin\\Desktop\\C1NWind.mdb"
Dim conQuery As String = "SELECT * FROM Cars WHERE Brand = 'BMW'"
Dim da As New OleDbDataAdapter(conQuery, conPath)

' fill the data table
da.Fill(dt)

' bind FlexChart to the data table
FlexChart1.DataSource = dt

' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()
Dim series3 As New C1.Win.Chart.Series()

' add the series to the series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
FlexChart1.Series.Add(series3)

' name the series
series1.Name = "Liter"
series2.Name = "Cyl"
series3.Name = "MPG_City"
```

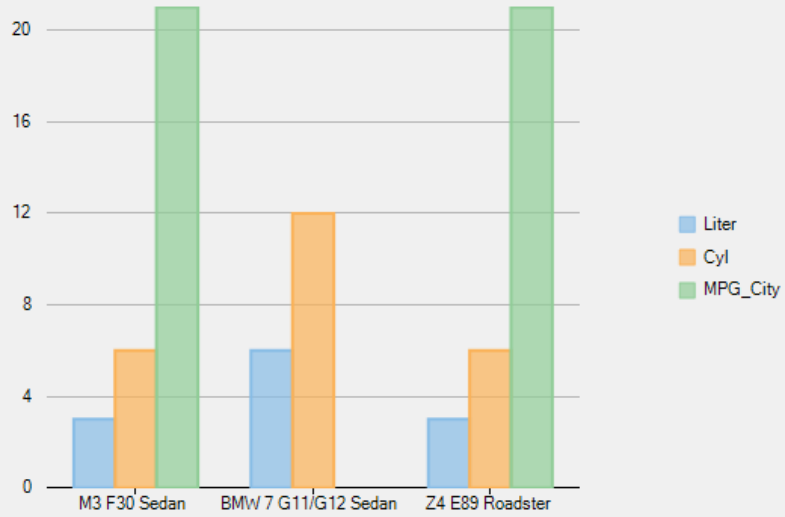
```
' bind X-axis and Y-axis
FlexChart1.BindingX = "Model"
series1.Binding = "Liter"
series2.Binding = "Cyl"
series3.Binding = "MPG_City"

' set the chart type
FlexChart1.ChartType = C1.Chart.ChartType.Column

' set the legend position
FlexChart1.Legend.Position = C1.Chart.Position.Right
```

- **C#**

```
DataTable dt = new DataTable();

// retrieve data from the data source
string conPath = @"Provider=Microsoft.Jet.OLEDB.4.0;
                Data Source=C:\\Users\\<User Name>\\Desktop\\C1NWind.mdb"; // Provide the local path of data source
string conQuery = "SELECT * FROM Cars WHERE Brand = 'BMW'";
OleDbDataAdapter da = new OleDbDataAdapter(conQuery, conPath);

// fill the data table
da.Fill(dt);

// bind FlexChart to the data table
flexChart1.DataSource = dt;

// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series3 = new C1.Win.Chart.Series();


// add the series to the series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);
flexChart1.Series.Add(series3);


// name the series
series1.Name = "Liter";
series2.Name = "Cyl";
series3.Name = "MPG_City";


// bind X-axis and Y-axis
flexChart1.BindingX = "Model";
series1.Binding = "Liter";
series2.Binding = "Cyl";
series3.Binding = "MPG_City";


// set the chart type
flexChart1.ChartType = C1.Chart.ChartType.Column;

// set the legend position
flexChart1.Legend.Position = C1.Chart.Position.Right;
```

Once you've added the code in the Form_Load event, run the application and observe the output.

Notice that the output appears with the plotted data, as shown below:



You have successfully created a simple FlexChart application.

This concludes the quick start.

Understanding FlexChart

To get started with the FlexChart control, you need thorough understanding of all FlexChart fundamentals and FlexChart types.

This section, therefore, takes you through the basics of the control in an easy-to-understand manner.

Click the following links to find information on all FlexChart basics and types:

- FlexChart Fundamentals
- FlexChart Types

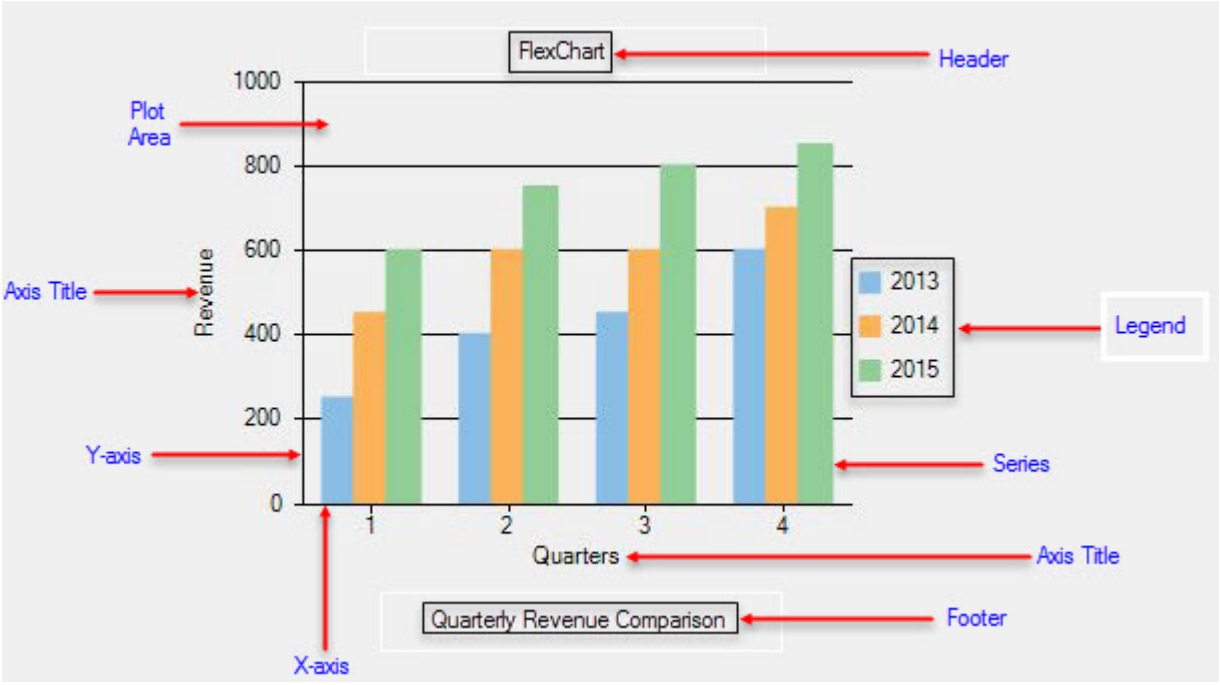
FlexChart Fundamentals

FlexChart consists of the following elements:

- Header and Footer
- Legend
- Axes
- Plot Area
- Series

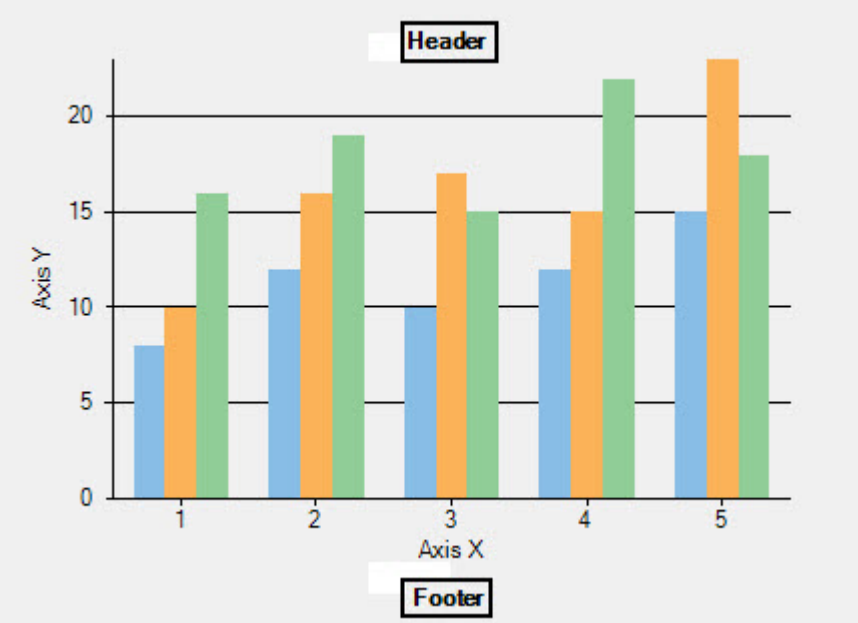
The control has a rich object model that represents these elements in terms of objects and provides relevant properties for the same.

The following image displays the various elements:



Header and Footer

Header and Footer are used to display descriptive and relevant information with respect to the chart.



In FlexChart, these elements are set by using the Header and the Footer property. Both properties return a ChartTitle object comprising the properties mentioned below:

Property	Description

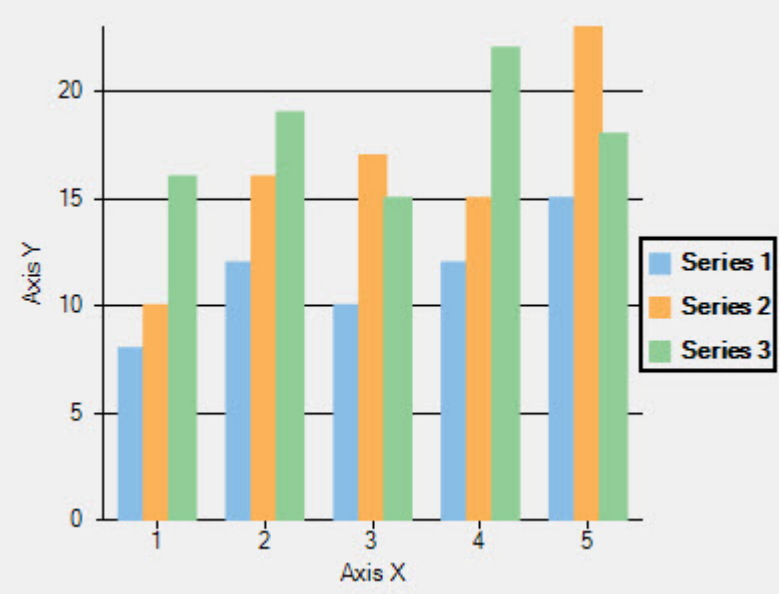
Content	Sets the title content (header or footer).
Style	Contains properties to set the font and the colors of the title.
HorizontalAlignment	Determines the horizontal position of the title.
Border	Determines whether the title has border.
BorderStyle	Contains properties that set the style of the title border.

You can add Header and Footer into FlexChart either at design-time or through code.

To add the elements at design-time, you need to navigate to the Header and the Footer node in the Properties window. And to specify the elements at run-time, you need to set the Content property in the code behind.

Legend

The Legend displays an entry for each data series in the chart. It represents the mapping between colors, symbols, and data series.



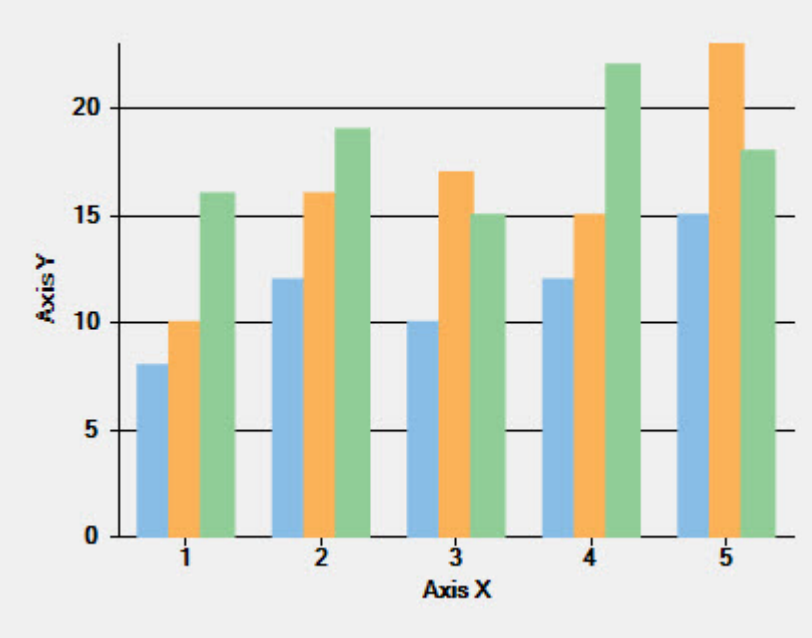
In FlexChart, the Legend is set by using the Legend property that returns the Legend object, which has the following properties:

Property	Description
Style	Contains properties that set the style of the legend.
Position	Determines the position of the legend.

For more details on Legend, refer to FlexChart Legend.

Axes

In a chart, there are two primary axes: X and Y. There are, of course, exceptions when you work with pie charts.



In FlexChart, X-axis and Y-axis are represented by the AxisX and the AxisY property respectively. Both the properties return an Axis object comprising the properties given below:

Layout and Style Properties

Property	Description
AxisLine	Determines whether the axis line is visible.
AxisType	Contains the type of the axis.
Position	Sets the position of the axis.
Reversed	Reverses the direction of the axis.
Style	Contains properties that set the style of the axis.
Title	Sets the title text to display next to the axis.

Axis Label Properties

Property	Description
Format	Sets the format string for the axis labels.
LabelAlignment	Sets the alignment of the axis labels.
LabelAngle	Set the rotation angle of the axis labels.
Labels	Determines whether the axis labels are visible.
MajorUnit	Sets the number of units between axis labels.

Scaling, Tick Mark, and Gridline Properties

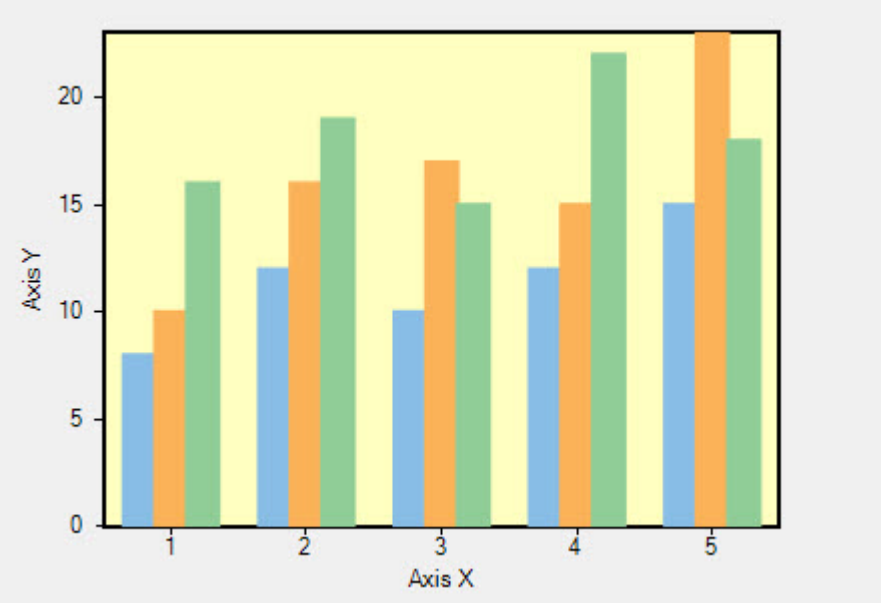
Property	Description
ActualMax	Gets the current axis maximum value.

ActualMin	Gets the current axis minimum value.
MajorGrid	Determines whether the axis includes gridlines.
MajorGridStyle	Contains properties to control the appearance of the grid lines drawn perpendicular to the major tick marks.
MajorTickMarks	Sets the location of the axis tick marks.
Max	Sets the maximum value for the axis.
Min	Sets the minimum value for the axis.
Origin	Sets the value at which an axis crosses the perpendicular axis.

For more information on Axes, refer to FlexChart Axes.

Plot Area

The Plot Area contains data plotted against X-axis and Y-axis.



In FlexChart, the Plot Area can be customized by the following properties of the ChartStyle object:

Property	Description
Fill	Sets the color of the Plot Area.
Font	Sets the font of the Plot Area.
Stroke	Sets the stroke brush of the Plot Area.
StrokeDashPattern	Sets the stroke dash pattern of the Plot Area.
StrokeWidth	Sets the stroke width of the Plot Area.

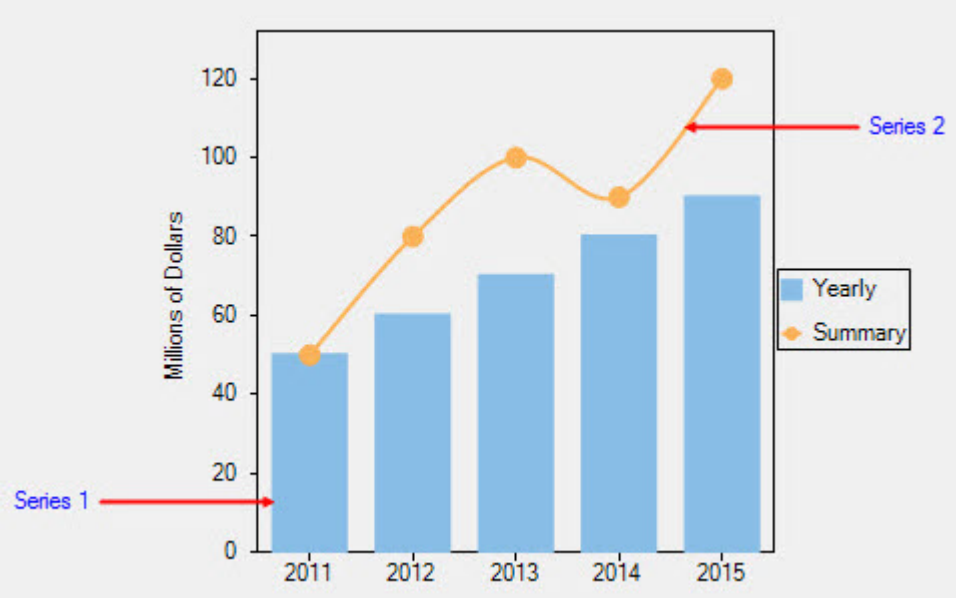
In addition, FlexChart allows creating multiple plot areas that increase data visibility by displaying a single series in a separate plot area.

For more information about multiple plot areas, refer to Multiple Plot Areas.

Series

Series are the groupings of the related points of data inside the Plot Area of the chart.

The following image illustrates data series in FlexChart:



Data series in FlexChart are controlled by the Series object comprising the following properties:

Property	Description
AxisX	Sets the series X-axis.
AxisY	Sets the series Y-axis.
Binding	Sets the name of the property that contains Y values for the series.
BindingX	Sets the name of the property that contains X values for the series.
ChartType	Sets the series chart type.
DataSource	Sets the collection of objects containing the series data.
Name	Sets the series text that is displayed in the legend.
Style	Sets the style of the series.
SymbolMarker	Sets the shape of the marker to be used for each data point of the series. This property applies to Scatter, LineSymbols, and SplineSymbols chart types only.
SymbolSize	Sets the size of the symbols used to render the series.
SymbolStyle	Sets the style of the symbols used in the series.
Visibility	Determines whether the series is visible and sets the position of the series, if it's visible.

For more information on series, refer to FlexChart series.

FlexChart Types

FlexChart offers a comprehensive set of chart types to meet all data visualization requirements of your end-users.

Below is a full listing of all the chart types offered by the control. Based upon the chart type you would like to use in your application, click the corresponding link to avail key information on the same.

- Area Chart
- Bar Chart
- Bubble Chart
- Column Chart
- Financial Charts
- Funnel Chart
- Histogram Chart
- Line Chart
- LineSymbols Chart
- Mixed Chart
- RangedHistogram Chart
- Scatter Chart
- Spline Chart
- SplineArea Chart
- SplineSymbols Chart
- Step Chart

Area

The Area Chart depicts change in data over a period of time. It represents data series by connecting data points against Y-axis and filling the area between the series and X-axis. In addition, the chart displays data series in the same order in which they are added—back-to-front.

FlexChart allows you to create the Area Chart both at design-time and run-time. At design-time, you need to set the ChartType property to Area in the Properties window; at run-time, you need to set the same property in code.

You can set the Stacking property to Stacked or Stacked100pc to create the stacking Area Chart.

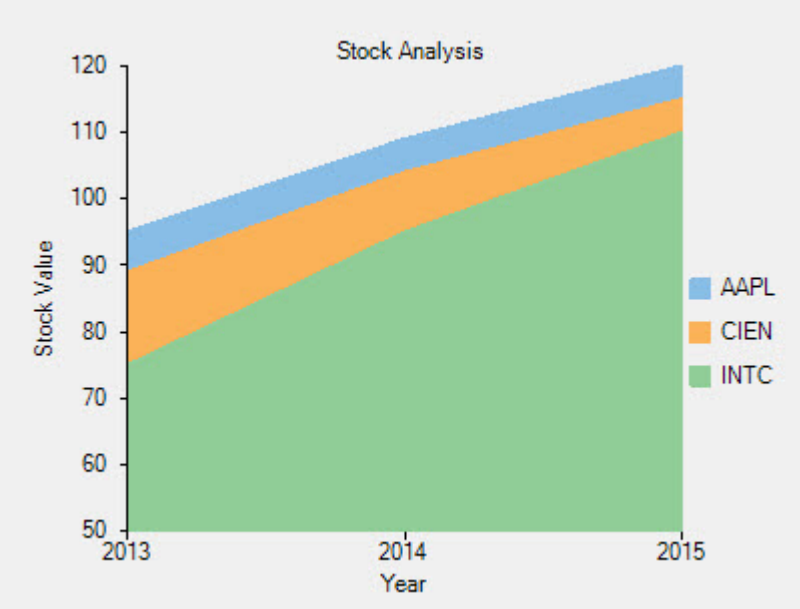
Suppose there are three stocks, namely AAPL, CIEN, and INTC that have stayed atop from 2013 to 2015 consecutively in the stock market. At the end of 2015, an analysis is done to identify the rise or decline in these stocks (based on their stock values) in the current year over the previous year. The analysis is based upon the values of the respective stocks in 2013, 2014, and 2015.

Since stock values represent data that varies over a period of time, the Area Chart is apt for this scenario.

Sample Data Table

Stock Name	2013	2014	2015
AAPL	95	109	120
CIEN	89	104	115
INTC	75	95	110

Area Chart



The above chart represents the rise of a stock by plotting the stock values in three consecutive years. The three areas for three different stocks have been rendered using three different colors.

- Number of series: three (AAPL, CIEN, and INTC)
- Number of Y values per point: one

The following code implements the above-mentioned scenario:

- **Visual Basic**

```
' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()
Dim series3 As New C1.Win.Chart.Series()

' add data points into the data series
series1.BindingX = "X"
series1.Binding = "Y"
series1.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2013, 95),
New System.Drawing.Point(2014, 109),
New System.Drawing.Point(2015, 120)}
series1.Name = "AAPL"

series2.BindingX = "X"
series2.Binding = "Y"
series2.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2013, 89),
New System.Drawing.Point(2014, 104),
New System.Drawing.Point(2015, 115)}
series2.Name = "CIEN"

series3.BindingX = "X"
series3.Binding = "Y"
series3.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2013, 75),
New System.Drawing.Point(2014, 95),
```

```
New System.Drawing.Point(2015, 110)}
series3.Name = "INTC"

' add the data series to the data series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
FlexChart1.Series.Add(series3)

' set the chart type to area
FlexChart1.ChartType = C1.Chart.ChartType.Area
```

- **C#**

```
// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series3 = new C1.Win.Chart.Series();

// add data points into the data series
series1.BindingX = "X";
series1.Binding = "Y";
series1.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2013,95),
new System.Drawing.Point(2014,109),
new System.Drawing.Point(2015,120) };
series1.Name = "AAPL";

series2.BindingX = "X";
series2.Binding = "Y";
series2.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2013,89),
new System.Drawing.Point(2014,104),
new System.Drawing.Point(2015,115) };
series2.Name = "CIEN";

series3.BindingX = "X";
series3.Binding = "Y";
series3.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2013,75),
new System.Drawing.Point(2014,95),
new System.Drawing.Point(2015,110) };
series3.Name = "INTC";

// add the data series to the data series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);
flexChart1.Series.Add(series3);

// set the chart type to area
flexChart1.ChartType = C1.Chart.ChartType.Area;
```

Bar

The Bar Chart compares values across various categories or displays variations in a data series over time. The chart displays horizontal bars for data series plotted against X-axis and arranges categories or items on Y-axis.

To create the Bar Chart at design-time, you need to set the ChartType property to Bar in the Properties window. And

to create the same at run-time, you need to set the property in code behind.

To create the stacking Bar Chart, you need to set the Stacking property either to Stacked or Stacked100pc.

Consider that a retailer wants to order six different types of products in beverages for his store. The order is to be placed with a distributor who provides the retailer with the requisite information regarding the products. The information comprises the price of each unit of a product, the number of units available in stock, and the number of units available on order.

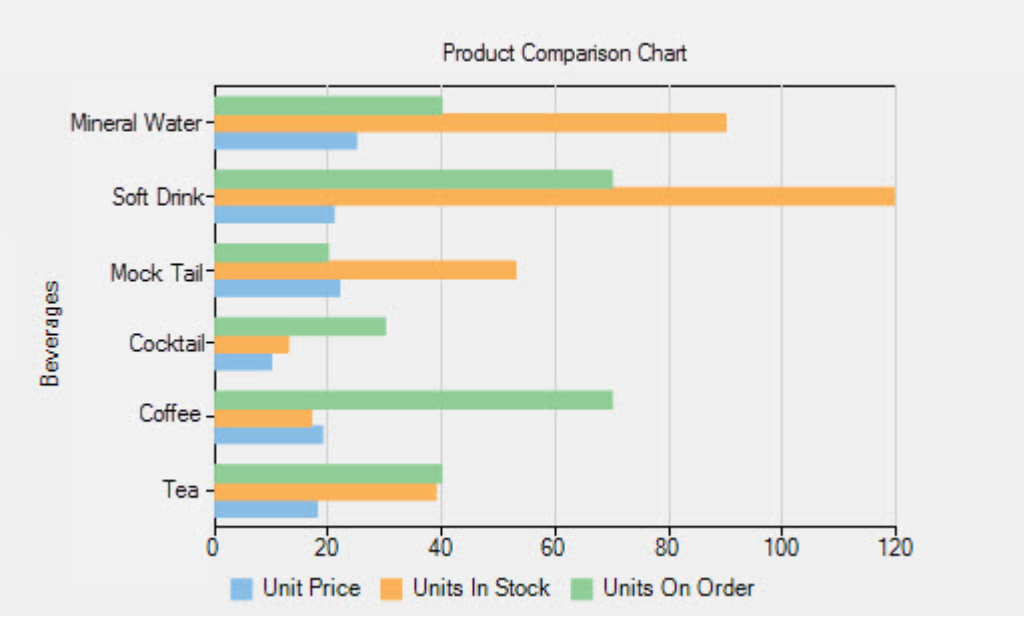
Here are the criteria that the retailer follows to order the products: unit price < $25, units in stock > 20, and units available on order > 20.

Let's prepare a comparison chart to help the retailer decide the products he can order. Since the number of products to be compared is six, the Bar Chart is apt for this case.

Sample Data Table

Beverages	Unit Price ($)	Units In Stock	Units On Order
Tea	18	39	40
Coffee	19	17	70
Cocktail	10	13	30
Mock Tail	22	53	20
Soft Drink	21	120	70
Mineral Water	25	90	40

Bar Chart



The above chart compares six different products on the basis of their unit price, units available in stock, and units available on order. The bars for the three series have been rendered using different colors.

- Number of series: three (Unit Price, Units In Stock, and Units On Order)
- Number of Y values per point: one
- Number of products to be compared: six (whose Product ID < 7)

See the following code to implement the scenario:

- **Visual Basic**

```
' create a datatable
Dim dt As New DataTable("Product Comparison")

' add columns to the datatable
dt.Columns.Add("Beverages", GetType(String))
dt.Columns.Add("Unit Price", GetType(Integer))
dt.Columns.Add("Units In Stock", GetType(Integer))
dt.Columns.Add("Units On Order", GetType(Integer))

' add rows to the datatable
dt.Rows.Add("Tea", 18, 39, 40)
dt.Rows.Add("Coffee", 19, 17, 70)
dt.Rows.Add("Cocktail", 10, 13, 30)
dt.Rows.Add("Mock Tail", 22, 53, 20)
dt.Rows.Add("Soft Drink", 21, 120, 70)
dt.Rows.Add("Mineral Water", 25, 90, 40)

' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()
Dim series3 As New C1.Win.Chart.Series()

' add the data series to the data series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
FlexChart1.Series.Add(series3)

' specify the datasource for the chart
FlexChart1.DataSource = dt

' bind the X-axis
FlexChart1.BindingX = "Beverages"

' bind the Y axes
series1.Binding = "Unit Price"
series2.Binding = "Units In Stock"
series3.Binding = "Units On Order"

' name the series
series1.Name = "Unit Price"
series2.Name = "Units In Stock"
series3.Name = "Units On Order"

' set the chart type to bar
FlexChart1.ChartType = C1.Chart.ChartType.Bar
```

- **C#**

```
// create a datatable
DataTable dt = new DataTable("Product Comparison");

// add columns to the datatable
dt.Columns.Add("Beverages", typeof(string));
dt.Columns.Add("Unit Price", typeof(int));
dt.Columns.Add("Units In Stock", typeof(int));
```

```
dt.Columns.Add("Units On Order", typeof(int));

// add rows to the datatable
dt.Rows.Add("Tea", 18, 39, 40);
dt.Rows.Add("Coffee", 19, 17, 70);
dt.Rows.Add("Cocktail", 10, 13, 30);
dt.Rows.Add("Mock Tail", 22, 53, 20);
dt.Rows.Add("Soft Drink", 21, 120, 70);
dt.Rows.Add("Mineral Water", 25, 90, 40);

// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series3 = new C1.Win.Chart.Series();

// add the data series to the data series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);
flexChart1.Series.Add(series3);

// specify the datasource for the chart
flexChart1.DataSource = dt;

// bind the X-axis
flexChart1.BindingX = "Beverages";

// bind the Y axes
series1.Binding = "Unit Price";
series2.Binding = "Units In Stock";
series3.Binding = "Units On Order";

// name the series
series1.Name = "Unit Price";
series2.Name = "Units In Stock";
series3.Name = "Units On Order";

// set the chart type to bar
flexChart1.ChartType = C1.Chart.ChartType.Bar;
```

Bubble

The Bubble Chart, which is basically a type of the Scatter Chart, is used for graphical representation of multi-dimensional data. It displays an additional data value at each point by changing its size. The chart type represents data points in the form of bubbles (data markers) whose X and Y coordinates are determined by two data values and whose size indicates the value of a third variable.

To create the Bubble Chart, you need to set the ChartType property to Bubble either in the Properties window (at design-time) or code behind (at run-time).

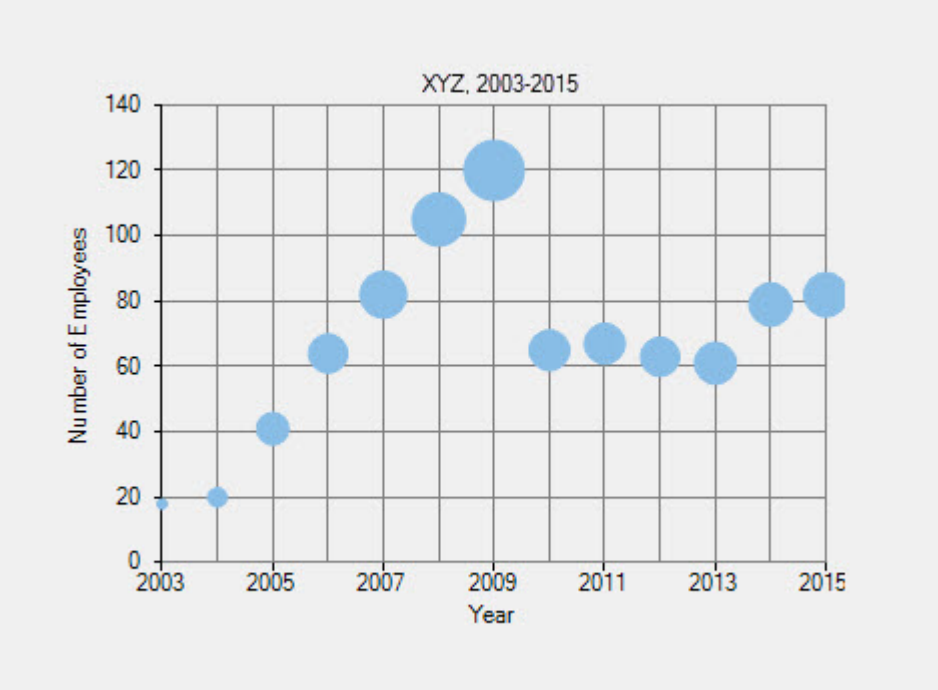
Let's assume that there is a company XYZ, founded in 2003. The HR of the company needs to prepare an analysis report showing the development of the company from 2003 to 2015. For the analysis, the parameters to be considered are the average number of employees each year and the annual revenue in that particular year.

We can use the Bubble Chart to create the analysis report because we need to represent two data values (Y-values) for each X-value. In the chart, the vertical axis displays the number of employees each year and the size of the bubble represents the annual revenue for the same year.

Sample Data Table

Year	Number of Employees	Annual Revenue (1000 $)
2003	18	50
2004	20	55
2005	41	80
2006	64	100
2007	82	130
2008	105	160
2009	120	200
2010	65	105
2011	67	106
2012	63	100
2013	61	110
2014	79	115
2015	82	120

Bubble Chart



The above chart shows the development of the company XYZ in the period 2003-2015.

- Number of series: one (Number of Employees and Annual Revenue)
- Number of Y values per point: two

Below is the implementation in code:

- **Visual Basic**

```
' create a datatable
Dim dt As New DataTable("Company XYZ,2003-2015")

' add columns to the datatable
dt.Columns.Add("Year", GetType(Integer))
dt.Columns.Add("Number of Employees", GetType(Integer))
dt.Columns.Add("Annual Revenue", GetType(Integer))

' add rows to the datatable
dt.Rows.Add(2003, 18, 50)
dt.Rows.Add(2004, 20, 55)
dt.Rows.Add(2005, 41, 80)
dt.Rows.Add(2006, 64, 100)
dt.Rows.Add(2007, 82, 130)
dt.Rows.Add(2008, 105, 160)
dt.Rows.Add(2009, 120, 200)
dt.Rows.Add(2010, 65, 105)
dt.Rows.Add(2011, 67, 106)
dt.Rows.Add(2012, 63, 100)
dt.Rows.Add(2013, 61, 110)
dt.Rows.Add(2014, 79, 115)
dt.Rows.Add(2015, 82, 120)

' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()

' add the data series to the data series collection
FlexChart1.Series.Add(series1)

' specify the datasource for the chart
FlexChart1.DataSource = dt

' bind X-axis and Y-axis
series1.BindingX = "Year"
series1.Binding = "Number of Employees,Annual Revenue"

' set the chart type to bubble
FlexChart1.ChartType = C1.Chart.ChartType.Bubble
```

- **C#**

```
// create a datatable
DataTable dt = new DataTable("Company XYZ,2003-2015");

// add columns to the datatable
dt.Columns.Add("Year", typeof(int));
dt.Columns.Add("Number of Employees", typeof(int));
dt.Columns.Add("Annual Revenue", typeof(int));

// add rows to the datatable
dt.Rows.Add(2003, 18, 50);
dt.Rows.Add(2004, 20, 55);
dt.Rows.Add(2005, 41, 80);
dt.Rows.Add(2006, 64, 100);
dt.Rows.Add(2007, 82, 130);
dt.Rows.Add(2008, 105, 160);
dt.Rows.Add(2009, 120, 200);
```

```
dt.Rows.Add(2010, 65, 105);
dt.Rows.Add(2011, 67, 106);
dt.Rows.Add(2012, 63, 100);
dt.Rows.Add(2013, 61, 110);
dt.Rows.Add(2014, 79, 115);
dt.Rows.Add(2015, 82, 120);

// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();

// add the data series to the data series collection
flexChart1.Series.Add(series1);

// specify the datasource for the chart
flexChart1.DataSource = dt;

// bind X-axis and Y-axis
series1.BindingX = "Year";
series1.Binding = "Number of Employees,Annual Revenue";

// set the chart type to bubble
flexChart1.ChartType = C1.Chart.ChartType.Bubble;
```

Column

The Column Chart, just like the Bar Chart, represents variation in a data series over time or compares different items. It displays values of one or more items as vertical bars against Y-axis and arranges items or categories on X-axis.

You need to set the ChartType property to Column either in the Property window or in code behind to create the Column Chart.

Set the Stacking property to Stacked or Stacked100pc to create the stacking Column Chart.

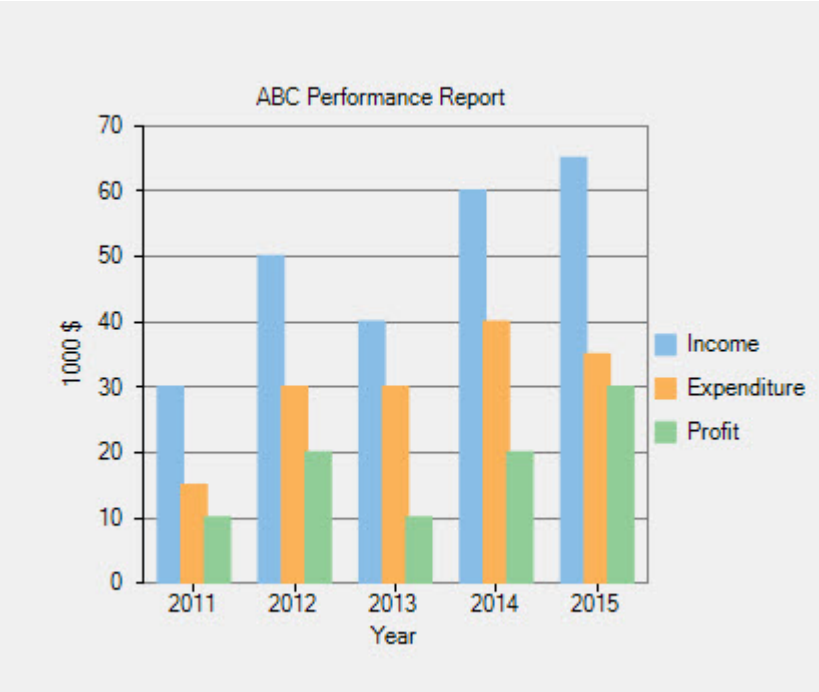
Let there be a company ABC that generates its performance report (of five years) for the period 2011-2015. The company calculates its annual profit by taking into account its income and expenditure in a year. Using the data, the company compares its annual profit in a particular year with that of the preceding year, thereby measuring its growth over the previous year.

Since the number of items to be compared are five, the Column Chart is apt for creating this report.

Sample Data Table

Year	Income (1000 $)	Expenditure (1000 $)	Profit (1000 $)
2011	30	20	10
2012	50	30	20
2013	40	30	10
2014	60	40	20
2015	65	35	30

Column Chart



The above chart displays variation in the income, expenditure, and profit of the company ABC over the period 2011-2015.

- Number of series: three (Income, Expenditure, and Profit)
- Number of Y values per point: one

The code below demonstrates the implementation:

- **Visual Basic**

```
' clear data series collection
flexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()
Dim series3 As New C1.Win.Chart.Series()

' add data points into the data series
series1.BindingX = "X"
series1.Binding = "Y"
series1.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2011, 30),
New System.Drawing.Point(2012, 50),
New System.Drawing.Point(2013, 40),
New System.Drawing.Point(2014, 60),
New System.Drawing.Point(2015, 65)}
series1.Name = "Income"

series2.BindingX = "X"
series2.Binding = "Y"
series2.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2011, 15),
New System.Drawing.Point(2012, 30),
New System.Drawing.Point(2013, 30),
New System.Drawing.Point(2014, 40),
New System.Drawing.Point(2015, 35)}
series2.Name = "Expenditure"
```

```
series3.BindingX = "X"
series3.Binding = "Y"
series3.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2011, 10),
New System.Drawing.Point(2012, 20),
New System.Drawing.Point(2013, 10),
New System.Drawing.Point(2014, 20),
New System.Drawing.Point(2015, 30)}
series3.Name = "Profit"

' add the data series to the data series collection
flexChart1.Series.Add(series1)
flexChart1.Series.Add(series2)
flexChart1.Series.Add(series3)

' set the chart type to column
flexChart1.ChartType = C1.Chart.ChartType.Column
```

- **C#**

```
// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series3 = new C1.Win.Chart.Series();

// add data points into the data series
series1.BindingX = "X";
series1.Binding = "Y";
series1.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2011,30),
new System.Drawing.Point(2012,50),
new System.Drawing.Point(2013,40),
new System.Drawing.Point(2014,60),
new System.Drawing.Point(2015,65)};
series1.Name = "Income";

series2.BindingX = "X";
series2.Binding = "Y";
series2.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2011,15),
new System.Drawing.Point(2012,30),
new System.Drawing.Point(2013,30),
new System.Drawing.Point(2014,40),
new System.Drawing.Point(2015,35)};
series2.Name = "Expenditure";

series3.BindingX = "X";
series3.Binding = "Y";
series3.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2011,10),
new System.Drawing.Point(2012,20),
new System.Drawing.Point(2013,10),
new System.Drawing.Point(2014,20),
new System.Drawing.Point(2015,30)};
series3.Name = "Profit";

// add the data series to the data series collection
flexChart1.Series.Add(series1);
```

```
flexChart1.Series.Add(series2);
flexChart1.Series.Add(series3);

// set the chart type to column
flexChart1.ChartType = C1.Chart.ChartType.Column;
```

Financial

Financial charts are used to represent fluctuation in market or stock prices; nonetheless, these charts can also be used to represent scientific data.

The FlexChart control supports two types of financial charts: Candle Chart and HighLowOpenClose Chart.

To use these chart types, you need to set the ChartType property either to Candlestick or to HighLowOpenClose at design-time or run-time.

Following are the financial chart types:

- Candle Chart
- HighLowOpenClose Chart

Candle

The Candle Chart integrates Bar and Line charts to depict a range of values over time. It consists of visual elements known as candles that are further comprised of three elements: body, wick, and tail.

- The body represents the opening and the closing value, while the wick and the tail represent the highest and the lowest value respectively.
- A hollow body indicates a rising stock price (the closing value is greater than the opening value).
- A filled body indicates a falling stock price (the opening value is greater than the closing value).

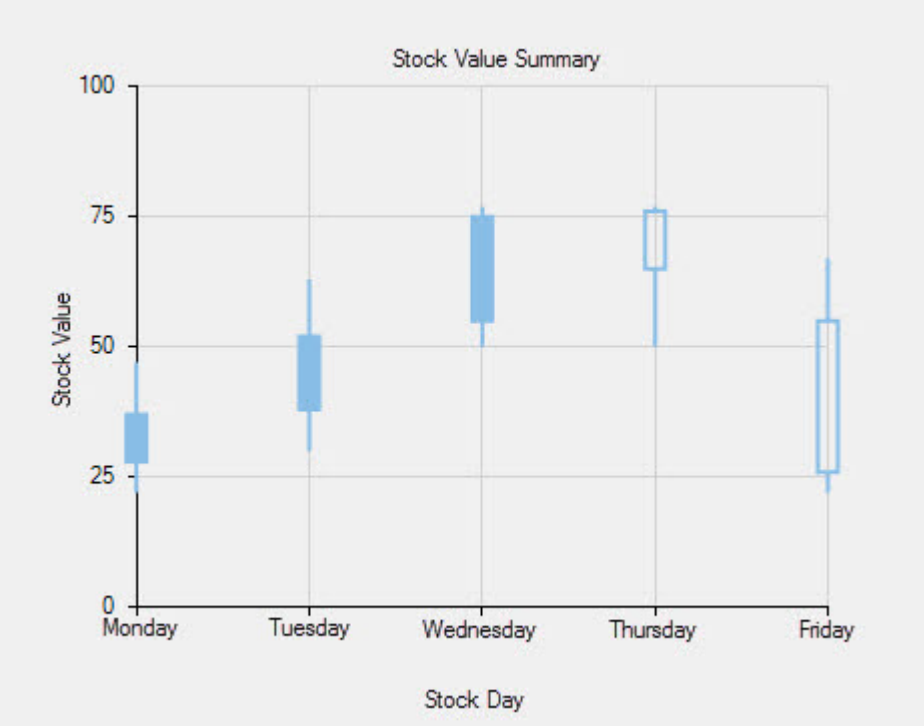
Consider a stock market scenario in which a stock is analyzed for five consecutive days in a particular week. The analysis is done to predict the prospective value of the stock. On the basis of the analysis, the general nature of the stock can be implied as well.

The Candle Chart is appropriate to represent the stock value summary.

Sample Data Table

Stock Day	Stock Open	Stock High	Stock Low	Stock Close
Monday	37	47	22	28
Tuesday	52	63	30	38
Wednesday	75	77	50	55
Thursday	65	77	50	76
Friday	26	67	22	55

Candle Chart



The above chart depicts the stock value summary for five consecutive days in a specific week.

- Number of series: one (Stock)
- Number of Y values per point: four

The following code shows the implementation:

- **Visual Basic**

```
' create a datatable
Dim dt As New DataTable("Stock Value Summary")

' add columns to the datatable
dt.Columns.Add("Stock Day", GetType(String))
dt.Columns.Add("Stock Open", GetType(Integer))
dt.Columns.Add("Stock High", GetType(Integer))
dt.Columns.Add("Stock Low", GetType(Integer))
dt.Columns.Add("Stock Close", GetType(Integer))

' add rows to the datatable
dt.Rows.Add("Monday", 37, 47, 22, 28)
dt.Rows.Add("Tuesday", 52, 63, 30, 38)
dt.Rows.Add("Wednesday", 75, 77, 50, 55)
dt.Rows.Add("Thursday", 65, 77, 50, 76)
dt.Rows.Add("Friday", 26, 67, 22, 55)

' clear data series collection
flexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()

' add the data series to the data series collection
flexChart1.Series.Add(series1)

' specify the datasource for the chart
flexChart1.DataSource = dt
```

```
' bind X-axis and Y-axis
FlexChart1.BindingX = "Stock Day"
series1.Binding = "Stock High,Stock Low,Stock Open,Stock Close"

' set the chart type to candlestick
flexChart1.ChartType = C1.Chart.ChartType.Candlestick
```

- **C#**

```
// create a datatable
DataTable dt = new DataTable("Stock Value Summary");

// add columns to the datatable
dt.Columns.Add("Stock Day", typeof(string));
dt.Columns.Add("Stock Open", typeof(int));
dt.Columns.Add("Stock High", typeof(int));
dt.Columns.Add("Stock Low", typeof(int));
dt.Columns.Add("Stock Close", typeof(int));

// add rows to the datatable
dt.Rows.Add("Monday", 37, 47, 22, 28);
dt.Rows.Add("Tuesday", 52, 63, 30, 38);
dt.Rows.Add("Wednesday", 75, 77, 50, 55);
dt.Rows.Add("Thursday", 65, 77, 50, 76);
dt.Rows.Add("Friday", 26, 67, 22, 55);

// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();

// add the data series to the data series collection
flexChart1.Series.Add(series1);

// specify the datasource for the chart
flexChart1.DataSource = dt;

// bind X-axis and Y-axis
flexChart1.BindingX = "Stock Day";
series1.Binding = "Stock High,Stock Low,Stock Open,Stock Close";

// set the chart type to candlestick
flexChart1.ChartType = C1.Chart.ChartType.Candlestick;
```

HighLowOpenClose

The HighLowOpenClose Chart is generally used in stock analysis. The chart combines four independent values to supply high, low, open, and close data values for each data point in a series.

Consider a stock market scenario in which a stock is to be analyzed in terms of its price for one week. The analysis needs to depict the stock price summary, so that the growth/decline can easily be interpreted.

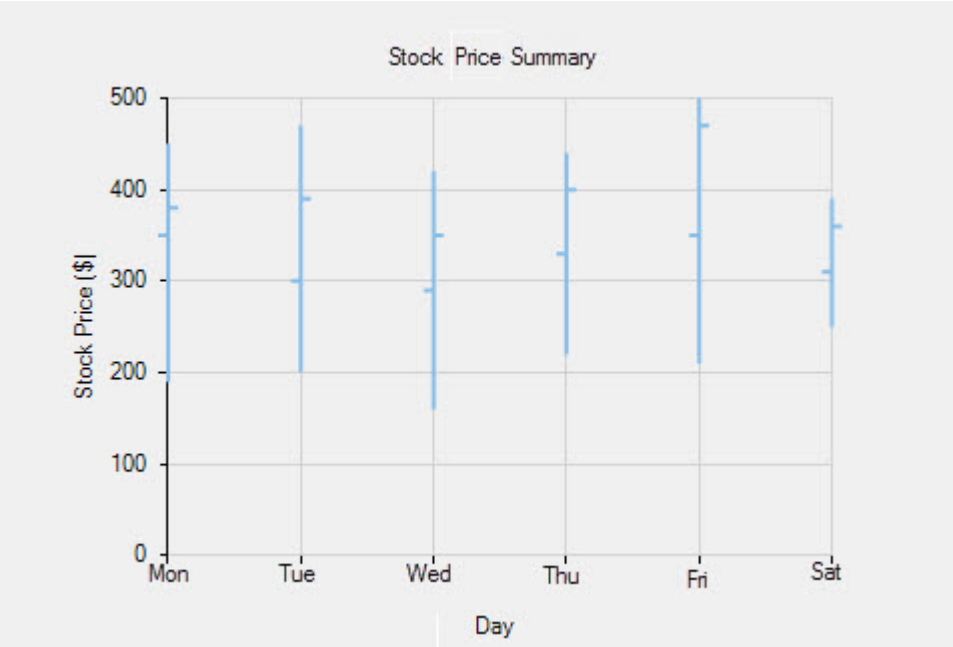
The HighLowOpenClose Chart can be used in this case.

Sample Data Table

Day	High	Low	Open	Close

Mon	450	190	350	380
Tue	470	200	300	390
Wed	420	160	290	350
Thu	440	220	330	400
Fri	500	210	350	470
Sat	390	250	310	360

HighLowOpenClose Chart



The above chart displays the stock summary for a particular week.

- Number of series: one (Stock)
- Number of Y values per point: four

See the following code for implementing the scenario:

- **Visual Basic**

```
' create a datatable
Dim dt As New DataTable("Stock Value Summary")

' add columns to the datatable
dt.Columns.Add("Day", GetType(String))
dt.Columns.Add("High", GetType(Integer))
dt.Columns.Add("Low", GetType(Integer))
dt.Columns.Add("Open", GetType(Integer))
dt.Columns.Add("Close", GetType(Integer))

' add rows to the datatable
dt.Rows.Add("Mon", 450, 190, 350, 380)
dt.Rows.Add("Tue", 470, 200, 300, 390)
dt.Rows.Add("Wed", 420, 160, 290, 350)
dt.Rows.Add("Thu", 440, 220, 330, 400)
dt.Rows.Add("Fri", 500, 210, 350, 470)
```

```
dt.Rows.Add("Sat", 390, 250, 310, 360)

' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()

' add the data series to the data series collection
FlexChart1.Series.Add(series1)

' specify the datasource for the chart
FlexChart1.DataSource = dt

' bind X-axis and Y-axis
FlexChart1.BindingX = "Day"
series1.Binding = "High,Low,Open,Close"

' set the chart type to candlestick
FlexChart1.ChartType = C1.Chart.ChartType.HighLowOpenClose
```

- **C#**

```
// create a datatable
DataTable dt = new DataTable("Stock Value Summary");

// add columns to the datatable
dt.Columns.Add("Day", typeof(string));
dt.Columns.Add("High", typeof(int));
dt.Columns.Add("Low", typeof(int));
dt.Columns.Add("Open", typeof(int));
dt.Columns.Add("Close", typeof(int));

// add rows to the datatable
dt.Rows.Add("Mon", 450, 190, 350, 380);
dt.Rows.Add("Tue", 470, 200, 300, 390);
dt.Rows.Add("Wed", 420, 160, 290, 350);
dt.Rows.Add("Thu", 440, 220, 330, 400);
dt.Rows.Add("Fri", 500, 210, 350, 470);
dt.Rows.Add("Sat", 390, 250, 310, 360);

// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();

// add the data series to the data series coltion
flexChart1.Series.Add(series1);

// specify the datasource for the chart
flexChart1.DataSource = dt;

// bind X-axis and Y-axis
flexChart1.BindingX = "Day";
series1.Binding = "High,Low,Open,Close";

// set the chart type to candlestick
flexChart1.ChartType = C1.Chart.ChartType.HighLowOpenClose;
```

Funnel

A funnel chart allows you to represent sequential stages in a linear process. For instance, a sales process that tracks prospects across the stages, such as Sales Prospects, Qualified Prospects, Price Quotes, Negotiations, and Closed Sales.

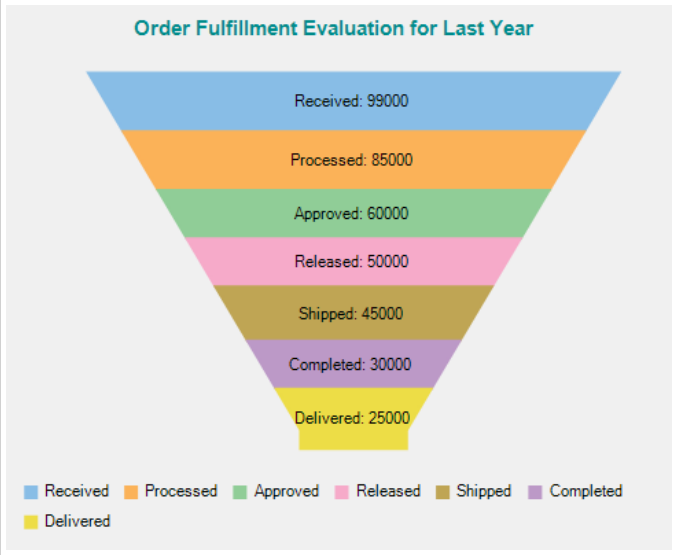
In the process, each stage represents a proportion (percentage) of the total. Therefore, the chart takes the funnel shape with the first stage being the largest and each following stage smaller than the predecessor.

Funnel charts are useful in identifying potential problem areas in processes where it is noticeable at what stages and rate the values decrease.

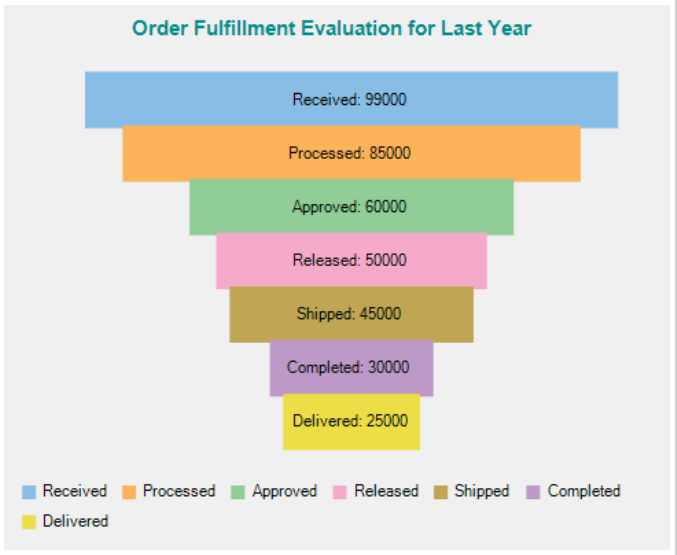
FlexChart offers the Funnel chart in two forms, as follows.

- **Trapezoid chart:** Contains a pair of parallel sides.
- **Stacked Bar chart**: Places related values on top of one another in the form of horizontal bars.

The following images show both Trapezoid and Stacked Bar charts displaying the number of orders across seven stages of an order fulfillment evaluation process.



Trapezoid Chart



Stacked Bar Chart

In FlexChart, use the Funnel chart by setting the ChartType property to **Funnel** from the ChartType enum. Specify the type of the Funnel chart as either Trapezoid or Stacked Bar chart by setting the FunnelType property to Default or Rectangle from the FunnelChartType enum.

In addition, change the dimensions of the neck of the Funnel chart, when set as Trapezoid chart, by setting the FunnelNeckWidth and FunnelNeckHeight properties. These properties are available in the ChartOptions class accessible through the Options property of the FlexChart class.

The following code manually creates data containing values for the amount of orders across seven stages of an order fulfillment process. The snippet sets the chart type as Funnel, specifies the dimensions of the Funnel neck, and sets Header, Legend, and Data Labels of the chart.

- **Visual Basic**

```
' clear the Series collection
FlexChart1.Series.Clear()

' create data
Dim data = New Object() {New With {
      .Name = "Received",
      .Value = 99000
}, New With {
      .Name = "Processed",
      .Value = 85000
}, New With {
      .Name = "Approved",
      .Value = 60000
}, New With {
      .Name = "Released",
      .Value = 50000
}, New With {
      .Name = "Shipped",
      .Value = 45000
}, New With {
      .Name = "Completed",
      .Value = 30000
},
    New With {
      .Name = "Delivered",
      .Value = 25000
}}

' create an instance of Series
Dim series1 As New C1.Win.Chart.Series()

' add the instance to the Series collection
FlexChart1.Series.Add(series1)

' set y-binding
```

```
series1.Binding = "Value"

' set x-binding and add data to the chart
FlexChart1.BindingX = "Name"
FlexChart1.DataSource = data

' set the chart type as Funnel
FlexChart1.ChartType = ChartType.Funnel

' set the Funnel chart type as Default (Trapezoid chart)
FlexChart1.Options.FunnelType = FunnelChartType.[Default]

' set the Funnel neck height
FlexChart1.Options.FunnelNeckHeight = 0.05

' set the Funnel neck width
FlexChart1.Options.FunnelNeckWidth = 0.2
```

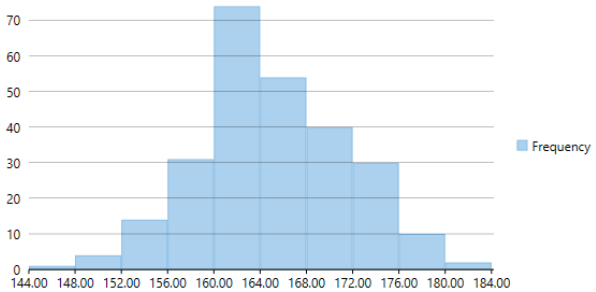
- **C#**

```
// clear the Series collection
flexChart1.Series.Clear();

// create data
var data = new[]
{
    new { Name = "Received", Value = 99000 },
    new { Name = "Processed", Value = 85000 },
    new { Name = "Approved", Value = 60000 },
    new { Name = "Released", Value = 50000 },
    new { Name = "Shipped", Value = 45000 },
    new { Name = "Completed", Value = 30000 },
    new { Name = "Delivered", Value = 25000 }
};

// create an instance of Series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();

// add the instance to the Series collection
flexChart1.Series.Add(series1);

// set y-binding
series1.Binding = "Value";

// set x-binding and add data to the chart
flexChart1.BindingX = "Name";
flexChart1.DataSource = data;

// set the chart type as Funnel
flexChart1.ChartType = ChartType.Funnel;

// set the Funnel chart type as Default (Trapezoid chart)
flexChart1.Options.FunnelType = FunnelChartType.Default;

// set the Funnel neck height
flexChart1.Options.FunnelNeckHeight = 0.05;

// set the Funnel neck width
flexChart1.Options.FunnelNeckWidth = 0.2;
```

Histogram

Histogram chart plots the frequency distribution of data against the defined class intervals or bins. These bins are created by dividing the raw data values into a series of consecutive and non-overlapping intervals. Based on the number of values falling in a particular bin, frequencies are then plotted as rectangular columns against continuous x-axis.

Following image illustrates a classic histogram chart, which depicts frequency distribution of scores obtained by students of a university in half yearly examinations.



Once you provide relevant data and the chart automatically calculates the intervals in which your data is grouped. However, if required, you can also specify the width of these intervals by setting the BinWidth property.

To create a histogram, you need to add the Histogram series and set the ChartType property to **Histogram**.

- **C#**

```
C1.Win.Chart.Histogram histogramSeries;
```

```
public Form1()
{
    InitializeComponent();
    SetupChart();
}
void SetupChart()
{
    flexChart1.Dock = DockStyle.Fill;
    flexChart1.Visible = true;
    flexChart1.BackColor = Color.White;

    flexChart1.BeginUpdate();
    flexChart1.Series.Clear();
    flexChart1.ChartType = ChartType.Histogram;

    histogramSeries = new C1.Win.Chart.Histogram();
    histogramSeries.Name = "Frequency";
    flexChart1.Series.Add(histogramSeries);
```

FlexChart generates frequency distribution for the data and plots the same in histogram, upon providing relevant data as shown in the following code snippet.

- **C#**

```
    flexChart1.DataSource = original;
    flexChart1.Binding = "Y";
    flexChart1.BindingX = "X";
    flexChart1.EndUpdate();
    flexChart1.AxisX.Format = "0.00";
}
```

Data Source

This example uses the following code.

- **C#**

```
var original = new List<Bin>() { new Bin{X=161.2,Y= 51.6}, new Bin{X=167.5,Y= 59.0}, new Bin{X=159.5,Y= 49.2}, new Bin{X=157.0,Y= 63.0}, new Bin{X=155.8,Y= 53.6},
new Bin{X=170.0,Y= 59.0}, new Bin{X=159.1,Y= 47.6}, new Bin{X=166.0,Y= 69.8}, new Bin{X=176.2,Y= 66.8}, new Bin{X=160.2,Y= 75.2},
new Bin{X=172.5,Y= 55.2}, new Bin{X=170.9,Y= 54.2}, new Bin{X=172.9,Y= 62.5}, new Bin{X=153.4,Y= 42.0}, new Bin{X=160.0,Y= 50.0},
new Bin{X=147.2,Y= 49.8}, new Bin{X=168.2,Y= 49.2}, new Bin{X=175.0,Y= 73.2}, new Bin{X=157.0,Y= 47.8}, new Bin{X=167.6,Y= 68.8},
new Bin{X=159.5,Y= 50.6}, new Bin{X=175.0,Y= 82.5}, new Bin{X=166.8,Y= 57.2}, new Bin{X=176.5,Y= 87.8}, new Bin{X=170.2,Y= 72.8},
new Bin{X=174.0,Y= 54.5}, new Bin{X=173.0,Y= 59.8}, new Bin{X=179.9,Y= 67.3}, new Bin{X=170.5,Y= 67.8}, new Bin{X=160.0,Y= 47.0},
new Bin{X=154.4,Y= 46.2}, new Bin{X=162.0,Y= 55.0}, new Bin{X=176.5,Y= 83.0}, new Bin{X=160.0,Y= 54.4}, new Bin{X=152.0,Y= 45.8},
new Bin{X=162.1,Y= 53.6}, new Bin{X=170.0,Y= 73.2}, new Bin{X=160.2,Y= 52.1}, new Bin{X=161.3,Y= 67.9}, new Bin{X=166.4,Y= 56.6},
    new Bin{X=168.9,Y= 62.3}, new Bin{X=163.8,Y= 58.5}, new Bin{X=167.6,Y= 54.5}, new Bin{X=160.0,Y= 50.2}, new Bin{X=161.3,Y= 60.3},
    new Bin{X=167.6,Y= 58.3}, new Bin{X=165.1,Y= 56.2}, new Bin{X=160.0,Y= 50.2}, new Bin{X=170.0,Y= 72.9}, new Bin{X=157.5,Y= 59.8},
    new Bin{X=167.6,Y= 61.0}, new Bin{X=160.7,Y= 69.1}, new Bin{X=163.2,Y= 55.9}, new Bin{X=152.4,Y= 46.5}, new Bin{X=157.5,Y= 54.3},
    new Bin{X=168.3,Y= 54.8}, new Bin{X=180.3,Y= 60.7}, new Bin{X=165.5,Y= 60.0}, new Bin{X=165.0,Y= 62.0}, new Bin{X=164.5,Y= 60.3},
    new Bin{X=156.0,Y= 52.7}, new Bin{X=160.0,Y= 74.3}, new Bin{X=163.0,Y= 62.0}, new Bin{X=165.7,Y= 73.1}, new Bin{X=161.0,Y= 80.0},
    new Bin{X=162.0,Y= 54.7}, new Bin{X=166.0,Y= 53.2}, new Bin{X=174.0,Y= 75.7}, new Bin{X=172.7,Y= 61.1}, new Bin{X=167.6,Y= 55.7},
    new Bin{X=151.1,Y= 48.7}, new Bin{X=164.5,Y= 52.3}, new Bin{X=163.5,Y= 50.0}, new Bin{X=152.0,Y= 59.3}, new Bin{X=169.0,Y= 62.5},
    new Bin{X=164.0,Y= 55.7}, new Bin{X=161.2,Y= 54.8}, new Bin{X=155.0,Y= 45.9}, new Bin{X=170.0,Y= 70.6}, new Bin{X=176.2,Y= 67.2},
    new Bin{X=170.0,Y= 69.4}, new Bin{X=162.5,Y= 58.2}, new Bin{X=170.3,Y= 64.8}, new Bin{X=164.1,Y= 71.6}, new Bin{X=169.5,Y= 52.8},
    new Bin{X=163.2,Y= 59.8}, new Bin{X=154.5,Y= 49.0}, new Bin{X=159.8,Y= 50.0}, new Bin{X=173.2,Y= 69.2}, new Bin{X=170.0,Y= 55.9},
    new Bin{X=161.4,Y= 63.4}, new Bin{X=169.0,Y= 58.2}, new Bin{X=166.2,Y= 58.6}, new Bin{X=159.4,Y= 45.7}, new Bin{X=162.5,Y= 52.2},
    new Bin{X=159.0,Y= 48.6}, new Bin{X=162.8,Y= 57.8}, new Bin{X=159.0,Y= 55.6}, new Bin{X=179.8,Y= 66.8}, new Bin{X=162.9,Y= 59.4},
    new Bin{X=161.0,Y= 53.6}, new Bin{X=151.1,Y= 73.2}, new Bin{X=168.2,Y= 53.4}, new Bin{X=168.9,Y= 69.0}, new Bin{X=173.2,Y= 58.4},
    new Bin{X=171.8,Y= 56.2}, new Bin{X=178.0,Y= 70.6}, new Bin{X=164.3,Y= 59.8}, new Bin{X=163.0,Y= 72.0}, new Bin{X=168.5,Y= 65.2},
    new Bin{X=166.8,Y= 56.6}, new Bin{X=172.7,Y= 105.2}, new Bin{X=163.5,Y= 51.8}, new Bin{X=169.4,Y= 63.4}, new Bin{X=167.8,Y= 59.0},
    new Bin{X=159.5,Y= 47.6}, new Bin{X=167.6,Y= 63.0}, new Bin{X=161.2,Y= 55.2}, new Bin{X=160.0,Y= 45.0}, new Bin{X=163.2,Y= 54.0},
    new Bin{X=162.2,Y= 50.2}, new Bin{X=161.3,Y= 60.2}, new Bin{X=149.5,Y= 44.8}, new Bin{X=157.5,Y= 58.8}, new Bin{X=163.2,Y= 56.4},
    new Bin{X=172.7,Y= 62.0}, new Bin{X=155.0,Y= 49.2}, new Bin{X=156.5,Y= 67.2}, new Bin{X=164.0,Y= 53.8}, new Bin{X=160.9,Y= 54.4},
    new Bin{X=162.8,Y= 58.0}, new Bin{X=167.0,Y= 59.8}, new Bin{X=160.0,Y= 54.8}, new Bin{X=160.0,Y= 43.2}, new Bin{X=168.9,Y= 60.5},
    new Bin{X=158.2,Y= 46.4}, new Bin{X=156.0,Y= 64.4}, new Bin{X=160.0,Y= 48.8}, new Bin{X=167.1,Y= 62.2}, new Bin{X=158.0,Y= 55.5},
    new Bin{X=167.6,Y= 57.8}, new Bin{X=156.0,Y= 54.6}, new Bin{X=162.1,Y= 59.2}, new Bin{X=173.4,Y= 52.7}, new Bin{X=159.8,Y= 53.2},
    new Bin{X=170.5,Y= 64.5}, new Bin{X=159.2,Y= 51.8}, new Bin{X=157.5,Y= 56.0}, new Bin{X=161.3,Y= 63.6}, new Bin{X=162.6,Y= 63.2},
    new Bin{X=160.0,Y= 59.5}, new Bin{X=168.9,Y= 56.8}, new Bin{X=165.1,Y= 64.1}, new Bin{X=162.6,Y= 50.0}, new Bin{X=165.1,Y= 72.3},
    new Bin{X=166.4,Y= 55.0}, new Bin{X=160.0,Y= 55.9}, new Bin{X=152.4,Y= 60.4}, new Bin{X=170.2,Y= 69.1}, new Bin{X=162.6,Y= 84.5},
    new Bin{X=170.2,Y= 55.9}, new Bin{X=158.8,Y= 55.5}, new Bin{X=172.7,Y= 69.5}, new Bin{X=167.6,Y= 76.4}, new Bin{X=162.6,Y= 61.4},
    new Bin{X=167.6,Y= 65.9}, new Bin{X=156.2,Y= 58.6}, new Bin{X=175.2,Y= 66.8}, new Bin{X=172.1,Y= 56.6}, new Bin{X=162.6,Y= 58.6},
    new Bin{X=160.0,Y= 55.9}, new Bin{X=165.1,Y= 59.1}, new Bin{X=182.9,Y= 81.8}, new Bin{X=166.4,Y= 70.7}, new Bin{X=165.1,Y= 56.8},
    new Bin{X=177.8,Y= 60.0}, new Bin{X=165.1,Y= 58.2}, new Bin{X=175.3,Y= 72.7}, new Bin{X=154.9,Y= 54.1}, new Bin{X=158.8,Y= 49.1},
    new Bin{X=172.7,Y= 75.9}, new Bin{X=168.9,Y= 55.0}, new Bin{X=161.3,Y= 57.3}, new Bin{X=167.6,Y= 55.0}, new Bin{X=165.1,Y= 65.5},
    new Bin{X=175.3,Y= 65.5}, new Bin{X=157.5,Y= 48.6}, new Bin{X=163.8,Y= 58.6}, new Bin{X=167.6,Y= 63.6}, new Bin{X=165.1,Y= 55.2},
    new Bin{X=165.1,Y= 62.7}, new Bin{X=168.9,Y= 56.6}, new Bin{X=162.6,Y= 53.9}, new Bin{X=164.5,Y= 63.2}, new Bin{X=176.5,Y= 73.6},
    new Bin{X=168.9,Y= 62.0}, new Bin{X=175.3,Y= 63.6}, new Bin{X=159.4,Y= 53.2}, new Bin{X=160.0,Y= 53.4}, new Bin{X=170.2,Y= 55.0},
    new Bin{X=162.6,Y= 70.5}, new Bin{X=167.6,Y= 54.5}, new Bin{X=162.6,Y= 54.5}, new Bin{X=160.7,Y= 55.9}, new Bin{X=160.0,Y= 59.0},
    new Bin{X=157.5,Y= 63.6}, new Bin{X=162.6,Y= 54.5}, new Bin{X=152.4,Y= 47.3}, new Bin{X=170.2,Y= 67.7}, new Bin{X=165.1,Y= 80.9},
    new Bin{X=172.7,Y= 70.5}, new Bin{X=165.1,Y= 60.9}, new Bin{X=170.2,Y= 63.6}, new Bin{X=170.2,Y= 54.5}, new Bin{X=170.2,Y= 59.1},
    new Bin{X=161.3,Y= 70.5}, new Bin{X=167.6,Y= 52.7}, new Bin{X=167.6,Y= 62.7}, new Bin{X=165.1,Y= 86.3}, new Bin{X=162.6,Y= 66.4},
    new Bin{X=152.4,Y= 67.3}, new Bin{X=168.9,Y= 63.0}, new Bin{X=170.2,Y= 73.6}, new Bin{X=175.2,Y= 62.3}, new Bin{X=175.2,Y= 57.7},
    new Bin{X=160.0,Y= 55.4}, new Bin{X=165.1,Y= 104.1}, new Bin{X=174.0,Y= 55.5}, new Bin{X=170.2,Y= 77.3}, new Bin{X=160.0,Y= 80.5},
    new Bin{X=167.6,Y= 64.5}, new Bin{X=167.6,Y= 72.3}, new Bin{X=167.6,Y= 61.4}, new Bin{X=154.9,Y= 58.2}, new Bin{X=162.6,Y= 81.8},
    new Bin{X=175.3,Y= 63.6}, new Bin{X=171.4,Y= 53.4}, new Bin{X=157.5,Y= 54.5}, new Bin{X=165.1,Y= 53.6}, new Bin{X=160.0,Y= 60.0},
    new Bin{X=174.0,Y= 73.6}, new Bin{X=162.6,Y= 61.4}, new Bin{X=174.0,Y= 55.5}, new Bin{X=162.6,Y= 63.6}, new Bin{X=161.3,Y= 60.9},
    new Bin{X=156.2,Y= 60.0}, new Bin{X=149.9,Y= 46.8}, new Bin{X=169.5,Y= 57.3}, new Bin{X=160.0,Y= 64.1}, new Bin{X=175.3,Y= 63.6},
    new Bin{X=169.5,Y= 67.3}, new Bin{X=160.0,Y= 75.5}, new Bin{X=172.7,Y= 68.2}, new Bin{X=162.6,Y= 61.4}, new Bin{X=157.5,Y= 76.8},
    new Bin{X=176.5,Y= 71.8}, new Bin{X=164.4,Y= 55.5}, new Bin{X=160.7,Y= 48.6}, new Bin{X=174.0,Y= 66.4}, new Bin{X=163.8,Y= 67.3}
};
```

Note that x-axis of Histogram chart can be shared by other chart series, which can be displayed together with the classic histogram series.

Back to Top

Line

The Line Chart displays trends over a period of time by connecting different data points in a series with a straight line. It treats the input as categorical information that is evenly spaced along the X-axis.

You can create the Line Chart by setting the ChartType property to Line either at design-time or in code behind.

To create the stacking Line Chart, you need to set the Stacking property to Stacked or Stacked100pc.

Let there be a school XYZ with an initial strength of 3000 in 2011. The management of the school decides to find out the maximum, minimum, and average number of students in the school from 2011 to 2015. In addition, the management needs to know the time period in which the largest number of students left the school.

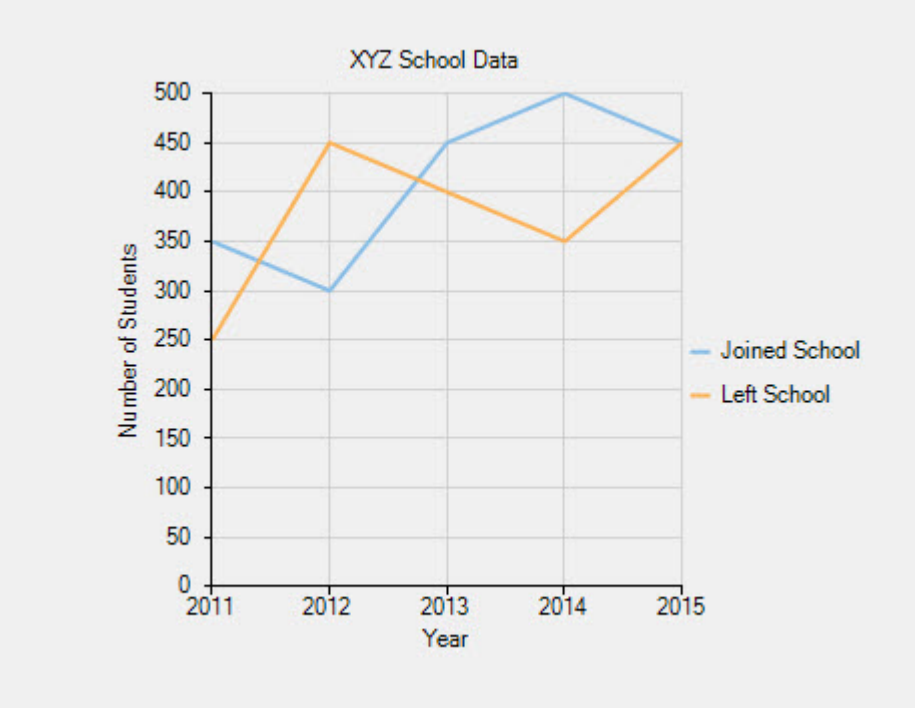
Let us visualize the given data by using the Line Chart.

Sample Data Table

Year	Joined School	Left School
2011	350	250
2012	300	450
2013	450	400
2014	500	350
2015	450	450

Important Points

Line Chart



The above chart displays the number of students who joined and left the school from 2011 to 2015.

- Number of series: two (Joined School and Left School)
- Number of Y values per point: one

Below is the code that implements the aforementioned example:

- **Visual Basic**

```
' clear data series collection
FlexChart1.Series.Clear()
```

```
' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()

' add data points into the data series
series1.BindingX = "X"
series1.Binding = "Y"
series1.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2011, 350),
New System.Drawing.Point(2012, 300),
New System.Drawing.Point(2013, 450),
New System.Drawing.Point(2014, 500),
New System.Drawing.Point(2015, 450)}
series1.Name = "Joined School"

series2.BindingX = "X"
series2.Binding = "Y"
series2.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2011, 250),
New System.Drawing.Point(2012, 450),
New System.Drawing.Point(2013, 400),
New System.Drawing.Point(2014, 350),
New System.Drawing.Point(2015, 450)}
series2.Name = "Left School"

' add the data series to the data series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)

' set the chart type to line
FlexChart1.ChartType = C1.Chart.ChartType.Line
```

- **C#**

```
// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();

// add data points into the data series
series1.BindingX = "X";
series1.Binding = "Y";
series1.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2011,350),
new System.Drawing.Point(2012,300),
new System.Drawing.Point(2013,450),
new System.Drawing.Point(2014,500),
new System.Drawing.Point(2015,450)};
series1.Name = "Joined School";

series2.BindingX = "X";
series2.Binding = "Y";
series2.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2011,250),
new System.Drawing.Point(2012,450),
new System.Drawing.Point(2013,400),
new System.Drawing.Point(2014,350),
new System.Drawing.Point(2015,450)};
series2.Name = "Left School";
```

```
// add the data series to the data series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);

// set the chart type to line
flexChart1.ChartType = C1.Chart.ChartType.Line;
```

LineSymbols

The LineSymbols Chart is a combination of the Line Chart and the Scatter Chart. The chart displays trends in data at equal intervals and visualizes relationship between two variables related to the same event. It plots data points by using symbols and connects the data points by using straight lines.

You need to set the ChartType property to LineSymbols either at design-time or at run-time to create the LineSymbols Chart.

You can set the Stacking property to Stacked or Stacked100pc to create the stacking LineSymbols Chart.

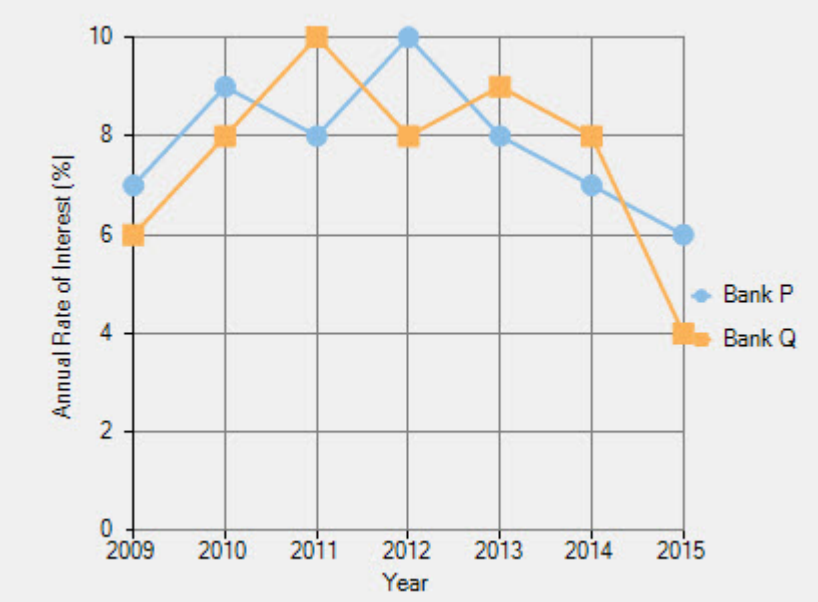
Let's take a scenario in which two banks P and Q declare their fixed annual rates of interest on the amounts invested with them by investors for the period 2009-2015. The rates of interest offered by these banks differ from year to year based on the variations in the economy of the country.

Now, we need to represent the annual rates of interest (in %) offered by these banks visually. And we are going to use the LineSymbols Chart to do the same.

Sample Data Table

Bank	2009	2010	2011	2012	2013	2014	2015
P	7	9	8	10	8	7	6
Q	6	8	10	8	9	8	4

LineSymbols Chart



The above chart displays the annual rates of interest offered by two banks from 2009 to 2015.

- Number of series: two (P and Q)
- Number of Y values per point: one

Below is the implementation in code:

- **Visual Basic**

```
' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()

' add data points into the data series
series1.BindingX = "X"
series1.Binding = "Y"
series1.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2009, 7),
New System.Drawing.Point(2010, 9),
New System.Drawing.Point(2011, 8),
New System.Drawing.Point(2012, 10),
New System.Drawing.Point(2013, 8),
New System.Drawing.Point(2014, 7),
New System.Drawing.Point(2015, 6)}
series1.Name = "Bank P"

series2.BindingX = "X"
series2.Binding = "Y"
series2.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(2009, 6),
New System.Drawing.Point(2010, 8),
New System.Drawing.Point(2011, 10),
New System.Drawing.Point(2012, 8),
New System.Drawing.Point(2013, 9),
New System.Drawing.Point(2014, 8),
New System.Drawing.Point(2015, 4)}
series2.Name = "Bank Q"

' add the data series to the data series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)

' set the chart type to linesymbols
FlexChart1.ChartType = C1.Chart.ChartType.LineSymbols
```

- **C#**

```
// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();

// add data points into the data series
series1.BindingX = "X";
series1.Binding = "Y";
series1.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2009,7),
new System.Drawing.Point(2010,9),
new System.Drawing.Point(2011,8),
```

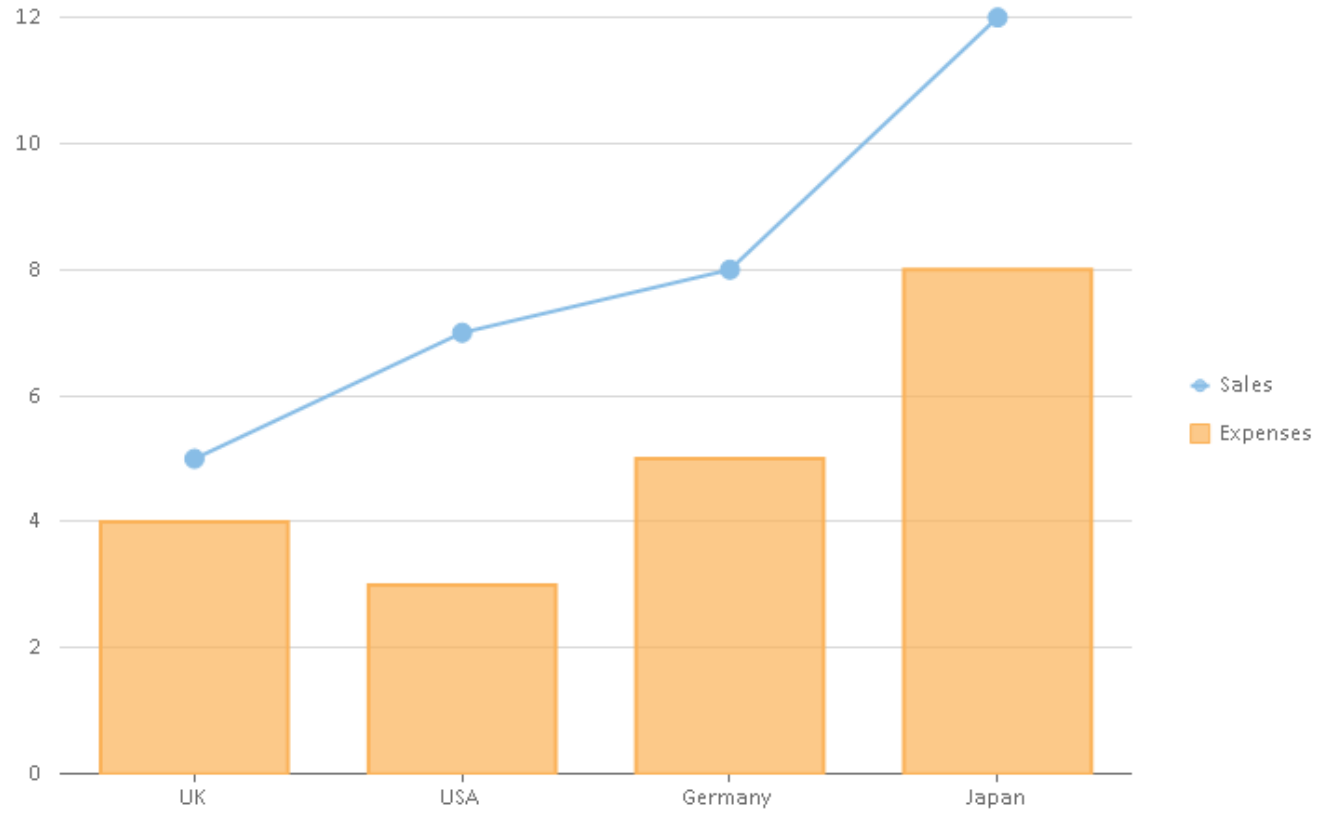
```
new System.Drawing.Point(2012,10),
new System.Drawing.Point(2013,8),
new System.Drawing.Point(2014,7),
new System.Drawing.Point(2015,6)};
series1.Name = "Bank P";

series2.BindingX = "X";
series2.Binding = "Y";
series2.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(2009,6),
new System.Drawing.Point(2010,8),
new System.Drawing.Point(2011,10),
new System.Drawing.Point(2012,8),
new System.Drawing.Point(2013,9),
new System.Drawing.Point(2014,8),
new System.Drawing.Point(2015,4)};
series2.Name = "Bank Q";

// add the data series to the data series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);

// set the chart type to linesymbols
flexChart1.ChartType = C1.Chart.ChartType.LineSymbols;
```

Mixed

FlexChart allows you to create mixed charts that offer two key advantages, as follows:

- **Combining chart types:** Combine two or more chart types in a single chart, for instance, area-bar, bar-line, bar-scatter etc. Plot different metrics in a chart using different chart types and let the end user interpret data easily. In FlexChart, specify a chart type for each series to combine several chart types. To specify the chart type for a series, set the ChartType property of the SeriesBase class. Setting this property overrides the ChartType property set for the chart.
- **Plotting multiple datasets:** Plot data from multiple datasets in a single chart by specifying data source for a series. This is useful when the data to plot lies at multiple places. To specify the data source for a series, set the DataSource property of the Series class. Setting this property overrides the DataSource property set for the chart.

The following image displays a mixed chart that combines column and line symbols chart types. The chart plots and compares sales and expenses data of four countries.



The following code sets the Column chart type for FlexChart and overrides it by setting the LineSymbols chart type for the Sales series, thereby implementing mixed charts.

- **Visual Basic**

```
FlexChart1.Series.Clear()

' Add data series
Dim s1 = New Series()
s1.Binding = InlineAssignHelper(s1.Name, "Sales")
s1.ChartType = C1.Chart.ChartType.LineSymbols
FlexChart1.Series.Add(s1)

Dim s2 = New Series()
s2.Binding = InlineAssignHelper(s2.Name, "Expenses")
FlexChart1.Series.Add(s2)

' Set x-binding and add data to the chart
FlexChart1.BindingX = "Country"
FlexChart1.ChartType = C1.Chart.ChartType.Column
FlexChart1.DataSource = New () {New With {
    Key .Country = "UK",
    Key .Sales = 5,
    Key .Expenses = 4
}, New With {
    Key .Country = "USA",
    Key .Sales = 7,
    Key .Expenses = 3
}, New With {
    Key .Country = "Germany",
    Key .Sales = 8,
    Key .Expenses = 5
}, New With {
```

```
    Key .Country = "Japan",
    Key .Sales = 12,
    Key .Expenses = 8
}}
```

- **C#**

```
flexChart1.Series.Clear();

// Add data series
var s1 = new Series();
s1.Binding = s1.Name = "Sales";
s1.ChartType = C1.Chart.ChartType.LineSymbols;
flexChart1.Series.Add(s1);

var s2 = new Series();
s2.Binding = s2.Name = "Expenses";
flexChart1.Series.Add(s2);

// Set x-binding and add data to the chart
flexChart1.BindingX = "Country";
flexChart1.ChartType = C1.Chart.ChartType.Column;
flexChart1.DataSource = new[]
{
    new { Country = "UK", Sales = 5, Expenses = 4},
    new { Country = "USA", Sales = 7, Expenses = 3},
    new { Country = "Germany", Sales = 8, Expenses = 5},
    new { Country = "Japan", Sales = 12, Expenses = 8},
};
```

RangedHistogram

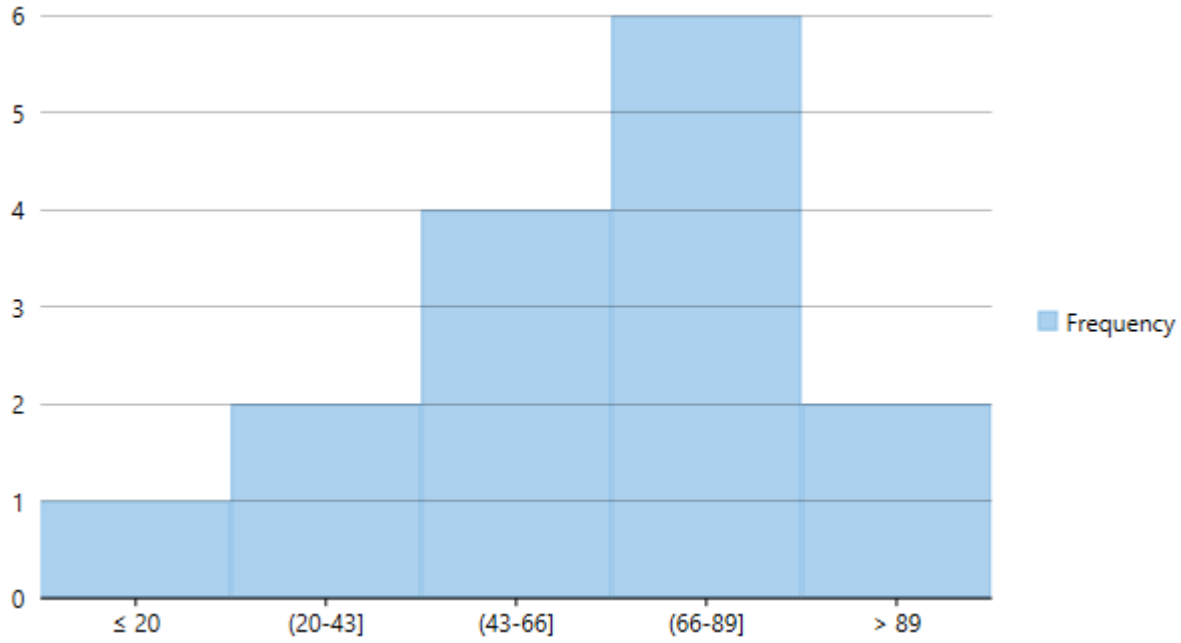
RangedHistogram is a modern Excel-like histogram chart that helps visualize frequency distribution on y axis, against **ranged** x axis. Like Histogram chart type, bins are created by dividing the raw data values into a series of consecutive, non-overlapping intervals. Based on the number of values falling in a particular bin, frequencies are then plotted as rectangular columns against x-axis.

RangedHistogram plots frequency distribution for the provided data in non-category and category modes.

Non-Category Mode

In non-category mode, the original data points are binned into intervals or ranges. These intervals are then plotted on x-axis, and y-axis shows frequency distribution for respective ranges. FlexChart automatically calculates the intervals in which your data is grouped.
However, you can control this behavior by specifying the HistogramBinning through BinMode property. Moreover, you can further set BinWidth, NumberOfBins, values for UnderflowBin and OverflowBin, and specify whether to ShowUnderflowBin and ShowOverflowBin.

The following image illustrates frequency distribution for units sold of various products of a retail store in non-category mode.



To create a RangedHistogram, for a given data in non-category mode, you need to add the RangedHistogram series and set the ChartType property to **RangedHistogram**, as shown in the following code snippet.

- ***C#***

```
public partial class RHistogram : Form
{

    C1.Win.Chart.RangedHistogram histogramRSeries;
    public RHistogram()
    {
        InitializeComponent();
        SetRangedChart();
    }

    void SetRangedChart()
    {
        flexChart2.Dock = DockStyle.Fill;
        flexChart2.Visible = true;
        flexChart2.BackColor = Color.White;

        flexChart2.BeginUpdate();
        flexChart2.Series.Clear();
        flexChart2.ChartType = ChartType.RangedHistogram;

        histogramRSeries = new C1.Win.Chart.RangedHistogram();
        histogramRSeries.Name = "Frequency";
        flexChart2.Series.Add(histogramRSeries);

        flexChart2.DataSource = new[]
        {
             new { Name = "Stationery", Value = 20 },
                new { Name = "Books", Value = 35 },
                new { Name = "Toys and Games", Value = 40 },
                new { Name = "Stationery", Value = 55 },
                new { Name = "Books", Value = 80 },
                new { Name = "Toys and Games", Value = 60 },
                new { Name = "Stationery", Value = 61 },
                new { Name = "Books", Value = 85 },
```

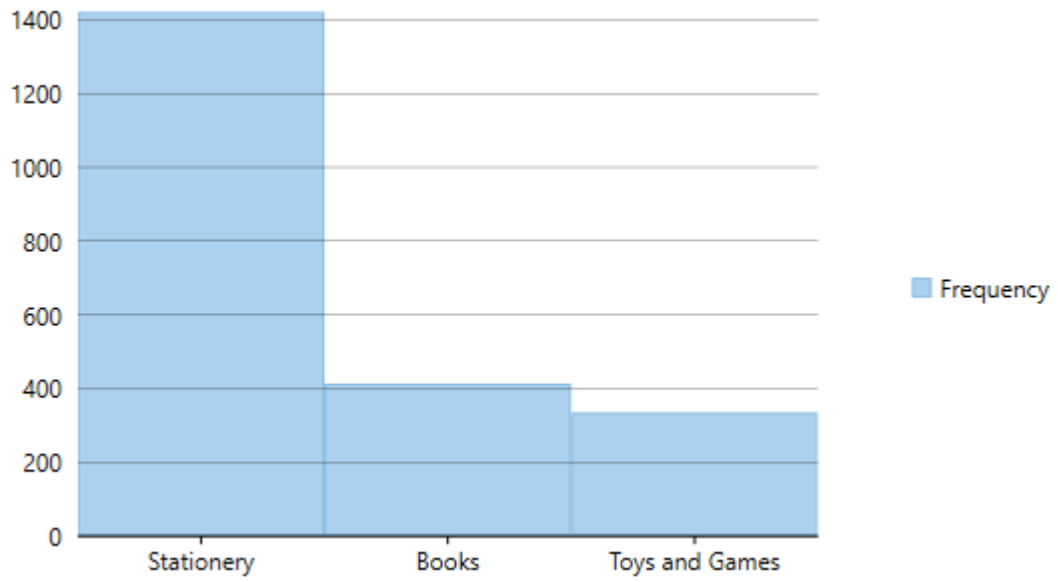
```
                new { Name = "Toys and Games", Value = 80 },
                new { Name = "Stationery", Value = 64 },
                new { Name = "Books", Value = 80 },
                new { Name = "Toys and Games", Value = 75 },
                new { Name = "Stationery", Value = 1222 },
                new { Name = "Books", Value = 133 },
                new { Name = "Toys and Games", Value = 80 },
            };

            flexChart2.Binding = "Value";
            histogramRSeries.BinMode = HistogramBinning.NumberOfBins;
            histogramRSeries.NumberOfBins = 5;
            histogramRSeries.OverflowBin = 89;
            histogramRSeries.UnderflowBin = 20;
            histogramRSeries.ShowOverflowBin = true;
            histogramRSeries.ShowUnderflowBin = true;
            flexChart2.EndUpdate();
        }

}
```

Category Mode

In Category mode, frequency data is exclusively grouped in categories (which are plotted on x-axis) as provided by the original data and y-axis depicts cumulative frequency for the respective categories. Category mode is enabled for RangedHistogram series by setting the BindingX property.

In this mode, BinMode, BinWidth, NumberOfBins, OverflowBin, and UnderflowBin properties for RangedHistogram series are ignored.

The following image illustrates frequency distribution for units sold of 3 categories of products Stationery items, Books, and Toys and Games of a retail store in category mode.



To create a RangedHistogram for a given data in category mode, you need to add the RangedHistogram series, set the ChartType property to **RangedHistogram** and set the BindingX property, as shown in the following code snippet.

- **C#**

```
public partial class RHistogram : Form
{
```

```
    C1.Win.Chart.RangedHistogram histogramRSeries;
    public RHistogram()
    {
        InitializeComponent();
        SetRangedChart();
    }

    void SetRangedChart()
    {
        flexChart2.Dock = DockStyle.Fill;
        flexChart2.Visible = true;
        flexChart2.BackColor = Color.White;

        flexChart2.BeginUpdate();
        flexChart2.Series.Clear();
        flexChart2.ChartType = ChartType.RangedHistogram;

        histogramRSeries = new C1.Win.Chart.RangedHistogram();
        histogramRSeries.Name = "Frequency";
        flexChart2.Series.Add(histogramRSeries);

        flexChart2.DataSource = new[]
        {
             new { Name = "Stationery", Value = 20 },
                new { Name = "Books", Value = 35 },
                new { Name = "Toys and Games", Value = 40 },
                new { Name = "Stationery", Value = 55 },
                new { Name = "Books", Value = 80 },
                new { Name = "Toys and Games", Value = 60 },
                new { Name = "Stationery", Value = 61 },
                new { Name = "Books", Value = 85 },
                new { Name = "Toys and Games", Value = 80 },
                new { Name = "Stationery", Value = 64 },
                new { Name = "Books", Value = 80 },
                new { Name = "Toys and Games", Value = 75 },
                new { Name = "Stationery", Value = 1222 },
                new { Name = "Books", Value = 133 },
                new { Name = "Toys and Games", Value = 80 },
        };

        flexChart2.Binding = "Value";
        flexChart2.BindingX = "Name";
        histogramRSeries.BinMode = HistogramBinning.NumberOfBins;
        histogramRSeries.NumberOfBins = 5;
        histogramRSeries.OverflowBin = 89;
        histogramRSeries.UnderflowBin = 20;
        histogramRSeries.ShowOverflowBin = true;
        histogramRSeries.ShowUnderflowBin = true;
        flexChart2.EndUpdate();
    }

}
```

Note that unlike traditional Histogram, other chart types cannot be plotted using the same x axis values as RangedHistogram.

Scatter

The Scatter Chart, which is also known as the XY Chart, depicts relationship among items of different data series. In

simple terms, it is a plot of X values and Y values along the two axes. The data points are not connected and can be customized using different symbols. This chart type is normally used to represent scientific data, and can highlight the deviation of assembled data from predicted data or result.

To create the Scatter Chart, you can set the ChartType property to Scatter at design-time as well as run-time.

Set the Stacking property to Stacked or Stacked100pc to create the stacking Scatter Chart.

Suppose that there is an ABC Warehouse that deals in garments and accessories. The inventory department and the sales department of the warehouse try discerning the relationship between the profit earned and amount sold at specific sales amounts in 2013, 2014, and 2015. Apparently, they wish to put a specific limit at the quantity of items purchased at the beginning of a year to prevent the unsold items from getting tampered or wasted at the end of the year.

Let us use the Scatter Chart to depict the required relationship.

Sample Data Table

2013

Profit %	Sales Amount (Lakhs)
43	10
44	20
50	30
35	40
60	50

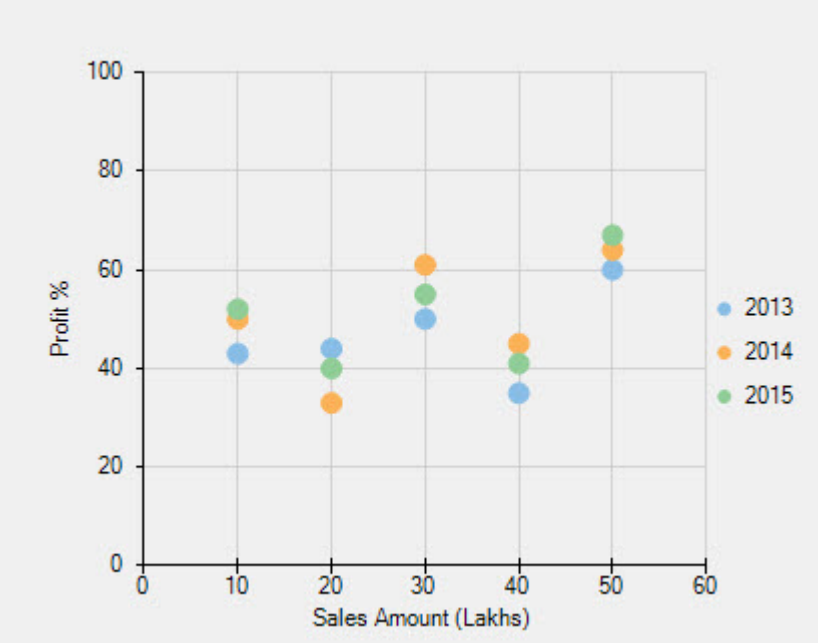
2014

Profit %	Sales Amount (Lakhs)
50	10
33	20
61	30
45	40
64	50

2015

Profit %	Sales Amount (Lakhs)
52	10
40	20
55	30
41	40
67	50

Scatter Chart



The above chart depicts the relationship between profit % and sales amount in three years.

- Number of series: three (2013. 2014, and 2015)
- Number of Y values per point: one

Below is the code implementing the scenario:

- **Visual Basic**

```
' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()
Dim series3 As New C1.Win.Chart.Series()

' add data points into the data series
series1.BindingX = "X"
series1.Binding = "Y"
series1.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(10, 43),
New System.Drawing.Point(20, 44),
New System.Drawing.Point(30, 50),
New System.Drawing.Point(40, 35),
New System.Drawing.Point(50, 60)}
series1.Name = "2013"

series2.BindingX = "X"
series2.Binding = "Y"
series2.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(10, 50),
New System.Drawing.Point(20, 33),
New System.Drawing.Point(30, 61),
New System.Drawing.Point(40, 45),
New System.Drawing.Point(50, 64)}
series2.Name = "2014"

series3.BindingX = "X"
series3.Binding = "Y"
```

```
series3.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(10, 52),
New System.Drawing.Point(20, 40),
New System.Drawing.Point(30, 55),
New System.Drawing.Point(40, 41),
New System.Drawing.Point(50, 67)}
series3.Name = "2015"

' add the data series to the data series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
FlexChart1.Series.Add(series3)

' set the chart type to scatter
FlexChart1.ChartType = C1.Chart.ChartType.Scatter
```

- **C#**

```
// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series3 = new C1.Win.Chart.Series();

// add data points into the data series
series1.BindingX = "X";
series1.Binding = "Y";
series1.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(10,43),
new System.Drawing.Point(20,44),
new System.Drawing.Point(30,50),
new System.Drawing.Point(40,35),
new System.Drawing.Point(50,60)};
series1.Name = "2013";

series2.BindingX = "X";
series2.Binding = "Y";
series2.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(10,50),
new System.Drawing.Point(20,33),
new System.Drawing.Point(30,61),
new System.Drawing.Point(40,45),
new System.Drawing.Point(50,64)};
series2.Name = "2014";

series3.BindingX = "X";
series3.Binding = "Y";
series3.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(10,52),
new System.Drawing.Point(20,40),
new System.Drawing.Point(30,55),
new System.Drawing.Point(40,41),
new System.Drawing.Point(50,67)};
series3.Name = "2015";

// add the data series to the data series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);
flexChart1.Series.Add(series3);
```

```
// set the chart type to scatter
flexChart1.ChartType = C1.Chart.ChartType.Scatter;
```

Spline

The Spline Chart is similar to the line chart except that it connects data points by using splines rather than straight lines. The chart is used as an alternative to the line chart, but more specifically for representing data that requires the use of curve fittings.

You need to set the ChartType property to Spline either from the Properties window or in code behind to create the Spline Chart.

You can set the Stacking property to Stacked or Stacked100pc to create the stacking Spline Chart.

Consider that you are the sales analyst of a company ABC that releases its annual sales report every year. Being the sales analyst, you are asked to compare the sales of last two years, thereby creating a sales comparison report. The report needs to show the increasing/decreasing trends of the sales in the two years based upon the given sales data.

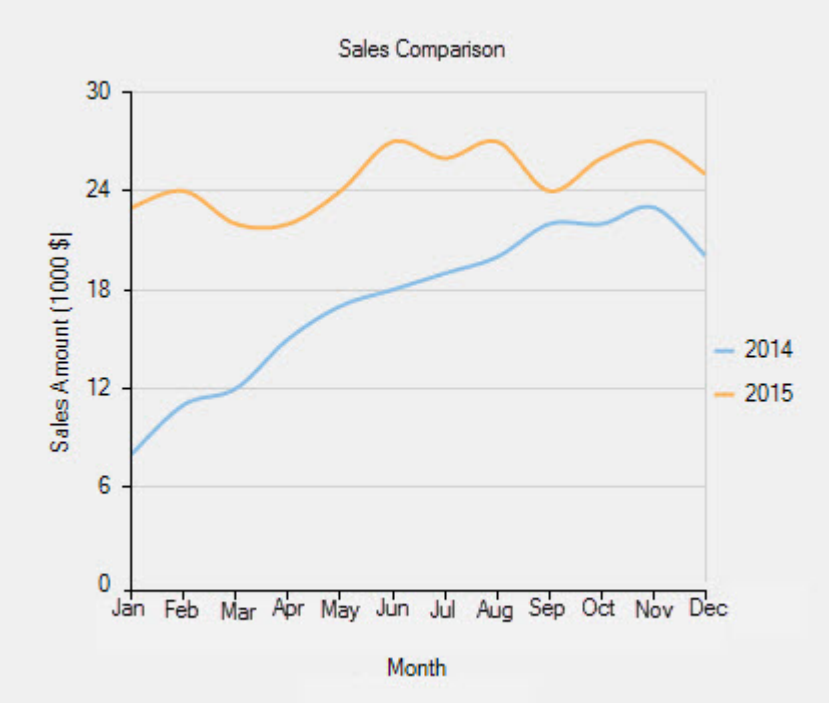
You can use the Spline Chart to create the required sales comparison report.

Sample Data Table

Month	Sales in 2014 (1000 $)	Sales in 2015 (1000 $)
Jan	8	23
Feb	11	24
Mar	12	22
Apr	15	22
May	17	24
Jun	18	27
Jul	19	26
Aug	20	27
Sep	22	24
Oct	22	26
Nov	23	27
Dec	20	25

Important Points

Spline Chart



The above chart shows comparison between ABC sales for 2014 and 2015.

- Number of series: two (2014 and 2015)
- Number of Y values per point: one

Here is the code demonstrating the implementation:

- **Visual Basic**

```
' create a datatable
Dim dt As New DataTable("Sales - 2014 vs 2015")

' add columns to the datatable
dt.Columns.Add("Month", GetType(String))
dt.Columns.Add("Sales in 2014", GetType(Integer))
dt.Columns.Add("Sales in 2015", GetType(Integer))

' add rows to the datatable
dt.Rows.Add("Jan", 8, 23)
dt.Rows.Add("Feb", 11, 24)
dt.Rows.Add("Mar", 12, 22)
dt.Rows.Add("Apr", 15, 22)
dt.Rows.Add("May", 17, 24)
dt.Rows.Add("Jun", 18, 27)
dt.Rows.Add("Jul", 19, 26)
dt.Rows.Add("Aug", 20, 27)
dt.Rows.Add("Sep", 22, 24)
dt.Rows.Add("Oct", 22, 26)
dt.Rows.Add("Nov", 23, 27)
dt.Rows.Add("Dec", 20, 25)

' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()
```

```
' add the data series to the data series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)

' name the series
series1.Name = "2014"
series2.Name = "2015"

' specify the datasource for the chart
FlexChart1.DataSource = dt

' bind X-axis and Y-axis
FlexChart1.BindingX = "Month"
series1.Binding = "Sales in 2014"
series2.Binding = "Sales in 2015"

' set the chart type to spline
FlexChart1.ChartType = C1.Chart.ChartType.Spline
```

- **C#**

```
// create a datatable
DataTable dt = new DataTable("Sales - 2014 vs 2015");

// add columns to the datatable
dt.Columns.Add("Month", typeof(string));
dt.Columns.Add("Sales in 2014", typeof(int));
dt.Columns.Add("Sales in 2015", typeof(int));

// add rows to the datatable
dt.Rows.Add("Jan", 8, 23);
dt.Rows.Add("Feb", 11, 24);
dt.Rows.Add("Mar", 12, 22);
dt.Rows.Add("Apr", 15, 22);
dt.Rows.Add("May", 17, 24);
dt.Rows.Add("Jun", 18, 27);
dt.Rows.Add("Jul", 19, 26);
dt.Rows.Add("Aug", 20, 27);
dt.Rows.Add("Sep", 22, 24);
dt.Rows.Add("Oct", 22, 26);
dt.Rows.Add("Nov", 23, 27);
dt.Rows.Add("Dec", 20, 25);

// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();

// add the data series to the data series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);

// name the series
series1.Name = "2014";
series2.Name = "2015";

// specify the datasource for the chart
flexChart1.DataSource = dt;
```

```
// bind X-axis and Y-axis
flexChart1.BindingX = "Month";
series1.Binding = "Sales in 2014";
series2.Binding = "Sales in 2015";

// set the chart type to spline
flexChart1.ChartType = C1.Chart.ChartType.Spline;
```

SplineArea

The SplineArea chart is just like the area chart with the only difference in the manner in which data points are connected. The SplineArea chart connects data points by using splines instead of straight lines, and fills the area enclosed by the splines.

You can set the ChartType property to SplineArea either at design-time or in code to create the SplineArea Chart.

To create the stacking SplineArea Chart, set the Stacking property to Stacked or Stacked100pc.

At the end of 2014, an international business magazine released a report ranking the topmost automobiles of the year on the basis of their resale values. The higher the resale value of an automobile, the better the ranking of the automobile. The same magazine released another report at the end of 2015 listing the best automobiles of 2015 on the basis of their resale values. There were a few vehicles that were common in both the reports.

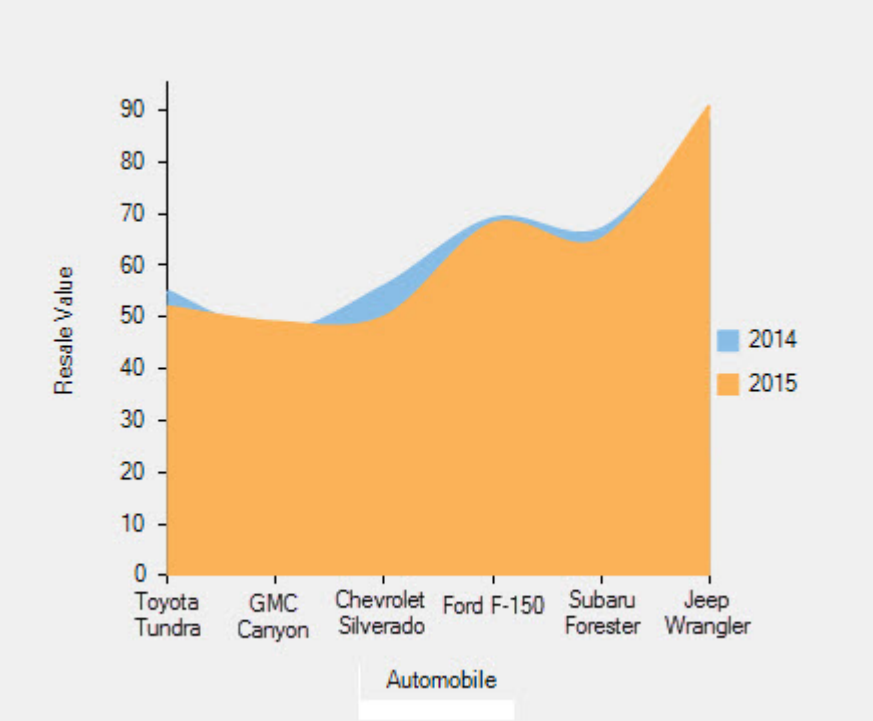
Now, you need to compare the resale values of those vehicles in 2014 with that of 2015 with the help of the given data.

Let us do the same by using the SplineArea Chart.

Sample Data Table

Automobile	Resale Value in 2014 (%)	Resale Value in 2015 (%)
Toyota Tundra	55	52
GMC Canyon	47	49
Chevrolet Silverado	56	50
Ford F-150	69	68
Subaru Forester	67	65
Jeep Wrangler	88	91

SplineArea Chart



The above chart represents comparison in the resale values of specific automobiles in 2014 with that of 2015.

- Number of series: two (2014 and 2015)
- Number of Y values per point: one

Here is the implementation in code:

- **Visual Basic**

```
' create a datatable
Dim dt As New DataTable("Resale Value 2014 vs 2015")

' add columns to the datatable
dt.Columns.Add("Automobile", GetType(String))
dt.Columns.Add("Resale Value in 2014", GetType(Integer))
dt.Columns.Add("Resale Value in 2015", GetType(Integer))

' add rows to the datatable
dt.Rows.Add("Toyota Tundra", 55, 52)
dt.Rows.Add("GMC Canyon", 47, 49)
dt.Rows.Add("Chevrolet Silverado", 56, 50)
dt.Rows.Add("Ford F-150", 69, 68)
dt.Rows.Add("Subaru Forester", 67, 65)
dt.Rows.Add("Jeep Wrangler", 88, 91)

' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()

' add the data series to the data series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)

' specify the datasource for the chart
```

```
FlexChart1.DataSource = dt

' bind the X-axis
FlexChart1.BindingX = "Automobile"

' bind the Y axes
series1.Binding = "Resale Value in 2014"
series2.Binding = "Resale Value in 2015"

' name the series
series1.Name = "2014"
series2.Name = "2015"

' set the chart type to splinearea
FlexChart1.ChartType = C1.Chart.ChartType.SplineArea
```

- **C#**

```
// create a datatable
DataTable dt = new DataTable("Resale Value 2014 vs 2015");

// add columns to the datatable
dt.Columns.Add("Automobile", typeof(string));
dt.Columns.Add("Resale Value in 2014", typeof(int));
dt.Columns.Add("Resale Value in 2015", typeof(int));

// add rows to the datatable
dt.Rows.Add("Toyota Tundra", 55, 52);
dt.Rows.Add("GMC Canyon", 47, 49);
dt.Rows.Add("Chevrolet Silverado", 56, 50);
dt.Rows.Add("Ford F-150", 69, 68);
dt.Rows.Add("Subaru Forester", 67, 65);
dt.Rows.Add("Jeep Wrangler", 88, 91);

// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();

// add the data series to the data series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);

// specify the datasource for the chart
flexChart1.DataSource = dt;

// bind the X-axis
flexChart1.BindingX = "Automobile";

// bind the Y axes
series1.Binding = "Resale Value in 2014";
series2.Binding = "Resale Value in 2015";

// name the series
series1.Name = "2014";
series2.Name = "2015";

// set the chart type to splinearea
flexChart1.ChartType = C1.Chart.ChartType.SplineArea;
```

SplineSymbols

The SplineSymbols Chart combines the Spline Chart and the Scatter Chart. The chart plots data points by using symbols and connects those data points by using splines.

To create the SplineSymbols Chart, you can set the ChartType property to SplineSymbols in the Properties window or you can also set the property programmatically.

Set the Stacking property to Stacked or Stacked100pc to create the stacking SplineSymbols Chart.

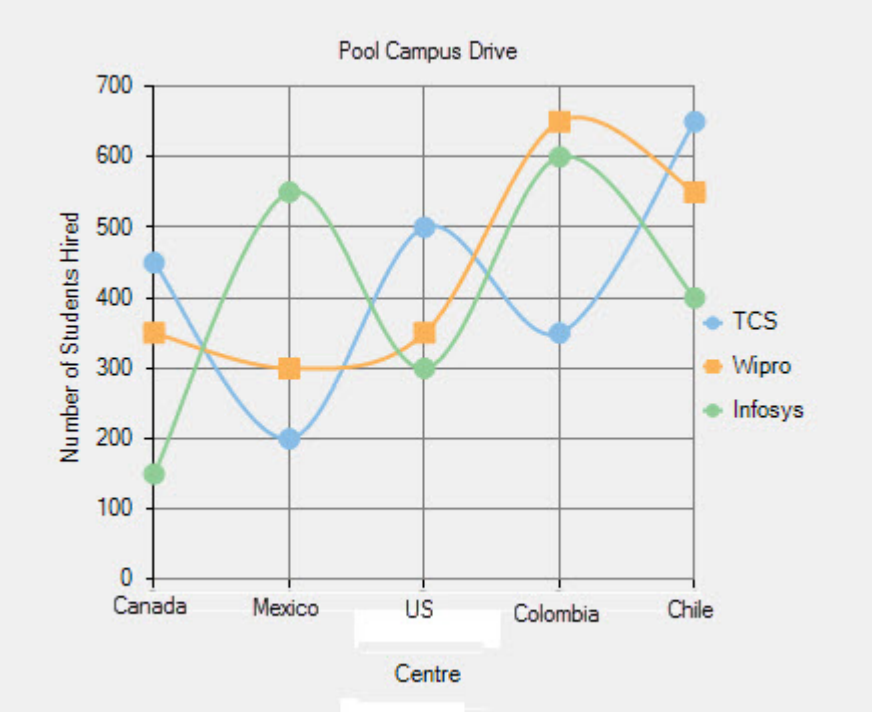
Take an example of a pool campus drive in which three IT organizations participate and hire students for their Canada, Mexico, US, Colombia, and Chile centres. The participating organizations are TCS, Wipro, and Infosys.

Let us visualize the given data by using the SplineSymbols Chart to depict the number of students hired by the companies for their various centres.

Sample Data Table

Centre	Hired by TCS	Hired by Wipro	Hired by Infosys
Canada	450	350	150
Mexico	200	300	550
US	500	350	300
Colombia	350	650	600
Chile	650	550	400

SplineSymbols Chart



The above chart displays the number of students hired by TCS, Wipro, and Infosys for their five different centres.

- Number of series: three (TCS, Wipro, and Infosys)
- Number of Y values per point: one

See the following code for implementation:

- **Visual Basic**

```
' create a datatable
Dim dt As New DataTable("Pool Campus Drive")

' add columns to the datatable
dt.Columns.Add("Centre", GetType(String))
dt.Columns.Add("Hired by TCS", GetType(Integer))
dt.Columns.Add("Hired by Wipro", GetType(Integer))
dt.Columns.Add("Hired by Infosys", GetType(Integer))

' add rows to the datatable
dt.Rows.Add("Canada", 450, 350, 150)
dt.Rows.Add("Mexico", 200, 300, 550)
dt.Rows.Add("US", 500, 350, 300)
dt.Rows.Add("Colombia", 350, 650, 600)
dt.Rows.Add("Chile", 650, 550, 400)

' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()
Dim series3 As New C1.Win.Chart.Series()

' add the data series to the data series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
FlexChart1.Series.Add(series3)

' specify the datasource for the chart
FlexChart1.DataSource = dt

' bind the X-axis
FlexChart1.BindingX = "Centre"

' bind the Y axes
series1.Binding = "Hired by TCS"
series2.Binding = "Hired by Wipro"
series3.Binding = "Hired by Infosys"

' name the series
series1.Name = "TCS"
series2.Name = "Wipro"
series3.Name = "Infosys"

' set the chart type to splinesymbols
FlexChart1.ChartType = C1.Chart.ChartType.SplineSymbols
```

- **C#**

```
// create a datatable
DataTable dt = new DataTable("Pool Campus Drive");

// add columns to the datatable
dt.Columns.Add("Centre", typeof(string));
dt.Columns.Add("Hired by TCS", typeof(int));
dt.Columns.Add("Hired by Wipro", typeof(int));
```

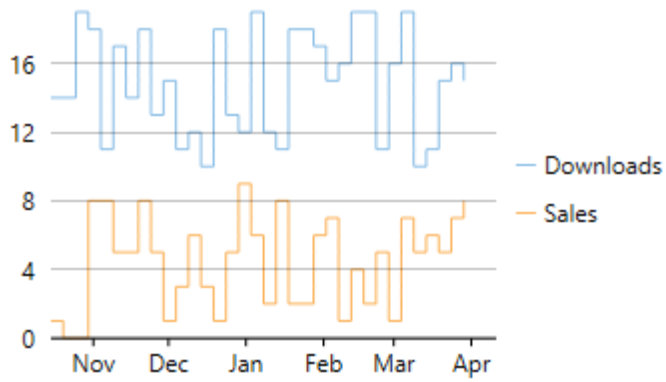
```
dt.Columns.Add("Hired by Infosys", typeof(int));

// add rows to the datatable
dt.Rows.Add("Canada", 450, 350, 150);
dt.Rows.Add("Mexico", 200, 300, 550);
dt.Rows.Add("US", 500, 350, 300);
dt.Rows.Add("Colombia", 350, 650, 600);
dt.Rows.Add("Chile", 650, 550, 400);

// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series3 = new C1.Win.Chart.Series();

// add the data series to the data series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);
flexChart1.Series.Add(series3);

// specify the datasource for the chart
flexChart1.DataSource = dt;

// bind the X-axis
flexChart1.BindingX = "Centre";

// bind the Y axes
series1.Binding = "Hired by TCS";
series2.Binding = "Hired by Wipro";
series3.Binding = "Hired by Infosys";

// name the series
series1.Name = "TCS";
series2.Name = "Wipro";
series3.Name = "Infosys";

// set the chart type to splinesymbols
flexChart1.ChartType = C1.Chart.ChartType.SplineSymbols;
```

Step

Step charts use horizontal and vertical lines to present data that show sudden changes along y-axis by discrete amount. These charts help display changes that are sudden and irregular but stay constant till the next change. Step charts enable judging trends in data along with the duration for which the trend remained constant.

Consider a use case where you want to visualize and compare weekly sales and units downloaded of a software. As both of these values vary with discrete amounts, you can use step chart to visualize them. As shown in the image below, apart from depicting the change in sales these charts also show the exact time of change and the duration for which sales were constant. Moreover, you can easily identify the magnitude of respective changes by simply looking at the chart.

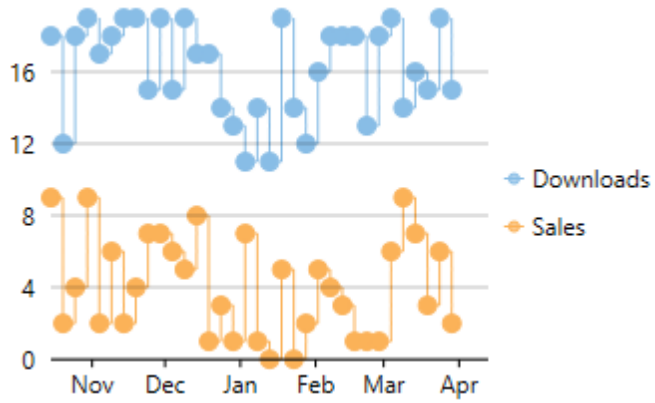
FlexChart supports Step chart, StepSymbols chart, and StepArea or filled step chart. The following table gives detailed explanation of these chart types.



Step Chart

Step chart is similar to the Line chart, except that Line chart uses shortest distance to connect consecutive data points, while Step chart connects them with horizontal and vertical lines. These horizontal and vertical lines give the chart step-like appearance.

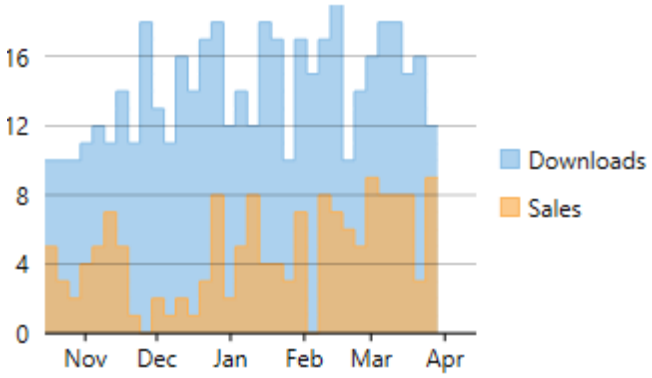
While the line charts depict change and its trend, the Step charts also help in judging the magnitude and the intermittent pattern of the change.



StepSymbols Chart

StepSymbols chart combines the Step chart and the Scatter chart. FlexChart plots data points by using symbols and connects those data points with horizontal and vertical step lines.

Here, the data points are marked using symbols and, therefore, help mark the beginning of an intermittent change.



StepArea Chart

StepArea chart combines the Step chart and the Area chart. It is similar to Area chart with the difference in the manner in which data points are connected. FlexChart plots the data points using horizontal and vertical step lines, and then fills the area between x-axis and the step lines.

These are based on Step charts, and are commonly used to compare discrete and intermittent changes between two or more quantities. This gives the chart stacked appearance, where related data points of the multiple series seem stacked above the other.

For example, number of units downloaded and sales of a software for a particular time duration can be easily compared as shown in the image.

To create Step chart, you need to populate appropriate data in chart and set the ChartType property to **Step.** However, to create a StepArea or a StepSymbol chart, you need to set the **ChartType** property to **StepArea** or **StepSymbols** respectively.

- **C#**

```
flexChart1.ChartType = ChartType.Step;   // ChartType.StepArea or ChartType.StepSymbols
```

Working with FlexChart

To work with FlexChart and use it for developing your applications, you should know how to leverage several features and functionality offered by the control.

This section provides important conceptual, task-based information on features and functionality offered by FlexChart.

The below-mentioned links take you to the sections that discuss the different ways in which you can work with FlexChart.

- Data
- Appearance
- End-User Interaction
- Design-Time Support
- FlexChart Elements

Data

Data is the first and foremost requirement of a chart. Without data, a chart cannot possibly visualize or display anything. Thus, while working with a chart, your first job is to get your chart to display data, so that you can work with and interpret the data accordingly.

When it comes to your chart data, there are three stages that sum up the specification, representation, and interpretation of the data:

- Providing data
- Plotting data

Access these sections to go through these stages with reference to FlexChart at length.

Providing Data

To get your chart to plot data, you first need to provide data to the chart.

Following are the three approaches that you can use to provide data to FlexChart:

- Loading data from pre-defined arrays
- Entering data manually at design-time or by using code
- Binding data using a data source

Loading Data from Arrays

To load data into FlexChart from a predefined array, you can set the data source of the series by using the DataSource property, which accepts any collection of objects.

You can use the Point array that enables you to set an ordered pair of X and Y coordinates of the integer type in a two dimensional plane. Or you can use the PointF array that lets you set an ordered pair of X and Y coordinates of the floating type in a two dimensional plane.

> You can use the Point/PointF array to enter data in case of charts that only use two fields, such as Bar, Column, Scatter, Line, and so on. For charts like Bubble, Candle, and HighLowOpenClose that use more than two fields, you can opt for an alternative method. You can create a data table manually, and then bind the axes of the chart

> to the appropriate fields of the data table.

Besides using objects like Point/PointF, you can use an anonymous object array to load data into FlexChart.

Furthermore, there is the SetData() that takes X and Y arrays of the double type. You just need to add a series to FlexChart and pass X and Y arrays of the double type in the SetData() to load data into FlexChart.

For details on adding data to series using the SetData(), refer to Adding Data to Series.

Let's use one of these approaches to load data into FlexChart.

Consider an IT company that is making a software to regulate and maintain banking/financial activities and data for a reputed international bank. There are three primary stages involved in the making of the software: designing (1), coding (2), and testing (3). To carry out these phases, the company undertakes both offshore and onsite activities.

Offshore refers to the work done at the developer's premises, while onsite refers to the work done at the customer's premises.

Apparently, the company wants to analyze the amount of work distribution in terms of man-hours for the software between offshore and onsite activities.

Here's the data table:

Work Distribution	Designing (1)	Coding (2)	Testing (3)
Offshore	100	400	280
Onsite	80	100	150

Since we need to compare the work distribution between offshore and onsite activities, we can use the Column Chart for this scenario.

See the following code for the implementation:

- **Visual Basic**

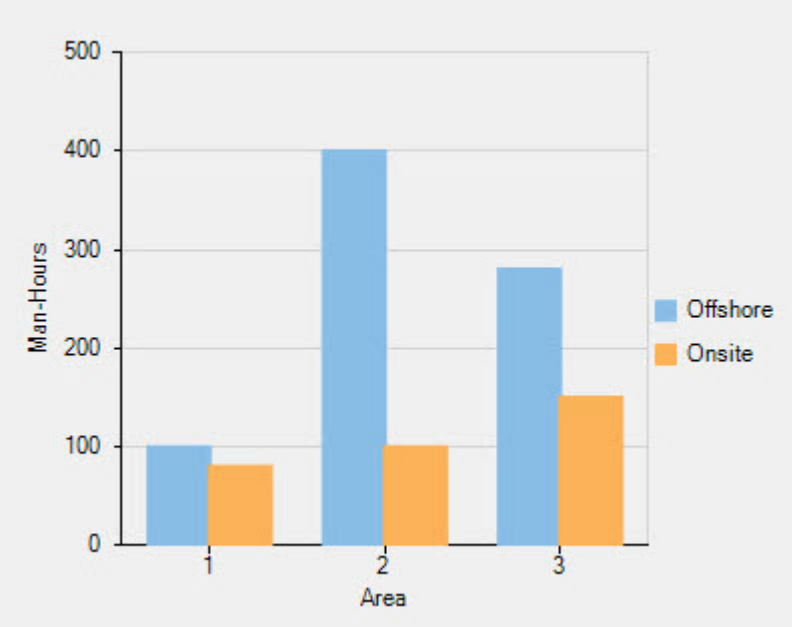
```
' clear data series collection
FlexChart1.Series.Clear()

' create data series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()

' add data points into the data series
series1.BindingX = "X"
series1.Binding = "Y"
series1.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(1, 100),
New System.Drawing.Point(2, 400),
New System.Drawing.Point(3, 280)}
series1.Name = "Offshore"

series2.BindingX = "X"
series2.Binding = "Y"
series2.DataSource = New System.Drawing.Point() {
New System.Drawing.Point(1, 80),
New System.Drawing.Point(2, 100),
New System.Drawing.Point(3, 150)}
series2.Name = "Onsite"

' add the data series to the data series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
```

```
' set the chart type to column
FlexChart1.ChartType = C1.Chart.ChartType.Column
```

- **C#**

```
// clear data series collection
flexChart1.Series.Clear();

// create data series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();

// add data points into the data series
series1.BindingX = "X";
series1.Binding = "Y";
series1.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(1,100),
new System.Drawing.Point(2,400),
new System.Drawing.Point(3,280)};
series1.Name = "Offshore";

series2.BindingX = "X";
series2.Binding = "Y";
series2.DataSource = new System.Drawing.Point[] {
new System.Drawing.Point(1,80),
new System.Drawing.Point(2,100),
new System.Drawing.Point(3,150)};
series2.Name = "Onsite";

// add the data series to the data series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);

// set the chart type to column
flexChart1.ChartType = C1.Chart.ChartType.Column;
```

Once you have run the code, the following output appears:



Entering Data Manually at Design-Time or Programmatically

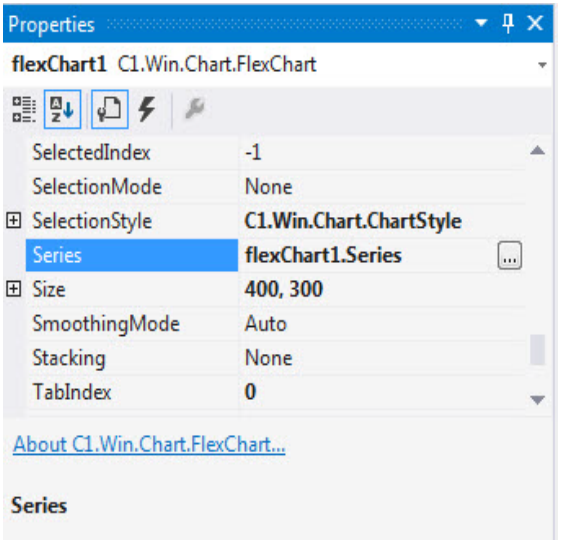
FlexChart allows you to enter data manually in two ways: at design-time or using code.

To enter data manually using code, you can assign different object arrays to a series using the DataSource property.

For complete information on how to load data through different object arrays, refer to Loading Data from Static Arrays.

At design-time, you can manually enter data through Series Collection Editor, which you can access by clicking the Ellipsis button next to the Series property in the Properties window.

Refer to Series Collection Editor for more details.



Binding Data Using a Data Source

Binding data means connecting one or more data consumers to a data provider in a synchronized manner. When data bound, the chart uses all of the bound data as its source of data for the specified series, and represents the data on the chart surface as per the series and chart properties.

To bind data, you first need to create a data source. A number of data sources are available including ADO.NET data source objects, such as DataView, DataTable, DataSet, and DataViewManager. In addition, third-party data sources, such as C1ExpressTable, C1ExpressView, C1ExpressConnection, C1DataView, C1DataTableSource, and C1DataSet are also available.

The following sections explain data binding at length:

- Binding Data to FlexChart
- Binding FlexChart Directly to the Data Source

Binding Data to FlexChart

Data is bound to FlexChart through the following steps: setting the DataSource property and setting the BindingX and

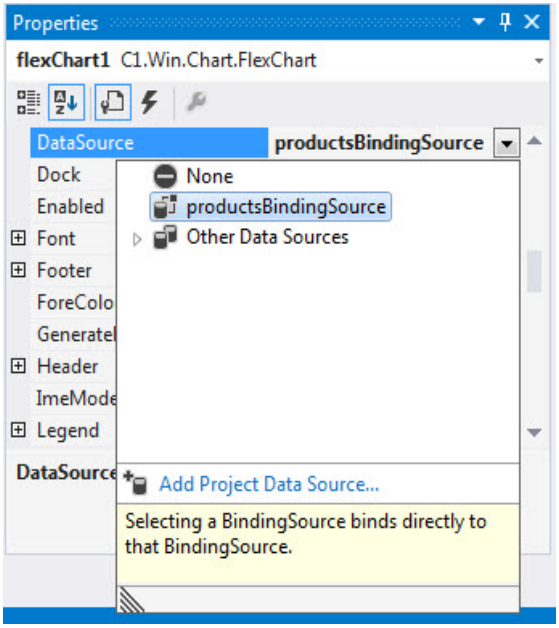
the Binding property.

Step 1: Setting the DataSource property

At Design-Time

Set the DataSource property at design-time by performing the following steps:

1. Select **FlexChart** from the **Properties** window.
2. Select the desired data source from the **DataSource** property drop-down list box.



In Code

Set the DataSource property by using the following code:

- **Visual Basic**

```
FlexChart1.DataSource = bindingSource1
```

- **C#**

```
flexChart1.DataSource = bindingSource1;
```

> You can set the DataSource property both at the chart level and at the series level. To set the property at the series level, navigate to **Series** in the Properties window. Click the **Ellipsis** button and open Series Collection Editor. Then, navigate to the DataSource property, and select the relevant data source from the drop-down list box.

Step 2: Setting the BindingX and the Binding property

Once you have set the DataSource property, you need to specify the fields to apply to the data series in the chart. In simple terms, you need to specify the fields to set data values for primary X and Y arrays and also for secondary Y arrays (depending upon the chart type).

You need to set the BindingX property to the field containing X values and the Binding property to the field(s) containing Y values.

At Design-Time

Perform the following steps:

1. Navigate to the **BindingX** and the **Binding** property.
2. Select the desired field from the **BindingX** drop-down list box to bind to X data array.
3. Select the desired field from the **Binding** drop-down list box to bind to Y data array.

In Code

Use the following code to set the BindingX and the Binding property at run-time:

- **Visual Basic**

```
FlexChart1.BindingX = "Id"
series1.Binding = "Sum"
series2.Binding = "Cost"
```

- **C#**

```
flexChart1.BindingX = "Id";
series1.Binding = "Sum";
series2.Binding = "Cost";
```

Like the DataSource property, you can set the Binding X and the Binding property at the chart as well as the series level. To set the properties at the series level, navigate to **Series** in the **Properties** window and click the **Ellipsis** button next to **Series**. In **Series Collection Editor**, navigate to the **BindingX** and the **Binding** property. Then, select the appropriate fields from the drop-down list box of the properties.

Binding FlexChart Directly to the Data Source

Since there is a layer between the data source and the actual chart, the data often needs to be summarized before it can be plotted; however, the data to be plotted sometimes may already be available in a data view or another data source object. And therefore, you can bind the chart directly to the data source object in such cases.

To bind the FlexChart control directly to the data source, you first need to set the DataSource property to the data source object, for instance, the data view, data table, or binding source. Next, you need to bind individual series of the chart to the fields present in the data source object by using the BindingX and the Binding property.

Let's use a case study of a food chain restaurant that specializes in a variety of foods and beverages. They hold special deals/discounts for their regular customers twice a month. Recently, they've decided to give out discounts on every food item purchased if the customer has bought at least five food items. However, the discounts to be given are decided on the basis of the quantities of the items purchased.

A customer, which is basically an organization by the name of QUICK-Stop, has ordered a couple of items in considerable quantities. The customer is billed the requisite amount on the order (order id: 10273) and is provided with the computer generated invoice that comprises the following details:

Product	Unit Price ($)	Quantity	Discount (%)
Chang	15.20	25	20
Queso Cabrales	16.80	50	20
Nord-Ost Matjeshering	20.70	35	20
Escargots de Bourgogne	10.60	30	20

Let us now visualize the invoice data by using FlexChart and interpret the data appropriately for all the products.

Here, we are using the Order Details data table in the C1NWind.mdb database.

Use the below-mentioned code to implement the case study:

- **Visual Basic**

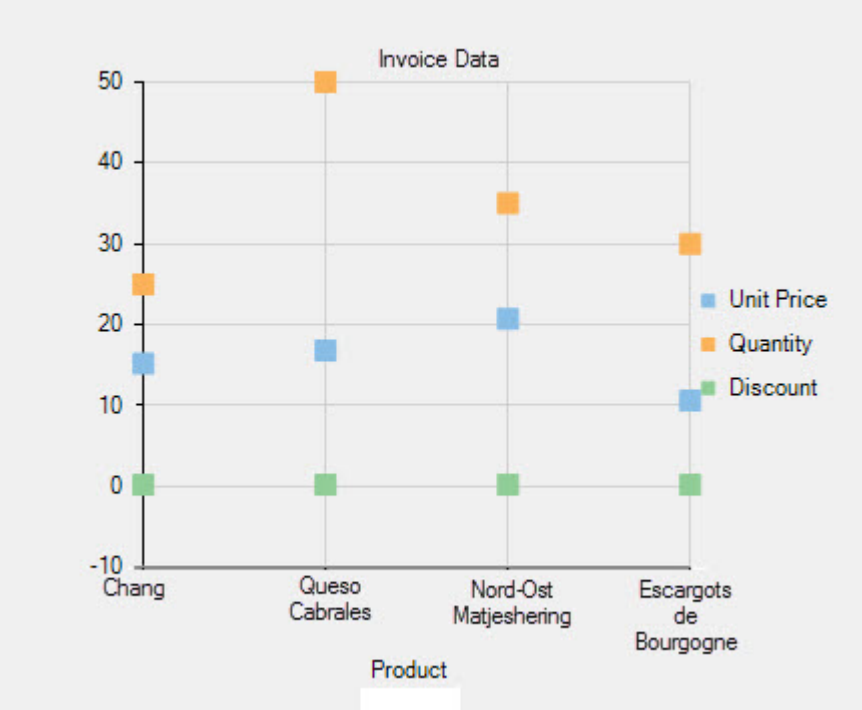
```
' retrieve data from the data source
Dim conPath As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & vbCr & vbLf &
                        "Data Source=C:\\Users\\Windows 8.1\\Desktop\\C1NWind.mdb"
Dim conQuery As String = "SELECT * FROM [Order Details] WHERE OrderID = 10327"
Dim da As New OleDbDataAdapter(conQuery, conPath)

' fill the dataset
da.Fill(ds)

' bind the binding source to the dataset
bs.DataSource = ds.Tables(0)

' bind the chart to the binding source
FlexChart1.DataSource = bs

' clear the series collection
FlexChart1.Series.Clear()

' create new series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()
Dim series3 As New C1.Win.Chart.Series()

' add the created series to the series collection
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
FlexChart1.Series.Add(series3)

' name the series
series1.Name = "Unit Price"
series2.Name = "Quantity"
series3.Name = "Discount"

' bind X-axis
FlexChart1.BindingX = "Product"

' bind Y axes
series1.Binding = "UnitPrice"
series2.Binding = "Quantity"
series3.Binding = "Discount"

' set the chart type
FlexChart1.ChartType = C1.Chart.ChartType.Scatter
```

- **C#**

```
// retrieve data from the data source
string conPath = @"Provider=Microsoft.Jet.OLEDB.4.0;
                Data Source=C:\\Users\\Windows 8.1\\Desktop\\C1NWind.mdb";
string conQuery = "SELECT * FROM [Order Details] WHERE OrderID = 10327";
OleDbDataAdapter da = new OleDbDataAdapter(conQuery, conPath);

// fill the dataset
da.Fill(ds);
```

```
// bind the binding source to the dataset
bs.DataSource = ds.Tables[0];

// bind the chart to the binding source
flexChart1.DataSource = bs;

// clear the series collection
flexChart1.Series.Clear();

// create new series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series3 = new C1.Win.Chart.Series();

// add the created series to the series collection
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);
flexChart1.Series.Add(series3);

// name the series
series1.Name = "Unit Price";
series2.Name = "Quantity";
series3.Name = "Discount";

// bind X-axis
flexChart1.BindingX = "Product";

// bind Y axes
series1.Binding = "UnitPrice";
series2.Binding = "Quantity";
series3.Binding = "Discount";

// set the chart type
flexChart1.ChartType = C1.Chart.ChartType.Scatter;
```

Following is the output:



Plotting Data

FlexChart plots data bound in the form of fields or data arrays when relevant values are set in the BindingX and the Binding property.

You require setting the values in the BindingX and the Binding property as per the desired chart type . For instance, in case of the Scatter Chart, you need to set a single value (field) in both the BindingX and the Binding property. However, in case of the Bubble Chart, you need to set a single value (field) in the BindingX property and two values (fields, one for specifying Y-values and another for specifying the size of the bubble) in the Binding property.

See the following code snippets for reference:

- **Visual Basic**

```
' in case of the Scatter Chart
FlexChart1.BindingX = "Beverages"
series1.Binding = "Unit Price"
```

- **C#**

```
// in case of the Scatter Chart
flexChart1.BindingX = "Beverages";
series1.Binding = "Unit Price";
```

- **Visual Basic**

```
' in case of the Bubble Chart
FlexChart1.BindingX = "Year"
series1.Binding = "Number of Employees,Annual Revenue"
```

- **C#**

```
// in case of the Bubble Chart
series1.BindingX = "Year";
series1.Binding = "Number of Employees,Annual Revenue";
```

Once the data is plotted, you can work on it to visualize data that suits your requirements.

Go through the sections given below to learn how to customize series and plot irregular data.

- Interpolating Null Values

Customizing Series

Once the series have been displayed in the chart, you can customize the displayed series to manage the same more efficiently.

FlexChart allows you to customize series by showing or hiding a series either in the Plot Area or the Legend or both.

The following sections detail the different customizations:

- Showing or Hiding a Series

Showing or Hiding a Series

If there are hundreds of series to be displayed in your chart, you would certainly need to manage the same due to the space limitation of the chart.

In FlexChart, you can manage series by using the Visibility property of a series. The Visibility property accepts values of the SeriesVisibility enumerated type.

You can set the property to the following different values to show or hide a series:

Value	Description
SeriesVisibility.Visible	The series is displayed in the Plot as well as the Legend.
SeriesVisibility.Plot	The series is displayed in the Plot, but hidden in the Legend.
SeriesVisibility.Legend	The series is displayed in the Legend, but hidden in the Plot.
SeriesVisibility.Hidden	The series is hidden in the Plot as well as the Legend.

Let's use the same scenario used in the Bar Chart. Refer to Bar Chart to see the complete scenario.

In the same scenario, suppose that you want to display comparison between products in terms of only unit price and units in stock.

To do the same, you can very well use the Visibility property.

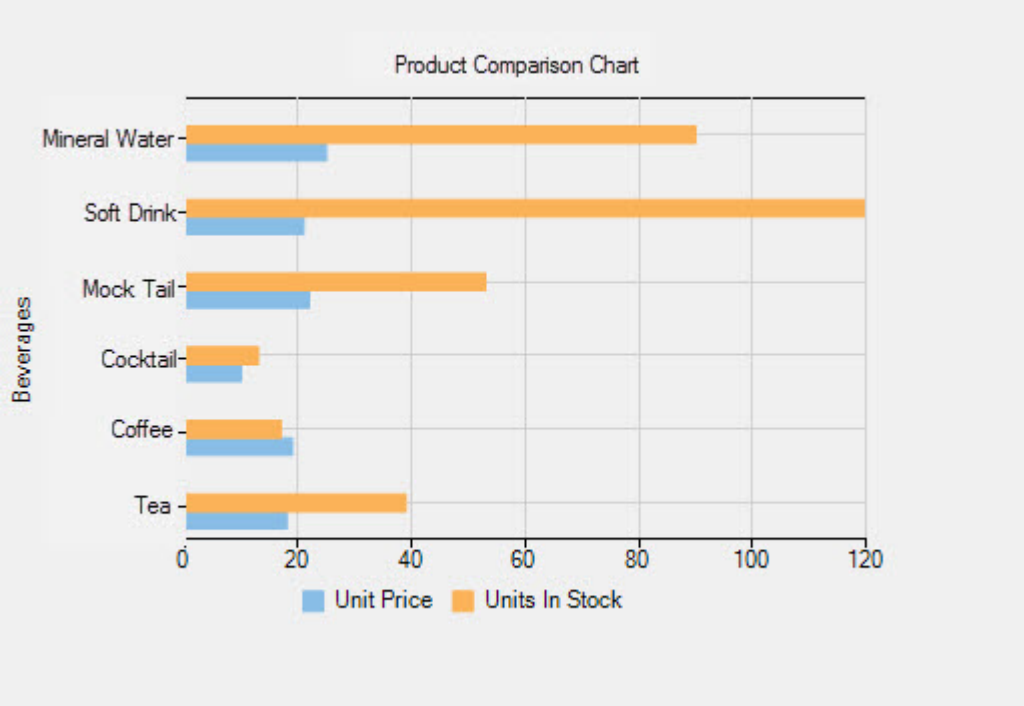
Here's the code snippet for reference:

- **Visual Basic**

```
' name the series
series1.Name = "Unit Price"
series2.Name = "Units In Stock"
series3.Name = "Units On Order"

' to hide Units On Order
series3.Visibility = C1.Chart.SeriesVisibility.Hidden
```

- **C#**

```
// name the series
series1.Name = "Unit Price";
series2.Name = "Units In Stock";
series3.Name = "Units On Order";

// to hide Units On Order
series3.Visibility = C1.Chart.SeriesVisibility.Hidden;
```



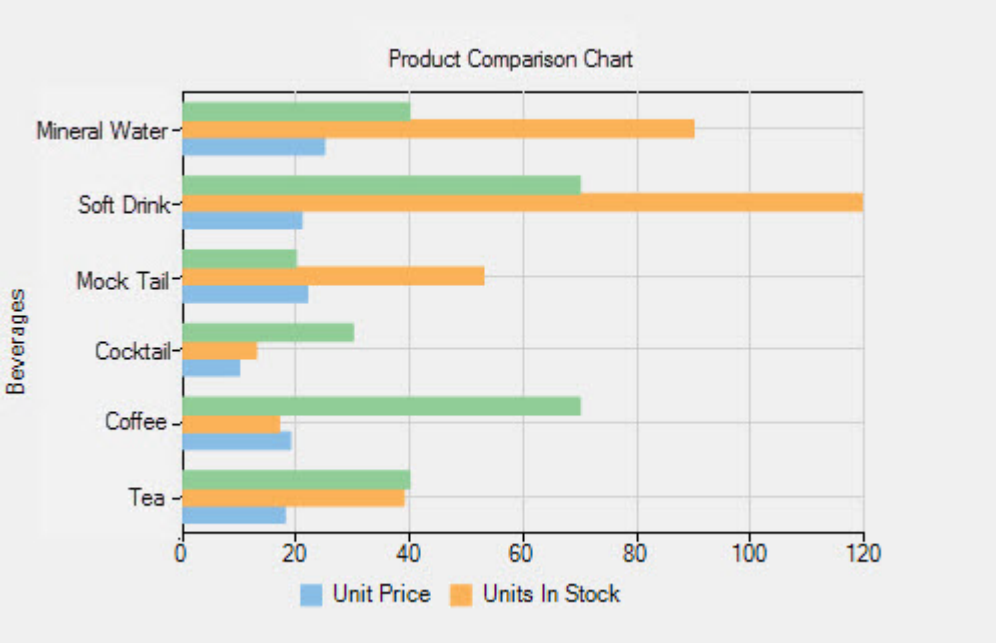
If you want to show the series (Units On Order) in the Plot Area, but at the same not display it in the Legend, you can use the Visibility property in the following manner:

- **Visual Basic**

```
' name the series
series1.Name = "Unit Price"
series2.Name = "Units In Stock"
series3.Name = "Units On Order"

' to show Units On Order in the Plot but not in the Legend
series3.Visibility = C1.Chart.SeriesVisibility.Plot
```

- **C#**

```
// name the series
series1.Name = "Unit Price";
series2.Name = "Units In Stock";
series3.Name = "Units On Order";

// to show Units On Order in the Plot but not in the Legend
series3.Visibility = C1.Chart.SeriesVisibility.Plot;
```



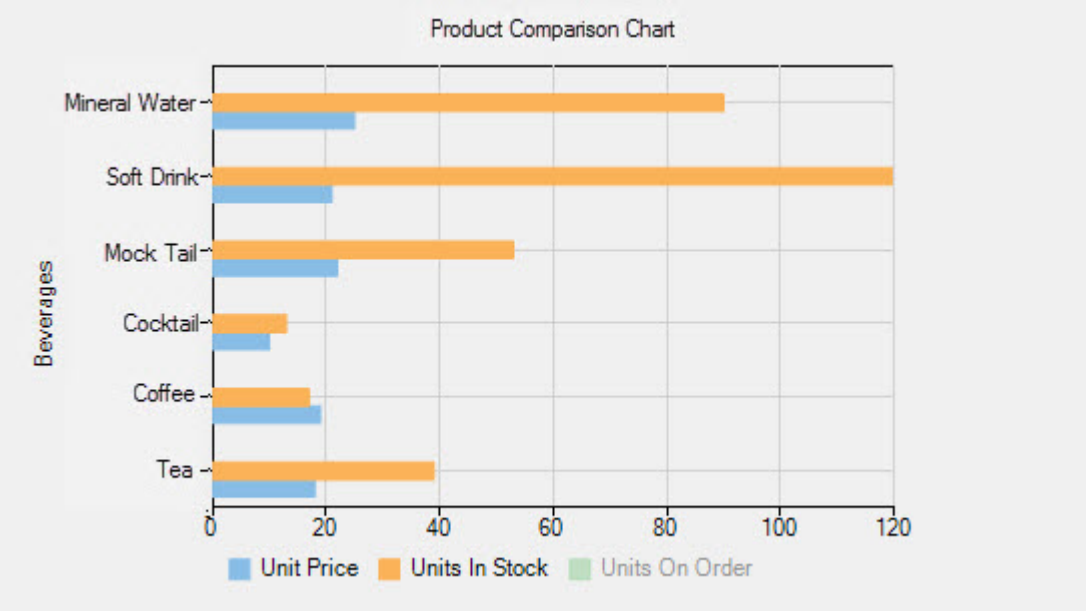
Similarly, for displaying the series in the Legend, but not in the Plot Area, you can set the Visibility property as follows:

- **Visual Basic**

```
' name the series
series1.Name = "Unit Price"
series2.Name = "Units In Stock"
series3.Name = "Units On Order"

' to show Units On Order in the Legend but not in the Plot
series3.Visibility = C1.Chart.SeriesVisibility.Legend
```

- **C#**

```
// name the series
series1.Name = "Unit Price";
series2.Name = "Units In Stock";
series3.Name = "Units On Order";

// to show Units On Order in the Legend but not in the Plot
series3.Visibility = C1.Chart.SeriesVisibility.Legend;
```



Interpolating Null Values

Often, there are null values in the data fields of a data table that you bind to FlexChart for plotting data. Due to the presence of null values, FlexChart creates gaps once it has plotted the data. The gaps present in the plotted data make data look inconsistent and incomplete.

FlexChart allows you to deal with such inconsistencies by using the InterpolateNulls property. You can set the InterpolateNulls property, so that the chart automatically fills in gaps created by null values in data.

> The InterpolateNulls property is applicable only for the Line Chart and the Area Chart.

Here is how you can set the InterpolateNulls property:

- **Visual Basic**

```
FlexChart1.Options.InterpolateNulls = True
```

- **C#**

```
flexChart1.Options.InterpolateNulls = true;
```

Appearance

The appearance of a chart determines its overall look and feel. A good and clean appearance draws your audiences toward the visual representation of your data. And it also adds to the ease of interpreting the data.

You can customize the appearance of FlexChart in a variety of ways that are mentioned below:

- Colors
- Fonts
- Symbol Styles for Series
- Themes

Colors

Colors are used to enhance the visual impact of a chart. You can customize colors by choosing colors interactively, setting chart palette, specifying RGB values, specifying hue, saturation, and brightness, or using transparent colors.

FlexChart lets you customize both background and foreground colors for the entire chart as well as the following elements:

- Series
- Header and Footer
- Legend
- Plot Area
- Label

Click the links given below to learn how to use colors in different ways.

- Choosing Colors Interactively
- Setting Chart Palette
- Specifying RGB Colors
- Specifying Hue, Saturation, and Brightness
- Using Transparent Colors

Choosing Colors Interactively

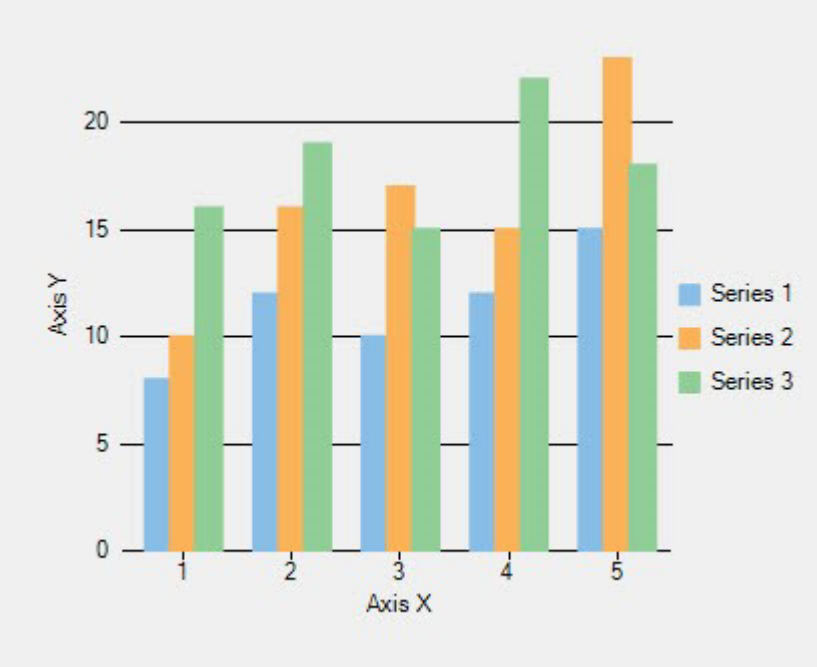
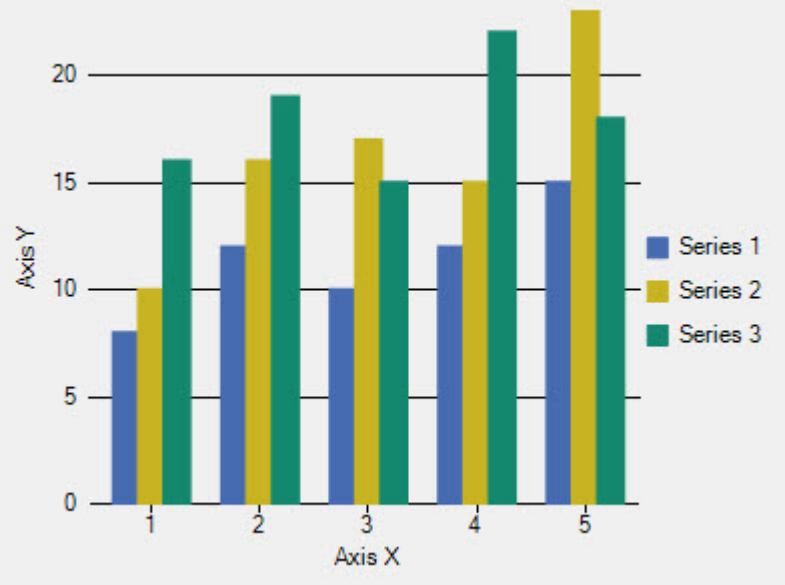
Colors can be chosen interactively by using .NET's color dialog that works like the standard windows color dialog. You can choose from Windows basic colors or customer colors, or you can interactively choose from a full color spectrum.

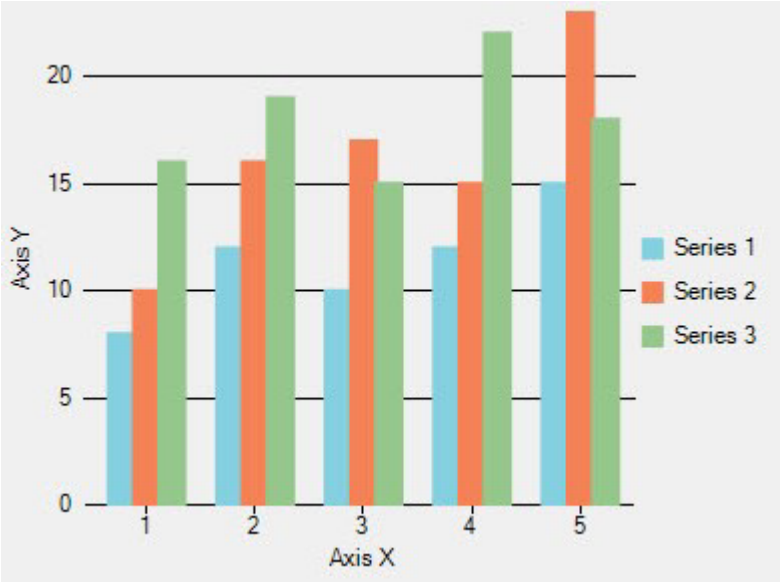
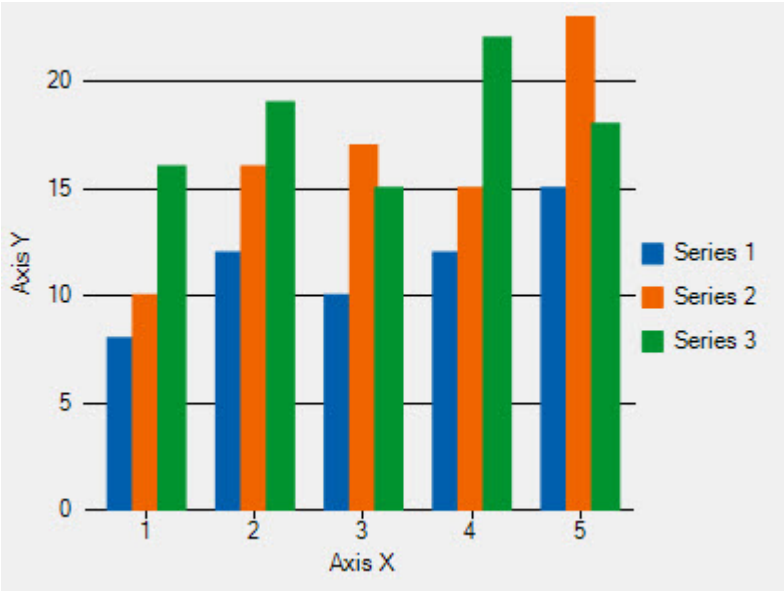
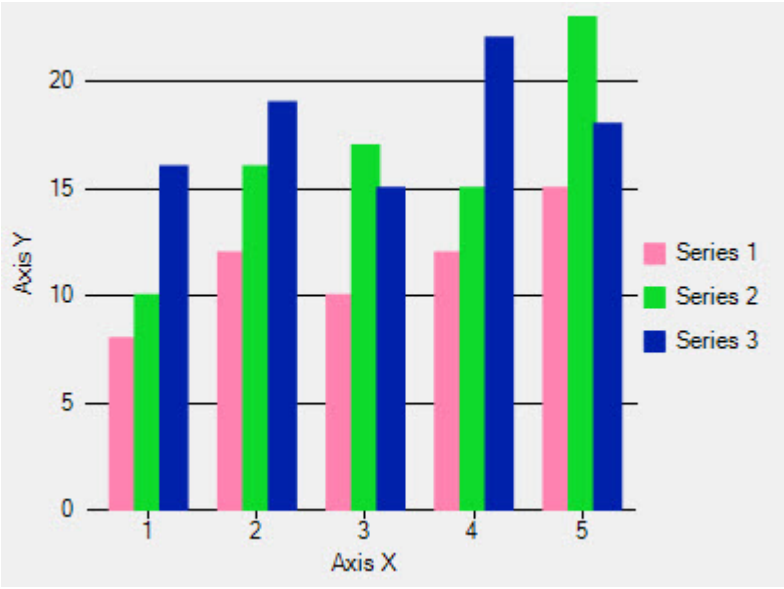
Setting FlexChart Palette

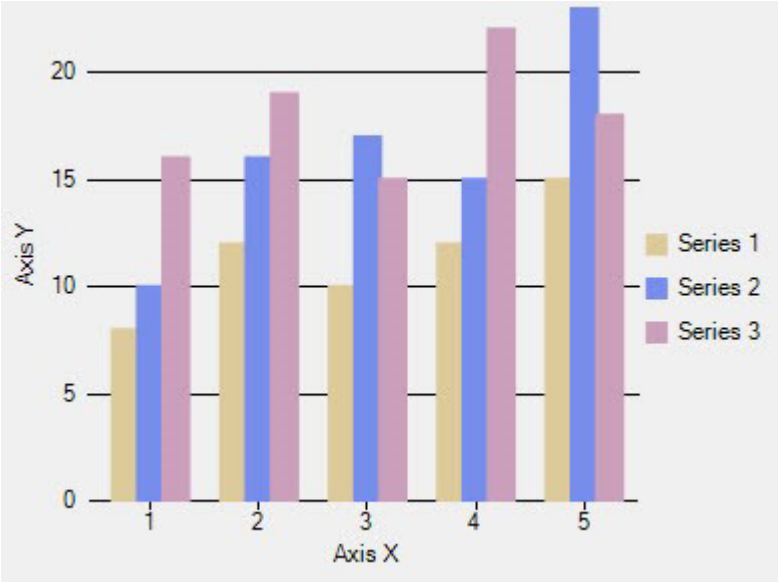
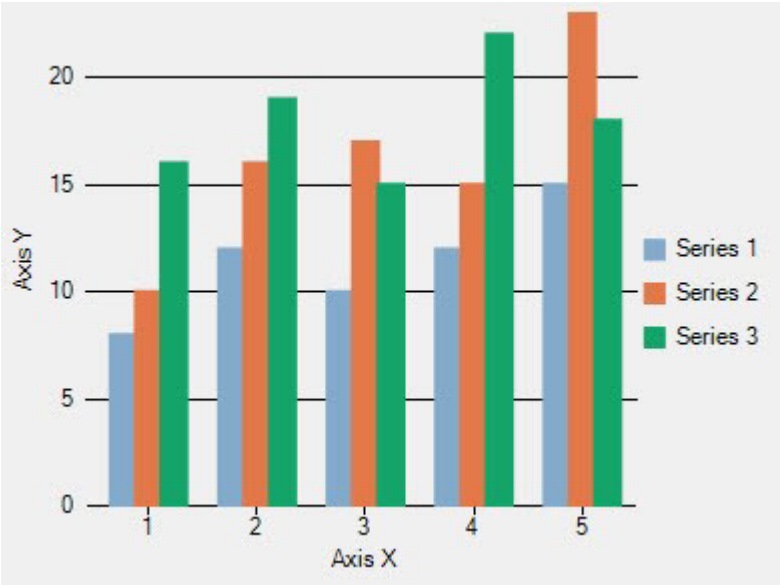
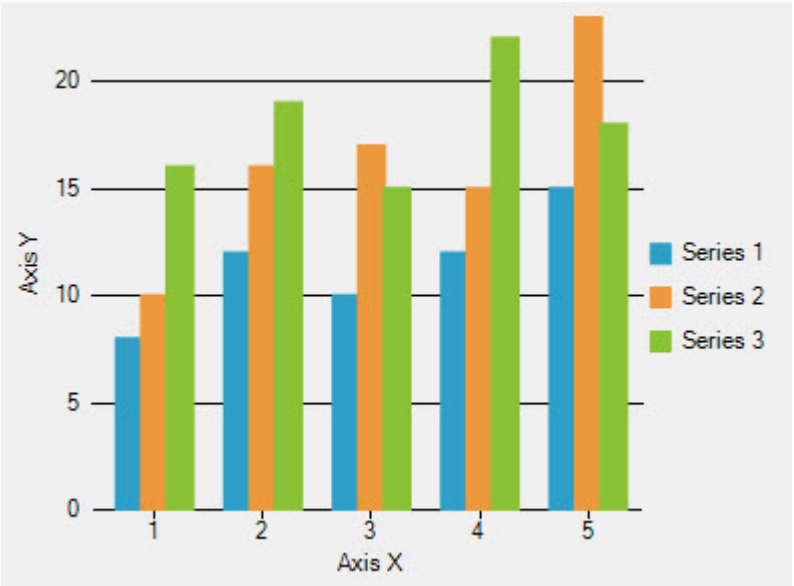
You can set the desired FlexChart palette by using the Palette property. By default, FlexChart uses the **Palette.Standard** setting that specifies the standard chart palette.

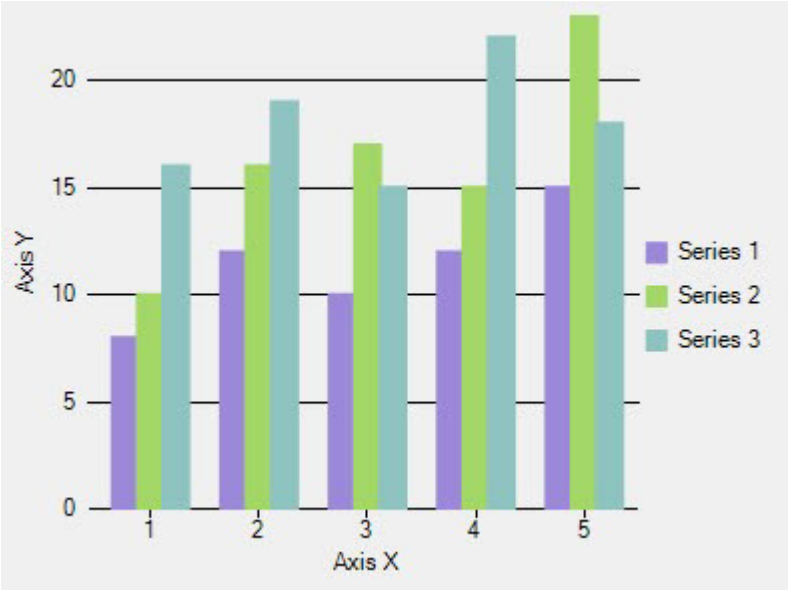
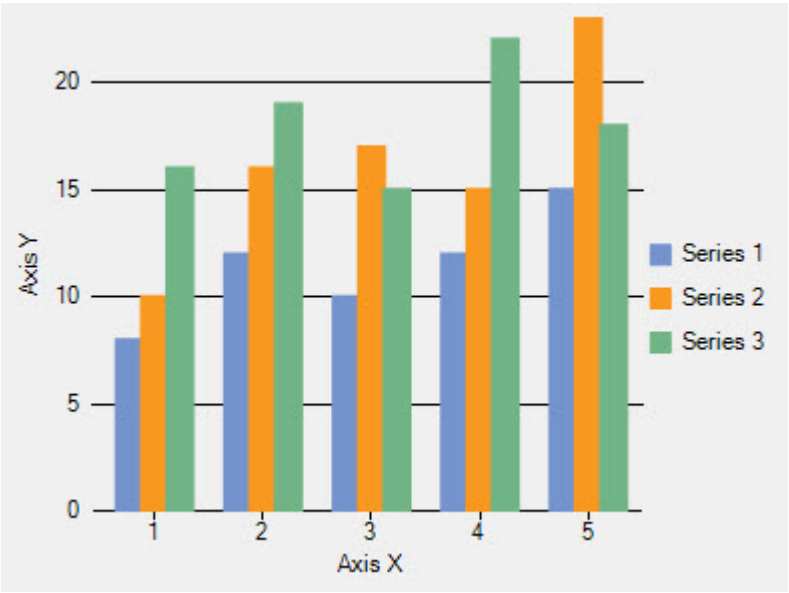
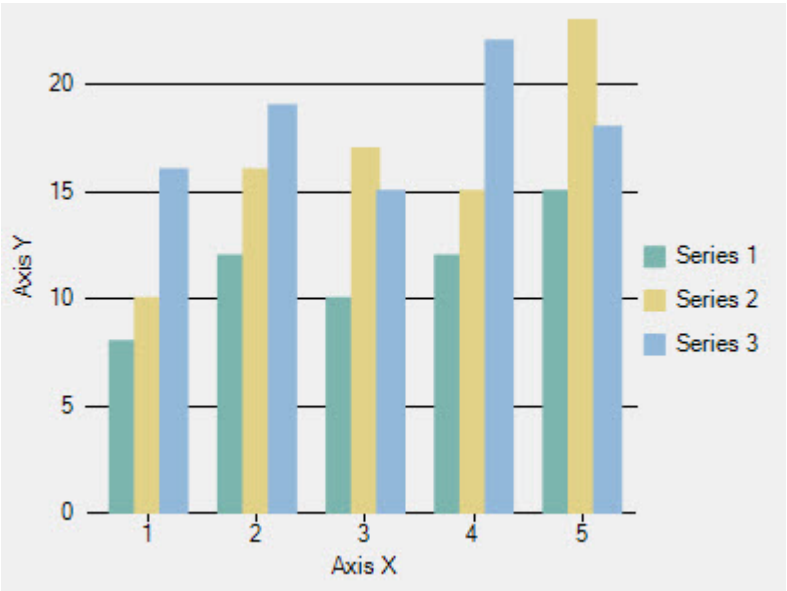
Here are the available palettes in FlexChart:

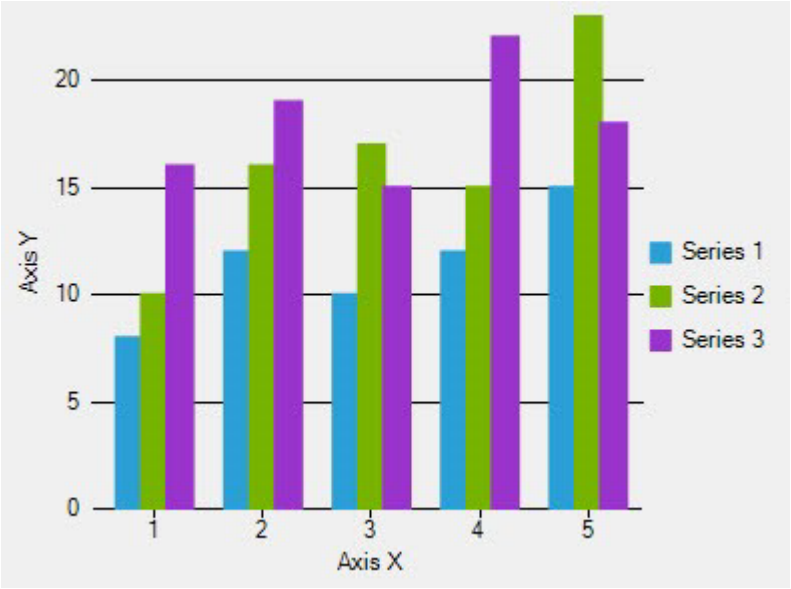
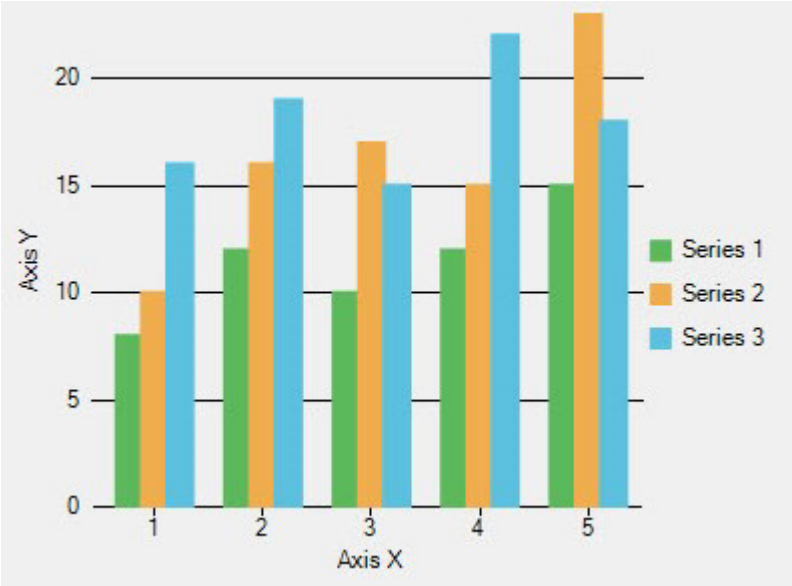
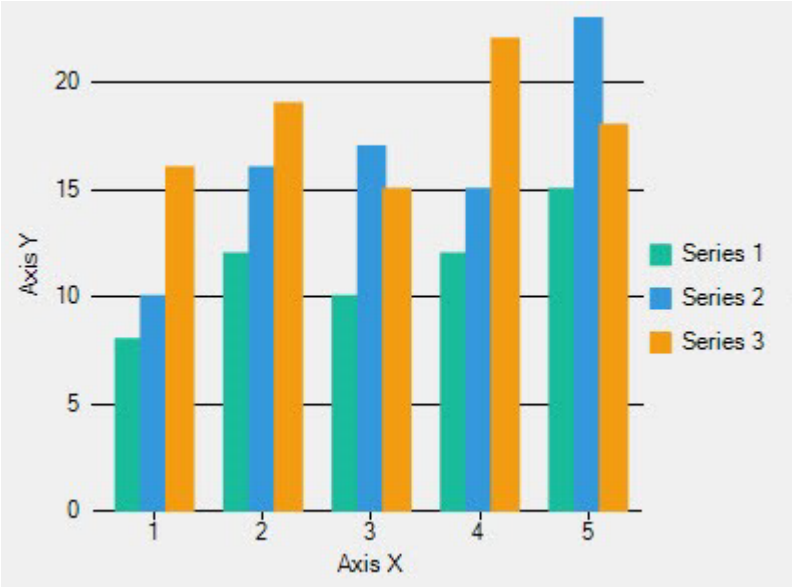
Palette Setting	Preview

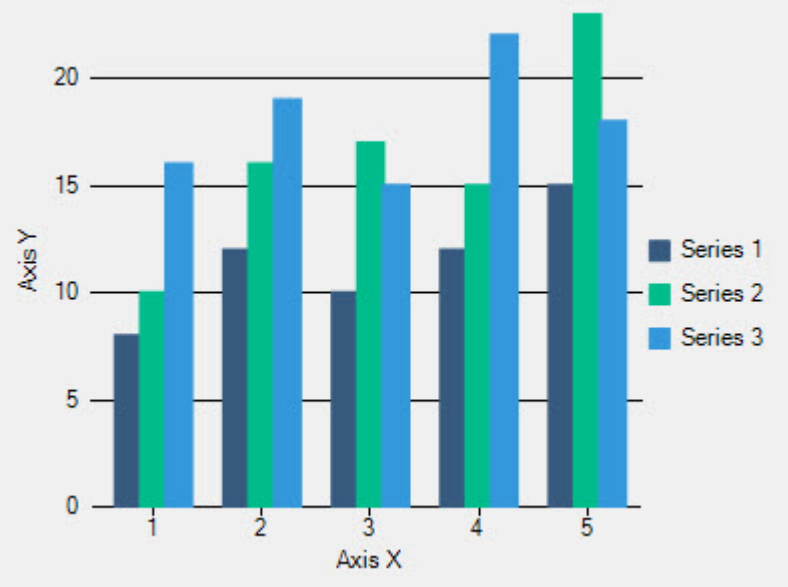
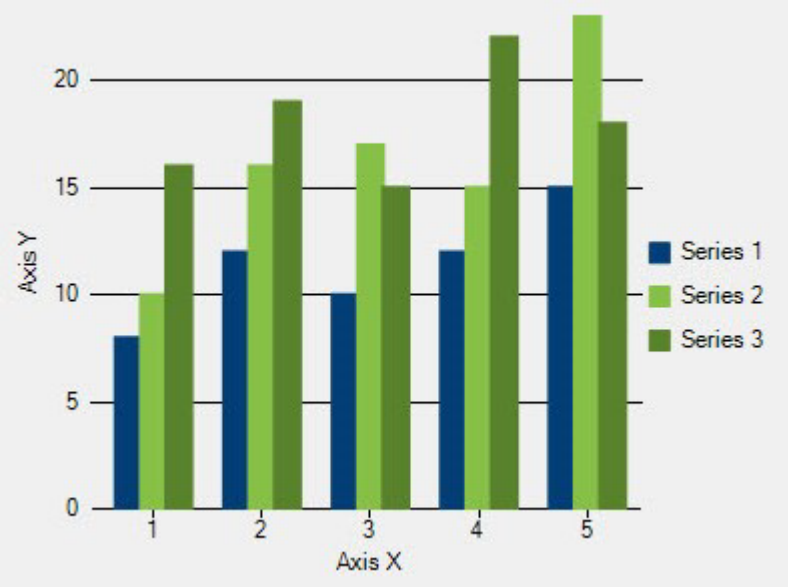
Standard (default)	
Cocoa	

Coral	
Dark	
Highcontrast	

Light	
Midnight	
Modern	

Organic	
Slate	
Zen	

Cyborg	
Superhero	
Flatly	

Darkly	
Cerulan	
Custom	Copies the currently specified palette into the custom group.

Specifying RGB Colors

A color can be specified by its RGB components, useful for matching another RGB color. RGB color values combine hexadecimal values for the red, green, and blue components of a color. "00" is the smallest value a component can have; "ff" is the largest value. For example, "#ff00ff" specifies magenta (the maximum value of red and blue combined with no green).

Specifying Hue, Saturation, and Brightness

In addition to a color being specified by its RGB components, it can also be represented by its hue, saturation, and brightness. The hue, saturation, and brightness are all aspects of the red, green, and blue color scheme. The hue is the specific tone of the color wheel made up of red, green, and blue tones. The saturation is the intensity of the hue from

gray tone to a pure vivid tone. And the brightness is the lightness or darkness of a tone.

Using Transparent Colors

The background and foreground of all elements except the chart itself can be "Transparent".

When a background or foreground is transparent, the chart uses the color of the element outside it for the background. For example, the header would have the background of the chart itself when its background is set to Transparent.

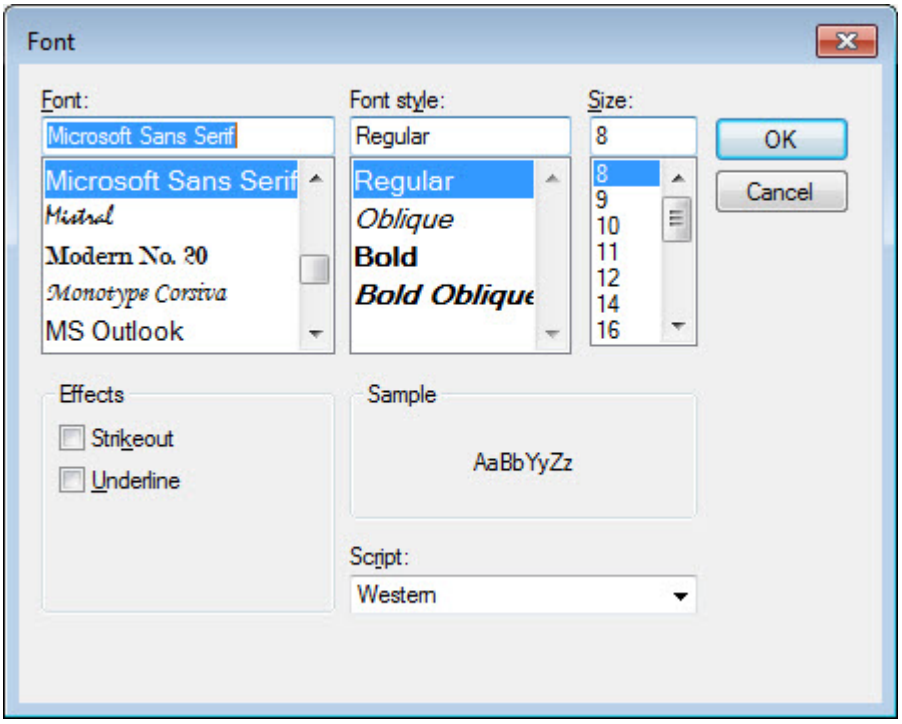
In other words, if the background color of the element is transparent then its background is not drawn. If the foreground color of the element is transparent, then the foreground (for example, the text of a title) is not drawn.

The transparent color properties are located under the Style nodes, found at design time on the Control, Header, Footer, Legend, ChartArea, and ChartLabels objects in the Visual Studio Properties window.

Fonts

Fonts, when customized with respect to various chart elements, enhance the impact of the chart. You can adjust the font size of an element to make it more suitable as per the overall size of the chart.

Font properties are located under the Font node of the Properties window. To change or customize fonts in FlexChart, you need to use the standard .NET Font property editor, as shown below:



Symbol Styles for Series

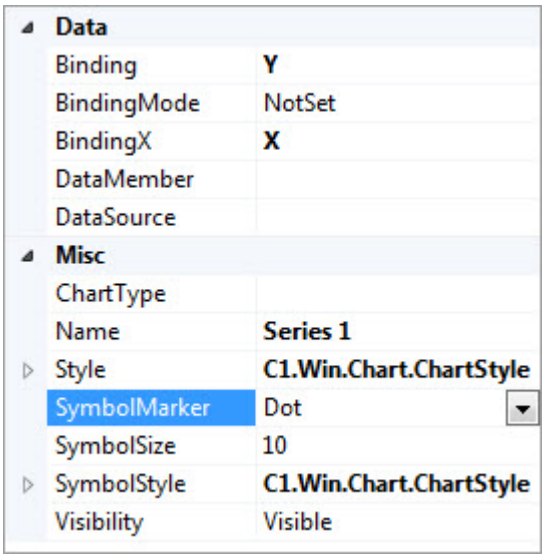
Depending upon the requirements, you may need to customize the appearance of series in the chart.

FlexChart allows you to customize series in the chart with the SymbolMarker, SymbolSize, and SymbolStyle properties located in Series Collection Editor.

Refer to Series Collection Editor for more information.

The SymbolMarker property allows you to set the shape of the marker to be used for each data point in the series.

The SymbolSize property enables you to set the size (in pixels) of the symbols used to render the series. And the SymbolStyle property allows you to set the fill color, font, stroke color, and stroke width of the symbols used in the series.



Below is a table that lists how these properties affect each chart type:

Value	Effect for SymbolMarker	Effect for SymbolSize	Effect for SymbolStyle
ChartType.Column	No effect	No effect	No effect
ChartType.Bar	No effect	No effect	No effect
ChartType.Line	No effect	No effect	No effect
ChartType.Scatter	Changes the symbol marker	Changes the symbol size	Changes the symbol style
ChartType.LineSymbols	Changes the symbol marker	Changes the symbol size	Changes the symbol style
ChartType.Area	No effect	No effect	No effect
ChartType.Spline	No effect	No effect	No effect
ChartType.SplineSymbols	Changes the symbol marker	Changes the symbol size	Changes the symbol style
ChartType.SplineArea	No effect	No effect	No effect
ChartType.Bubble	Changes the symbol marker	No effect	Changes the symbol style
ChartType.Candlestick	No effect	Changes the symbol size	No effect
ChartType.HighLowOpenClose	No effect	Changes the symbol size	No effect

The SymbolSize property has no effect on the Bubble Chart; however, you can change the size of the bubble in the Bubble Chart by setting the BubbleMaxSize and the BubbleMinSize property located in the Options node of the Properties window.

Themes

There are a number of predefined themes that can be applied to FlexChart or they can be even modified with the Themes designer.

For more information, refer to Themes for WinForms documentation.

End-User Interaction

When it comes to the functionality and features of the chart, you may have specific requirements that can be accommodated only through a few specific tools.

Therefore, to accommodate such requirements, FlexChart renders a set of conversion methods and interactive built-in tools. These tools help you customize and develop your applications further.

Go to the following sections for information on end-user interaction:

- ToolTips
- Axis Scrollbar
- Range Selector

ToolTips

Tooltips are pop-ups that appear while hovering over data points or series in a chart. They provide additional, valuable information about chart data in scenarios, as follows:

- **Single series chart**: Tooltips display data values and series name.
- **Mixed charts**: Tooltips display multiple data values for multiple series for a single category.
- **Pie charts**: Tooltips display name and percentage share or value of slices.

A tooltip displays Y value of a data point in FlexChart by default. However, FlexChart allows creating and formatting custom content in tooltips using pre-defined parameters and formats. In addition, the control allows creating a shared tooltip when working with mixed charts.

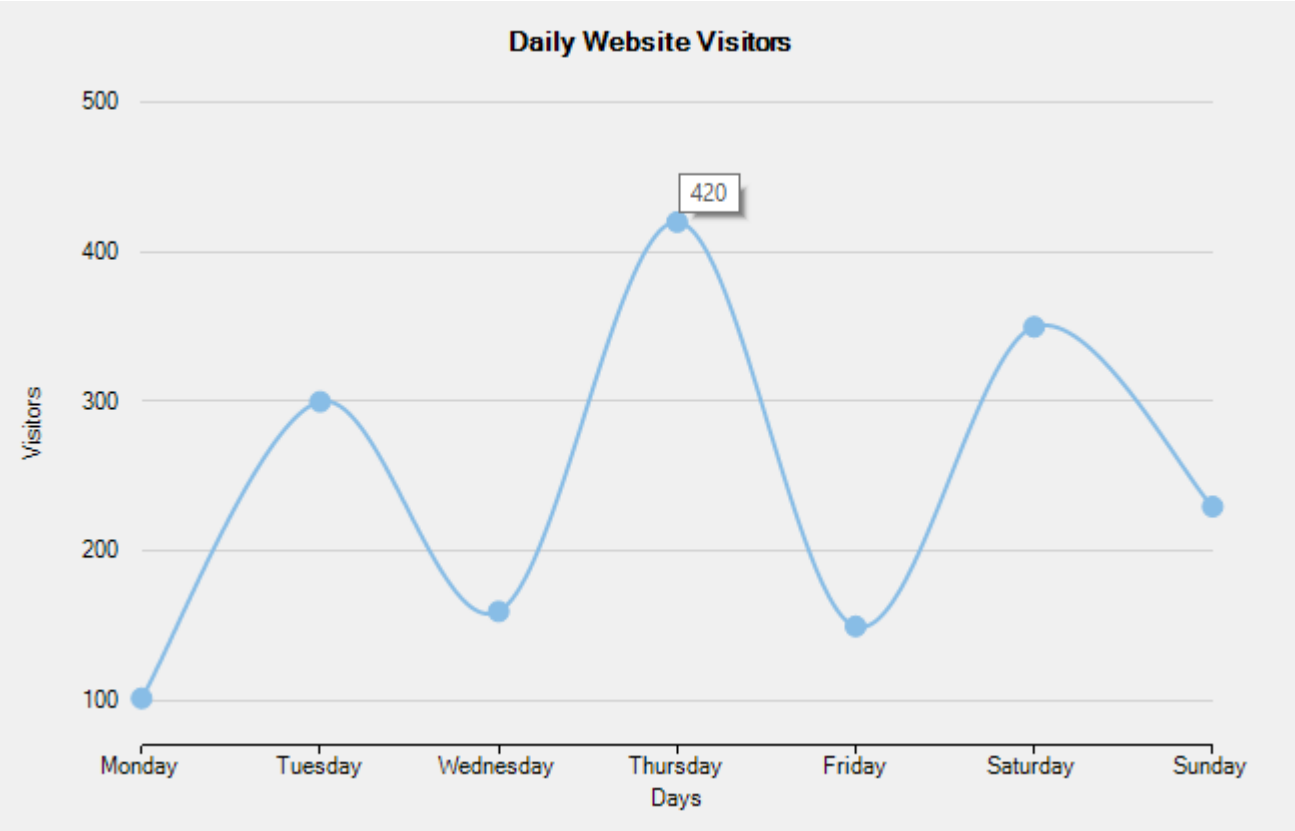
To know more about tooltips in FlexChart, see the following topics:

- Default Tooltip
- Customizing Tooltip Content
- Formatting Tooltip Content
- Shared Tooltip

Default Tooltip

FlexChart displays a default tooltip when you hover over a data point or series. The default tooltip shows the Y value of the hovered data point. FlexChart generates the default tooltip using the underlying data when custom content is not present.

The following image displays the default tooltip showing the data value of a data point.



Customizing Tooltip Content

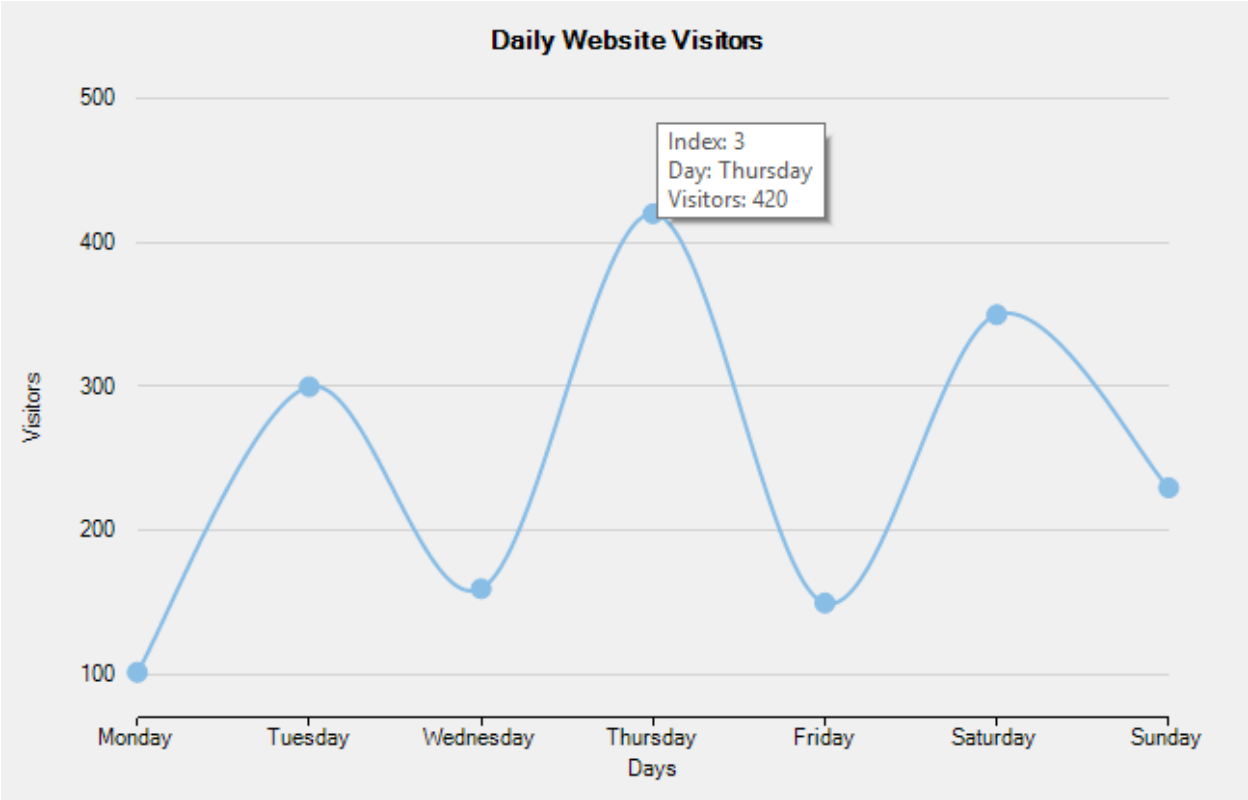
FlexChart simplifies customizing tooltip content by allowing you to set pre-defined parameters in the Content property of the tooltip.

To customize content in a tooltip, set the pre-defined parameters in the template string of the Content property of the ChartToolTip class.

The table below lists the pre-defined parameters applicable for tooltip content customization.

Parameter	Description
x	Shows the X value of the data point.
y	Shows the Y value of the data point.
value	Shows the Y value of the data point.
name	Shows the X value of the data point.
seriesName	Shows the name of the series.
pointIndex	Shows the index of the data point.

The following image displays customized tooltip content showing the index and the data point values.



The following code compares and displays data of number of daily website visitors in a specific week. The code shows how to configure the Content property to customize tooltip content.

- **Visual Basic**

```
'set the Content property
FlexChart1.ToolTip.Content = "Index: {pointIndex}" &
    vbLf & "Day: {name}" &
    vbLf & "{seriesName}: {Visitors}"
```

- **C#**

```
// set the Content property
flexChart1.ToolTip.Content = "Index: {pointIndex}\nDay: {name}\n{seriesName}: {Visitors}";
```

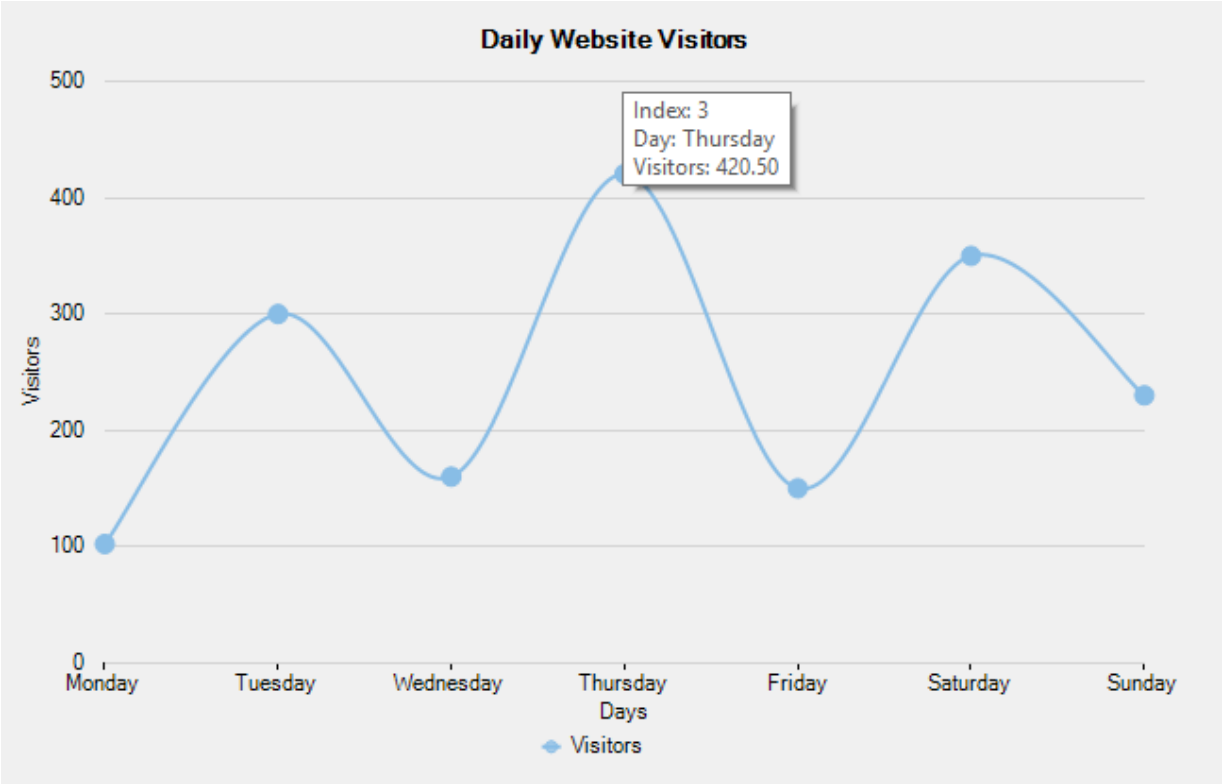
Formatting Tooltip Content

In FlexChart, it is possible to display number separators, current symbols, or date/time formats to add more details into tooltips.

FlexChart enables you to format the custom content in the tooltip by using standard and custom format strings. These format strings are a variety of Numeric and DateTime formats provided by .NET.

For information about these format strings, refer to Numeric and DateTime format strings.

The following image displays customized tooltip content showing the index and formatted values of the data point.



The following code compares and displays data of number of daily website visitors in a specific week. The code shows how to configure the Content property to format tooltip content.

- **Visual Basic**

```
'configure the Content property
FlexChart1.ToolTip.Content = "Index: {pointIndex}" &
    vbLf &
    "Day: {name}" &
    vbLf &
    "{seriesName}: {Visitors:F}"
```

- **C#**

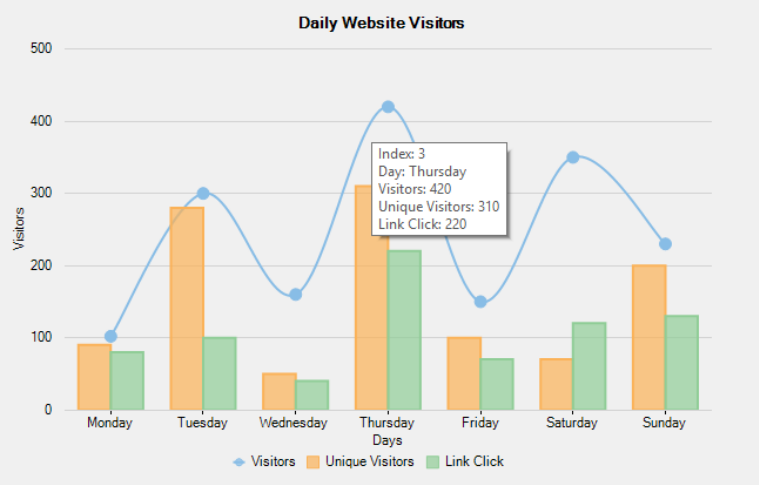
```
flexChart1.ToolTip.Content = "Index: {pointIndex}\nDay: {name}\n{seriesName}: {Visitors:F}";
```

Shared Tooltip

A shared tooltip is a single tooltip that highlights all data values for a single X value in the chart.

In a chart containing mixed chart types, you often require displaying multiple Y values for a common X value through a single tooltip. In such cases, FlexChart tooltips can be used as shared tooltips by setting the Content property accordingly.

The following image displays a shared tooltip showing Y values for all series at a single X value.



The code compares and displays data of number of daily website visitors, unique visitors, and link clicks in a specific week. The code shows how to set the Content property to create a shared tooltip.

- **Visual Basic**

```
'configure the Content property
FlexChart1.ToolTip.Content = "Index: {pointIndex}" &
    vbLf & "Day: {name}" &
    vbLf & "Visitors: {Visitors}" &
    vbLf & "Unique Visitors: {Unique Visitors}" &
    vbLf & "Link Click: {Link Click}"
```

- **C#**

```
//configure the Content property
flexChart1.ToolTip.Content = "Index: {pointIndex}\nDay: {name}\nVisitors: {Visitors}\nUnique Visitors: {Unique Visitors}\nLink Click: {Link Click}";
```

Axis Scrollbar

The presence of a large number of values or data in charts makes data interpretation difficult, especially in compact user interfaces. Axis Scrollbars solve this problem by letting you easily interpret closely related data within a specific range.

FlexChart allows you to add Axis Scrollbar to primary axes (X and Y axes) as well as secondary axes. To add Axis Scrollbar to an axis, you need to create an instance of the C1.Win.Chart.Interaction.AxisScrollbar class. The instance accepts a parameter that is an object of the C1.Win.Chart.Axis class type. In other words, you need to pass an Axis object while creating the AxisScrollbar instance to display the scrollbar for an axis.

The AxisScrollbar class provides the Maximum and the Minimum property to set the maximum and the minimum possible value of the scrollbar respectively. The class also provides the LowerValue and the UpperValue property to set the lower value and the upper value of the control respectively. The lower and upper values change when the scrollbar is resized or moved. And the ValueChanged event fires when any of the LowerValue or the UpperValue property changes.

To set the horizontal or the vertical orientation of the scrollbar, you can use the Orientation property. When the property changes, the OrientationChanged event fires.

See the following code snippet for reference:

- **Visual Basic**

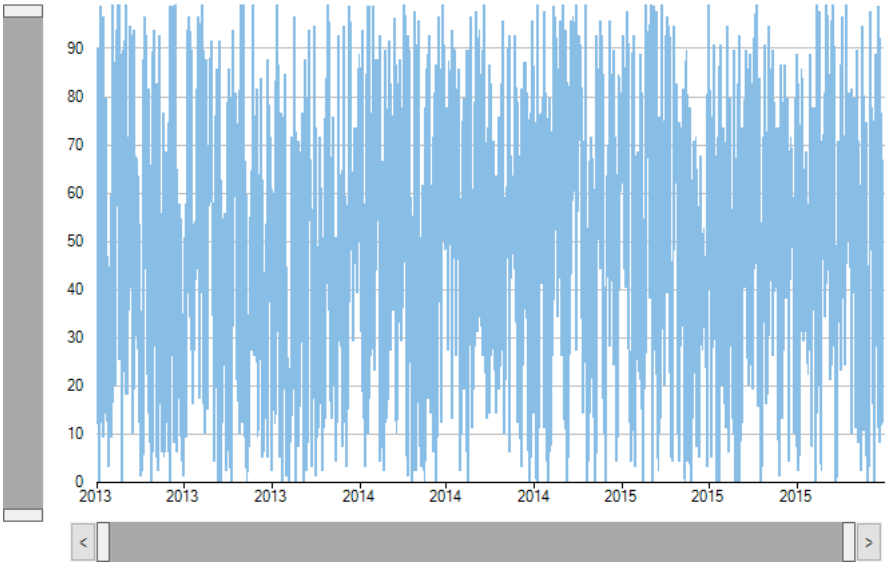
```
Imports C1.Win.Chart
Imports C1.Win.Chart.Interaction
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Linq
Imports System.Text
Imports System.Threading.Tasks
Imports System.Windows.Forms
Public Class Form1
    Private _horizontalAxisScrollbar As AxisScrollbar
    Private _verticalAxisScrollbar As AxisScrollbar
    Public Sub New()
        InitializeComponent()
        SetupChart()
    End Sub
    Public Sub SetupRangeSelector()
        If _horizontalAxisScrollbar IsNot Nothing OrElse _verticalAxisScrollbar IsNot Nothing Then
            Return
        End If

        ' add horizontal AxixScrollbar
        _horizontalAxisScrollbar = New AxisScrollbar(flexChart1.AxisX)
        AddHandler _horizontalAxisScrollbar.ValueChanged, AddressOf XRangeSelector_ValueChanged

        ' add vartical AxixScrollbar
        _verticalAxisScrollbar = New AxisScrollbar(flexChart1.AxisY)
        _verticalAxisScrollbar.ScrollButtonsVisible = False
        AddHandler _verticalAxisScrollbar.ValueChanged, AddressOf YRangeSelector_ValueChanged
    End Sub

    Private Sub XRangeSelector_ValueChanged(sender As Object, e As EventArgs)
        flexChart1.AxisX.Min = _horizontalAxisScrollbar.LowerValue
```

```
        flexChart1.AxisX.Max = _horizontalAxisScrollbar.UpperValue
    End Sub
    Private Sub YRangeSelector_ValueChanged(sender As Object, e As EventArgs)
        flexChart1.AxisY.Min = _verticalAxisScrollbar.LowerValue
        flexChart1.AxisY.Max = _verticalAxisScrollbar.UpperValue
    End Sub

    Private Sub SetupChart()
        Dim rnd = New Random()
        Dim pointsCount = rnd.[Next](1, 30)

        Dim pointsList = New List(Of DataItem)()
        Dim [date] As New DateTime(DateTime.Now.Year - 3, 1, 1)
        While [date].Year < DateTime.Now.Year
            pointsList.Add(New DataItem() With {
                .[Date] = [date],
                .Series1 = rnd.[Next](100)
            })
            [date] = [date].AddDays(1)
        End While

        flexChart1.BeginUpdate()
        flexChart1.Series.Clear()
        flexChart1.ChartType = C1.Chart.ChartType.Line

        flexChart1.BindingX = "Date"
        flexChart1.Series.Add(New Series() With {
            .Name = "Series1",
            .Binding = "Series1"
        })
        flexChart1.DataSource = pointsList.ToArray()

        flexChart1.EndUpdate()
    End Sub

    Private Class DataItem
        Public Property Series1() As Integer
            Get
                Return m_Series1
            End Get
            Set
                m_Series1 = Value
            End Set
        End Property
        Private m_Series1 As Integer
        Public Property [Date]() As DateTime
            Get
                Return m_Date
            End Get
            Set
                m_Date = Value
            End Set
        End Property
        Private m_Date As DateTime
    End Class

    Private Sub flexChart1_Rendered(sender As Object, e As RenderEventArgs)
        SetupRangeSelector()
    End Sub
End Class
```

- **C#**

```
using C1.Win.Chart;
using C1.Win.Chart.Interaction;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AxisScrollbarCS
{
    public partial class Form1 : Form
    {
        AxisScrollbar _horizontalAxisScrollbar;
        AxisScrollbar _verticalAxisScrollbar;
        public Form1()
        {
            InitializeComponent();
            SetupChart();
        }
        public void SetupRangeSelector()
        {
            if (_horizontalAxisScrollbar != null || _verticalAxisScrollbar != null) return;

            // add horizontal AxixScrollbar
            _horizontalAxisScrollbar = new AxisScrollbar(flexChart1.AxisX);
            _horizontalAxisScrollbar.ValueChanged += XRangeSelector_ValueChanged;
```

```
            // add vartical AxixScrollbar
            _verticalAxisScrollbar = new AxisScrollbar(flexChart1.AxisY);
            _verticalAxisScrollbar.ScrollButtonsVisible = false;
            _verticalAxisScrollbar.ValueChanged += YRangeSelector_ValueChanged;
        }

        void XRangeSelector_ValueChanged(object sender, EventArgs e)
        {
            flexChart1.AxisX.Min = _horizontalAxisScrollbar.LowerValue;
            flexChart1.AxisX.Max = _horizontalAxisScrollbar.UpperValue;
        }
        void YRangeSelector_ValueChanged(object sender, EventArgs e)
        {
            flexChart1.AxisY.Min = _verticalAxisScrollbar.LowerValue;
            flexChart1.AxisY.Max = _verticalAxisScrollbar.UpperValue;
        }

        void SetupChart()
        {
            var rnd = new Random();
            var pointsCount = rnd.Next(1, 30);

            var pointsList = new List<DataItem>();
            for (DateTime date = new DateTime(DateTime.Now.Year - 3, 1, 1); date.Year < DateTime.Now.Year; date = date.AddDays(1))
            {
                pointsList.Add(new DataItem()
                {
                    Date = date,
                    Series1 = rnd.Next(100)
                });
            }

            flexChart1.BeginUpdate();
            flexChart1.Series.Clear();
            flexChart1.ChartType = C1.Chart.ChartType.Line;

            flexChart1.BindingX = "Date";
            flexChart1.Series.Add(new Series() { Name = "Series1", Binding = "Series1" });
            flexChart1.DataSource = pointsList.ToArray();

            flexChart1.EndUpdate();
        }

        class DataItem
        {
            public int Series1 { get; set; }
            public DateTime Date { get; set; }
        }

        private void flexChart1_Rendered(object sender, RenderEventArgs e)
        {
            SetupRangeSelector();
        }
    }
}
```



Range Selector

While scrollbars are a traditional way of scrolling the chart, Range Selector is a more modern approach, which lets the user visualize where the selected range sits in the complete data range.

FlexChart's Range Selector lets you select a range of numeric data with lower value thumb and upper value thumb. These thumbs define the start and end values of the range. On dragging the thumb towards

left (or down) on the range bar, you reduce its value, and dragging it towards the right (or up) increases the value on the range bar.

You can add Range Selector by creating an instance of the C1.Win.Chart.Interaction.RangeSelector class. The instance accepts a parameter of the C1.Win.Chart.FlexChart class type. The C1RangeSelector class inherits the C1.Win.C1Input.C1RangeSlider class. You can use the LowerValue and the UpperValue property provided by C1RangeSlider to set the lower and the upper value of the range selector respectively. The ValueChanged event fires when any of the LowerValue or the UpperValue property is changed.

To set the horizontal or the vertical orientation of the range selector, you can use the Orientation property. When the property is changed, the OrientationChanged event fires.

Here is the code snippet showing the implementation:

- **Visual Basic**

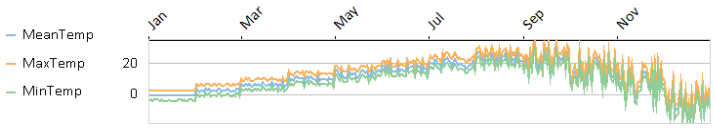
```
Private XRangeSelector As C1.Win.Chart.Interaction.RangeSelector

Public Sub New()
    InitializeComponent()
    SetupCharts()
End Sub

Public Sub SetupRangeSelector()
    If XRangeSelector IsNot Nothing Then
        Return
    End If

    ' add X-RangeSelector
    XRangeSelector = New C1.Win.Chart.Interaction.RangeSelector(FlexChart2)

    AddHandler XRangeSelector.ValueChanged, AddressOf XRangeSelector_ValueChanged
End Sub

Private Sub XRangeSelector_ValueChanged(sender As Object, e As EventArgs)
    FlexChart1.AxisX.Min = XRangeSelector.LowerValue
    FlexChart1.AxisX.Max = XRangeSelector.UpperValue
End Sub

Private Sub SetupCharts()
    Dim rnd = New Random()
    Dim pointsCount = rnd.[Next](1, 30)

    Dim temperaturePoints = New List(Of DataItem)()
    Dim [date] As New DateTime(2016, 1, 1)
    While [date].Year = 2016
        Dim newItem = New DataItem()
        newItem.[Date] = [date]
        If [date].Month <= 8 Then
            newItem.MaxTemp = rnd.[Next](3 * [date].Month, 4 * [date].Month)
        Else
            newItem.MaxTemp = rnd.[Next]((13 - [date].Month - 2) * [date].Month, (13 - [date].Month) * [date].Month)
        End If
        newItem.MinTemp = newItem.MaxTemp - rnd.[Next](6, 8)
        newItem.MeanTemp = (newItem.MaxTemp + newItem.MinTemp) / 2
        newItem.MeanPressure = rnd.[Next](980, 1050)
        newItem.Presipitation = If(rnd.[Next](5) = 1, rnd.[Next](0, 20), 0)
        temperaturePoints.Add(newItem)
        [date] = [date].AddDays(1)
    End While

    'Setup flexChart1
    FlexChart1.BeginUpdate()
    FlexChart1.ChartType = C1.Chart.ChartType.Line
    FlexChart1.Series.Clear()

    FlexChart1.BindingX = "Date"
    FlexChart1.Series.Add(New Series() With {
    .Name = "MeanTemp",
    .Binding = "MeanTemp"
})
    FlexChart1.Series.Add(New Series() With {
    .Name = "MaxTemp",
    .Binding = "MaxTemp"
})
    FlexChart1.Series.Add(New Series() With {
    .Name = "MinTemp",
    .Binding = "MinTemp"
})
    FlexChart1.DataSource = temperaturePoints.ToArray()
    FlexChart1.EndUpdate()

    'Setup flexChart2
    FlexChart2.BeginUpdate()
    FlexChart2.ChartType = C1.Chart.ChartType.Line
    FlexChart2.Series.Clear()

    FlexChart2.BindingX = "Date"
    FlexChart2.Series.Add(New Series() With {
    .Name = "MeanTemp",
    .Binding = "MeanTemp"
})
    FlexChart2.DataSource = temperaturePoints.ToArray()
    FlexChart2.EndUpdate()
End Sub

Class DataItem
    Public Property MaxTemp() As Integer
        Get
            Return m_MaxTemp
        End Get
        Set
            m_MaxTemp = Value
        End Set
    End Property
    Private m_MaxTemp As Integer
    Public Property MinTemp() As Integer
        Get
            Return m_MinTemp
        End Get
        Set
            m_MinTemp = Value
        End Set
    End Property
```

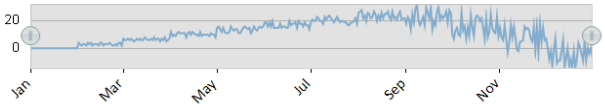
```
    Private m_MinTemp As Integer
    Public Property MeanTemp() As Integer
        Get
            Return m_MeanTemp
        End Get
        Set
            m_MeanTemp = Value
        End Set
    End Property
    Private m_MeanTemp As Integer
    Public Property MeanPressure() As Integer
        Get
            Return m_MeanPressure
        End Get
        Set
            m_MeanPressure = Value
        End Set
    End Property
    Private m_MeanPressure As Integer
    Public Property Presipitation() As Integer
        Get
            Return m_Presipitation
        End Get
        Set
            m_Presipitation = Value
        End Set
    End Property
    Private m_Presipitation As Integer
    Public Property [Date]() As DateTime
        Get
            Return m_Date
        End Get
        Set
            m_Date = Value
        End Set
    End Property
    Private m_Date As DateTime
End Class


Private Sub FlexChart2_Rendered(sender As Object, e As RenderEventArgs)
    SetupRangeSelector()
    FlexChart1_Rendered(sender, e)
End Sub

Private Sub FlexChart1_Rendered(sender As Object, e As RenderEventArgs)
    Dim flexChart = TryCast(sender, FlexChart)
    If flexChart Is Nothing Then
        Return
    End If

    Using pen = New Pen(New SolidBrush(Color.LightGray))
        Dim rect = New Rectangle(CInt(flexChart.PlotRect.X), CInt(flexChart.PlotRect.Y), CInt(flexChart.PlotRect.Width), CInt(flexChart.PlotRect.Height))
        e.Graphics.DrawRectangle(pen, rect)
    End Using
End Sub
```

- **C#**

```
C1.Win.Chart.Interaction.RangeSelector XRangeSelector;

public Form1()
{
    InitializeComponent();
    SetupCharts();
}

public void SetupRangeSelector()
{
    if (XRangeSelector != null) return;

    // add X-RangeSelector
    XRangeSelector = new C1.Win.Chart.Interaction.RangeSelector(flexChart2);

    XRangeSelector.ValueChanged += XRangeSelector_ValueChanged;
}

void XRangeSelector_ValueChanged(object sender, EventArgs e)
{
    flexChart1.AxisX.Min = XRangeSelector.LowerValue;
    flexChart1.AxisX.Max = XRangeSelector.UpperValue;

}

void SetupCharts()
{
    var rnd = new Random();
    var pointsCount = rnd.Next(1, 30);

    var temperaturePoints = new List<DataItem>();
    for (DateTime date = new DateTime(2016, 1, 1); date.Year == 2016; date = date.AddDays(1))
    {
        var newItem = new DataItem();
        newItem.Date = date;
        if (date.Month <= 8)
            newItem.MaxTemp = rnd.Next(3 * date.Month, 4 * date.Month);
        else
            newItem.MaxTemp = rnd.Next((13 - date.Month - 2) * date.Month, (13 - date.Month) * date.Month);
        newItem.MinTemp = newItem.MaxTemp - rnd.Next(6, 8);
        newItem.MeanTemp = (newItem.MaxTemp + newItem.MinTemp) / 2;
        newItem.MeanPressure = rnd.Next(980, 1050);
        newItem.Presipitation = rnd.Next(5) == 1 ? rnd.Next(0, 20) : 0;
        temperaturePoints.Add(newItem);
    }

    //Setup flexChart1
    flexChart1.BeginUpdate();
    flexChart1.ChartType = C1.Chart.ChartType.Line;
    flexChart1.Series.Clear();
```

```
    flexChart1.BindingX = "Date";
    flexChart1.Series.Add(new Series() { Name = "MeanTemp", Binding = "MeanTemp" });
    flexChart1.Series.Add(new Series() { Name = "MaxTemp", Binding = "MaxTemp" });
    flexChart1.Series.Add(new Series() { Name = "MinTemp", Binding = "MinTemp" });
    flexChart1.DataSource = temperaturePoints.ToArray();
    flexChart1.EndUpdate();

    //Setup flexChart2
    flexChart2.BeginUpdate();
    flexChart2.ChartType = C1.Chart.ChartType.Line;
    flexChart2.Series.Clear();

    flexChart2.BindingX = "Date";
    flexChart2.Series.Add(new Series() { Name = "MeanTemp", Binding = "MeanTemp" });
    flexChart2.DataSource = temperaturePoints.ToArray();
    flexChart2.EndUpdate();
}

class DataItem
{
    public int MaxTemp { get; set; }
    public int MinTemp { get; set; }
    public int MeanTemp { get; set; }
    public int MeanPressure { get; set; }
    public int Presipitation { get; set; }
    public DateTime Date { get; set; }
}

private void flexChart2_Rendered(object sender, RenderEventArgs e)
{
    SetupRangeSelector();
    flexChart1_Rendered(sender, e);
}

private void flexChart1_Rendered(object sender, RenderEventArgs e)
{
    var flexChart = (sender as FlexChart);
    if (flexChart == null) return;

    using (var pen = new Pen(new SolidBrush(Color.LightGray)))
    {
        var rect = new Rectangle(
            (int)flexChart.PlotRect.X,
            (int)flexChart.PlotRect.Y,
            (int)flexChart.PlotRect.Width,
            (int)flexChart.PlotRect.Height
            );
        e.Graphics.DrawRectangle(pen, rect);
    }
}
```

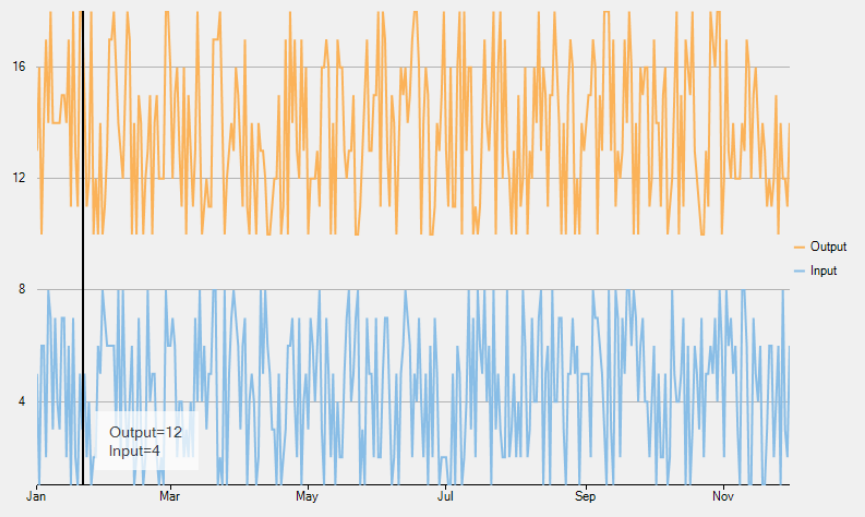




Line Marker

LineMarker displays the precise data values for a given position on the chart by dragging horizontal or vertical lines over the plot with an attached label. It is useful in scenarios, where a user has a lot of data in a line or area chart, or if a user wants to display data from multiple series in a single label. With built-in interactions, such as Drag and Move, a user can drag the line marker and more precisely select the data point on the chart.

The following image shows LineMarker displaying data values for a specific position on the chart.



To create a line marker and use it in FlexChart, create an instance of the C1.Win.Chart.Interaction.LineMarker class. While creating the instance, pass a parameter that is an object of the C1.Win.Chart.FlexChart type. Use the Lines property provided by LineMarker to set the visibility of the LineMarker lines. The Lines property accepts the following values from the LineMarkerLines enumeration:

- **Both**: Shows both vertical and horizontal lines
- **Horizontal**: Shows a horizontal line
- **Vertical**: Shows a vertical line
- **None**: Shows no line

The LineMarker class also provides the Alignment property to set the alignment of the line marker. In addition, set the interaction mode of the line marker by setting the Interaction property to any of the following values in the LineMarkerInteraction enumeration:

- **Drag**: The line marker moves when the user drags the line
- **Move** (Default): The line marker moves with the pointer
- **None**: The user specifies the position by clicking.

If you set the Interaction property to Drag, you need to set the DragContent and the DragLines property to specify whether the content and values linked with the line marker lines are draggable or not. In addition, the LineMarker class provides the Content property that you can use to customize the text content of the line marker.

The following code snippet implements the aforementioned classes and properties.

- **Visual Basic**

```
Imports System.Windows.Forms
Imports C1.Win.Chart.Interaction
Imports System.Drawing
Imports C1.Win.Chart
Imports System.Collections.Generic

Partial Public Class Form1
    Inherits Form
    Private lineMarker As C1.Win.Chart.Interaction.LineMarker

    Public Sub New()
        InitializeComponent()
        SetupChart()
    End Sub

    Private Sub SetupChart()

        Dim rnd = New Random()
        Dim pointsCount = rnd.[Next](1, 30)

        Dim pointsList = New List(Of DataItem)()
        Dim [date] As New DateTime(DateTime.Now.Year - 1, 1, 1)
        While [date].Month < DateTime.Now.Month + 2
            pointsList.Add(New DataItem() With {
                .[Date] = [date],
                .Input = rnd.[Next](1, 9),
                .Output = rnd.[Next](10, 19)
            })
            [date] = [date].AddDays(1)
        End While

        FlexChart1.BeginUpdate()
        FlexChart1.Series.Clear()
        FlexChart1.ChartType = C1.Chart.ChartType.Line

        FlexChart1.BindingX = "Date"

        Dim outputSerie = New C1.Win.Chart.Series() With {
            .Name = "Output",
            .Binding = "Output"
        }
        outputSerie.Style.StrokeColor = Color.FromArgb(255, 251, 178, 88)
        FlexChart1.Series.Add(outputSerie)
```

```
        Dim inputSerie = New C1.Win.Chart.Series() With {
            .Name = "Input",
            .Binding = "Input"
        }
        inputSerie.Style.StrokeColor = Color.FromArgb(255, 136, 189, 230)
        FlexChart1.Series.Add(inputSerie)

        FlexChart1.DataSource = pointsList.ToArray()

        FlexChart1.EndUpdate()

        AddHandler FlexChart1.Rendered, AddressOf flexChart1_Rendered
    End Sub

    Private Sub flexChart1_Rendered(sender As Object, e As RenderEventArgs)
        If lineMarker Is Nothing Then
            Dim lineMarker As New C1.Win.Chart.Interaction.LineMarker(FlexChart1)
            lineMarker.LineWidth = 2
            lineMarker.DragThreshold = 10
            lineMarker.Content = "Output={Output}" & vbLf & "Input={Input}"
            lineMarker.Alignment = LineMarkerAlignment.Auto
            lineMarker.Interaction = LineMarkerInteraction.Move
            lineMarker.Lines = LineMarkerLines.Vertical
        End If
    End Sub

    Private Class DataItem
        Public Property Input() As Integer
            Get
                Return m_Input
            End Get
            Set
                m_Input = Value
            End Set
        End Property
        Private m_Input As Integer
        Public Property Output() As Integer
            Get
                Return m_Output
            End Get
            Set
                m_Output = Value
            End Set
        End Property
        Private m_Output As Integer
        Public Property [Date]() As DateTime
            Get
                Return m_Date
            End Get
            Set
                m_Date = Value
            End Set
        End Property
        Private m_Date As DateTime
    End Class
End Class
```

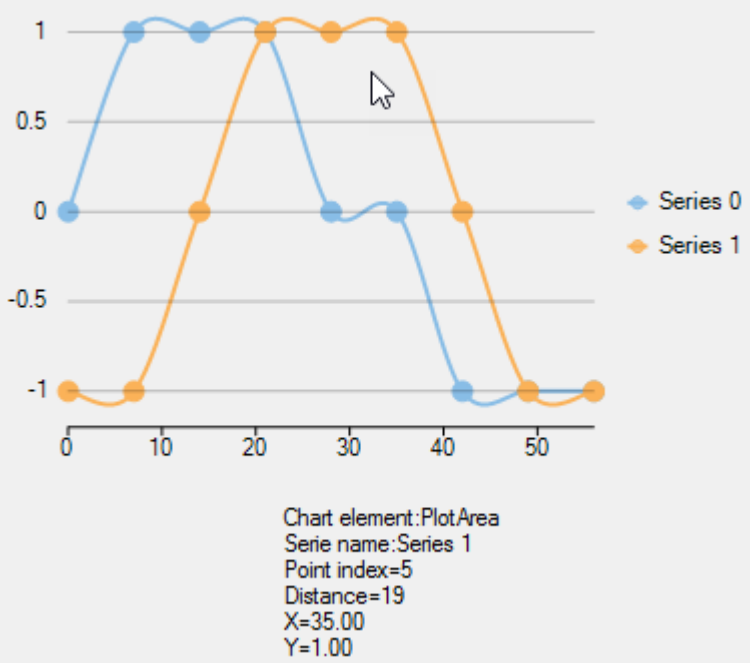
- **C#**

```
using System;
using System.Windows.Forms;
using C1.Win.Chart.Interaction;
using System.Drawing;
using C1.Win.Chart;
using System.Collections.Generic;

namespace LineMarker
{
    public partial class Form1 : Form
    {
        C1.Win.Chart.Interaction.LineMarker lineMarker;

        public Form1()
        {
            InitializeComponent();
            SetupChart();
        }

        void SetupChart()
        {

            var rnd = new Random();
            var pointsCount = rnd.Next(1, 30);

            var pointsList = new List<DataItem>();
            for (DateTime date = new DateTime(DateTime.Now.Year - 1, 1, 1); date.Month < DateTime.Now.Month + 2; date = date.AddDays(1))
            {
                pointsList.Add(new DataItem()
                {
                    Date = date,
                    Input = rnd.Next(1, 9),
                    Output = rnd.Next(10, 19)
                });
            }

            flexChart1.BeginUpdate();
            flexChart1.Series.Clear();
```

```
            flexChart1.ChartType = C1.Chart.ChartType.Line;

            flexChart1.BindingX = "Date";

            var outputSerie = new C1.Win.Chart.Series() { Name = "Output", Binding = "Output" };
            outputSerie.Style.StrokeColor = Color.FromArgb(255, 251, 178, 88);
            flexChart1.Series.Add(outputSerie);

            var inputSerie = new C1.Win.Chart.Series() { Name = "Input", Binding = "Input" };
            inputSerie.Style.StrokeColor = Color.FromArgb(255, 136, 189, 230);
            flexChart1.Series.Add(inputSerie);

            flexChart1.DataSource = pointsList.ToArray();

            flexChart1.EndUpdate();

            flexChart1.Rendered += flexChart1_Rendered;
        }

        void flexChart1_Rendered(object sender, RenderEventArgs e)
        {
            if (lineMarker == null)
            {
                C1.Win.Chart.Interaction.LineMarker lineMarker = new C1.Win.Chart.Interaction.LineMarker(flexChart1);
                lineMarker.LineWidth = 2;
                lineMarker.DragThreshold = 10;
                lineMarker.Content = "Output={Output}\nInput={Input}";
                lineMarker.Alignment = LineMarkerAlignment.Auto;
                lineMarker.Interaction = LineMarkerInteraction.Move;
                lineMarker.Lines = LineMarkerLines.Vertical;
            }
        }

        class DataItem
        {
            public int Input { get; set; }
            public int Output { get; set; }
            public DateTime Date { get; set; }
        }

    }
}
```

Hit Test

FlexChart supports hit testing, which enables you to fetch information about a specific point in the control at run-time. The information obtained about the pointed coordinate can then be reused to drill down the chart data, to set alerts, or to enable other user interaction functionalities.



FlexChart supports hit testing by utilizing HitTest() method. This method takes two overloads, one accepts pointer

location and the other accepts pointer location, MeasureOption, and seriesIndex value in the control as parameters; and returns an object of HitTestInfo class, which provides the following information about the pointer location:

- Chart element comprising the pointer
- Distance of the pointer location from the closest data point in chart, if the pointer is within plot area. Distance is returned as Double.Nan, if the pointer is outside the plot area.
  This distance value varies depending upon the MeasureOption being used. For example, with MeasureOption.X, the Distance is calculated along the X-axis only.
- Data object corresponding to the closest data point
- Index of the nearest data point
- Series name that the nearest data point belongs to
- X value of the nearest data point
- Y value of the nearest data point

Note that, the mouse coordinates that are passed to **HitTest()** method are in pixels and are relative to the upper left corner of the form.

In this example, **HitTest()** method is called on MouseMove event of the FlexChart control. Here, the point coordinates of pointer location are passed as parameter to HitTest() method.

To enable hit testing in FlexChart, follow these steps:

1. **Add a data bound FlexChart control**
2. **Subscribe to a Mouse event**
3. **Invoke chart's HitTest method in mouse event handler**
4. **Use the information returned by HitTestInfo object**

Back to Top

1. **Add a data bound FlexChart control**

   Add an instance of FlexChart control to your application, and bind it to an appropriate data source, as shown in the below code snippet.

   - **C#**

```
const int SERIES_NUMBER = 2;
void SetupChart()
{
    flexChart2.BeginUpdate();
    flexChart2.Series.Clear();

    // Create sample data and add it to the chart
    var offset = 0;
    for (int i = 0; i < SERIES_NUMBER; i++)
    {
        Series tbs = new Series()
        {
            Name = "Series " + i.ToString(),
            DataSource = DataCreator.Create(x => Math.Sin(x + offset), 0, 60, 7),
            Binding = "YVals",
            BindingX = "XVals"
        };

        flexChart2.ChartType = C1.Chart.ChartType.SplineSymbols;

        flexChart2.Series.Add(tbs);
        offset += 5;
    }
    flexChart2.EndUpdate();
```

```
}
```

Back to Top

2. **Subscribe to a Mouse event**

   Subscribe to a mouse event to capture the pointer coordinates, as shown in the below code snippet.

   - **C#**

```
flexChart2.MouseDoubleClick += flexChart2_MouseDoubleClick;
```

Back to Top

3. **Invoke chart's HitTest method in mouse event handler**

   In the respective event handler, invoke the **HitTest()** method and pass the captured mouse pointer coordinates, as shown in the below code snippet.

   - **C#**

```
private void flexChart2_MouseDoubleClick(object sender, MouseEventArgs e)
{
    // Show information about chart element under mouse cursor
    var ht = flexChart2.HitTest(e.Location);
```

Back to Top

4. **Use the information returned by HitTestInfo object**

   The information regarding mouse pointer location, as returned by the **HitTestInfo** object, can then be reused. For example in the below code snippet, the values returned by **HitTestInfo** object are converted to string and displayed in a TextBlock.

   - **C#**

```
    ...
    var result = new StringBuilder();
    result.AppendLine(string.Format("Chart element:{0}", ht.ChartElement));
    if (ht.Series != null)
        result.AppendLine(string.Format("Serie name:{0}", ht.Series.Name));

    //result.AppendLine(String.Format("Item", ht.Item));
    if (ht.PointIndex > 0)
        result.AppendLine(string.Format("Point index={0:0}", ht.PointIndex));
    if (ht.Distance > 0)
        result.AppendLine(string.Format("Distance={0:0}", ht.Distance));
    if (ht.X != null)
        result.AppendLine(string.Format("X={0:0.00}", ht.X));
    if (ht.Y != null)
        result.AppendLine(string.Format("Y={0:0.00}", ht.Y));
    lblPosition.Text = result.ToString();
}
```

Back to Top

Design-Time Support

FlexChart offers collection editors that simplify working with the object model. Collection editors allow you to access several useful properties from a single place. And this eradicates the need for you to drill down properties in the Properties window.

Know more about collection editors by referring to the following section:

- Collection Editors

Collection Editors

FlexChart renders a number of collection editors that let you apply properties to different chart elements at design-time.

The control provides you with the below-mentioned collection editors:

- Series Collection Editor

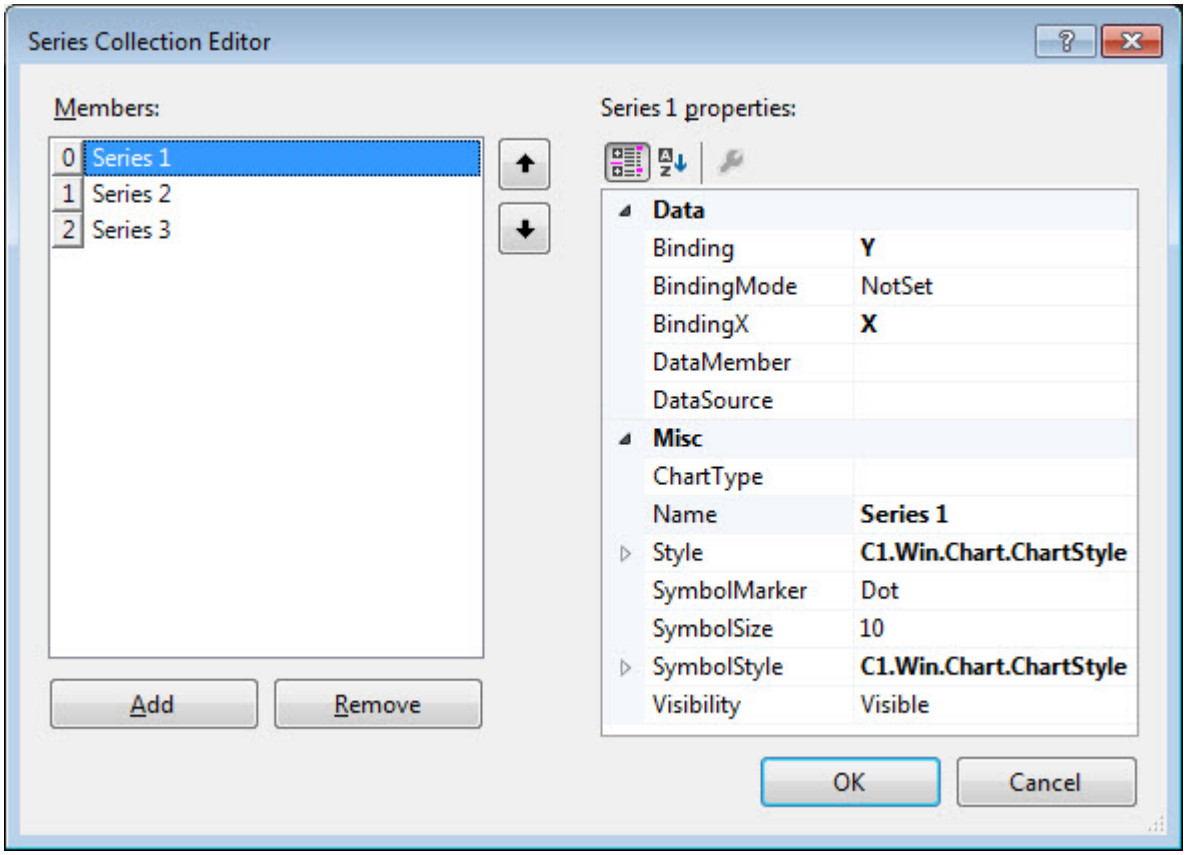
Series Collection Editor

Series Collection Editor allows you to add or remove series and modify their properties.

For information on the Series object, refer to Series.

Accessing Series Collection Editor

1. Right-click the **FlexChart** control.
2. Select **Properties** from the context menu.
3. In the **Properties** window, click the **ellipsis** button next to the **Series** property.
   Series Collection Editor appears, as shown below:



The following table lists the properties available in Series Collection Editor:

Properties	Description
DataSource	Sets the collection of objects containing the series data.
ChartType	Sets the series chart type.
Name	Sets the name of the series.
Visibility	Sets the visibility of the series.

FlexChart Elements

You can customize the elements of a chart to make the chart look more professional and visually appealing.

FlexChart consists of Axes, Legend, and Titles. These elements have already been discussed briefly in FlexChart Fundamentals.

Below are the sections that focus upon the customization of these elements with respect to FlexChart.

- FlexChart Axes
- FlexChart Axes Labels
- FlexChart Annotations
- FlexChart Legend
- FlexChart Series
- FlexChart Data Labels

Axes

Charts generally have two axes for measuring and categorizing data: a vertical axis (Y-axis) and a horizontal axis (X-axis). The vertical axis is also known as value axis, and the horizontal axis is also called category axis.

Not all charts depict axes in the same manner. For instance, Scatter charts and Bubble charts depict numeric values on the vertical axis as well as the horizontal axis to represent discreet or continuous numerical data. A real-time example could be how Internet Usage (Hours per Week) is plotted against different Age Groups. Here, both the items will have numeric values, and data points will be plotted corresponding to their numeric values on X and Y axes.

Other charts, such as Line, Column, Bar, and Area display numeric values on the vertical axis and categories on the horizontal axis. A real-time example could be how Internet Usage (Hours per Week) is plotted against different regions. Here, regions will be textual categories plotted on the horizontal axis.

However, FlexChart provides great flexibility, thereby allowing you to display numeric values on both X and Y axes even in case of Bar, Line, and Area charts. Also, FlexChart doesn't require any additional settings to display different types of values.

An axis in FlexChart is represented by the Axis class. You can access the primary axes of FlexChart by using the AxisX and the AxisY property.

The primary X-axis is rendered horizontally at the bottom, and the primary Y-axis is rendered vertically at the left. You can, however, create exceptions to this rule by customizing the primary axes and also by using multiple axes.

While working with FlexChart, you can change the way tick marks and axes labels appear. You can even lessen the number of axes labels on X and Y axes by specifying the number of units between values on the axes. In addition, you can modify the alignment and orientation of the labels and change the format of the numbers to be depicted. You can style the axes and change their position as per your requirements as well.

The sections, which are mentioned below, explain different customizations and modifications possible with respect to the FlexChart axes.

Axes Position

FlexChart lets you change the position of the axes by using the Position property.

The Position property for an axis can be set to the following values in the Position enumeration:

Property	Description
Position.Auto	Positions the item automatically.
Position.Bottom	Positions the item at the bottom.
Position.Left	Positions the item at the left.
Position.None	Hides the item.
Position.Right	Positions the item at the right.
Position.Top	Positions the item at the top.

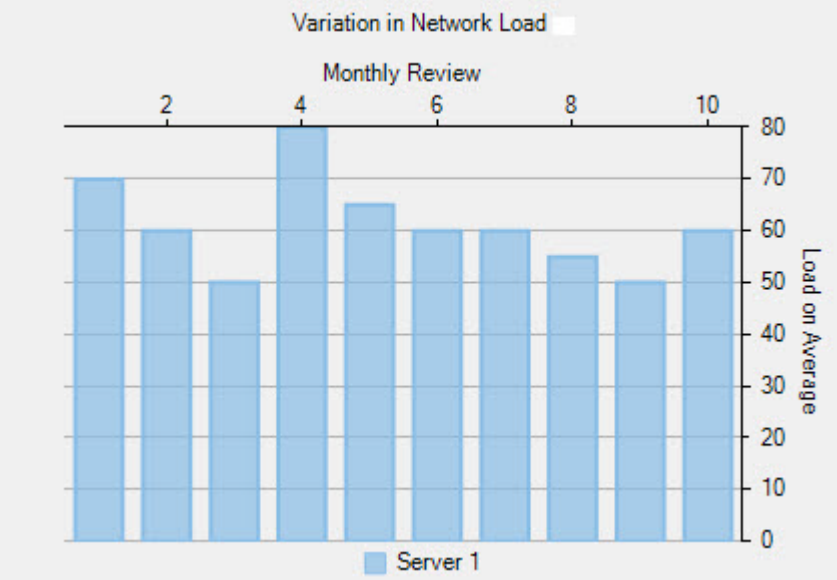
Here is the sample code:

- **Visual Basic**

```
' this causes X axis to be rendered at the top horizontally
FlexChart1.AxisX.Position = C1.Chart.Position.Top

' this causes Y axis to be rendered at the right vertically
FlexChart1.AxisY.Position = C1.Chart.Position.Right
```

- **C#**

```
// this causes X axis to be rendered at the top horizontally
flexChart1.AxisX.Position = C1.Chart.Position.Top;

// this causes Y axis to be rendered at the right vertically
flexChart1.AxisY.Position = C1.Chart.Position.Right;
```



Axes Styling

FlexChart allows you to style the axes by using different properties mentioned below:

Property	Description
FillColor	Sets the fill color.
Font	Sets the font of the element.
StrokeColor	Sets the stroke color.
StrokeWidth	Sets the stroke width.

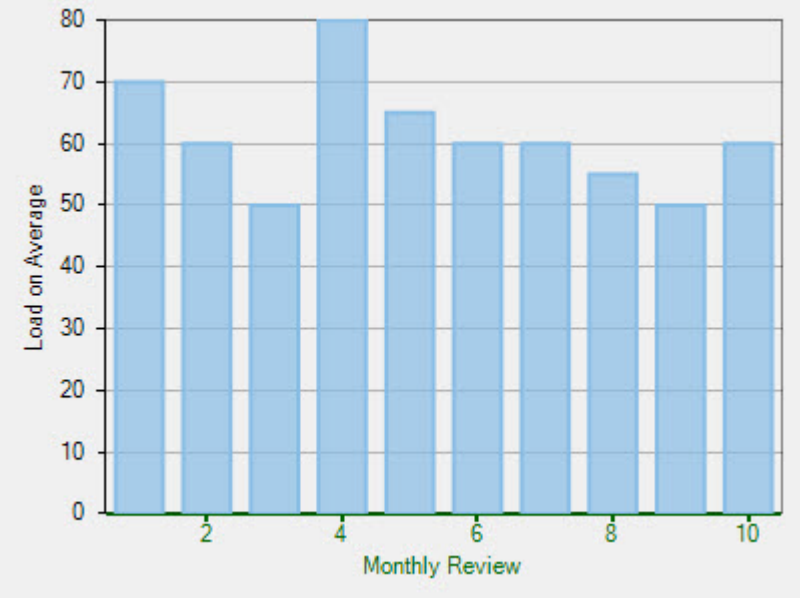
The sample code is given below:

- **Visual Basic**

```
' set the stroke color
FlexChart1.AxisX.Style.StrokeColor = System.Drawing.Color.DarkGreen

' specify the stroke width
FlexChart1.AxisX.Style.StrokeWidth = 2
```

- **C#**

```
// set the stroke color
flexChart1.AxisX.Style.StrokeColor = System.Drawing.Color.DarkGreen;

// specify the stroke width
flexChart1.AxisX.Style.StrokeWidth = 2;
```



Axes Title

After creating a chart, you can add a title to any vertical or horizontal axis in the chart. An axis title displays information regarding what is displayed along the axis. And it enables end-users viewing the chart to understand what the data is about. It is however not possible to add axis titles to charts without axes, for instance Pie Chart.

In FlexChart, you can set the axis title by using the Title property, which accepts a string.

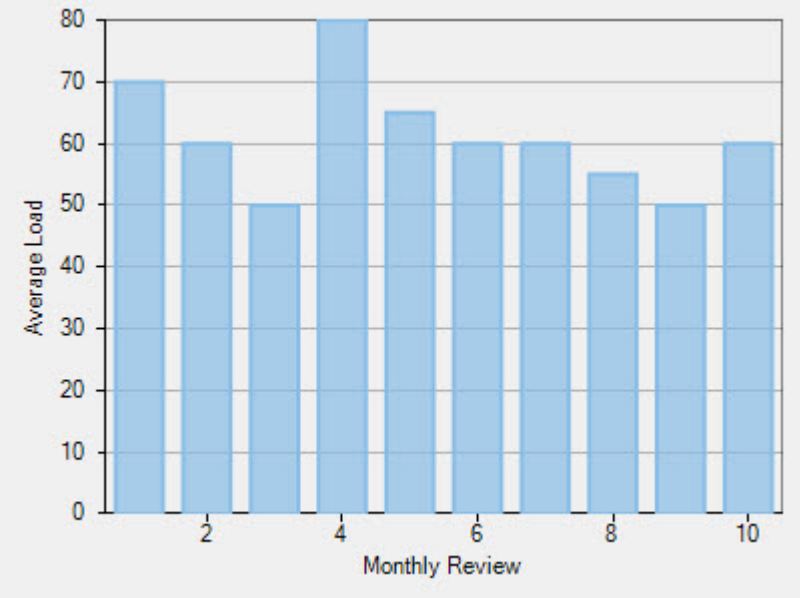
See the following code snippet:

- **Visual Basic**

```
' set the X axis title
FlexChart1.AxisX.Title = "Monthly Review"

' set the Y axis title
FlexChart1.AxisY.Title = "Average Load"
```

- **C#**

```
// set the X axis title
flexChart1.AxisX.Title = "Monthly Review";

// set the Y axis title
flexChart1.AxisY.Title = "Average Load";
```



Axes Tick Marks

Axes tick marks are the points at which labels are plotted on the axes. In other words, they are the small marks that identify the position of items on the axes. In addition, they divide axes into equal sections by a value determined by specific properties of an axis. And their location controls the location of grid lines.

When it comes to axes tick marks, a chart is rendered with two types basically: major tick marks and minor tick marks. Major tick marks are rendered automatically when an axis intersects the interval grid lines. And minor tick marks are rendered between major tick marks.

By default, FlexChart sets up X-axis with major tick marks and Y-axis with no tick marks.

You can, however, use the MajorTickMarks and the MinorTickMarks property to manipulate the position of the major tick marks and the minor tick marks respectively.

Both the properties can be set to any of the following TickMark enumeration values:

Values	Description
TickMark.Cross	Tick marks cross the axis.
TickMark.Inside	Tick marks appear inside the plot.
TickMark.None	Tick marks don't appear.
TickMark.Outside	Tick marks appear outside the plot.

See the following code sample:

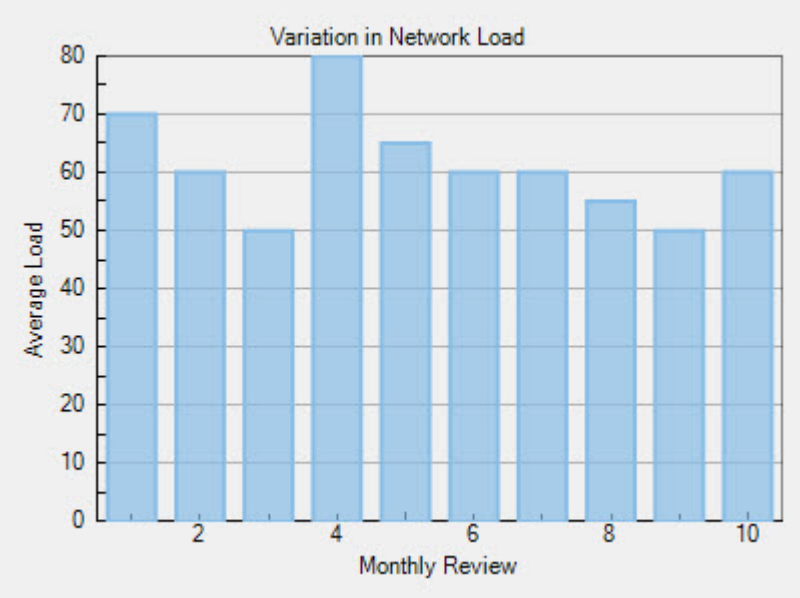
- **Visual Basic**

```
' set the major tick marks for X axis and Y axis
FlexChart1.AxisX.MajorTickMarks = C1.Chart.TickMark.Inside
FlexChart1.AxisY.MajorTickMarks = C1.Chart.TickMark.Inside

' set the minor tick marks for X axis and Y axis
FlexChart1.AxisX.MinorTickMarks = C1.Chart.TickMark.Inside
```

```
FlexChart1.AxisY.MinorTickMarks = C1.Chart.TickMark.Inside
```

- **C#**

```
// set the major tick marks for X axis and Y axis
flexChart1.AxisX.MajorTickMarks = C1.Chart.TickMark.Inside;
flexChart1.AxisY.MajorTickMarks = C1.Chart.TickMark.Inside;

// set the minor tick marks for X axis and Y axis
flexChart1.AxisX.MinorTickMarks = C1.Chart.TickMark.Inside;
flexChart1.AxisY.MinorTickMarks = C1.Chart.TickMark.Inside;
```



Axes Grid Lines

Axes grid lines extend from any vertical or horizontal axis across the plot area of the chart. They are displayed for major and minor units and aligned with major and minor tick marks displayed on the axes. These auxiliary lines form a grid that improves the readability of the chart, especially when you are looking for exact values.

Primarily, axes grid lines are of two types: major grid lines and minor grid lines. The lines perpendicular to major tick marks at major unit intervals are major gridlines, while those perpendicular to minor tick marks at minor unit intervals are minor grid lines.

In FlexChart, major grid lines are controlled by the MajorGrid property, while minor grid lines are controlled by the MinorGrid property. In addition, the appearances of the major and the minor grid lines are controlled by the MajorGridStyle and the MinorGridStyle property respectively.

Using these properties, you can display horizontal as well as vertical grid lines to make the FlexChart data easier to read.

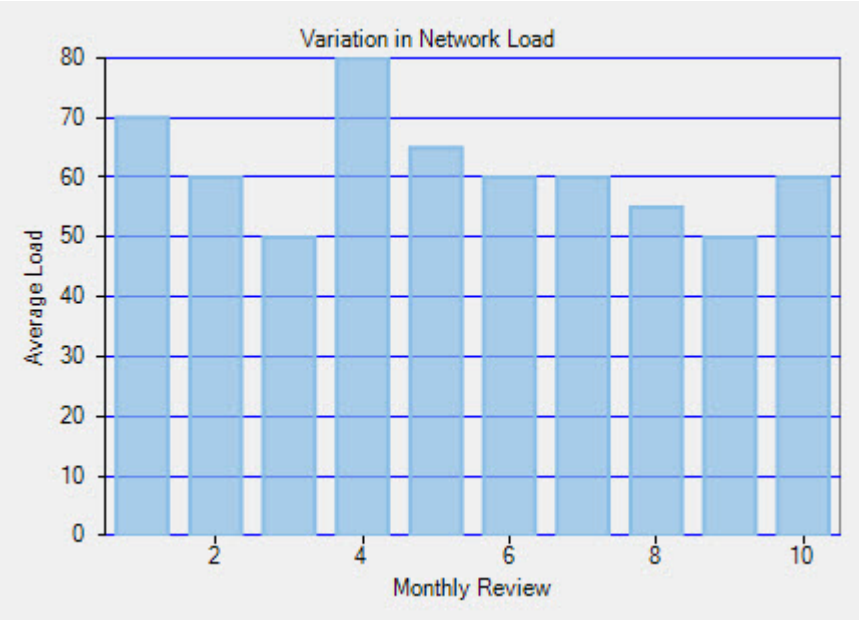
The code below illustrates how to set these properties.

- **Visual Basic**

```
' set the major grid lines for X-axis and Y-axis
FlexChart1.AxisX.MajorGrid = False
FlexChart1.AxisY.MajorGrid = True

' style the grid lines
FlexChart1.AxisX.MajorGridStyle.StrokeColor = System.Drawing.Color.Blue
FlexChart1.AxisY.MajorGridStyle.StrokeColor = System.Drawing.Color.Blue
```

- **C#**

```
// set the major grid lines for X-axis and Y-axis
flexChart1.AxisX.MajorGrid = false;
flexChart1.AxisY.MajorGrid = true;

// style the grid lines
flexChart1.AxisX.MajorGridStyle.StrokeColor = System.Drawing.Color.Blue;
flexChart1.AxisY.MajorGridStyle.StrokeColor = System.Drawing.Color.Blue;
```



Axes Bounds

If you want to display a specific portion of the chart in terms of data, you can do so by fixing the axes bounds. With axes bounds, the chart determines the extent of each axis by reckoning the lowest and the highest data values.

FlexChart enables you to set axes bounds by setting the Min and the Max property for the axes.

The following code shows how to set the Min and the Max property:

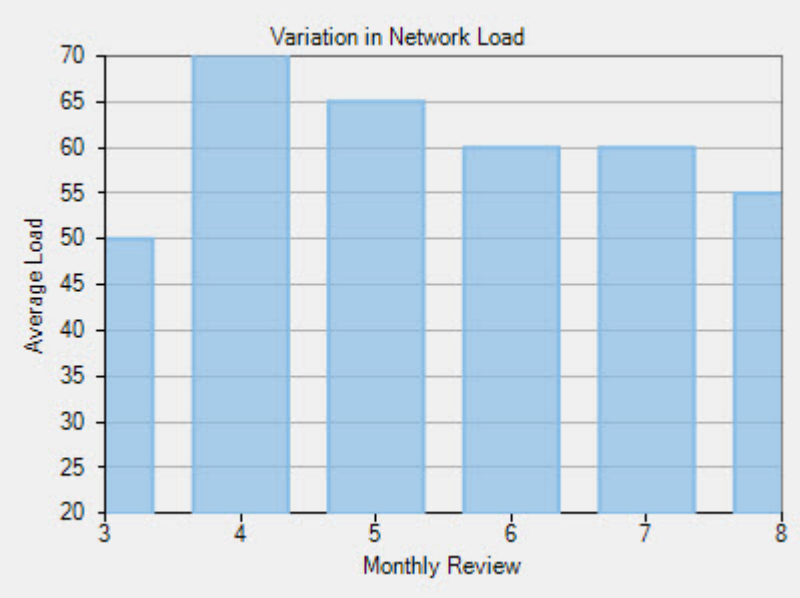
- **Visual Basic**

```
' set Min and Max values for X-axis
FlexChart1.AxisX.Min = 3
FlexChart1.AxisX.Max = 8

' set Min and Max values for Y-axis
FlexChart1.AxisY.Min = 20
FlexChart1.AxisY.Max = 70
```

- **C#**

```
// set Min and Max values for X-axis
flexChart1.AxisX.Min = 3;
flexChart1.AxisX.Max = 8;

// set Min and Max values for Y-axis
flexChart1.AxisY.Min = 20;
```

```
flexChart1.AxisY.Max = 70;
```



Axes Scaling

Sometimes, you require distinguishing the data plotted by the series in the chart. The need arises when the data points of the series do not fall in the same range. In other words, the Y axes of the series contain values in different ranges. For instance, there could be two series. The Y values for one might lie between 0 and 100 and that for the other between 0 and -100. In addition, the data of the series could require different scales altogether. In such cases, displaying the Y values of the series on a single Y-axis can confuse the interpretation of the data and overlap the same as well.

FlexChart allows you to deal with such cases by letting you scale the axes (primary and additional) by using the Min, the Max, and the MajorUnit properties. You can even apply these properties to an additional Y-axis for plotting the Y values for one of the series for better data representation.

Axes Reversing

When a dataset contains X or Y values that lie in a large range, the general chart setup sometimes doesn't display the information most effectively. Often, the chart data may look more appealing with the axes reversed.

You can reverse the axes in FlexChart by using the Reversed property.

Setting the Reversed property for the axes to True reverses the axes. This means that the maximum value along the axis takes the place of the minimum value, and the minimum value along the axis takes the place of the maximum value.

Initially, the chart displays the minimum value on the left of X-axis, and at the bottom of Y-axis. However, the Reversed property for the axes juxtaposes the maximum and minimum values.

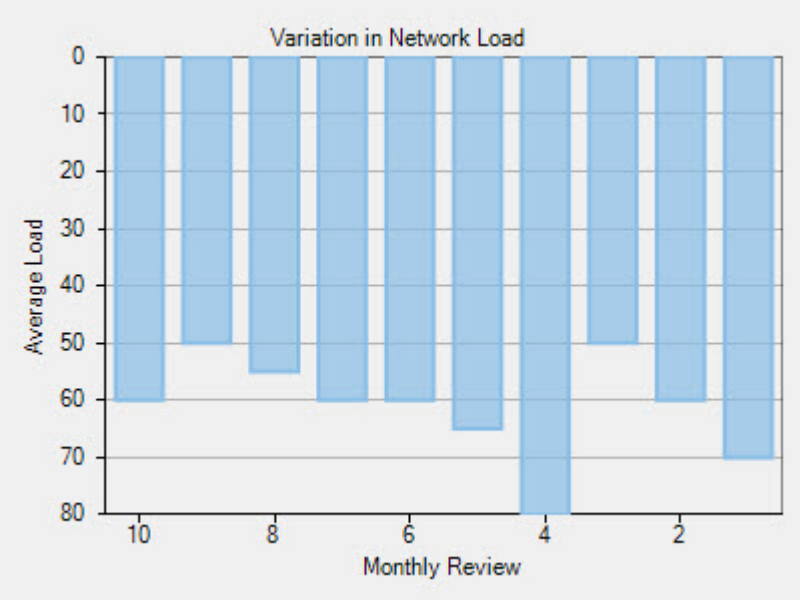
Here is the sample code:

- **Visual Basic**

```
FlexChart1.AxisX.Reversed = True
FlexChart1.AxisY.Reversed = True
```

- **C#**

```
flexChart1.AxisX.Reversed = true;
flexChart1.AxisY.Reversed = true;
```

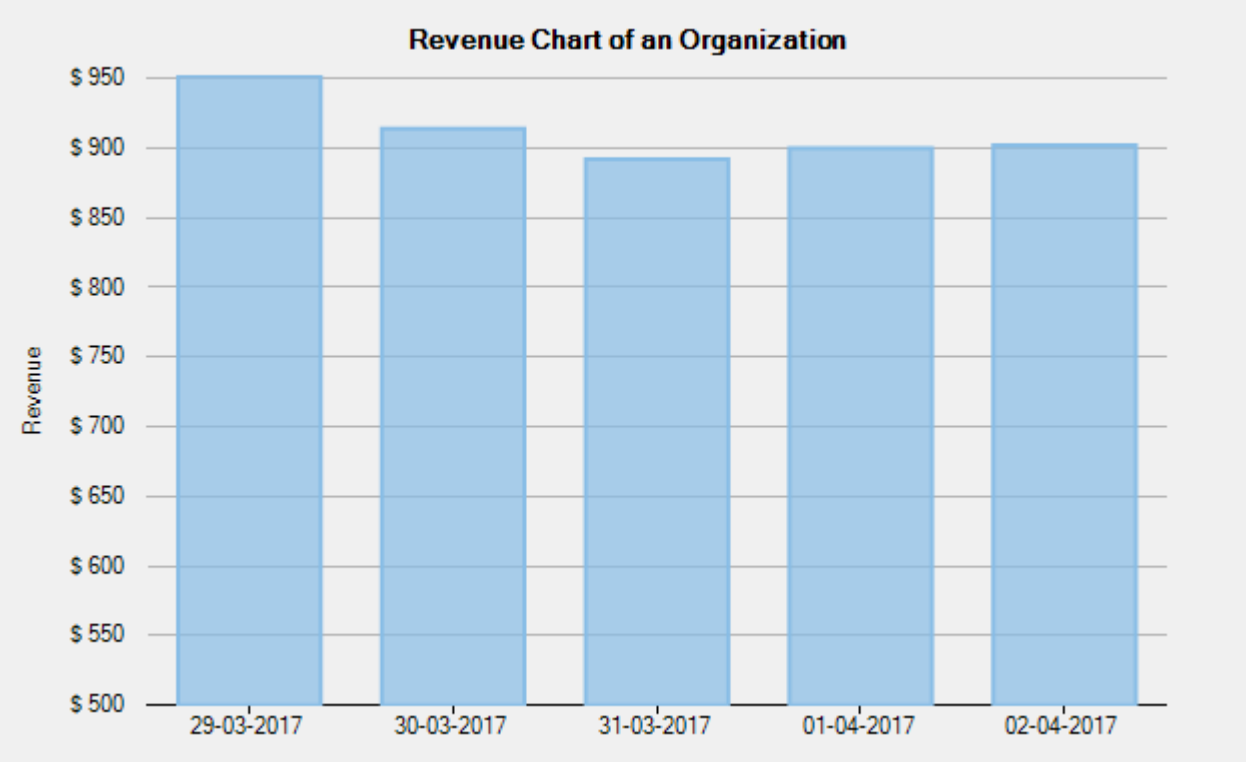


Axes Binding

Axis binding enables you to override the default axes labels that the axes display based on the chart binding. In other words, axis binding lets you show axes labels from a data source other than the chart data source.

FlexChart allows you to bind axes to a data source using the DataSource property of the Axis class. Specify the fields containing values for the axes labels in the data source using the Binding property of the Axis class.

The following image displays labels on Y-axis from the fields not part of the chart data source.



The following code uses revenue data of an organization in a given year. The chart datasource contains the revenue data in the Euro currency. To replace euro currency axis labels with USD currency labels, the code binds Y-axis to a data source containing USD data.

- **Visual Basic**

```
' bind Y axis to a data source
FlexChart1.AxisY.DataSource = axisDataSource

' specify fields containing values for axis labels
FlexChart1.AxisY.Binding = "Value,Text"
```

- **C#**

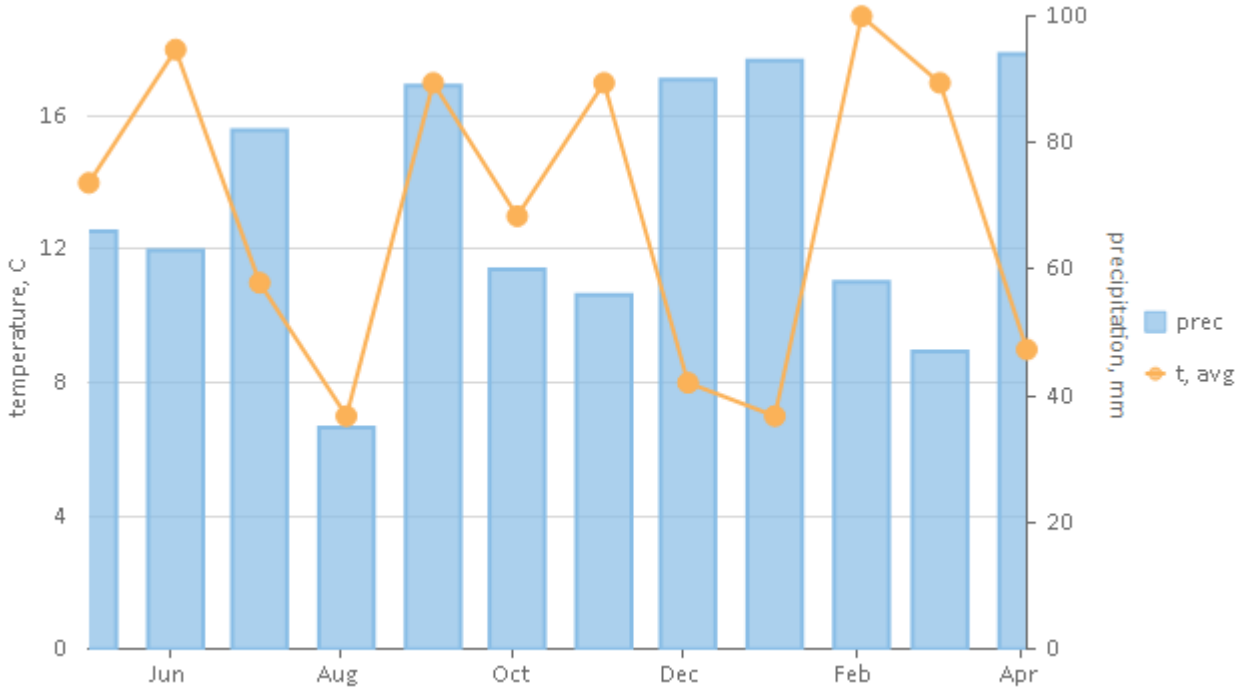
```
// bind Y axis to a data source
flexChart1.AxisY.DataSource = axisDataSource;

// specify fields containing values for axis labels
flexChart1.AxisY.Binding = "Value,Text";
```

Multiple Axes

Although a chart contains primary X and Y axes, you may still sometimes require additional axes to fulfill your requirements. For example, you may want to plot series of a significantly different range of values in a chart. In addition, you may want to plot entirely different values (of different types) within a single chart. With just two axes, it would not be possible to display data in such scenarios effectively. In such cases, using secondary axes would come in handy. To use secondary axes, you can plot multiple series in a single chart with their own X and Y axes.

FlexChart allows you to work with multiple axes easily. You just need to create additional axes as per your requirements, and then bind the same to the AxisX and the AxisY property of a series.

The following image shows two Y axes. one primary and another auxillary, along with X axis in FlexChart.



The following code snippet demonstrates how you can create and use multiple axes in FlexChart:

- **Visual Basic**

```
Partial Public Class Form1
    Inherits Form
    Private npts As Integer = 12
    Private rnd As New Random()
    Public Sub New()
        InitializeComponent()

        ' Create sample data
        Dim data = New List(Of DataItem)()
        Dim dt = DateTime.Today
        For i As Object = 0 To npts - 1
            data.Add(New DataItem() With {
                .Time = dt.AddMonths(i),
                .Precipitation = rnd.[Next](30, 100),
                .Temperature = rnd.[Next](7, 20)
            })
        Next

        FlexChart1.BeginUpdate()
        FlexChart1.Series.Clear()

        ' 1st series with auxiliary axis
        Dim series1 = New Series()
        series1.Name = "prec"
        series1.Binding = "Precipitation"
        series1.ChartType = ChartType.Column
        series1.AxisY = New Axis() With {
            .Position = Position.Right,
            .Min = 0,
            .Max = 100,
            .Title = "precipitation, mm"
        }
```

```
        ' 2nd series
        Dim series2 = New Series()
        series2.Name = "t, avg"
        series2.Binding = "Temperature"
        series2.ChartType = ChartType.LineSymbols

        ' Add data to the chart
        FlexChart1.Series.Add(series1)
        FlexChart1.Series.Add(series2)
        FlexChart1.BindingX = "Time"
        FlexChart1.DataSource = data


        FlexChart1.ChartType = ChartType.Column

        ' Set axis appearance
        FlexChart1.AxisY.Title = "temperature, C"
        FlexChart1.AxisY.Min = 0
        FlexChart1.AxisY.MajorGrid = True
        FlexChart1.EndUpdate()
    End Sub
    Public Class DataItem
        Public Property Time() As DateTime
            Get
                Return m_Time
            End Get
            Set
                m_Time = Value
            End Set
        End Property
        Private m_Time As DateTime

        Public Property Precipitation() As Integer
            Get
                Return m_Precipitation
            End Get
            Set
                m_Precipitation = Value
            End Set
        End Property
        Private m_Precipitation As Integer
        Public Property Temperature() As Integer
            Get
                Return m_Temperature
            End Get
            Set
                m_Temperature = Value
            End Set
        End Property
        Private m_Temperature As Integer
    End Class
End Class
```

- **C#**

```
public partial class Form1 : Form
{
    int npts = 12;
    Random rnd = new Random();
    public Form1()
    {
        InitializeComponent();
```

```
        // Create sample data
        var data = new List<DataItem>();
        var dt = DateTime.Today;
        for (var i = 0; i < npts; i++)
        {
            data.Add(new DataItem()
            {
                Time = dt.AddMonths(i),
                Precipitation = rnd.Next(30, 100),
                Temperature = rnd.Next(7, 20)
            });
        }

        flexChart1.BeginUpdate();
        flexChart1.Series.Clear();

        // 1st series with auxiliary axis
        var series1 = new Series();
        series1.Name = "prec";
        series1.Binding = "Precipitation";
        series1.ChartType = ChartType.Column;
        series1.AxisY = new Axis()
        {
            Position = Position.Right,
            Min = 0,
            Max = 100,
            Title = "precipitation, mm"
        };

        // 2nd series
        var series2 = new Series();
        series2.Name = "t, avg";
        series2.Binding = "Temperature";
        series2.ChartType = ChartType.LineSymbols;

        // Add data to the chart
        flexChart1.Series.Add(series1);
        flexChart1.Series.Add(series2);
        flexChart1.BindingX = "Time";
        flexChart1.DataSource = data;


        flexChart1.ChartType = ChartType.Column;

        // Set axis appearance
        flexChart1.AxisY.Title = "temperature, C";
        flexChart1.AxisY.Min = 0;
        flexChart1.AxisY.MajorGrid = true;
        flexChart1.EndUpdate();
    }
    public class DataItem
    {
        public DateTime Time { get; set; }

        public int Precipitation { get; set; }
        public int Temperature { get; set; }
    }
```

Axes Labels

Axes labels are the values that appear along the axes. By default, axes labels are determined on the basis of the axes data points and the generated intervals.

In FlexChart, you can change the look, format, and alignment of the axes labels using the properties listed below.

Property	Description
Format	Specifies the format string used for the axes labels.
LabelAlignment	Sets the alignment of the axes labels.
LabelAngle	Specifies the rotation angle of the labels.
Labels	Indicates whether the axes labels are visible.
OverlappingLabels	Indicates how to handle overlapping labels.

The following sections discuss how to work with these properties:

- Axes Labels Format
- Axes Labels Rotation
- Axes Labels Visibility
- Axes Labels Overlap

Axes Labels Format

By default, axis labels are determined automatically based on the data points and generated intervals of the axes. However, you can still format the axis labels by using the Format property to cover your requirements better.

The Format property accepts values from the Standard .Net Format string.

Axes Labels Rotation

When the horizontal axis is crowded with axis labels, you need to rotate the labels to avoid the cluttered look. Rotating the labels allows you to accommodate a large number of labels in a limited space on the axis.

You can use the LabelAngle property to rotate axes labels in anticlockwise direction in FlexChart.

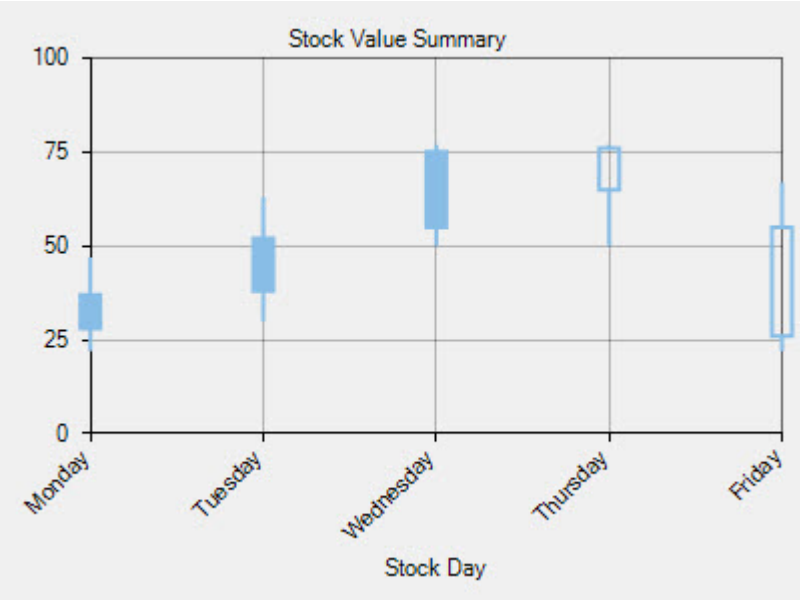
See the code given below for reference.

- **Visual Basic**

```
' set the LabelAngle property
FlexChart1.AxisX.LabelAngle = 45
```

- **C#**

```
// set the LabelAngle property
flexChart1.AxisX.LabelAngle = 45;
```



Axes Labels Visibility

FlexChart enables you to show or hide axis labels with the Labels property. You can set the property to False for a specific axis, if you want to hide axis labels along the axis. The default value of the Labels property is True.

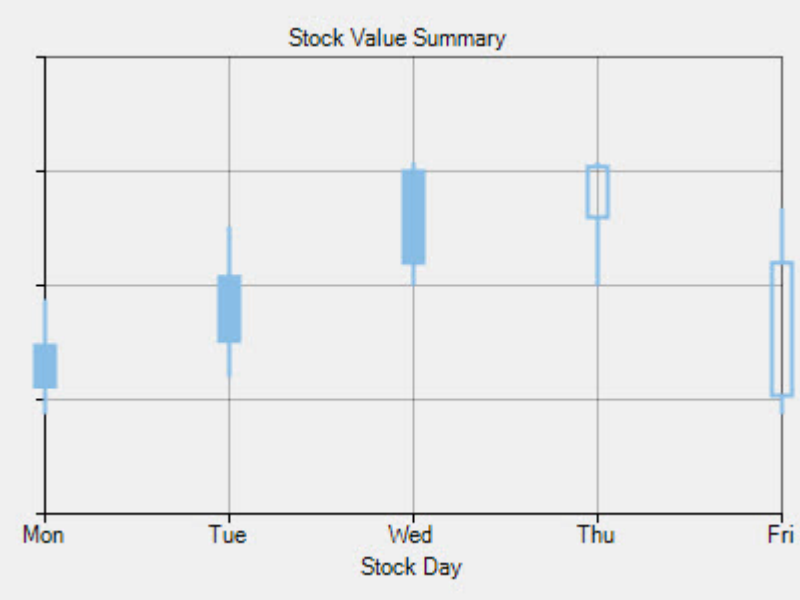
See the following code snippet:

- **Visual Basic**

```
' set the Labels property
FlexChart1.AxisY.Labels = False
```

- **C#**

```
// set the Labels property
flexChart1.AxisY.Labels = false;
```



Axes Labels Overlap

In case there are overlapping labels in the chart for any reason, you can manage the same using the OverlappingLabels property.

The OverlappingLabels property accepts the following values in the OverlappingLabels enumeration:

Property	Description
OverlappingLabels.Auto	Hides overlapping labels.
OverlappingLabels.Show	Shows all labels including the overlapping ones.

Here is the code snippet:

- **Visual Basic**

```
FlexChart1.AxisX.OverlappingLabels = C1.Chart.OverlappingLabels.Auto
```

- **C#**

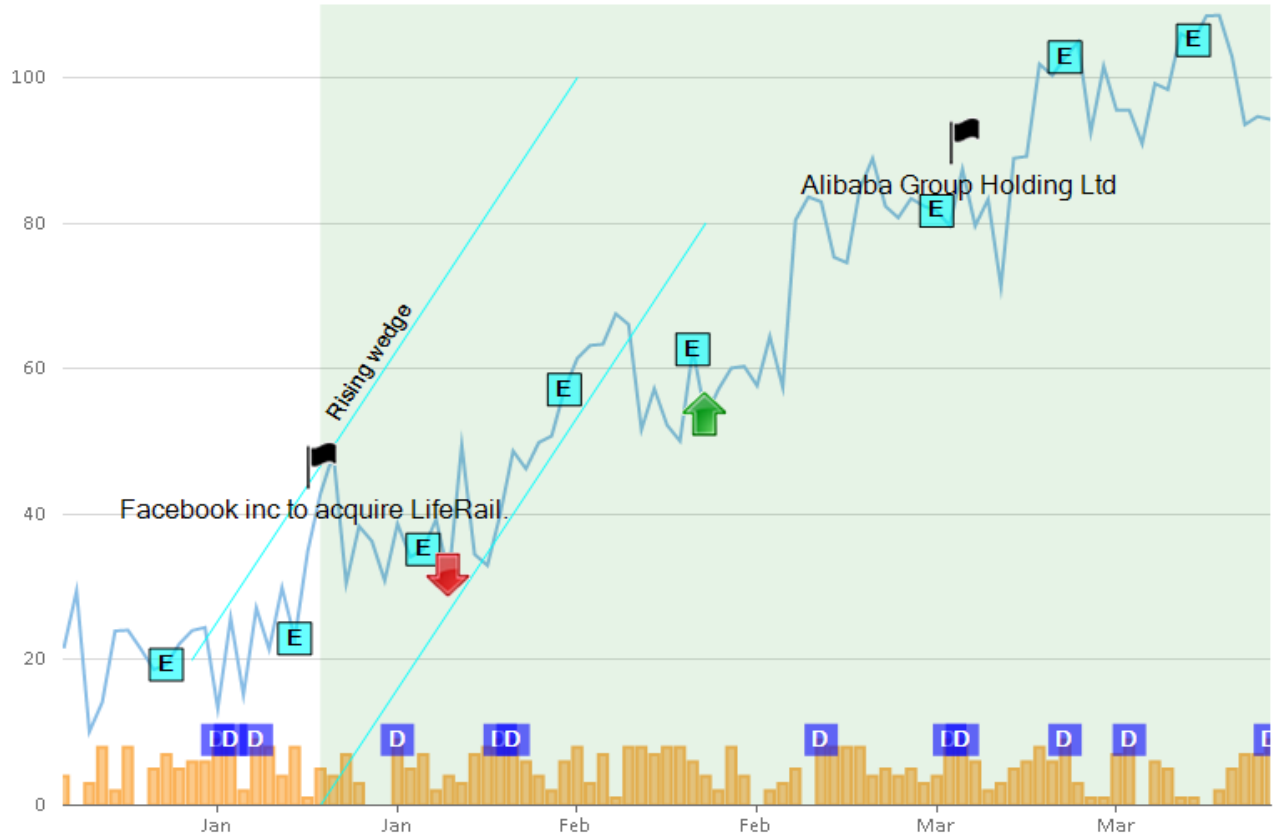
```
flexChart1.AxisX.OverlappingLabels = C1.Chart.OverlappingLabels.Auto;
```

Annotations

Annotations are visual elements used to mark or highlight specific areas in a chart. They include texts, images, and shapes that can be used to display and highlight important information about specific data points. The primary purpose of using annotations in a chart is to communicate the chart data clearly.

FlexChart provides eight types of annotations in three categories: shape, text, and image annotations. Each annotation type allows you to make chart data informative in different ways. Those include displaying information in circle, rectangle, polygon, and other shapes, and highlighting data through explanatory notes or images.

In addition, annotations in FlexChart can be positioned in the chart using attachment modes, such as Absolute, Relative, Data Index, and Data Coordinate. Both annotations and their content are customizable through styling properties of font, color, and stroke. They can be made interactive by adding tooltips, especially image annotations.



To explore annotations, refer to the following sections:

- Adding Annotations
- Positioning Annotations
- Customizing Annotations
- Types of Annotations
- Creating Callouts

Adding Annotations

FlexChart enables you to add annotations in an annotation layer, which contains the collection of all annotations in the chart.

To add annotations in FlexChart, follow these steps:

1. Create an annotation layer in FlexChart.
2. Add the annotation instance in the annotation layer.

To create an annotation layer in FlexChart, create an instance of the AnnotationLayer class and pass FlexChart as the parameter into it. To add an annotation into the annotation layer, create an instance of the respective annotation class based on its type. Add the annotation instance to the Annotations collection of the annotation layer.

The following code snippet illustrates how to create and add the Rectangle annotation to Annotation Layer in FlexChart.

- **Visual Basic**

```
' create an instance of AnnotationLayer and pass FlexChart as the parameter
Dim annotationLayer As New C1.Win.Chart.Annotation.AnnotationLayer(FlexChart1)

' create an instance of the Rectange annotation
Dim rect As New C1.Win.Chart.Annotation.Rectangle("Maximum Tax Revenue" & vbLf & "2013" & vbLf & "45000")

' add the annotation to the Annotations collection of the annotation layer
annotationLayer.Annotations.Add(rect)
```

- **C#**

```
// create an instance of AnnotationLayer and pass FlexChart as the parameter
C1.Win.Chart.Annotation.AnnotationLayer annotationLayer = new C1.Win.Chart.Annotation.AnnotationLayer(flexChart1);
```

```
// create an instance of the Rectange annotation
C1.Win.Chart.Annotation.Rectangle rect = new C1.Win.Chart.Annotation.Rectangle("Maximum Tax Revenue\n2013\n45000");

// add the annotation to the Annotations collection of the annotation layer
annotationLayer.Annotations.Add(rect);
```

Positioning Annotations

In FlexChart, positioning annotations includes two mechanisms (not necessarily in the same order), as follows:

- Positioning annotations relative to the chart.
- Positioning annotations relative to the data points.

Positioning Annotations Relative to the Chart

Positioning annotations relative to the chart includes specifying the attachment and the location of the annotations in the chart.

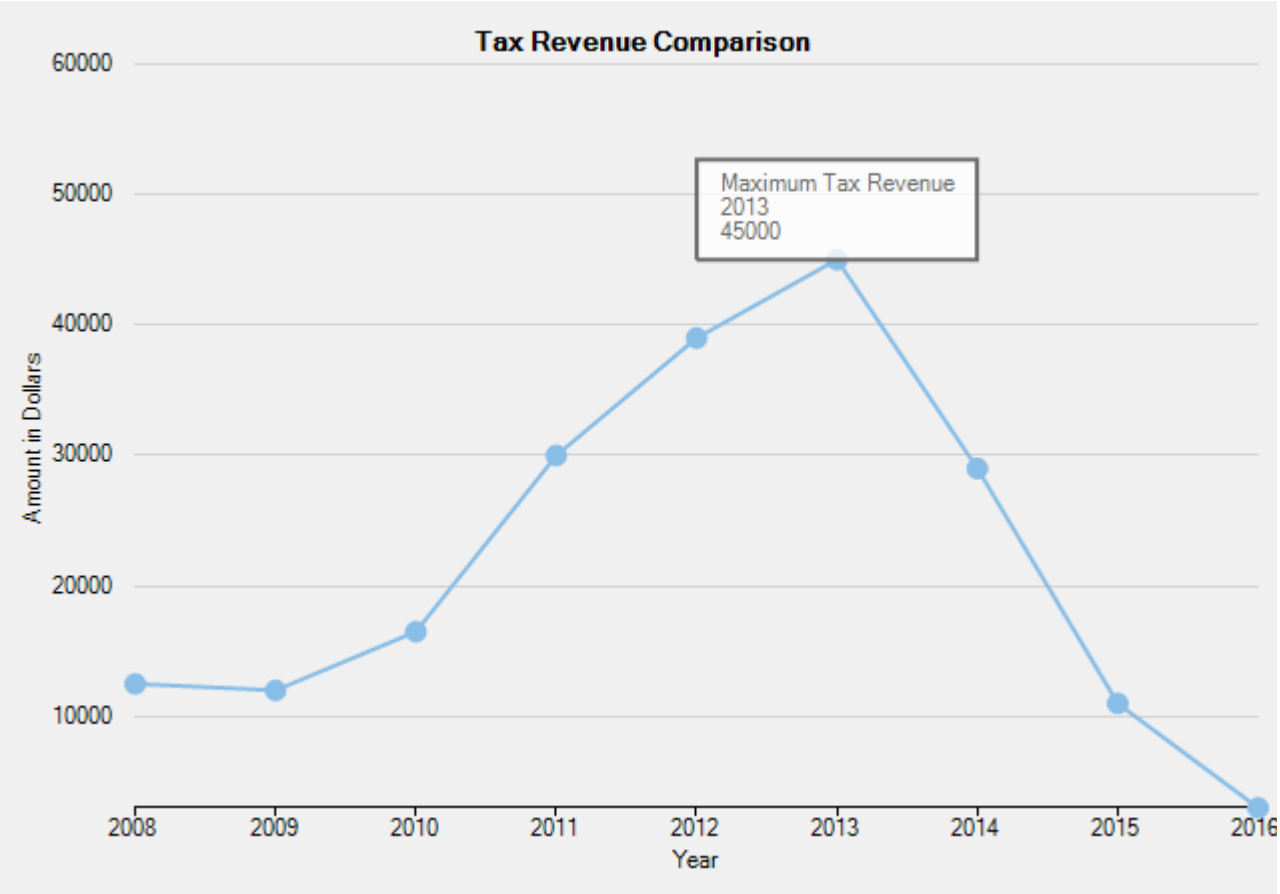
FlexChart provides four ways of attaching annotations, as follows:

- **Absolute**: This attachment indicates that the annotation is fixed and cannot move, irrespective of the resizing of the application. To set the absolute attachment, set the Attachment property to **Absolute** from the AnnotationAttachment enum. To set the location of the annotation in the absolute attachment mode, set the annotation's coordinates in pixels.
- **DataCoordinate**: This attachment indicates that the annotation is attached to a specific data point. To set this attachment, set the Attachment property to **DataCoordinate** from the AnnotationAttachment enum. To set the annotation's location, specify the annotation's data coordinates by setting the Location property.
- **DataIndex**: This attachment indicates that the annotation is attached to the series as per the series index and to the point as per the point index. To set this attachment, set the Attachment property to **DataIndex** from the AnnotationAttachment enum. To specify the annotation's location, set the SeriesIndex and the PointIndex properties.
- **Relative**: This attachment indicates that the annotation retains its location and dimensions relative to the chart. To set this attachment, set the Attachment property to **Relative** from the AnnotationAttachment enum. Specify the annotation's location using the Location property in terms of relative position inside the chart where (0, 0) is the top left corner and (1, 1) is the bottom right corner.

Positioning Annotations Relative to the Data Points

Specify the position of annotations with respect to the data points by setting the Position property from the AnnotationPosition enum.

The following image displays the Rectangle annotation highlighting the maximum tax revenue in the year, 2013.



The following code compares tax revenue data of nine consecutive years to display the maximum tax revenue. The code shows how to specify the attachment, location, and position of the Rectangle annotation to Annotation Layer in FlexChart.

- **Visual Basic**

```
' specify the attachment of the annotation
rect.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataIndex

' set the location of the annotation
rect.SeriesIndex = 0
rect.PointIndex = 5

' set the annotation position
rect.Position = C1.Chart.Annotation.AnnotationPosition.Top

' specify the annotation dimensions
rect.Height = 50
rect.Width = 140

' set the annotation style
rect.Style.FillColor = Color.FromArgb(200, Color.Transparent)
```

- **C#**

```
// specify the attachment of the annotation
rect.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataIndex;

// set the location of the annotation
rect.SeriesIndex = 0;
rect.PointIndex = 5;
```

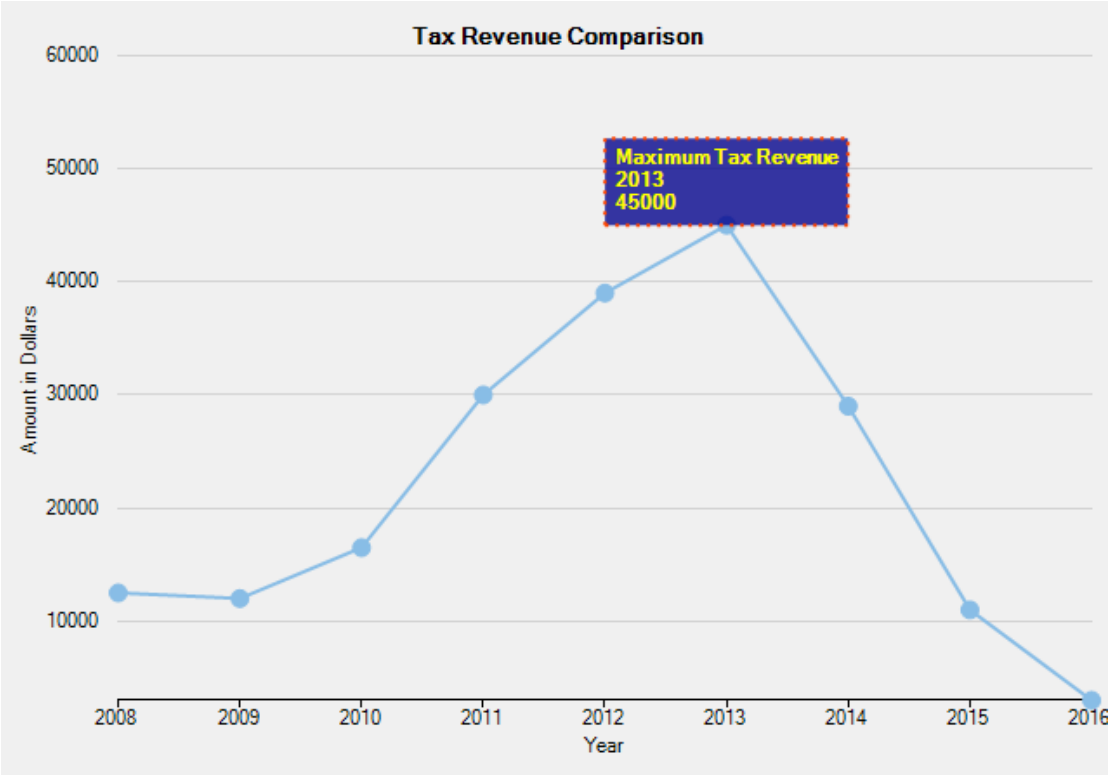
```
// set the annotation position
rect.Position = C1.Chart.Annotation.AnnotationPosition.Top;

// specify the annotation dimensions
rect.Height = 50;
rect.Width = 140;

// set the annotation style
rect.Style.FillColor = Color.FromArgb(200, Color.Transparent);
```

Customizing Annotations

FlexChart annotations are customizable in terms of dimensions (for shapes), scaling (for images), and content style (for all except images).

- **Dimensions**: Change the dimensions of all shapes using dimension properties of the respective classes. For instance, to change the dimensions of the Rectangle annotation, set the Height and the Width properties of the Rectangle class.
- **Style**: Customize the appearance of shape and text annotations in terms of color, font, and stroke by using the Style property of the AnnotationBase class.
- **Content Style**: Customize the appearance of content present in shape annotations by using the ContentStyle property of the AnnotationBase class.

The following image displays the Rectangle annotation customized to further highlight the maximum tax revenue in the year, 2013.



The following code compares tax revenue data of nine consecutive years to display the maximum tax revenue. The code shows how to set the Rectangle annotation's dimensions, customize its appearance and content.

- **Visual Basic**

```
' specify the annotation dimensions
rect.Height = 50
rect.Width = 140

' customize the annotation style and content style
rect.Style.FillColor = Color.FromArgb(200, Color.DarkBlue)
rect.Style.StrokeColor = Color.OrangeRed
```

```
rect.Style.StrokeWidth = 2
rect.Style.StrokeDashPattern = New Single() {1.0F, 2.0F}
rect.ContentStyle.StrokeColor = Color.Yellow
rect.ContentStyle.Font = New System.Drawing.Font(FontFamily.GenericSansSerif, 8.5F, FontStyle.Bold)
```

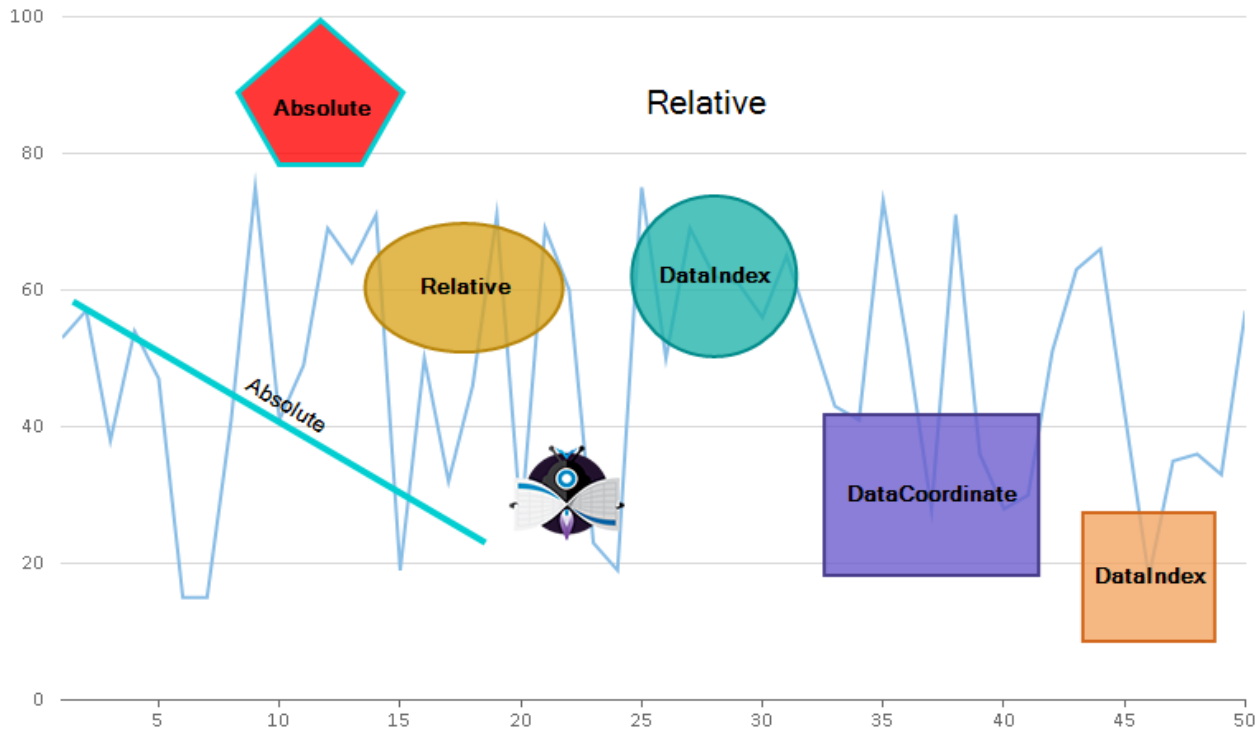
- **C#**

```
// specify the annotation dimensions
rect.Height = 50;
rect.Width = 140;

// customize the annotation style and content style
rect.Style.FillColor = Color.FromArgb(200, Color.DarkBlue);
rect.Style.StrokeColor = Color.OrangeRed;
rect.Style.StrokeWidth = 2;
rect.Style.StrokeDashPattern = new[] { 1f, 2f };
rect.ContentStyle.StrokeColor = Color.Yellow;
rect.ContentStyle.Font = new System.Drawing.Font(FontFamily.GenericSansSerif, 8.5F, FontStyle.Bold);
```

Types of Annotations

FlexChart provides eight types of annotations in three categories, as follows:

- **Shapes**: Include useful information at specific areas and highlight the areas within chart data by using shapes, such as Circle, Ellipsis, Rectangle, Square, Line, or Polygon.
- **Text**: Add descriptive notes or informative comments at specific points in the chart by using text annotations.
- **Image**: Add self-explanatory images to readily communicate chart data by using image annotations.



To explore different types of annotations provided by FlexChart, refer to the following sections:

- Shape Annotations
- Text Annotations
- Image Annotations

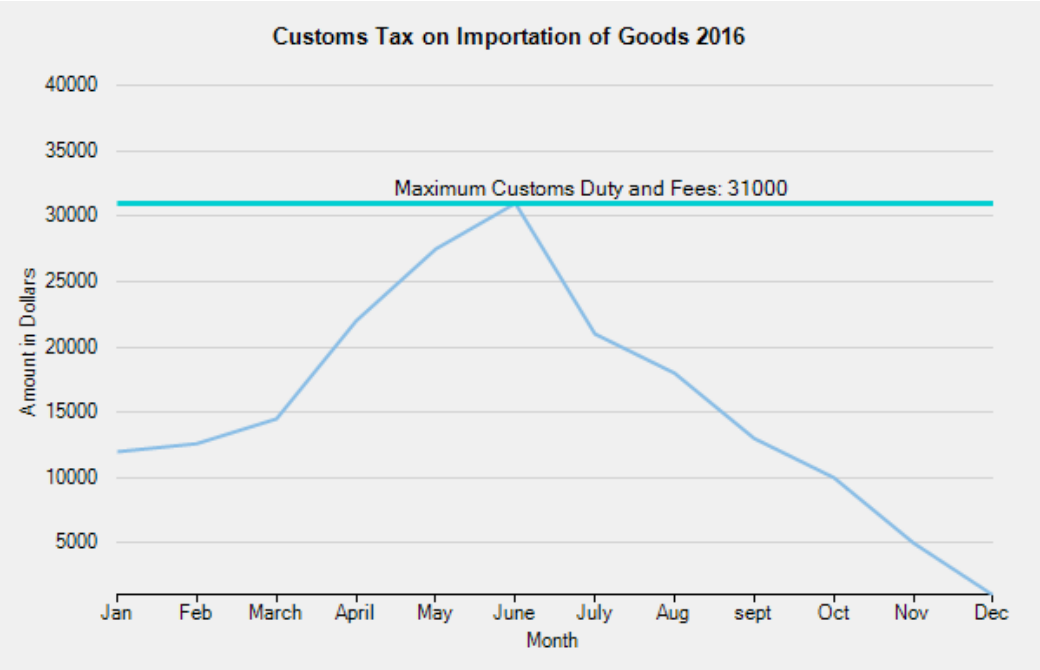
Shape Annotations

Shapes are beneficial for drawing the user's attention at specific areas where important data is highlighted.

FlexChart offers six shape annotations, as follows:

- Circle
- Ellipsis
- Line
- Polygon
- Rectangle
- Square

The following image shows the Line annotation highlighting the maximum customs duty and fees in the year, 2016.



To create any of these specific shapes, create an instance of the shape annotation's class. Set the dimensions of the shape by using the dimension properties from the corresponding class. For example, to create a line annotation, create an instance of the Line class. Specify the length of the line annotation or rotate it by setting the Start and the End properties of the Line class.

For any shape annotation, specify the text by setting the Content property of the Shape class, the base class for all shape annotations. In addition, other shapes like triangles and arrows can be created using the Polygon annotation in FlexChart.

The following code uses customs tax data on importation of goods for representing its increment or decrement for the year, 2016. The code shows how to add, position, and customize the Line annotation in FlexChart.

- **Visual Basic**

```
' create an instance of the Line annotation
Dim line As New C1.Win.Chart.Annotation.Line()

' specify the line content
line.Content = "Maximum Customs Duty and Fees: 31000"

' specify the start and end points of the line
line.Start = New PointF(0, 31000)
line.[End] = New PointF(12, 31000)

' specify the line attachment
line.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate

' set the line position
line.Position = C1.Chart.Annotation.AnnotationPosition.Top

' customize the line style and content style
```

```
line.Style.StrokeColor = Color.DarkTurquoise
line.Style.StrokeWidth = 3
line.ContentStyle.StrokeColor = Color.Black
line.ContentStyle.Font = New System.Drawing.Font(FontFamily.GenericSansSerif, 9, FontStyle.Regular)

' add the line to the Annotations collection of the annotation layer
annotationLayer.Annotations.Add(line)
```

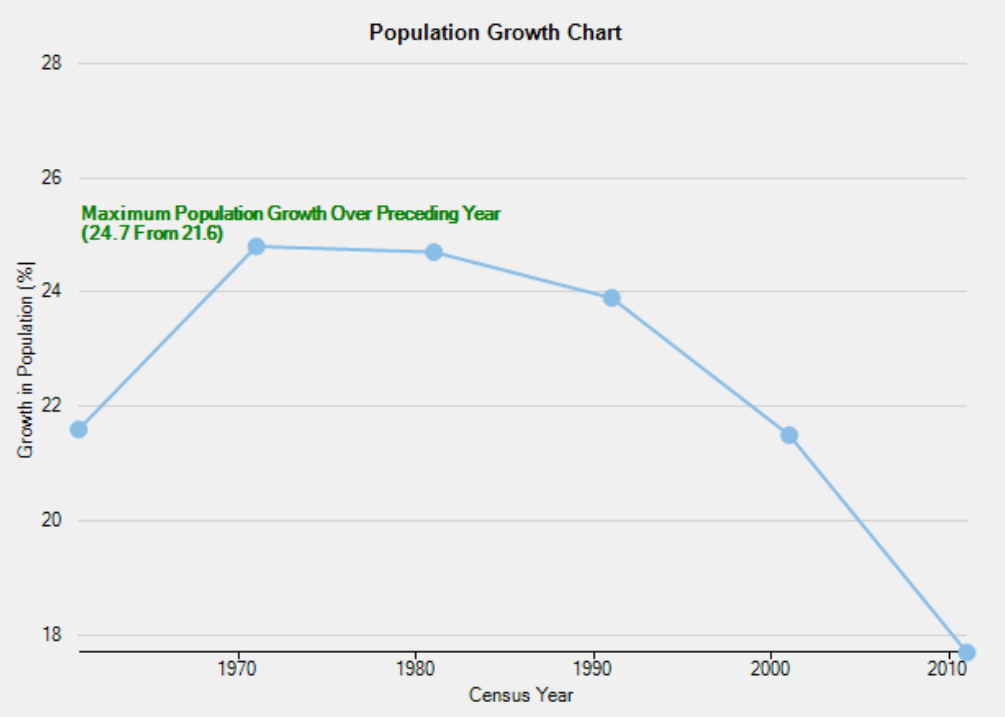
- **C#**

```
// create an instance of the Line annotation
C1.Win.Chart.Annotation.Line line = new C1.Win.Chart.Annotation.Line();

// specify the line content
line.Content = "Maximum Customs Duty and Fees: 31000";

// specify the start and end points of the line
line.Start = new PointF(0,31000);
line.End = new PointF(12,31000);

// specify the line attachment
line.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate;

// set the line position
line.Position = C1.Chart.Annotation.AnnotationPosition.Top;

// customize the line style and content style
line.Style.StrokeColor = Color.DarkTurquoise;
line.Style.StrokeWidth = 3;
line.ContentStyle.StrokeColor = Color.Black;
line.ContentStyle.Font = new System.Drawing.Font(FontFamily.GenericSansSerif, 9, FontStyle.Regular);

// add the line to the Annotations collection of the annotation layer
annotationLayer.Annotations.Add(line);
```

Text Annotation

Text annotations let you add additional information at specific data points to make the data informative. FlexChart lets you add single line as well as multiline text in text annotations.

The following image shows the Text annotation displaying the maximum population growth rate between 1961 and 2011.



To work with text annotation in FlexChart, create an instance of the Text class and set the Content property for the instance.

The following code compares population growth rates at specific years in five consecutive decades. The code shows how to add, position, and customize the Text annotation in FlexChart.

- **Visual Basic**

```
' create an instance of the Line annotation
Dim text As New C1.Win.Chart.Annotation.Text()

' specify the line content
text.Content = "Maximum Population Growth Over Preceding Year" + Environment.NewLine + "(24.7 From 21.6)"

' specify the start and end points of the line
text.Location = New PointF(1961.0F, 25.15F)

' specify the line attachment
text.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate

' set the line position
text.Position = C1.Chart.Annotation.AnnotationPosition.Top

' customize the line style and content style
text.Style.StrokeColor = Color.Green
text.Style.Font = New System.Drawing.Font(FontFamily.GenericSansSerif, 8, FontStyle.Bold)

' add the line to the Annotations collection of the annotation layer
annotationLayer.Annotations.Add(text)
```

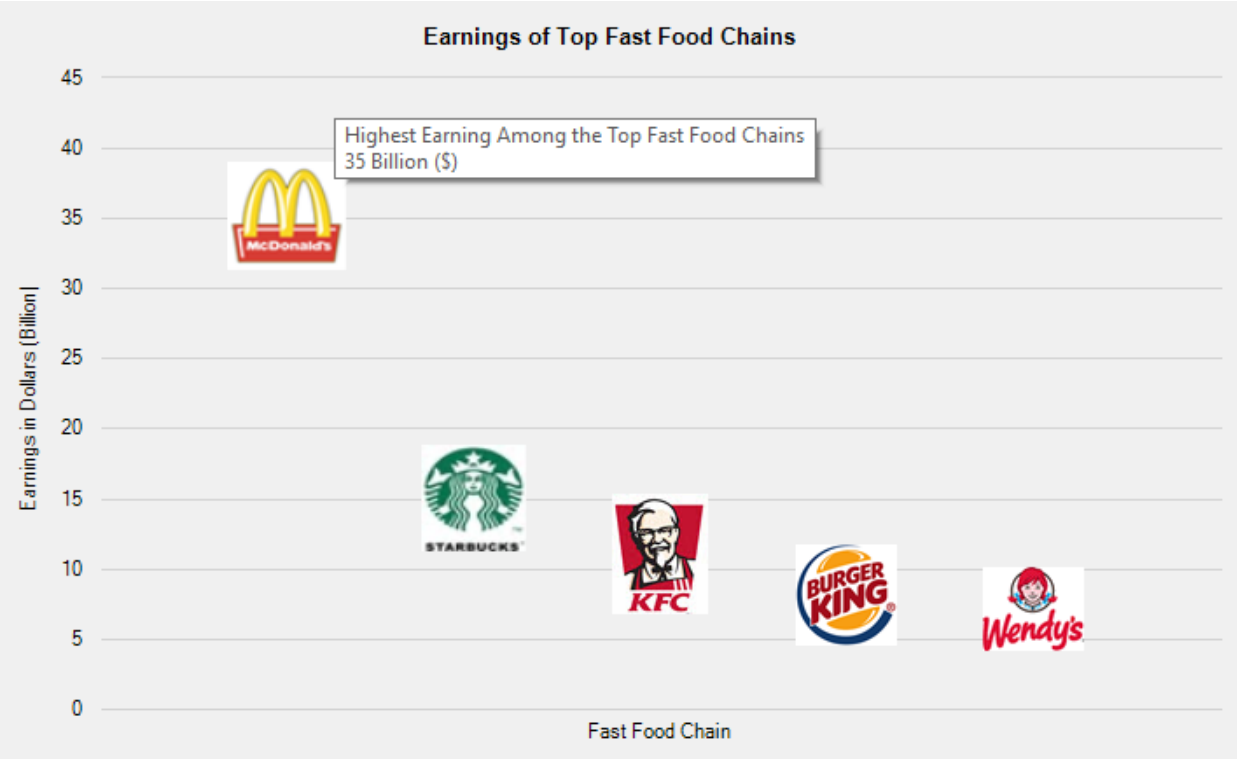
- **C#**

```
// create an instance of the Line annotation
C1.Win.Chart.Annotation.Text text = new C1.Win.Chart.Annotation.Text();

// specify the line content
text.Content = "Maximum Population Growth Over Preceding Year" + Environment.NewLine + "(24.7 From 21.6)";

// specify the start and end points of the line
text.Location = new PointF(1961F,25.15F);

// specify the line attachment
text.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate;

// set the line position
text.Position = C1.Chart.Annotation.AnnotationPosition.Top;

// customize the line style and content style
text.Style.StrokeColor = Color.Green;
text.Style.Font = new System.Drawing.Font(FontFamily.GenericSansSerif, 8, FontStyle.Bold);

// add the line to the Annotations collection of the annotation layer
annotationLayer.Annotations.Add(text);
```

Image Annotation

Image annotations create visual impact and allow users to quickly interpret the chart data. A great way to communicate informative data through image annotations is by adding tooltips.

The following image displays highest earning of a fast food chain among others using a tooltip with an image annotation.



To use image annotations in FlexChart, create an instance of the Image class and set an image for the instance by specifying the image path in the SourceImage property. Scale the image or adjust its size by setting the Height and the Width properties. To add tooltips to image annotations, set the TooltipText property of the AnnotationBase class for the image annotation instance.

The following code compares earnings of the top fast food chains in United States. The code shows how to add, position, and customize the Image annotation in FlexChart.

- **Visual Basic**

```
' create instances of the Image annotation
Dim image1 As New C1.Win.Chart.Annotation.Image()
Dim image2 As New C1.Win.Chart.Annotation.Image()
Dim image3 As New C1.Win.Chart.Annotation.Image()
Dim image4 As New C1.Win.Chart.Annotation.Image()
Dim image5 As New C1.Win.Chart.Annotation.Image()

' set the source image for the annotations
image1.SourceImage = Image.FromFile("C:\Resources\image1.png")
image2.SourceImage = Image.FromFile("C:\Resources\image2.png")
image3.SourceImage = Image.FromFile("C:\Resources\image3.png")
image4.SourceImage = Image.FromFile("C:\Resources\image4.png")
image5.SourceImage = Image.FromFile("C:\Resources\image5.png")

' specify the attachment of the images
image1.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate
image2.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate
image3.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate
image4.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate
image5.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate

' specify the location of the images
image1.Location = New PointF(1, 35)
image2.Location = New PointF(2, 15)
image3.Location = New PointF(3, 11)
image4.Location = New PointF(4, 8)
image5.Location = New PointF(5, 7)

' specify the position of the images
image1.Position = C1.Chart.Annotation.AnnotationPosition.Center
```

```
image2.Position = C1.Chart.Annotation.AnnotationPosition.Center
image3.Position = C1.Chart.Annotation.AnnotationPosition.Center
image4.Position = C1.Chart.Annotation.AnnotationPosition.Center
image5.Position = C1.Chart.Annotation.AnnotationPosition.Center

' scale the images
image1.Width = 68
image1.Height = 62
image2.Width = 60
image2.Height = 61

' add a tooltip
image1.TooltipText = "Highest Earning Among the Top Fast Food Chains" & vbLf & "35 Billion ($)"

' add the images to the Annotations collection of the annotation layer
annotationLayer.Annotations.Add(image1)
annotationLayer.Annotations.Add(image2)
annotationLayer.Annotations.Add(image3)
annotationLayer.Annotations.Add(image4)
annotationLayer.Annotations.Add(image5)
```

- **C#**

```
// create instances of the Image annotation
C1.Win.Chart.Annotation.Image image1 = new C1.Win.Chart.Annotation.Image();
C1.Win.Chart.Annotation.Image image2 = new C1.Win.Chart.Annotation.Image();
C1.Win.Chart.Annotation.Image image3 = new C1.Win.Chart.Annotation.Image();
C1.Win.Chart.Annotation.Image image4 = new C1.Win.Chart.Annotation.Image();
C1.Win.Chart.Annotation.Image image5 = new C1.Win.Chart.Annotation.Image();

// set the source image for the annotations
image1.SourceImage = Image.FromFile("C:\\Resources\\image1.png");
image2.SourceImage = Image.FromFile("C:\\Resources\\image2.png");
image3.SourceImage = Image.FromFile("C:\\Resources\\image3.png");
image4.SourceImage = Image.FromFile("C:\\Resources\\image4.png");
image5.SourceImage = Image.FromFile("C:\\Resources\\image5.png");

// specify the attachment of the images
image1.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate;
image2.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate;
image3.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate;
image4.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate;
image5.Attachment = C1.Chart.Annotation.AnnotationAttachment.DataCoordinate;

// specify the location of the images
image1.Location = new PointF(1, 35);
image2.Location = new PointF(2, 15);
image3.Location = new PointF(3, 11);
image4.Location = new PointF(4, 8);
image5.Location = new PointF(5, 7);

// specify the position of the images
image1.Position = C1.Chart.Annotation.AnnotationPosition.Center;
image2.Position = C1.Chart.Annotation.AnnotationPosition.Center;
image3.Position = C1.Chart.Annotation.AnnotationPosition.Center;
image4.Position = C1.Chart.Annotation.AnnotationPosition.Center;
image5.Position = C1.Chart.Annotation.AnnotationPosition.Center;

// scale the images
image1.Width = 68;
image1.Height = 62;
image2.Width = 60;
image2.Height = 61;

// add a tooltip
image1.TooltipText = "Highest Earning Among the Top Fast Food Chains\n35 Billion ($)";

// add the images to the Annotations collection of the annotation layer
```

```
annotationLayer.Annotations.Add(image1);
annotationLayer.Annotations.Add(image2);
annotationLayer.Annotations.Add(image3);
annotationLayer.Annotations.Add(image4);
annotationLayer.Annotations.Add(image5);
```

Creating Callouts

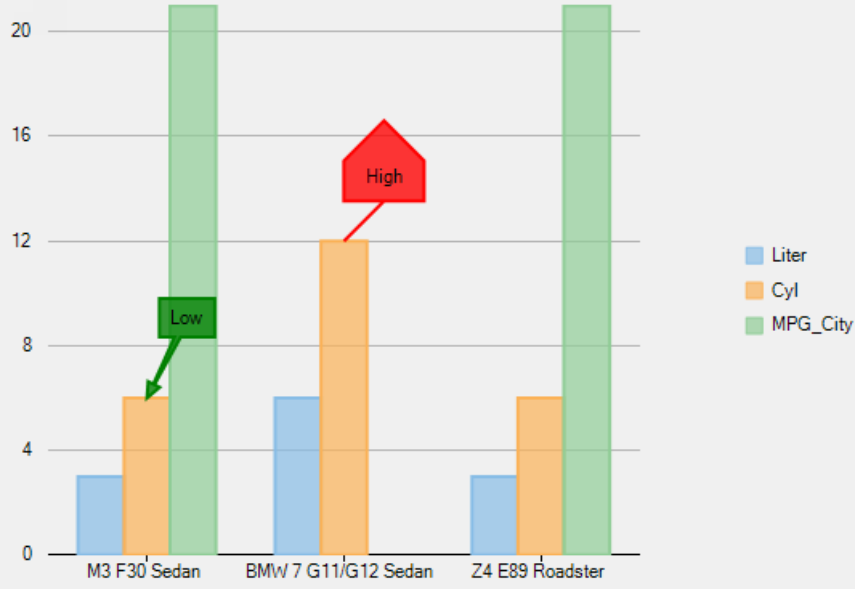
Callouts in charts are used to display the details of a data series or individual data points in an easy-to-read format. Callouts being connected with data points, help better visualize and comprehend chart data by minimizing visual disturbances in the chart area. In FlexChart, Polygon type annotations can be customized to create chart callouts with line or arrow connectors.

In this example, we are using sample created in the Quick Start topic to further create an arrow callout and polygon annotation with line connection. This is done with the help of the Points property and the ContentCenter property that define the coordinates of polygon vertices and annotation content center respectively.

To create callouts connected with respective data points, follow these steps:

- **Step 1: Create annotation with line connector**
- **Step 2: Create arrow annotation callout**
- **Step 3: Render the annotations in chart**

The following image illustrates polygon annotations connected to data points through arrow and line connectors.



Step 1: Create annotation with line connector

To create a line callout, use the following code.

- **Visual Basic**

```
    ...
    ' Create a line callout annotation of polygon type
    Dim lineCallout = New Polygon("High") With {
.Attachment = AnnotationAttachment.DataIndex,
.SeriesIndex = 1,
.PointIndex = 1,
.ContentCenter = New Point(25, -40)
}
    ' Create a list of points for the line callout annotation
    lineCallout.Points.Add(New PointF(0, 0))
    lineCallout.Points.Add(New PointF(25, -25))
    lineCallout.Points.Add(New PointF(50, -25))
    lineCallout.Points.Add(New PointF(50, -50))
    lineCallout.Points.Add(New PointF(25, -75))
    lineCallout.Points.Add(New PointF(0, -50))
    lineCallout.Points.Add(New PointF(0, -25))
    lineCallout.Points.Add(New PointF(25, -25))
    lineCallout.Points.Add(New PointF(0, 0))
```

```
    ' Stylise the line callout annotation of polygon type
    lineCallout.ContentStyle.StrokeColor = Color.Black
    lineCallout.Style.FillColor = Color.FromArgb(200, Color.Red)
    lineCallout.Style.StrokeColor = Color.Red
    ...
```

- **C#**

```
...
// Create a line callout annotation of polygon type
var lineCallout = new Polygon("High")
{
    Attachment = AnnotationAttachment.DataIndex,
    SeriesIndex = 1,
    PointIndex = 1,
    ContentCenter = new Point(25, -40),
    // Create a list of points for the line callout annotation
    Points = { new Point(0, 0), new Point(25, -25),
        new Point(50, -25), new Point(50, -50), new Point(25, -75),
        new Point(0, -50), new Point(0, -25), new Point(25, -25), new Point(0, 0) }
};
//Stylise the line callout annotation of polygon type
lineCallout.ContentStyle.StrokeColor = Color.Black;
lineCallout.Style.FillColor = Color.FromArgb(200, Color.Red);
lineCallout.Style.StrokeColor = Color.Red;
...
```

Back to Top

Step 2: Create arrow annotation callout

1. To create an arrow callout use the following code.
    - **Visual Basic**

```
 Private Sub SetUpAnnotations()
     'Create an annotation of Polygon type with arrow connector
     Dim arrowCallout = New Polygon("Low") With {
 .Attachment = AnnotationAttachment.DataIndex,
 .SeriesIndex = 1,
 .PointIndex = 0,
 .ContentCenter = New PointF(25, -50)
}
     'Create a list of points for the annotation with arrow connector
     Dim points As List(Of PointF) = GetPointsForArrowCallout(arrowCallout.ContentCenter.Value.X,
                                                              arrowCallout.ContentCenter.Value.Y,
                                                              "Low")
     For Each p As PointF In points
         arrowCallout.Points.Add(p)
     Next
     'Stylise the annotation with arrow connector
     arrowCallout.ContentStyle.StrokeColor = Color.Black
     arrowCallout.Style.FillColor = Color.FromArgb(200, Color.Green)
     arrowCallout.Style.StrokeColor = Color.Green
     ...
```

    - **C#**

```
private void SetUpAnnotations()
{
    // Create an arrow callout annotation of polygon type
    var arrowCallout = new Polygon("Low")
    {
        // Specified the Annotation coordinates by  Data Series Index and Data Point Index
        Attachment = AnnotationAttachment.DataIndex,
        SeriesIndex = 1,
        PointIndex = 0,
        ContentCenter = new PointF(25, -50)
    };

    // Create a list of points for arrow callout by calling GetPointsForArrowCallout()
    List<PointF> points = GetPointsForArrowCallout(arrowCallout.ContentCenter.Value.X,
        arrowCallout.ContentCenter.Value.Y, "Low");
    foreach (PointF p in points)
    {
        arrowCallout.Points.Add(p);
    }

    //Stylise the arrow callout annotation
    arrowCallout.ContentStyle.StrokeColor = Color.Black;
    arrowCallout.Style.FillColor = Color.FromArgb(200, Color.Green);
```

```
arrowCallout.Style.StrokeColor = Color.Green;
...
```

2. Define the GetPointsForArrowCallout() method to specify the points for arrow callout.
    1. To measure the size of content string in arrow callout, and reuse it to calculate and set the dimensions of arrow annotation, use the following code.
        - **Visual Basic**

```
Private Function GetPointsForArrowCallout(centerX As Single,
                                          centerY As Single,
                                          content As String)
    As List(Of PointF)
    ' Measure the size of the content string in arrow callout
    Dim size As _Size = _engine.MeasureString(content)

    ' Call method to calculate dimensions of the arrow annotation
    Return GetPointsForArrowCallout(centerX, centerY,
                                    CSng(size.Width) + 10, CSng(size.Height) + 10)
End Function
```

        - **C#**

```
private List<PointF> GetPointsForArrowCallout
    (float centerX, float centerY, string content)
{
    // Measure the size of the content string in arrow callout
    _Size size = _engine.MeasureString(content);

    // Call method to calculate dimensions of the arrow annotation
    return GetPointsForArrowCallout(centerX, centerY,
        (float)size.Width + 10, (float)size.Height + 10);
}
```

    2. To calculate the dimensions and points for arrow annotations, define the method overload GetPointsForArrowCallout() as shown below.
        - **Visual Basic**

```
Private Function GetPointsForArrowCallout(centerX As Single,
                                          centerY As Single,
                                          rectWidth As Single,
                                          rectHeight As Single)
    As List(Of PointF)
    Dim points = New List(Of PointF)()

    Dim rectLeft As Single = centerX - rectWidth / 2
    Dim rectRight As Single = centerX + rectWidth / 2
    Dim rectTop As Single = centerY - rectHeight / 2
    Dim rectBottom As Single = centerY + rectHeight / 2

    Dim angle As Single = CSng(Math.Atan2(-centerY, centerX))
    Dim angleOffset1 As Single = 0.4F
    Dim angleOffset2 As Single = 0.04F
    Dim arrowHeight As Single = 0.4F * rectHeight
    Dim hypotenuse As Single = CSng(arrowHeight / Math.Cos(angleOffset1))
    Dim subHypotenuse As Single = CSng(arrowHeight / Math.Cos(angleOffset2))

    Dim isNearBottom As Boolean = Math.Abs(rectTop) > Math.Abs(rectBottom)
    Dim nearHorizontalEdge As Single = If(isNearBottom, rectBottom, rectTop)
    Dim isNearRight As Boolean = Math.Abs(rectLeft) > Math.Abs(rectRight)
    Dim nearVerticalEdge As Single = If(isNearRight, rectRight, rectLeft)
    Dim isHorizontalCrossed As Boolean = Math.Abs(nearHorizontalEdge) > Math.Abs(nearVerticalEdge)
    Dim nearEdge As Single = If(isHorizontalCrossed, nearHorizontalEdge, nearVerticalEdge)

    Dim factor As Integer = If(nearEdge > 0, -1, 1)
    Dim crossedPointOffsetToCenter As Single = If(isHorizontalCrossed,
        CSng(rectHeight / (2 * Math.Tan(angle)) * factor),
        CSng(rectWidth * Math.Tan(angle) * factor / 2))

    ' Specify Arrow points
    points.Add(New PointF(0, 0))
    points.Add(New PointF(CSng(Math.Cos(angle + angleOffset1) * hypotenuse),
                          CSng(-Math.Sin(angle + angleOffset1) * hypotenuse)))
    points.Add(New PointF(CSng(Math.Cos(angle + angleOffset2) * subHypotenuse),
                          CSng(-Math.Sin(angle + angleOffset2) * subHypotenuse)))

    ' Specify Rectangle points
    If isHorizontalCrossed Then
        points.Add(New PointF(CSng(-nearEdge / Math.Tan(angle + angleOffset2)), CSng(nearEdge)))
        If isNearBottom Then
            points.Add(New PointF(rectLeft, rectBottom))
            points.Add(New PointF(rectLeft, rectTop))
            points.Add(New PointF(rectRight, rectTop))
```

```
            points.Add(New PointF(rectRight, rectBottom))
        Else
            points.Add(New PointF(rectRight, rectTop))
            points.Add(New PointF(rectRight, rectBottom))
            points.Add(New PointF(rectLeft, rectBottom))
            points.Add(New PointF(rectLeft, rectTop))
        End If
        points.Add(New PointF(CSng(-nearEdge / Math.Tan(angle - angleOffset2)), nearEdge))
    Else
        points.Add(New PointF(nearEdge, CSng(-nearEdge * Math.Tan(angle + angleOffset2))))
        If isNearRight Then
            points.Add(New PointF(rectRight, rectBottom))
            points.Add(New PointF(rectLeft, rectBottom))
            points.Add(New PointF(rectLeft, rectTop))
            points.Add(New PointF(rectRight, rectTop))
        Else
            points.Add(New PointF(rectLeft, rectTop))
            points.Add(New PointF(rectRight, rectTop))
            points.Add(New PointF(rectRight, rectBottom))
            points.Add(New PointF(rectLeft, rectBottom))
        End If
        points.Add(New PointF(nearEdge, CSng(-nearEdge * Math.Tan(angle - angleOffset2))))
    End If

    ' Arrow points
    points.Add(New PointF(CSng(Math.Cos(angle - angleOffset2) * subHypotenuse),
                          CSng(-Math.Sin(angle - angleOffset2) * subHypotenuse)))
    points.Add(New PointF(CSng(Math.Cos(angle - angleOffset1) * hypotenuse),
                          CSng(-Math.Sin(angle - angleOffset1) * hypotenuse)))
    Return points
End Function

Private Sub FlexChart1_Click(sender As Object, e As EventArgs) Handles FlexChart1.Click

End Sub
```

- **C#**

```
private List<PointF> GetPointsForArrowCallout
    (float centerX, float centerY, float rectWidth, float rectHeight)
{
    var points = new List<PointF>();

    float rectLeft = centerX - rectWidth / 2;
    float rectRight = centerX + rectWidth / 2;
    float rectTop = centerY - rectHeight / 2;
    float rectBottom = centerY + rectHeight / 2;

    float angle = (float)(Math.Atan2(-centerY, centerX));
    float angleOffset1 = 0.4f;
    float angleOffset2 = 0.04f;
    float arrowHeight = 0.4f * rectHeight;
    float hypotenuse = (float)(arrowHeight / Math.Cos(angleOffset1));
    float subHypotenuse = (float)(arrowHeight / Math.Cos(angleOffset2));

    bool isNearBottom = Math.Abs(rectTop) > Math.Abs(rectBottom);
    float nearHorizontalEdge = isNearBottom ? rectBottom : rectTop;
    bool isNearRight = Math.Abs(rectLeft) > Math.Abs(rectRight);
    float nearVerticalEdge = isNearRight ? rectRight : rectLeft;
    bool isHorizontalCrossed = Math.Abs(nearHorizontalEdge) > Math.Abs(nearVerticalEdge);
    float nearEdge = isHorizontalCrossed ? nearHorizontalEdge : nearVerticalEdge;

    int factor = nearEdge > 0 ? -1 : 1;
    float crossedPointOffsetToCenter = isHorizontalCrossed ?
        (float)(rectHeight / (2 * Math.Tan(angle)) * factor)
        : (float)(rectWidth * Math.Tan(angle) * factor / 2);

    // Specify Arrow points
    points.Add(new PointF(0, 0));
    points.Add(new PointF((float)(Math.Cos(angle + angleOffset1)
        * hypotenuse), (float)(-Math.Sin(angle + angleOffset1) * hypotenuse)));
    points.Add(new PointF((float)(Math.Cos(angle + angleOffset2)
        * subHypotenuse), (float)(-Math.Sin(angle + angleOffset2) * subHypotenuse)));

    // Specify Rectangle points
    if (isHorizontalCrossed)
    {
        points.Add(new PointF((float)(-nearEdge / Math.Tan(angle + angleOffset2)),
            (float)nearEdge));
```

```
            if (isNearBottom)
            {
                points.Add(new PointF(rectLeft, rectBottom));
                points.Add(new PointF(rectLeft, rectTop));
                points.Add(new PointF(rectRight, rectTop));
                points.Add(new PointF(rectRight, rectBottom));
            }
            else
            {
                points.Add(new PointF(rectRight, rectTop));
                points.Add(new PointF(rectRight, rectBottom));
                points.Add(new PointF(rectLeft, rectBottom));
                points.Add(new PointF(rectLeft, rectTop));
            }
            points.Add(new PointF((float)(-nearEdge / Math.Tan(angle - angleOffset2)),
                nearEdge));
        }
        else
        {
            points.Add(new PointF(nearEdge, (float)(-nearEdge * Math.Tan(angle + angleOffset2))));
            if (isNearRight)
            {
                points.Add(new PointF(rectRight, rectBottom));
                points.Add(new PointF(rectLeft, rectBottom));
                points.Add(new PointF(rectLeft, rectTop));
                points.Add(new PointF(rectRight, rectTop));
            }
            else
            {
                points.Add(new PointF(rectLeft, rectTop));
                points.Add(new PointF(rectRight, rectTop));
                points.Add(new PointF(rectRight, rectBottom));
                points.Add(new PointF(rectLeft, rectBottom));
            }
            points.Add(new PointF(nearEdge, (float)(-nearEdge * Math.Tan(angle - angleOffset2))));
        }

        // Arrow points
        points.Add(new PointF((float)(Math.Cos(angle - angleOffset2) * subHypotenuse),
            (float)(-Math.Sin(angle - angleOffset2) * subHypotenuse)));
        points.Add(new PointF((float)(Math.Cos(angle - angleOffset1) * hypotenuse),
            (float)(-Math.Sin(angle - angleOffset1) * hypotenuse)));
        return points;
    }
```

Back to Top

Step 3: Render the annotations in chart

To Render the annotations in chart, follow these steps:

1. Define global fields of AnnotationLayer class and render engine in Form class of your application.
   - **Visual Basic**

```
Dim annotationLayer As AnnotationLayer
Dim _engine As IRenderEngine
```

   - **C#**

```
AnnotationLayer annotationLayer;
IRenderEngine _engine;
```

2. Create an instance of AnnotationLayer class, in SetUpAnnotations() method.
   - **Visual Basic**

```
' Create an instance of AnnotationLayer class
annotationLayer = New AnnotationLayer(FlexChart1)
```

   - **C#**

```
// Create an instance of AnnotationLayer class
annotationLayer = new AnnotationLayer(flexChart1);
```

3. Add the line and arrow callout annotations to the annotationLayer, by using the following code.
   - **Visual Basic**

```
 'Add the polygon annotations with line and arrow connectors to the annotationLayer
 annotationLayer.Annotations.Add(arrowCallout)
 annotationLayer.Annotations.Add(lineCallout)
End Sub
```

   - **C#**

```
 //Add the line and arrow callout annotations to the annotationLayer
```

```
        annotationLayer.Annotations.Add(arrowCallout);
        annotationLayer.Annotations.Add(lineCallout);
    }
```

4. To render the callouts use the following code in the Rendered event of FlexChart.
    - **Visual Basic**

```
Private Sub FlexChart1_Rendered(sender As Object, e As C1.Win.Chart.RenderEventArgs)
    Handles FlexChart1.Rendered
    If _engine Is Nothing Then
        _engine = e.Engine
        SetUpAnnotations()
        FlexChart1.Invalidate()
    End If
End Sub
```

    - **C#**

```
private void flexChart1_Rendered(object sender, C1.Win.Chart.RenderEventArgs e)
{
    if (_engine == null)
    {
        _engine = e.Engine;
        SetUpAnnotations();
        flexChart1.Invalidate();
    }
}
```

Back to Top

Legend

By default, FlexChart doesn't display the Legend until the series are specified with the Name property available in Series Collection Editor.

Refer to Series Collection Editor for more details.

Once you've set the series with their relevant names, the Legend is displayed by FlexChart.

The following topics discuss how to customize legends.

- Legend Position
- Legend Style
- Legend Toggle
- Legend Text Wrap
- Legend Grouping

Legend Position

You can use the Position property to position the Legend relative to the Plot Area, as per your requirements.

The Position property can be set to any of the following values in the Position enumeration:

Property	Description
Position.Auto	Positions the legend automatically.
Position.Bottom	Positions the legend below the plot.
Position.Left	Positions the legend to the left of the plot.
Position.None	Hides the legend.
Position.Right (default value)	Positions the legend to the right of the plot.
Position.Top	Positions the legend above the plot.

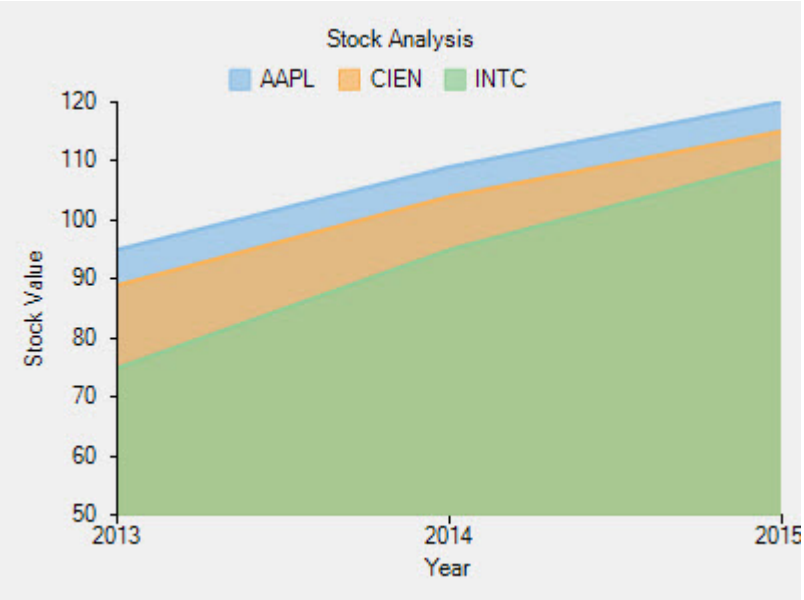
Here is the code snippet for setting the property:

- **Visual Basic**

```
' set the Legend position
FlexChart1.Legend.Position = C1.Chart.Position.Top
```

- **C#**

```
// set the Legend position
flexChart1.Legend.Position = C1.Chart.Position.Top;
```



Legend Style

FlexChart lets you can customize the Legend using the Style property.

The table below lists the properties available for customizing the Legend:

Property	Description
FillColor	Specifies the fill color.
Font	Sets the font of the Legend.
StrokeColor	Sets the stroke color.
StrokeWidth	Sets the stroke width.

Legend Toggle

FlexChart allows you to toggle the visibility of a series in the plot, when you click the series item in the legend by means of the LegendToggle property.

The default value of the LegendToggle property is False. To enable series toggling, you need to set the LegendToggle property to True.

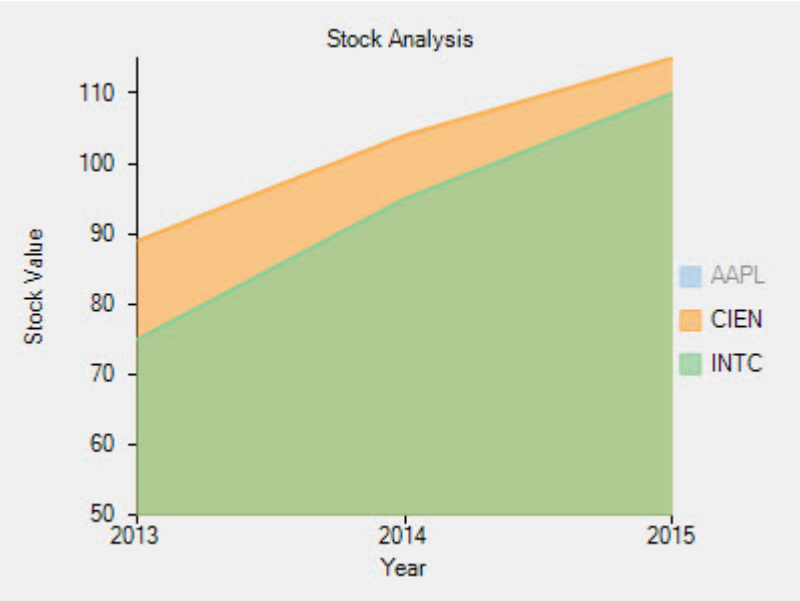
Here is the code snippet:

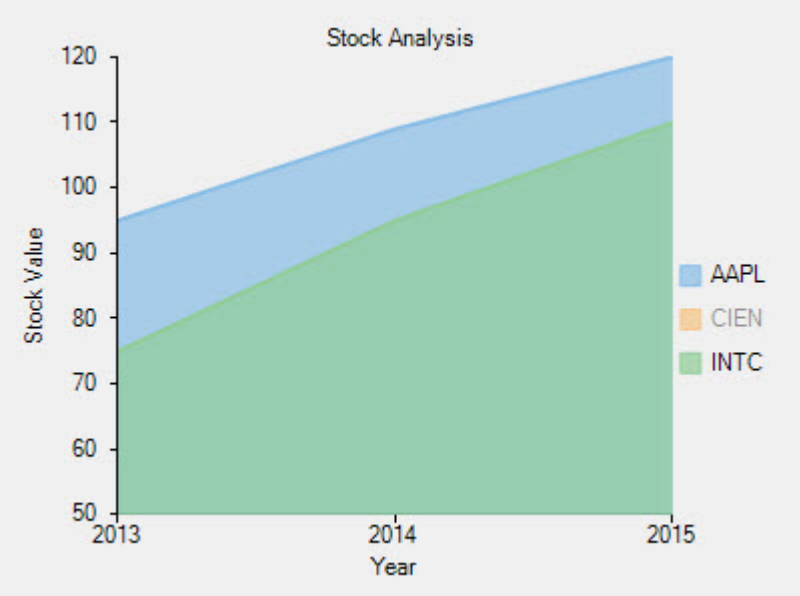
- **Visual Basic**

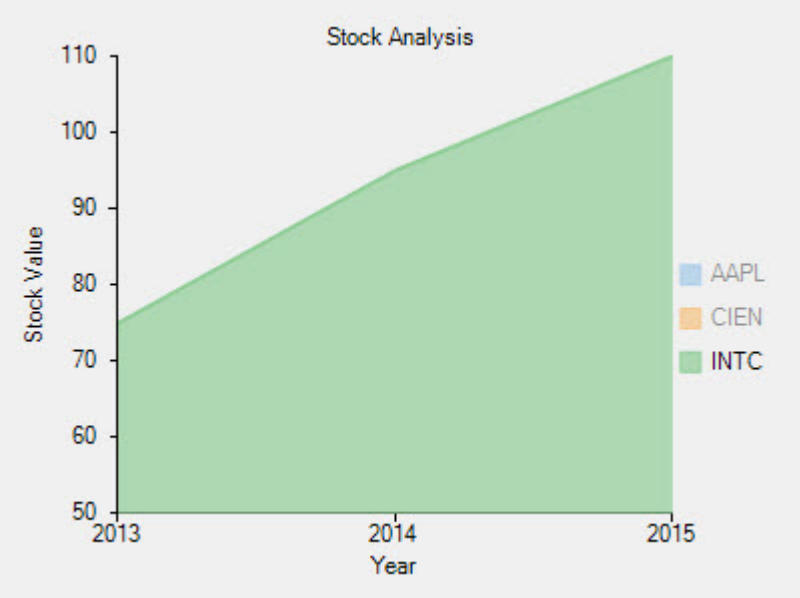
```
FlexChart1.LegendToggle = True
```

- **C#**

```
flexChart1.LegendToggle = true;
```







Legend Text Wrap

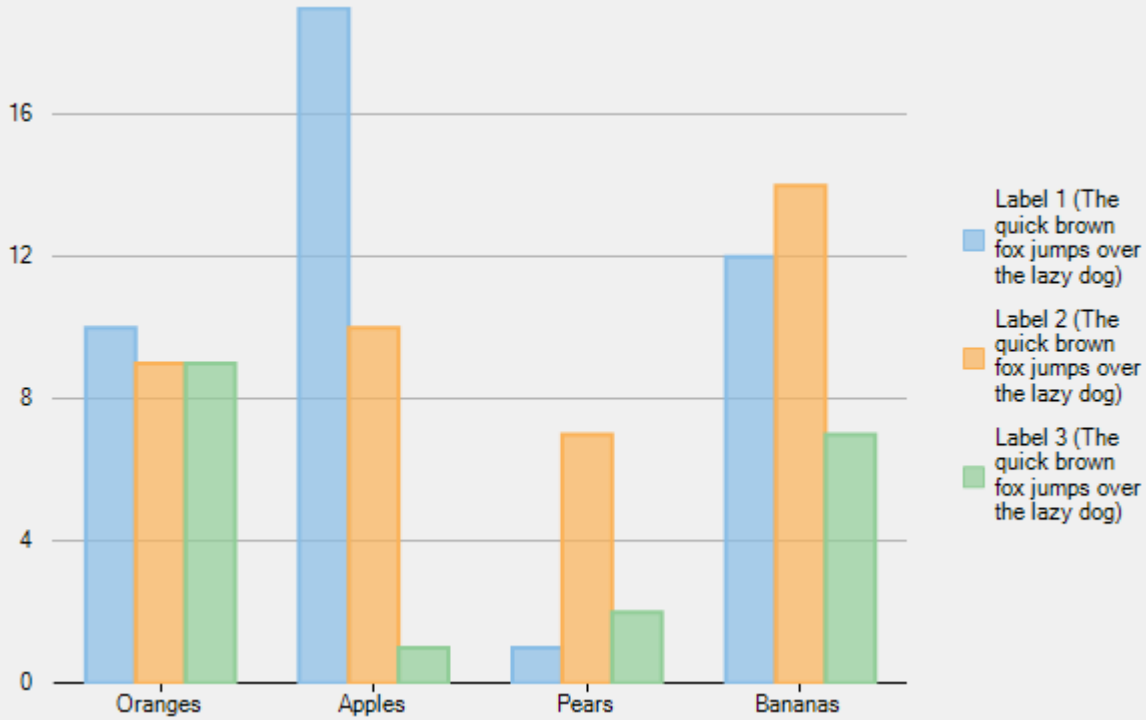
Legend text wrap is a feature to shorten the legend entries by either truncating or wrapping them into multiple lines. This feature gives user the flexibility to effectively utilize chart display area by adjusting the space occupied by legends.

FlexChart provides text wrapping for the legend text that exceeds the value specified in ItemMaxWidth property, which sets the maximum width of each legend entry. The control enables you to manage legend entries in following two ways:

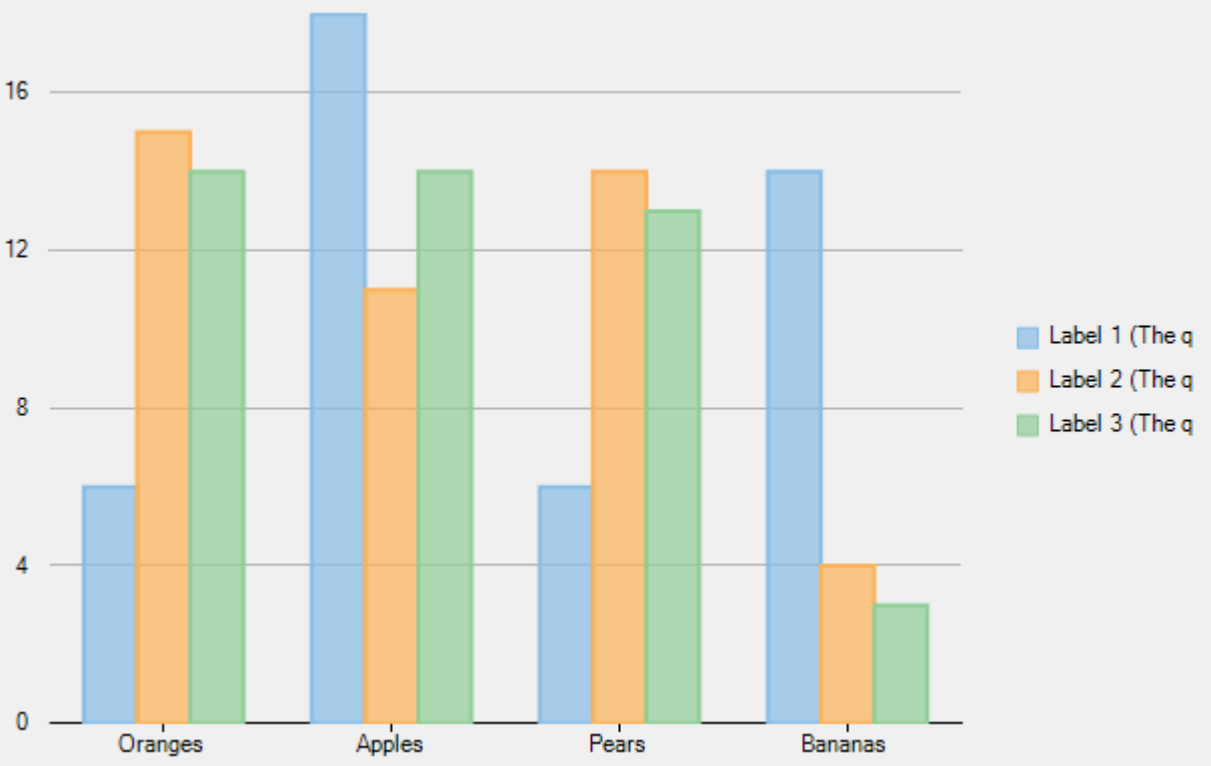
- **Wrap**: This mode allows you to wrap or break the legend entries into multiple lines. Wrapping text is useful when you need the entire legend text to be visible in chart area. To wrap legend texts in FlexChart, set the TextWrapping property to **Wrap** from the TextWrapping enum.
- **Truncate**: This mode allows you to shorten legend entries by cutting off the text from the end. If you want to truncate legend texts in FlexChart, set the **TextWrapping** property to **Truncate**.

In FlexChart, the maximum width set for the legend entries affects both text wrapping and text truncating. The greater the value set for maximum legend entry width, the less the legend text is wrapped or truncated.

The following image displays legend texts wrapped into multiple lines.



The following image displays truncated legend texts.



The following code shows how to implement legend text wrapping in FlexChart.

- **Visual Basic**

```
' set maximum width of legend items
FlexChart1.Legend.ItemMaxWidth = 80

' set legend text wrap mode
```

```
FlexChart1.Legend.TextWrapping = C1.Chart.TextWrapping.Wrap

' set legend position
FlexChart1.Legend.Position = C1.Chart.Position.Right
```

- **C#**

```
// set maximum width of legend items
flexChart1.Legend.ItemMaxWidth = 80;

// set legend text wrap mode
flexChart1.Legend.TextWrapping = C1.Chart.TextWrapping.Wrap;

// set legend position
flexChart1.Legend.Position = C1.Chart.Position.Right;
```

Legend Grouping

Legend group, as the name suggests, categorizes the legend entries of chart series based on the data represented by them. So, the multiple chart series with similar data can be better presented as groups in legend. This organizes the legends, which helps in better visualization and analysis of charts depicting multiple series.

FlexChart supports grouping respective legend items of different series in chart through LegendGroup property. By setting the LegendGroup property to a string value, you can specify the group name to which a particular series or legend item belongs. The series for which value of LegendGroup property is same are grouped together in the legend. However, if the LegendGroup property is not defined for a series then the series becomes a part of 0th group.

The value of LegendGroup property gets displayed as group title above the corresponding legend items. However, legend items that belong to the 0th group get displayed without any group title.

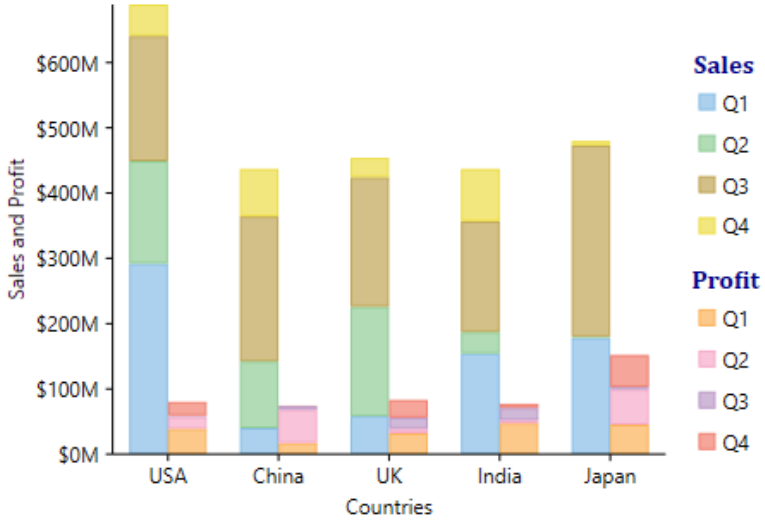
Positioning Legend Groups

The legend groups get positioned automatically with respect to each other depending on legend's position. For example, if the legends are positioned on top or bottom of the chart, then the legend groups are stacked horizontally one alongside the other. Whereas, if the legends are positioned to left or right of the chart, then the legend groups are stacked vertically one above the other.

Styling Legend Groups

FlexChart also supports styling and formatting of the legend group titles. The appearance of legend group titles can be customized by specifying the GroupHeaderStyle property.

Following image shows a stacked chart that plots country-wise sales and profit of a company for different quarters of a year. Here, the legend items have been grouped together as per the stacked series for quick and easy analysis. The image also shows how legend and the legend groups have got positioned vertically and how appearance of the group titles can be customized.



Following code snippet demonstrates how to group the legends of respective series together by setting the Series.LegendGroup property of those series to the desired group name. The code snippet also shows how GroupHeaderStyle property can be used to style headers of the legend groups.

- **C#**

```
// specify legend group for each series
flexChart1.Series[0].LegendGroup = "Sales";
flexChart1.Series[1].LegendGroup = "Profit";
flexChart1.Series[2].LegendGroup = "Sales";
flexChart1.Series[3].LegendGroup = "Profit";
flexChart1.Series[4].LegendGroup = "Sales";
flexChart1.Series[5].LegendGroup = "Profit";
flexChart1.Series[6].LegendGroup = "Sales";
flexChart1.Series[7].LegendGroup = "Profit";

// stylise legend group header
flexChart1.Legend.GroupHeaderStyle.Stroke = Brushes.DarkBlue;
flexChart1.Legend.GroupHeaderStyle.Font = new Font(FontFamily.GenericMonospace, 10f, FontStyle.Bold);

flexChart1.EndUpdate();
```

To add series to the chart, use the following code snippet.

- **C#**

```
    InitializeComponent();
    SetupChart();
}
void SetupChart()
{
    flexChart1.BeginUpdate();
    flexChart1.BackColor = Color.White;
    flexChart1.Series.Clear();

    // Add data series
    var s1 = new Series();
    s1.Binding = "SalesQ1";
    s1.Name = "Q1";
    flexChart1.Series.Add(s1);

    var s2 = new Series();
    s2.Binding = "ProfitQ1";
    s2.Name = "Q1";
    flexChart1.Series.Add(s2);

    var s3 = new Series();
    s3.Binding = "SalesQ2";
    s3.Name = "Q2";
    flexChart1.Series.Add(s3);

    var s4 = new Series();
    s4.Binding = "ProfitQ2";
    s4.Name = "Q2";
    flexChart1.Series.Add(s4);

    var s5 = new Series();
    s5.Binding = "SalesQ3";
    s5.Name = "Q3";
    flexChart1.Series.Add(s5);

    var s6 = new Series();
    s6.Binding = "ProfitQ3";
    s6.Name = "Q3";
    flexChart1.Series.Add(s6);

    var s7 = new Series();
    s7.Binding = "SalesQ4";
    s7.Name = "Q4";
    flexChart1.Series.Add(s7);

    var s8 = new Series();
```

```
    s8.Binding = "ProfitQ4";
    s8.Name = "Q4";
    flexChart1.Series.Add(s8);

    // Set x-binding and add data to the chart
    flexChart1.BindingX = "Country";
    flexChart1.DataSource = DataCreator.CreateCountry();

    // set FlexChart stacking type
    flexChart1.Stacking = C1.Chart.Stacking.Stacked;

    // specify stacking group for each series
    flexChart1.Series[0].StackingGroup = 0;
    flexChart1.Series[1].StackingGroup = 1;
    flexChart1.Series[2].StackingGroup = 0;
    flexChart1.Series[3].StackingGroup = 1;
    flexChart1.Series[4].StackingGroup = 0;
    flexChart1.Series[5].StackingGroup = 1;
    flexChart1.Series[6].StackingGroup = 0;
    flexChart1.Series[7].StackingGroup = 1;
```

Data Source

This example uses the following data.

- **C#**

```
class DataCreator
{
    public static CountryDataItem[] CreateCountry()
    {
        var countries = new string[] { "USA", "China", "UK", "India", "Japan" };
        var count = countries.Length;
        var result = new CountryDataItem[count];
        var rnd = new Random();
        for (var i = 0; i < count; i++)
            result[i] = new CountryDataItem()
            {
                Country = countries[i],
                SalesQ1 = rnd.Next(320),
                ProfitQ1 = rnd.Next(50),
                SalesQ2 = rnd.Next(200),
                ProfitQ2 = rnd.Next(60),
                SalesQ3 = rnd.Next(300),
                ProfitQ3 = rnd.Next(20),
                SalesQ4 = rnd.Next(120),
                ProfitQ4 = rnd.Next(50),
            };
        return result;
    }
}
public class CountryDataItem
{
    public string Country { get; set; }
    public double SalesQ1 { get; set; }
    public double ProfitQ1 { get; set; }
    public double SalesQ2 { get; set; }
    public double ProfitQ2 { get; set; }
    public double SalesQ3 { get; set; }
    public double ProfitQ3 { get; set; }
    public double SalesQ4 { get; set; }
    public double ProfitQ4 { get; set; }
}
```

Series

A series is a set of data or more specifically related data points that are plotted on a chart.

In FlexChart, a series is represented by the **Series** object, which provides the entire data plotted on the chart. And the

Flexchart.Series collection comprises all data series (**Series** objects) in the control.

You can assign any of the following properties to a series in FlexChart:

- An X-axis (**Series.AxisX**)
- A Y-axis (**Series.AxisY**)
- A property containing Y values for the series (**Series.Binding**)
- A property containing X values for the series (**Series.BindingX**)
- A chart type (**Series.ChartType**)
- A collection of objects containing the series data (**Series.DataSource**)
- A name (**Series.Name**)

A series consists of a collection of data points that you can customize using the following properties:

- To set the shape of the marker to be used for each data point in the series (**Series.SymbolMarker**)
- To set the size of the symbol used to render the series (**Series.SymbolSize**)
- To set the symbol style used in the data points in the series (**Series.SymbolStyle**)

Once you have set these properties in a series, the same settings are inherited by all data points.

Here are the links to key information regarding the Series object in FlexChart:

- Creating and Adding a Series
- Adding Data to Series
- Emphasizing Different Types of Data
- Customizing Series

Creating and Adding Series

By default, FlexChart displays one series at design-time. However, the control doesn't display the series at run-time until the chart is provided with data. For information on how to provide data to FlexChart, refer to Providing Data.

FlexChart enables you to create and add a series at design-time as well as run-time.

At Design-Time

Perform the following steps to create and add a series in FlexChart at design-time:

1. In the **Properties** window, navigate to the **Series** field.
2. Click the **Ellipsis** button next to the **Series** field.
   Series Collection Editor appears.

   > By default, FlexChart contains one series added in the Series collection. Thus, **Series Collection Editor** appears with a pre-added series, **Series 1** .
3. Click the **Add** button to add an additional series in the Series collection.
4. Repeat step 3 to add the required number of series.
5. Click the **OK** button.

At Run-Time

At run-time, you first need to create a series by using the **Series** object. And then, you need to add the series to the FlexChart Series collection using the **Add** method in the **FlexChart.Series** collection property.

The following code shows how to create and add a series in FlexChart at run-time.

- **Visual Basic**

```
' create series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()

' add the series
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
```

- **C#**

```
// create series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();

// add the series
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);
```

Adding Data to Series

FlexChart allows you to add data to a series easily by using the **SetData** method of the **Series** object. The SetData() accepts two arrays of the double type as parameters: one for specifying X values and another for specifying Y values.

Here is the code demonstrating the use of the SetData().

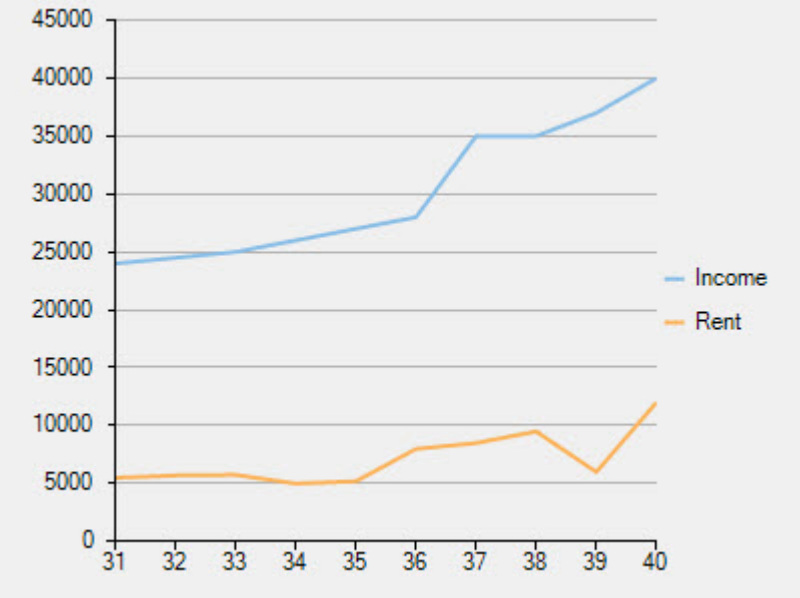
- **Visual Basic**

```
' name the series
series1.Name = "Income"
series2.Name = "Rent"

' add data to the series
series1.SetData(
    New Double() {31, 32, 33, 34, 35,
                  36, 37, 38, 39, 40},
    New Double() {24000, 24500, 25000, 26000, 27000,
                  28000, 35000, 35000, 37000, 40000})
series2.SetData(
    New Double() {31, 32, 33, 34, 35,
                  36, 37, 38, 39, 40},
    New Double() {5500, 5700, 5750, 5000, 5200,
                  8000, 8500, 9500, 6000, 12000})
```

- **C#**

```
// name the series
series1.Name = "Income";
series2.Name = "Rent";

// add data to the series
series1.SetData(
    new double[] { 31, 32, 33, 34, 35,
        36, 37, 38, 39, 40 },
    new double[] { 24000, 24500, 25000, 26000,
        27000, 28000, 35000, 35000, 37000, 40000 });
series2.SetData(
    new double[] { 31, 32, 33, 34, 35,
        36, 37, 38, 39, 40 },
    new double[] { 5500, 5700, 5750, 5000,
```

```
        5200, 8000, 8500, 9500, 6000, 12000 });
```



You need to clear the Series collection in FlexChart using the **Clear()** in the **FlexChart.Series** collection property before plotting series data. If the Series collection is not cleared, the default series entry will still be visible in the Legend. Using the Clear() overrides the default data and plots the provided data on the chart.

You can also add data to a series using the Point array in FlexChart. Refer to Loading Data from Arrays for more details.

When it comes to adding data to series, FlexChart provides even a more powerful way through binding. You can bind series in FlexChart with multiple data sources, which enables you to combine data from multiple data sources. To plot data from multiple data sources, you need to use the **Series.DataSource** property.

See the following code for reference.

- **Visual Basic**

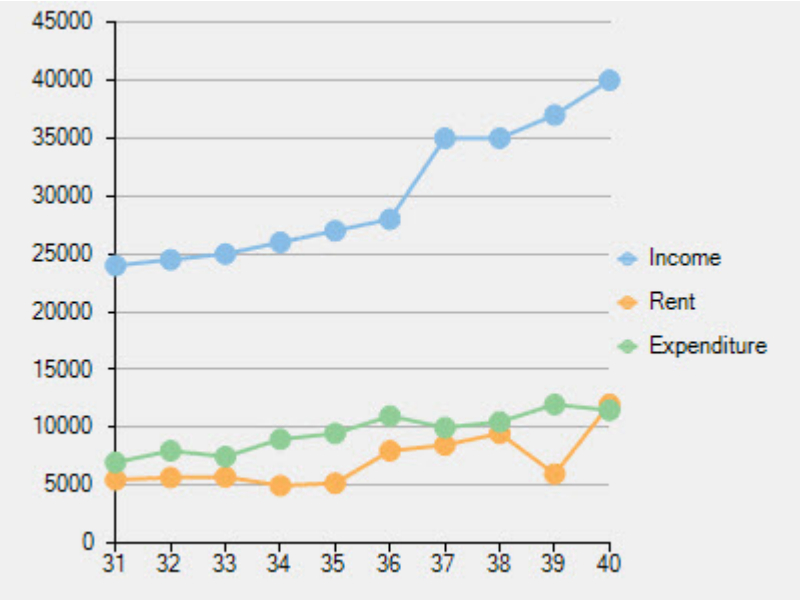
```
' create series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()
Dim series3 As New C1.Win.Chart.Series()

' add the series
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
FlexChart1.Series.Add(series3)

' name the series
series1.Name = "Income"
series2.Name = "Rent"
series3.Name = "Expenditure"

' create a datatable
Dim dt As New DataTable()

' add columns to the datatable
dt.Columns.Add("Age Group", GetType(Integer))
dt.Columns.Add("Expenditure", GetType(Integer))

' add rows to the datatable
```

```
dt.Rows.Add(31, 7000)
dt.Rows.Add(32, 8000)
dt.Rows.Add(33, 7500)
dt.Rows.Add(34, 9000)
dt.Rows.Add(35, 9500)
dt.Rows.Add(36, 11000)
dt.Rows.Add(37, 10000)
dt.Rows.Add(38, 10500)
dt.Rows.Add(39, 12000)
dt.Rows.Add(40, 11500)

' add data to the first two series
series1.SetData(New Double() {31, 32, 33, 34, 35, 36, _
    37, 38, 39, 40}, New Double() {24000, 24500, 25000, 26000, 27000, 28000, _
    35000, 35000, 37000, 40000})
series2.SetData(New Double() {31, 32, 33, 34, 35, 36, _
    37, 38, 39, 40}, New Double() {5500, 5700, 5750, 5000, 5200, 8000, _
    8500, 9500, 6000, 12000})

' set dt as the datasource of the third series
series3.DataSource = dt

' bind X-axis and Y-axis of the third series
series3.Binding = "Expenditure"
series3.BindingX = "Age Group"
```

- **C#**

```
// create series
C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();
C1.Win.Chart.Series series3 = new C1.Win.Chart.Series();

// add the series
flexChart1.Series.Add(series1);
flexChart1.Series.Add(series2);
flexChart1.Series.Add(series3);

// name the series
series1.Name = "Income";
series2.Name = "Rent";
series3.Name = "Expenditure";

// create a datatable
DataTable dt = new DataTable();

// add columns to the datatable
dt.Columns.Add("Age Group", typeof(int));
dt.Columns.Add("Expenditure", typeof(int));

// add rows to the datatable
dt.Rows.Add(31, 7000);
dt.Rows.Add(32, 8000);
dt.Rows.Add(33, 7500);
dt.Rows.Add(34, 9000);
dt.Rows.Add(35, 9500);
dt.Rows.Add(36, 11000);
dt.Rows.Add(37, 10000);
dt.Rows.Add(38, 10500);
dt.Rows.Add(39, 12000);
dt.Rows.Add(40, 11500);
```

```
// add data to the first two series
series1.SetData(
    new double[] { 31, 32, 33, 34, 35,
        36, 37, 38, 39, 40 },
    new double[] { 24000, 24500, 25000, 26000, 27000,
        28000, 35000, 35000, 37000, 40000 });
series2.SetData(
    new double[] { 31, 32, 33, 34, 35,
        36, 37, 38, 39, 40 },
        new double[] { 5500, 5700, 5750, 5000, 5200,
            8000, 8500, 9500, 6000, 12000 });

// set dt as the datasource of the third series
series3.DataSource = dt;

// bind X-axis and Y-axis of the third series
series3.Binding = "Expenditure";
series3.BindingX = "Age Group";
```



Emphasizing Different Types of Data

In a chart, there is often a common but crucial requirement to emphasize different types of data. Emphasizing or distinguishing different types of data is vital because chart data that is distinguishable is easier to interpret and understand.

FlexChart caters to this requirement by allowing you to combine two or more chart types in a single chart. For instance, you can combine the LineSymbols Chart with the Column Chart to make the chart data easier to interpret. You can use the **ChartType** property to specify the chart type at the series level for each series, thereby creating charts with multiple chart types.

The following code combines two chart types in a single chart.

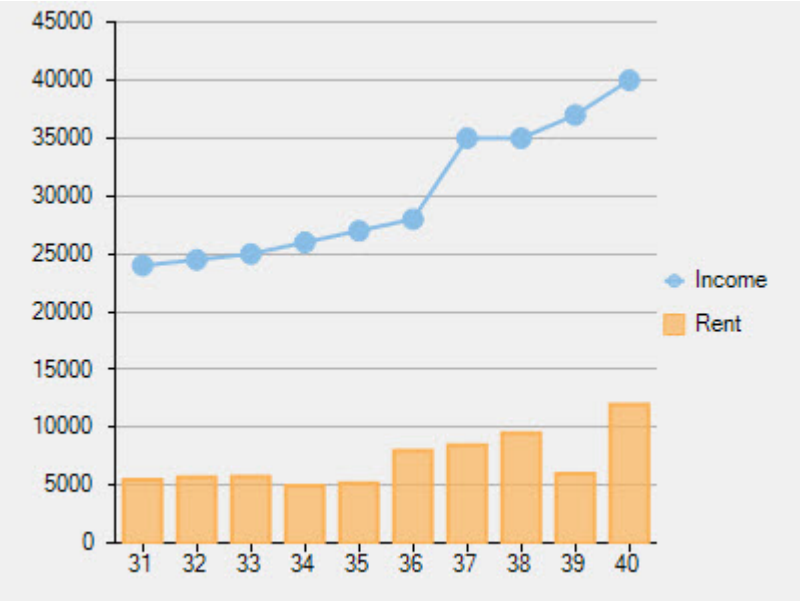
- **Visual Basic**

```
' create series
Dim series1 As New C1.Win.Chart.Series()
Dim series2 As New C1.Win.Chart.Series()

' add the series
```

```
FlexChart1.Series.Add(series1)
FlexChart1.Series.Add(series2)
```

- **C#**

```
// specify chart type for each series
series1.ChartType = C1.Chart.ChartType.LineSymbols;
series2.ChartType = C1.Chart.ChartType.Column;
```



With each series representing its own chart type, you can easily identify the data represented by a particular series.

Nevertheless, just combining multiple chart types in a single chart doesn't suffice when data to be plotted requires different units or scales. In such a case, you require plotting different data on different axes to emphasize their types. FlexChart fulfils this requirement as well by letting you work with multiple axes seamlessly.

For information on how to create and use multiple axes in FlexChart, refer to Multiple Axes.

Customizing Series

Once the series have been displayed in the chart, you can customize the displayed series to manage the same more efficiently.

FlexChart allows you to customize series by showing or hiding a series either in the Plot Area or the Legend or both. If there are hundreds of series to be displayed in your chart, you would certainly need to manage the same due to the space limitation of the chart.

In FlexChart, you can manage series by using the Visibility property of a series. The Visibility property accepts values of the SeriesVisibility enumerated type.

You can set the property to the following different values to show or hide a series:

Value	Description
SeriesVisibility.Visible	The series is displayed in the Plot as well as the Legend.
SeriesVisibility.Plot	The series is displayed in the Plot, but hidden in the Legend.
SeriesVisibility.Legend	The series is displayed in the Legend, but hidden in the Plot.
SeriesVisibility.Hidden	The series is hidden in the Plot as well as the Legend.

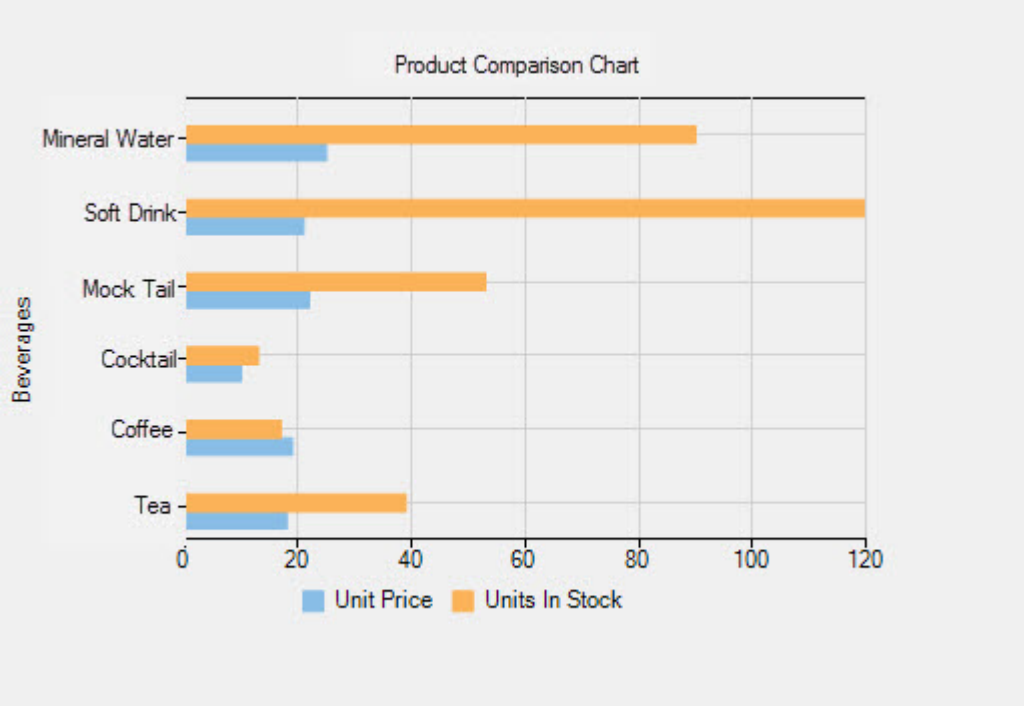
Here's the code snippet for reference:

- **Visual Basic**

```
' name the series
series1.Name = "Unit Price"
series2.Name = "Units In Stock"
series3.Name = "Units On Order"

' to hide Units On Order
series3.Visibility = C1.Chart.SeriesVisibility.Hidden
```

- **C#**

```
// name the series
series1.Name = "Unit Price";
series2.Name = "Units In Stock";
series3.Name = "Units On Order";

// to hide Units On Order
series3.Visibility = C1.Chart.SeriesVisibility.Hidden;
```



If you want to show the series (Units On Order) in the Plot Area, but at the same not display it in the Legend, you can use the Visibility property in the following manner:

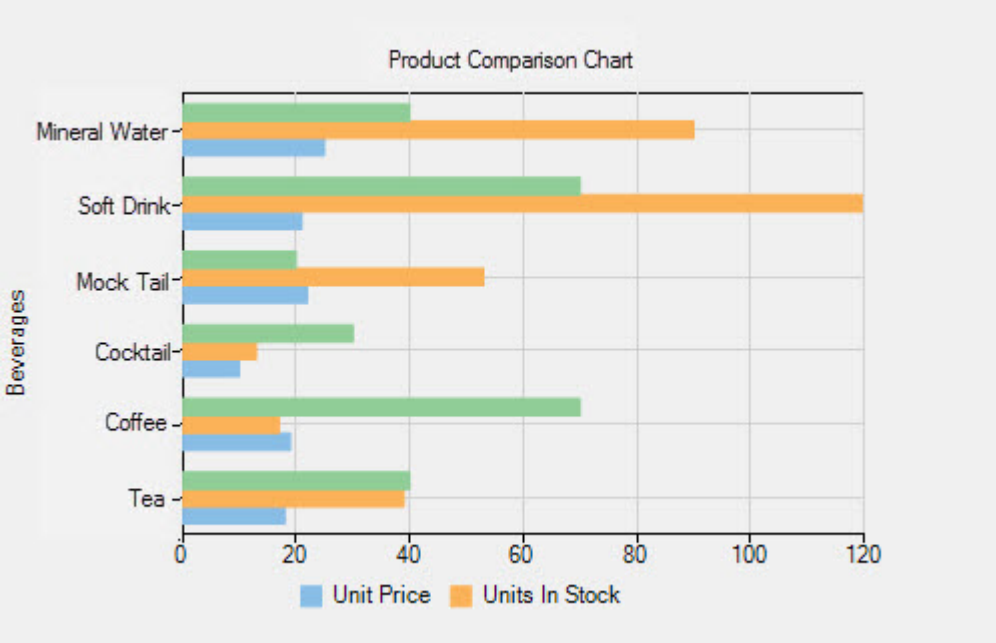
- **Visual Basic**

```
' name the series
series1.Name = "Unit Price"
series2.Name = "Units In Stock"
series3.Name = "Units On Order"

' to show Units On Order in the Plot but not in the Legend
series3.Visibility = C1.Chart.SeriesVisibility.Plot
```

- **C#**

```
// name the series
series1.Name = "Unit Price";
```

```
series2.Name = "Units In Stock";
series3.Name = "Units On Order";

// to show Units On Order in the Plot but not in the Legend
series3.Visibility = C1.Chart.SeriesVisibility.Plot;
```



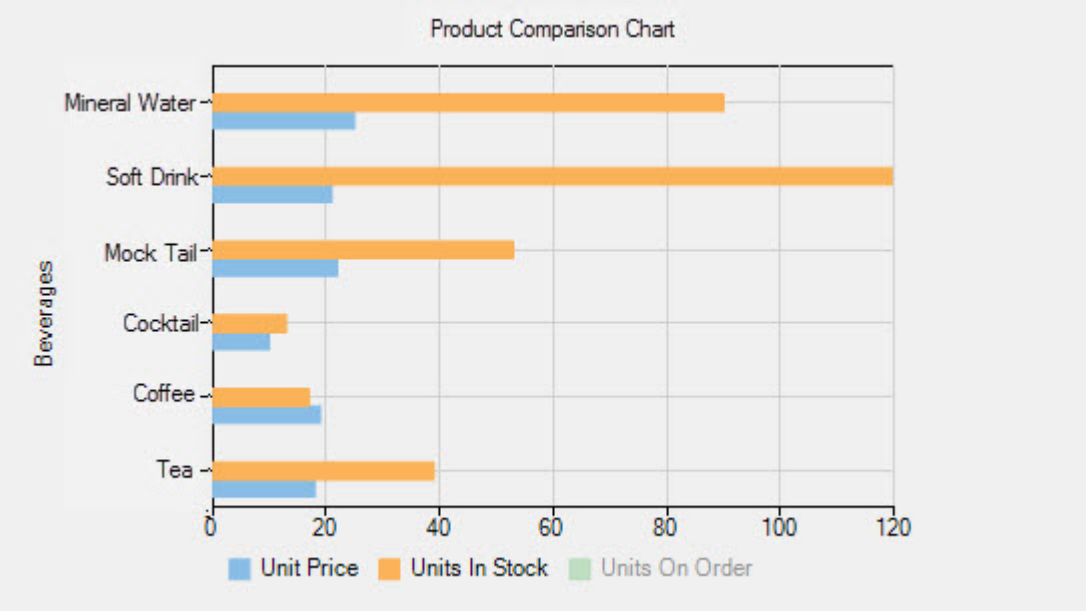
Similarly, for displaying the series in the Legend, but not in the Plot Area, you can set the Visibility property as follows:

- **Visual Basic**

```
' name the series
series1.Name = "Unit Price"
series2.Name = "Units In Stock"
series3.Name = "Units On Order"

' to show Units On Order in the Legend but not in the Plot
series3.Visibility = C1.Chart.SeriesVisibility.Legend
```

- **C#**

```
// name the series
series1.Name = "Unit Price";
series2.Name = "Units In Stock";
series3.Name = "Units On Order";

// to show Units On Order in the Legend but not in the Plot
series3.Visibility = C1.Chart.SeriesVisibility.Legend;
```



In addition, you can enhance the visual appeal of the series by setting different palettes for FlexChart. For more details, refer to Setting FlexChart Palette.

You can also work with different symbol styles to render visually appealing series in the chart. For more information, refer to Symbol styles for Series.

Box-and-Whisker

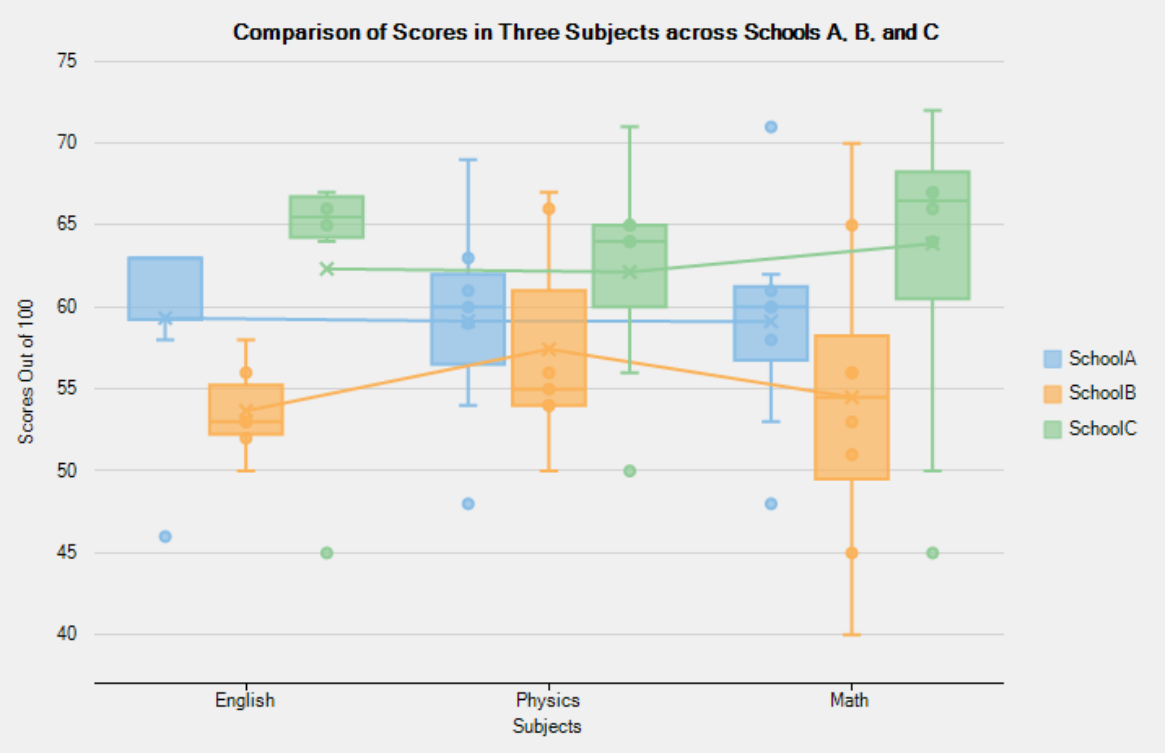
Box-and-Whisker series allows you to display groups of data into the range, quartiles, and median. The name itself suggests that the series depicts data through boxes and whiskers.

A box is the range showing the quartiles (lower and upper) and the median. Whiskers, on the other hand, are the lines extending vertically from the boxes. These lines indicate the data variability outside the lower and the upper quartiles. In addition, points that lie outside of these lines are known as outliers.

Box-and-Whisker series is ideal for visualizing statistical distribution or examining multiple sets of data graphically. In FlexChart, the series allows working with different features, as follows:

- **Quartile**: Specify whether you would like to calculate quartiles by including or excluding median. To specify quartile calculation, set the QuartileCalculation property from the QuartileCalculation enumeration.
- **Inner points**: Indicate whether to show or hide inner points by setting the ShowInnerPoints property.
- **Outliers**: Indicate whether to show outliers by setting the ShowOutliers property.
- **Mean line**: Display the mean line by setting the ShowMeanLine property.
- **Mean marks**: Show mean marks by setting the ShowMeanMarks property.

The following image displays quartiles, median, and whiskers for the data that compares scores of students in three subjects across different schools.



The following code uses data regarding scores obtained by students of schools A, B, and C in three subjects. The code illustrates how to implement Box-and-Whisker series in FlexChart.

- **Visual Basic**

```
Public Sub New()
    InitializeComponent()
    SetupChart()
End Sub

Private Sub InitializeComponent()
    Throw New NotImplementedException()
End Sub

Private Sub SetupChart()
    flexChart1.BeginUpdate()

    ' specify the data source
    flexChart1.DataSource = CreateForBoxWhisker()

    ' Set the property containing X values
    flexChart1.BindingX = "ClassName"

    ' clear the Series collection
    flexChart1.Series.Clear()

    ' create first Boxwhisker series and set various properties
    Dim boxWhiskerA As C1.Win.Chart.BoxWhisker = New BoxWhisker()
    boxWhiskerA.Name = "SchoolA"
    boxWhiskerA.Binding = "SchoolA"
    boxWhiskerA.QuartileCalculation = QuartileCalculation.InclusiveMedian
    boxWhiskerA.ShowInnerPoints = True
    boxWhiskerA.ShowOutliers = True
    boxWhiskerA.ShowMeanLine = True
    boxWhiskerA.ShowMeanMarks = True


    Dim bindAxis = New Y_AxisBinding() {New Y_AxisBinding() With {
        .d = 10,
        .str = "10 (Fail)"
    }, New Y_AxisBinding() With {
        .d = 20,
```

```
        .str = "20 (Fail)"
    }, New Y_AxisBinding() With {
        .d = 30,
        .str = "30 (Fail)"
    }, New Y_AxisBinding() With {
        .d = 40,
        .str = "40 (Pass)"
    }, New Y_AxisBinding() With {
        .d = 50,
        .str = "50 (Pass)"
    }, New Y_AxisBinding() With {
        .d = 60,
        .str = "60 (Pass)"
    },
        New Y_AxisBinding() With {
        .d = 70,
        .str = "70 (Pass)"
    }}


    ' create second Boxwhisker series and set various properties
    Dim boxWhiskerB As C1.Win.Chart.BoxWhisker = New BoxWhisker()
    boxWhiskerB.Name = "SchoolB"
    boxWhiskerB.Binding = "SchoolB"
    boxWhiskerB.QuartileCalculation = QuartileCalculation.InclusiveMedian
    boxWhiskerB.ShowInnerPoints = True
    boxWhiskerB.ShowOutliers = True
    boxWhiskerB.ShowMeanLine = True
    boxWhiskerB.ShowMeanMarks = True

    ' create third Boxwhisker series and set various properties
    Dim boxWhiskerC As C1.Win.Chart.BoxWhisker = New BoxWhisker()
    boxWhiskerC.Name = "SchoolC"
    boxWhiskerC.Binding = "SchoolC"
    boxWhiskerC.QuartileCalculation = QuartileCalculation.InclusiveMedian
    boxWhiskerC.ShowInnerPoints = True
    boxWhiskerC.ShowOutliers = True
    boxWhiskerC.ShowMeanLine = True
    boxWhiskerC.ShowMeanMarks = True

    ' add the series to the Series collection
    flexChart1.Series.Add(boxWhiskerA)
    flexChart1.Series.Add(boxWhiskerB)
    flexChart1.Series.Add(boxWhiskerC)

    flexChart1.EndUpdate()
End Sub

' create data source
Public Shared Function CreateForBoxWhisker() As List(Of ClassScore)
    Dim result = New List(Of ClassScore)()
    result.Add(New ClassScore() With {
        .ClassName = "English",
        .SchoolA = 46,
        .SchoolB = 53,
        .SchoolC = 66
    })
    result.Add(New ClassScore() With {
        .ClassName = "Physics",
        .SchoolA = 61,
        .SchoolB = 55,
        .SchoolC = 65
    })
    result.Add(New ClassScore() With {
        .ClassName = "English",
        .SchoolA = 58,
        .SchoolB = 56,
        .SchoolC = 67
    })
    result.Add(New ClassScore() With {
        .ClassName = "Math",
        .SchoolA = 58,
```

```
    .SchoolB = 51,
    .SchoolC = 64
})
result.Add(New ClassScore() With {
    .ClassName = "English",
    .SchoolA = 63,
    .SchoolB = 53,
    .SchoolC = 45
})
result.Add(New ClassScore() With {
    .ClassName = "English",
    .SchoolA = 63,
    .SchoolB = 50,
    .SchoolC = 65
})
result.Add(New ClassScore() With {
    .ClassName = "Math",
    .SchoolA = 60,
    .SchoolB = 45,
    .SchoolC = 67
})
result.Add(New ClassScore() With {
    .ClassName = "Math",
    .SchoolA = 62,
    .SchoolB = 53,
    .SchoolC = 66
})
result.Add(New ClassScore() With {
    .ClassName = "Physics",
    .SchoolA = 63,
    .SchoolB = 54,
    .SchoolC = 64
})
result.Add(New ClassScore() With {
    .ClassName = "English",
    .SchoolA = 63,
    .SchoolB = 52,
    .SchoolC = 67
})
result.Add(New ClassScore() With {
    .ClassName = "Physics",
    .SchoolA = 69,
    .SchoolB = 66,
    .SchoolC = 71
})
result.Add(New ClassScore() With {
    .ClassName = "Physics",
    .SchoolA = 48,
    .SchoolB = 67,
    .SchoolC = 50
})
result.Add(New ClassScore() With {
    .ClassName = "Physics",
    .SchoolA = 54,
    .SchoolB = 50,
    .SchoolC = 56
})
result.Add(New ClassScore() With {
    .ClassName = "Physics",
    .SchoolA = 60,
    .SchoolB = 56,
    .SchoolC = 64
})
result.Add(New ClassScore() With {
    .ClassName = "Math",
    .SchoolA = 71,
    .SchoolB = 65,
    .SchoolC = 50
})
result.Add(New ClassScore() With {
    .ClassName = "Math",
    .SchoolA = 48,
```

```
        .SchoolB = 70,
        .SchoolC = 72
    })
    result.Add(New ClassScore() With {
        .ClassName = "Math",
        .SchoolA = 53,
        .SchoolB = 40,
        .SchoolC = 80
    })
    result.Add(New ClassScore() With {
        .ClassName = "Math",
        .SchoolA = 60,
        .SchoolB = 56,
        .SchoolC = 67
    })
    result.Add(New ClassScore() With {
        .ClassName = "Math",
        .SchoolA = 61,
        .SchoolB = 56,
        .SchoolC = 45
    })
    result.Add(New ClassScore() With {
        .ClassName = "English",
        .SchoolA = 63,
        .SchoolB = 58,
        .SchoolC = 64
    })
    result.Add(New ClassScore() With {
        .ClassName = "Physics",
        .SchoolA = 59,
        .SchoolB = 54,
        .SchoolC = 65
    })

    Return result
End Function

Public Class ClassScore
    Public Property ClassName() As String
        Get
            Return m_ClassName
        End Get
        Set
            m_ClassName = Value
        End Set
    End Property
    Private m_ClassName As String
    Public Property SchoolA() As Double
        Get
            Return m_SchoolA
        End Get
        Set
            m_SchoolA = Value
        End Set
    End Property
    Private m_SchoolA As Double
    Public Property SchoolB() As Double
        Get
            Return m_SchoolB
        End Get
        Set
            m_SchoolB = Value
        End Set
    End Property
    Private m_SchoolB As Double
    Public Property SchoolC() As Double
        Get
            Return m_SchoolC
        End Get
        Set
            m_SchoolC = Value
        End Set
```

```
    End Property
    Private m_SchoolC As Double
End Class

Public Class Y_AxisBinding
    Public Property d() As Double
        Get
            Return m_d
        End Get
        Set
            m_d = Value
        End Set
    End Property
    Private m_d As Double
    Public Property str() As String
        Get
            Return m_str
        End Get
        Set
            m_str = Value
        End Set
    End Property
    Private m_str As String

    ' public int d { get; set; }

End Class
```

- **C#**

```
public Form1()
{
    InitializeComponent();
    SetupChart();
}
void SetupChart()
{
    flexChart1.BeginUpdate();

    // specify the data source
    flexChart1.DataSource = CreateForBoxWhisker();

    // Set the property containing X values
    flexChart1.BindingX = "ClassName";

    // clear the Series collection
    flexChart1.Series.Clear();

    // create first Boxwhisker series and set various properties
    C1.Win.Chart.BoxWhisker boxWhiskerA = new BoxWhisker();
    boxWhiskerA.Name = "SchoolA";
    boxWhiskerA.Binding = "SchoolA";
    boxWhiskerA.QuartileCalculation = QuartileCalculation.InclusiveMedian;
    boxWhiskerA.ShowInnerPoints = true;
    boxWhiskerA.ShowOutliers = true;
    boxWhiskerA.ShowMeanLine = true;
    boxWhiskerA.ShowMeanMarks = true;

    var bindAxis = new Y_AxisBinding[]
  {
        new Y_AxisBinding {d=10,str="10 (Fail)"},
        new Y_AxisBinding {d=20,str="20 (Fail)"},
        new Y_AxisBinding {d=30,str="30 (Fail)"},
        new Y_AxisBinding {d=40,str="40 (Pass)"},
        new Y_AxisBinding {d=50,str="50 (Pass)"},
        new Y_AxisBinding {d=60,str="60 (Pass)"},
        new Y_AxisBinding {d=70,str="70 (Pass)"},

  };


    // create second Boxwhisker series and set various properties
```

```
    C1.Win.Chart.BoxWhisker boxWhiskerB = new BoxWhisker();
    boxWhiskerB.Name = "SchoolB";
    boxWhiskerB.Binding = "SchoolB";
    boxWhiskerB.QuartileCalculation = QuartileCalculation.InclusiveMedian;
    boxWhiskerB.ShowInnerPoints = true;
    boxWhiskerB.ShowOutliers = true;
    boxWhiskerB.ShowMeanLine = true;
    boxWhiskerB.ShowMeanMarks = true;

    // create third Boxwhisker series and set various properties
    C1.Win.Chart.BoxWhisker boxWhiskerC = new BoxWhisker();
    boxWhiskerC.Name = "SchoolC";
    boxWhiskerC.Binding = "SchoolC";
    boxWhiskerC.QuartileCalculation = QuartileCalculation.InclusiveMedian;
    boxWhiskerC.ShowInnerPoints = true;
    boxWhiskerC.ShowOutliers = true;
    boxWhiskerC.ShowMeanLine = true;
    boxWhiskerC.ShowMeanMarks = true;

    // add the series to the Series collection
    flexChart1.Series.Add(boxWhiskerA);
    flexChart1.Series.Add(boxWhiskerB);
    flexChart1.Series.Add(boxWhiskerC);

    flexChart1.EndUpdate();
}

// create data source
public static List<ClassScore> CreateForBoxWhisker()
{
    var result = new List<ClassScore>();
    result.Add(new ClassScore() { ClassName = "English", SchoolA = 46, SchoolB = 53, SchoolC = 66 });
    result.Add(new ClassScore() { ClassName = "Physics", SchoolA = 61, SchoolB = 55, SchoolC = 65 });
    result.Add(new ClassScore() { ClassName = "English", SchoolA = 58, SchoolB = 56, SchoolC = 67 });
    result.Add(new ClassScore() { ClassName = "Math", SchoolA = 58, SchoolB = 51, SchoolC = 64 });
    result.Add(new ClassScore() { ClassName = "English", SchoolA = 63, SchoolB = 53, SchoolC = 45 });
    result.Add(new ClassScore() { ClassName = "English", SchoolA = 63, SchoolB = 50, SchoolC = 65 });
    result.Add(new ClassScore() { ClassName = "Math", SchoolA = 60, SchoolB = 45, SchoolC = 67 });
    result.Add(new ClassScore() { ClassName = "Math", SchoolA = 62, SchoolB = 53, SchoolC = 66 });
    result.Add(new ClassScore() { ClassName = "Physics", SchoolA = 63, SchoolB = 54, SchoolC = 64 });
    result.Add(new ClassScore() { ClassName = "English", SchoolA = 63, SchoolB = 52, SchoolC = 67 });
    result.Add(new ClassScore() { ClassName = "Physics", SchoolA = 69, SchoolB = 66, SchoolC = 71 });
    result.Add(new ClassScore() { ClassName = "Physics", SchoolA = 48, SchoolB = 67, SchoolC = 50 });
    result.Add(new ClassScore() { ClassName = "Physics", SchoolA = 54, SchoolB = 50, SchoolC = 56 });
    result.Add(new ClassScore() { ClassName = "Physics", SchoolA = 60, SchoolB = 56, SchoolC = 64 });
    result.Add(new ClassScore() { ClassName = "Math", SchoolA = 71, SchoolB = 65, SchoolC = 50 });
    result.Add(new ClassScore() { ClassName = "Math", SchoolA = 48, SchoolB = 70, SchoolC = 72 });
    result.Add(new ClassScore() { ClassName = "Math", SchoolA = 53, SchoolB = 40, SchoolC = 80 });
    result.Add(new ClassScore() { ClassName = "Math", SchoolA = 60, SchoolB = 56, SchoolC = 67 });
    result.Add(new ClassScore() { ClassName = "Math", SchoolA = 61, SchoolB = 56, SchoolC = 45 });
    result.Add(new ClassScore() { ClassName = "English", SchoolA = 63, SchoolB = 58, SchoolC = 64 });
    result.Add(new ClassScore() { ClassName = "Physics", SchoolA = 59, SchoolB = 54, SchoolC = 65 });

    return result;
}

public class ClassScore
{
    public string ClassName { get; set; }
    public double SchoolA { get; set; }
    public double SchoolB { get; set; }
    public double SchoolC { get; set; }
}

public class Y_AxisBinding
{
    public double d { get; set; }
    public string str { get; set; }

    // public int d { get; set; }

}
```

Error Bar

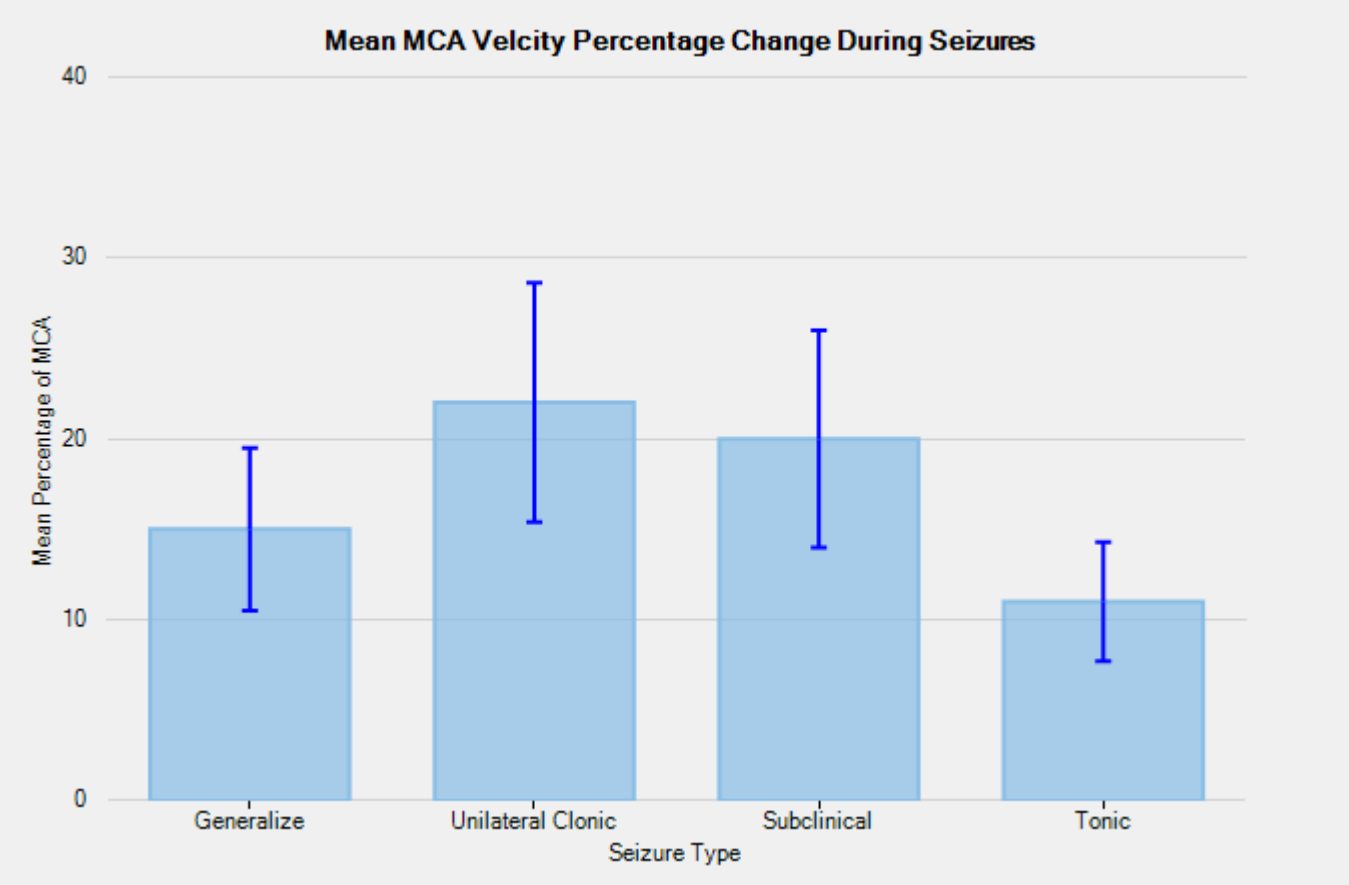
Error Bar series allows you to indicate variability of data or uncertainty in values. It enables you to display standard deviations and a range of error in variable data using error bars. Generally, results of scientific studies or experimental sciences use error bars in charts to depict variations in data from original values.

FlexChart lets you use Error Bar series in different chart types including Area, Column, Line, LineSymbols, Scatter, Spline, SplineArea, and SplineSymbols.

Error Bar series in FlexChart offers several features, as follows:

- **Error amount**: Set up error bars on all data points using different ways, such as a fixed value, percentage, standard error, or standard deviation. In addition, it is possible to set a custom value to show a precise error amount, if required. To display error bars in any of these ways, set the ErrorAmount property from the ErrorAmount enumeration.
- **Direction**: Show error bars in the Plus, the Minus, or even both directions by setting the Direction property from the ErrorBarDirection enumeration.
- **End style**: Display error bars with or without caps by setting the EndStyle property from the ErrorBarEndStyle enumeration.
- **Bar style**: Customize the appearance of error bars using the ErrorBarStyle property.

The following image displays Plus and Minus error amounts in the mean MCA (Middle Cerebral Artery) velocity data for different seizure types observed in children.



The following code uses mean percentage values of MCA velocity during different kinds of seizures in children. The codes shows how to implement ErrorBar series in FlexChart.

- **Visual Basic**

```
' create a datatable
```

```
Dim dt As New DataTable()

' add columns to the datatable
dt.Columns.Add("Seizure Type", GetType(String))
dt.Columns.Add("Mean MCA", GetType(Integer))

' add rows to the datatable
dt.Rows.Add("Generalize", 15)
dt.Rows.Add("Unilateral Clonic", 22)
dt.Rows.Add("Subclinical", 20)
dt.Rows.Add("Tonic", 11)

' clear data series collection
flexChart1.Series.Clear()

' create ErrorBar series
Dim errorBar As New C1.Win.Chart.ErrorBar()

' add the series to the data series collection
flexChart1.Series.Add(errorBar)

' specify the datasource for the chart
flexChart1.DataSource = dt

' bind X-axis and Y-axis
flexChart1.BindingX = "Seizure Type"
errorBar.Binding = "Mean MCA"

' specify error amount of the series
errorBar.ErrorAmount = C1.Chart.ErrorAmount.Percentage

' specify the direction of the error
errorBar.Direction = C1.Chart.ErrorBarDirection.Both

' specify the error value of the series
errorBar.ErrorValue = 0.3

' style the ErrorBar series
errorBar.EndStyle = C1.Chart.ErrorBarEndStyle.Cap
errorBar.ErrorBarStyle.StrokeWidth = 2
errorBar.ErrorBarStyle.StrokeColor = Color.Blue
```

- **C#**

```
// create a datatable
DataTable dt = new DataTable();

// add columns to the datatable
dt.Columns.Add("Seizure Type", typeof(string));
dt.Columns.Add("Mean MCA", typeof(int));

// add rows to the datatable
dt.Rows.Add("Generalize", 15);
dt.Rows.Add("Unilateral Clonic", 22);
dt.Rows.Add("Subclinical", 20);
dt.Rows.Add("Tonic", 11);

// clear data series collection
flexChart1.Series.Clear();

// create ErrorBar series
C1.Win.Chart.ErrorBar errorBar = new C1.Win.Chart.ErrorBar();
```

```
// add the series to the data series collection
flexChart1.Series.Add(errorBar);

// specify the datasource for the chart
flexChart1.DataSource = dt;

// bind X-axis and Y-axis
flexChart1.BindingX = "Seizure Type";
errorBar.Binding = "Mean MCA";

// specify error amount of the series
errorBar.ErrorAmount = C1.Chart.ErrorAmount.Percentage;

// specify the direction of the error
errorBar.Direction = C1.Chart.ErrorBarDirection.Both;

// specify the error value of the series
errorBar.ErrorValue = .3;

// style the ErrorBar series
errorBar.EndStyle = C1.Chart.ErrorBarEndStyle.Cap;
errorBar.ErrorBarStyle.StrokeWidth = 2;
errorBar.ErrorBarStyle.StrokeColor = Color.Blue;
```

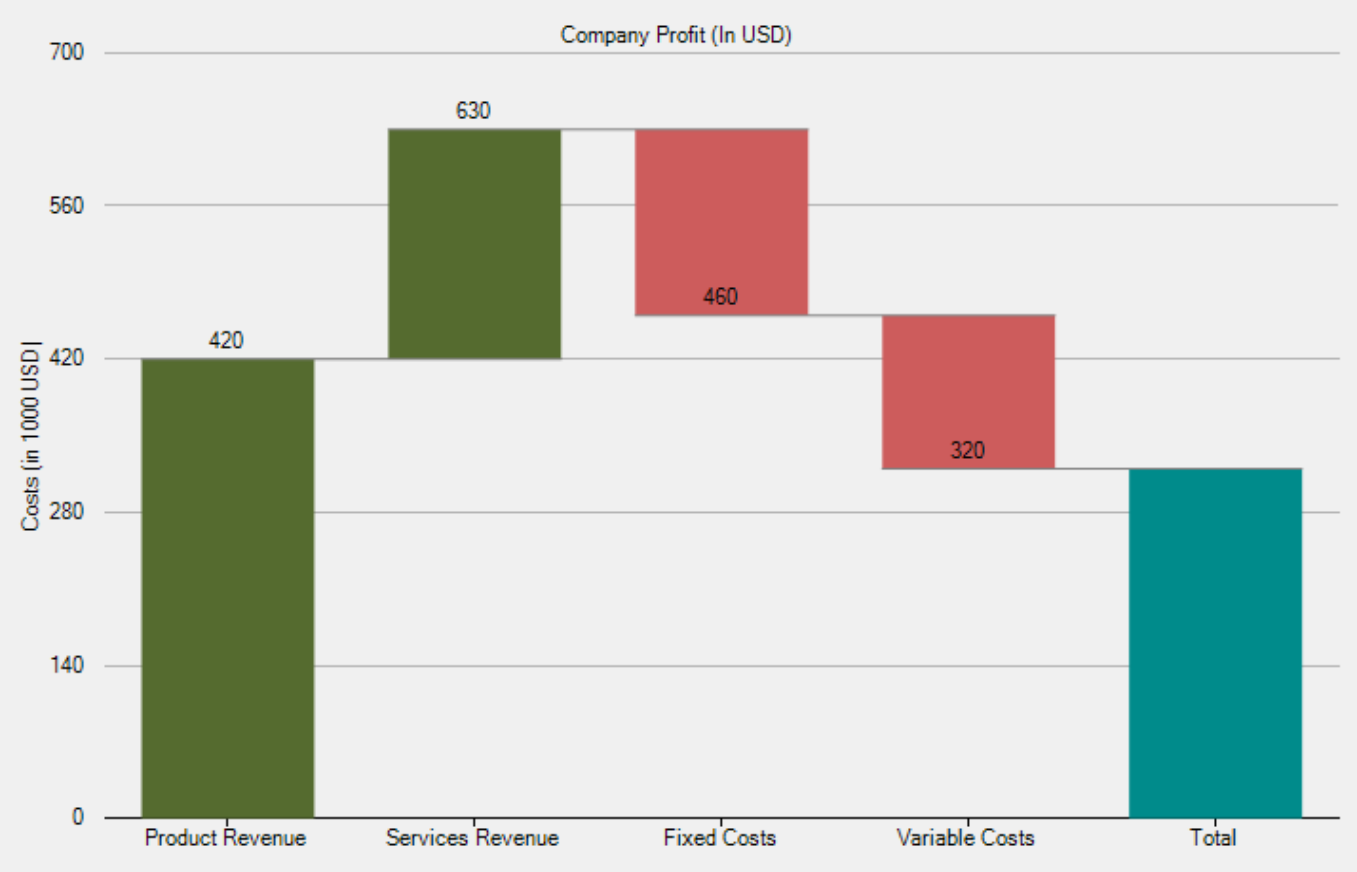
Waterfall Series

Waterfall series allows you to understand the cumulative effect of sequential positive or negative values. It is useful to understand the effect of a series of positive and negative values on an initial value. The series depicts color coded columns to easily distinguish positive values from negative values. Generally initial and final values are depicted by total columns, while intermediate values are represented by floating columns. It is recommended to use Waterfall series when there is a column of category text and a mix of positive and negative values. Such cases are mostly found in quantitative analysis like inventory analysis or performance analysis, where the chart shows the gradual transition in the quantitative value of an entity subjected to increment or decrement.

FlexChart provides features that can be implemented and customized for enhanced data visualization through Waterfall series.

- **Connector lines**: Connector lines are the lines that connect color coded columns to display the flow of data in the chart. To show connector lines, set the ConnectorLines property of the **WaterFall** class to True.
- **Connector lines customization**: Once displayed, connector lines can be customized using the ConnectorLineStyle property that allows you to access styling properties of the ChartStyle class.
- **Columns customization**: To clearly differentiate positive values from negative values or total, you can apply various styles to the columns showing these values. For that, you can use various properties, such as RisingStyle, FallingStyle, TotalStyle, or StartStyle provided by the Waterfall class.

The following image displays Waterfall series displaying the cumulative effect of sequential positive and negative values.



To work with Waterfall series in FlexChart, create an instance of the Waterfall class, which inherits the Series class. Add the created instance to the FlexChart Series collection accessible through the Series property of the FlexChart class.

The following code snippet illustrates how to set various properties while working with Waterfall series in FlexChart.

- **Visual Basic**

```
' create a datatable
Dim dt As New DataTable("Product Comparison")

' add columns to the datatable
dt.Columns.Add("Costs", GetType(String))
dt.Columns.Add("Amount", GetType(Integer))

' add rows to the datatable
dt.Rows.Add("Product Revenue", 420)
dt.Rows.Add("Services Revenue", 630)
dt.Rows.Add("Fixed Costs", 460)
dt.Rows.Add("Variable Costs", 320)

' clear data series collection
FlexChart1.Series.Clear()

' create a Waterfall series
Dim waterFall As New C1.Win.Chart.Waterfall()

' add the Waterfall series to the FlexChart series collection
FlexChart1.Series.Add(waterFall)

' specify the datasource for the FlexChart
FlexChart1.DataSource = dt
```

```
' bind the X-axis
FlexChart1.BindingX = "Costs"

' bind the Y axes
waterFall.Binding = "Amount"

' customize WaterFall connector lines
waterFall.ConnectorLines = True
waterFall.ConnectorLineStyle.StrokeWidth = 0.5F
waterFall.ConnectorLineStyle.StrokeColor = Color.Gray
waterFall.ConnectorLineStyle.Stroke = Brushes.Gray

' customize styles for rising, falling, and total values
waterFall.RisingStyle.FillColor = Color.DarkOliveGreen
waterFall.FallingStyle.FillColor = Color.IndianRed
waterFall.ShowTotal = True
waterFall.TotalStyle.FillColor = Color.DarkCyan
waterFall.TotalLabel = "Total"

' specify titles for FlexChart header and axes
FlexChart1.Header.Content = "Company Profit (In USD)"
FlexChart1.AxisY.Title = "Costs (in 1000 USD)"

' customize axes
FlexChart1.AxisY.MajorUnit = 140
FlexChart1.AxisY.Min = 0
FlexChart1.AxisY.Max = 700

' customize data labels
FlexChart1.DataLabel.Content = "{Amount}"
FlexChart1.DataLabel.Position = C1.Chart.LabelPosition.Top
```

- **C#**

```
// create a datatable
DataTable dt = new DataTable("Product Comparison");

// add columns to the datatable
dt.Columns.Add("Costs", typeof(string));
dt.Columns.Add("Amount", typeof(int));

// add rows to the datatable
dt.Rows.Add("Product Revenue", 420);
dt.Rows.Add("Services Revenue", 630);
dt.Rows.Add("Fixed Costs", 460);
dt.Rows.Add("Variable Costs", 320);

// clear data series collection
flexChart1.Series.Clear();

// create a Waterfall series
C1.Win.Chart.Waterfall waterFall = new C1.Win.Chart.Waterfall();

// add the Waterfall series to the FlexChart series collection
flexChart1.Series.Add(waterFall);

// specify the datasource for the FlexChart
flexChart1.DataSource = dt;

// bind the X-axis
flexChart1.BindingX = "Costs";
```

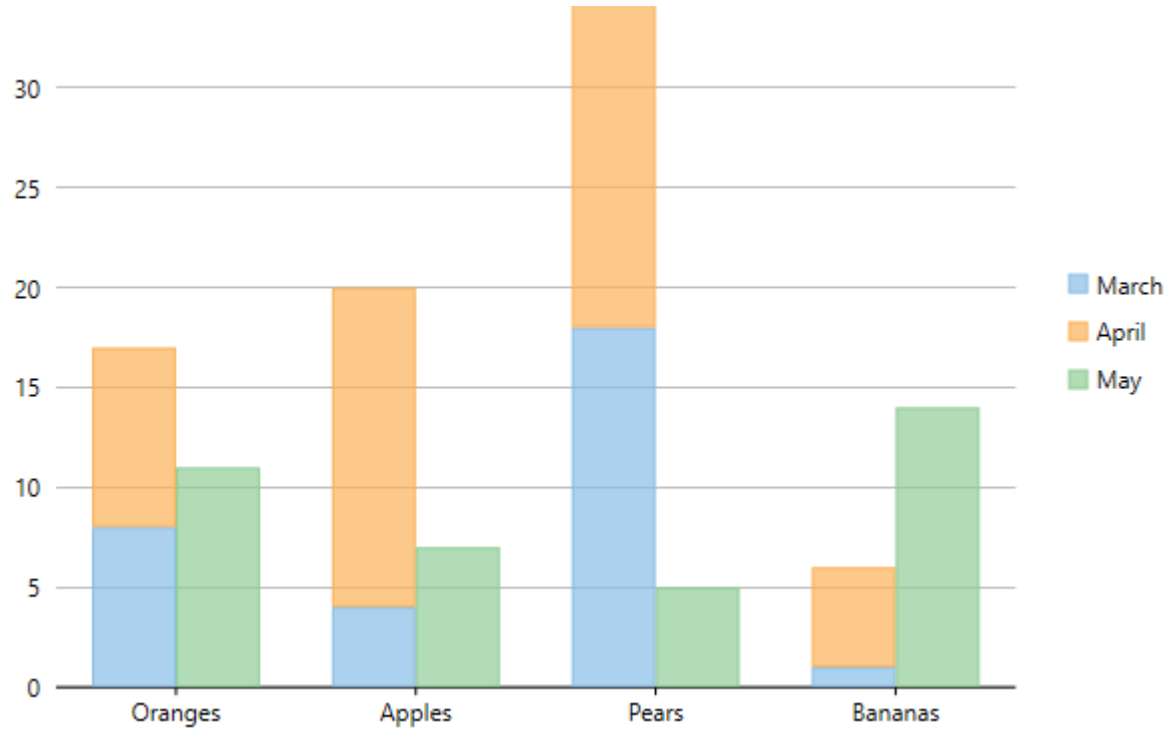
```
// bind the Y axes
waterFall.Binding = "Amount";

// customize WaterFall connector lines
waterFall.ConnectorLines = true;
waterFall.ConnectorLineStyle.StrokeWidth = 0.5F;
waterFall.ConnectorLineStyle.StrokeColor = Color.Gray;
waterFall.ConnectorLineStyle.Stroke = Brushes.Gray;

// customize styles for rising, falling, and total values
waterFall.RisingStyle.FillColor = Color.DarkOliveGreen;
waterFall.FallingStyle.FillColor = Color.IndianRed;
waterFall.ShowTotal = true;
waterFall.TotalStyle.FillColor = Color.DarkCyan;
waterFall.TotalLabel = "Total";

// specify titles for FlexChart header and axes
flexChart1.Header.Content = "Company Profit (In USD)";
flexChart1.AxisY.Title = "Costs (in 1000 USD)";

// customize axes
flexChart1.AxisY.MajorUnit = 140;
flexChart1.AxisY.Min = 0;
flexChart1.AxisY.Max = 700;

// customize data labels
flexChart1.DataLabel.Content = "{Amount}";
flexChart1.DataLabel.Position = C1.Chart.LabelPosition.Top;
```

Stacked Groups

FlexChart supports stacking and grouping of data items in column and bar charts. Stacking provides capabilities for stacking data items one on top of the other (in column chart) or side-by-side (in bar chart). Whereas, grouping enables clustering of the stacked data items in bar and column charts.

Stacked groups allow you to compare items across categories in a group. In addition, you can visualize relative difference between items in each group.

The following image displays stacked groups in FlexChart.



To stack specific series in a specific stacked group, set the index value of that stacked group in the StackingGroup property for the series. Note that Stacked groups in FlexChart are implementable when the Stacking property for FlexChart is set to either **Stacked** or **Stacked100pc**, which specifies how the data values of chart will be stacked.

The following code compares fruit data for three consecutive months and shows how to implement stacked groups in FlexChart.

- **Visual Basic**

```
FlexChart1.Series.Clear()

' Add three data series
Dim s1 = New Series()
s1.Binding = "March"
s1.Name = "March"
FlexChart1.Series.Add(s1)

Dim s2 = New Series()
s2.Binding = "April"
s2.Name = "April"
FlexChart1.Series.Add(s2)

Dim s3 = New Series()
s3.Binding = "May"
s3.Name = "May"
FlexChart1.Series.Add(s3)

' Set x-binding and add data to the chart
FlexChart1.BindingX = "Fruit"
FlexChart1.DataSource = DataCreator.CreateFruit()

' set FlexChart stacking type
FlexChart1.Stacking = C1.Chart.Stacking.Stacked

' specify stacking group for each series
FlexChart1.Series(0).StackingGroup = 0
FlexChart1.Series(1).StackingGroup = 0
```

```
FlexChart1.Series(2).StackingGroup = 1
```

- **C#**

```
flexChart1.Series.Clear();

// Add three data series
var s1 = new Series();
s1.Binding = s1.Name = "March";
flexChart1.Series.Add(s1);

var s2 = new Series();
s2.Binding = s2.Name = "April";
flexChart1.Series.Add(s2);

var s3 = new Series();
s3.Binding = s3.Name = "May";
flexChart1.Series.Add(s3);

// Set x-binding and add data to the chart
flexChart1.BindingX = "Fruit";
flexChart1.DataSource = DataCreator.CreateFruit();

// set FlexChart stacking type
flexChart1.Stacking = C1.Chart.Stacking.Stacked100pc;

// specify stacking group for each series
flexChart1.Series[0].StackingGroup = 0;
flexChart1.Series[1].StackingGroup = 0;
flexChart1.Series[2].StackingGroup = 1;
```

Data Labels

Data labels are the labels associated with data points to provide additional information about the data points. In other words, these labels can be defined as descriptive texts or values displayed over data points of the series. These labels are primarily used to highlight crucial data points, thereby enhancing the readability of the chart and allowing you to identify data quickly.

FlexChart offers support for highly customizable data labels that enable you to conveniently highlight chart data. And that in turn helps end-users to identify and interpret the chart data more efficiently. When it comes to working with data labels in FlexChart, the DataLabel property allows you to do so. By default, FlexChart does not display data labels; however, you can not only display data labels, but also customize them as per your requirements using various properties of the DataLabel and the DataLabelBase classes.

Below are the sections that describe how you can add data labels to data points and how you can control data labels in terms of their appearance and the data they display:

- Adding and Positioning Data Labels
- Formatting Data Labels

Adding and Positioning Data Labels

When added to data points in the chart, data labels make it easier to understand the chart data because they display

details about individual data points. These labels quickly highlight data that is both relevant and important.

There is a simple method to add data labels to data points while working with FlexChart. You just need to configure the Content property as per the type of entry you want to display in data labels. And you need to set the position of data labels using the Position property to display the data labels in the chart.

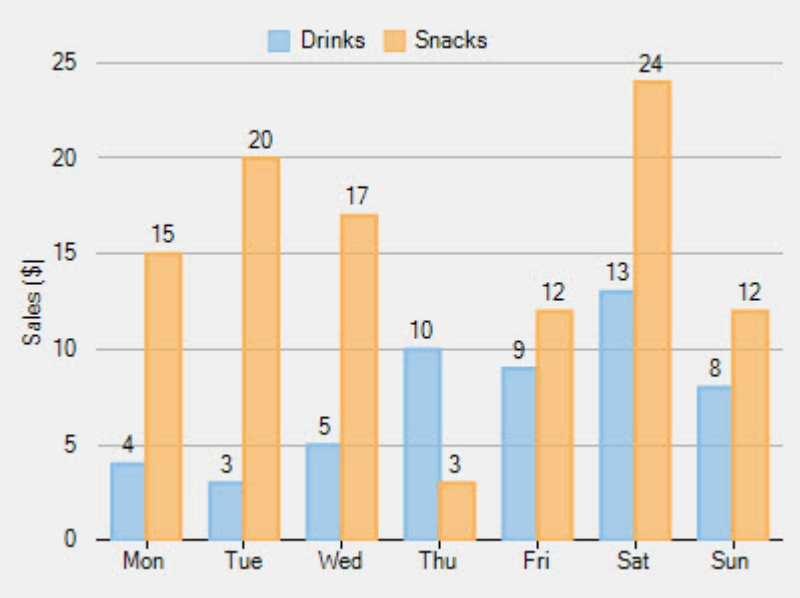
The table below lists the pre-defined parameters applicable for data label content customization.

Parameter	Description
x	Shows the X value of the data point.
y	Shows the Y value of the data point.
value	Shows the Y value of the data point.
name	Shows the X value of the data point.
seriesName	Shows the name of the series.
pointIndex	Shows the index of the data point.
P	Shows the percentage share with respect to the parent slice in Sunburst.
p	Shows the percentage share with respect to the whole chart in Sunburst.

See the following code snippet for reference.

- **C#**

```
// configure Content property
flexChart1.DataLabel.Content = "{value}";

// set the position of data labels
flexChart1.DataLabel.Position = C1.Chart.LabelPosition.Top;
```



Depending upon the chart type, you can select from different positioning options to position data labels perfectly in the chart. The Position property accepts the following values from the LabelPosition enumeration:

Property	Description
Top	Sets the labels above the data points.

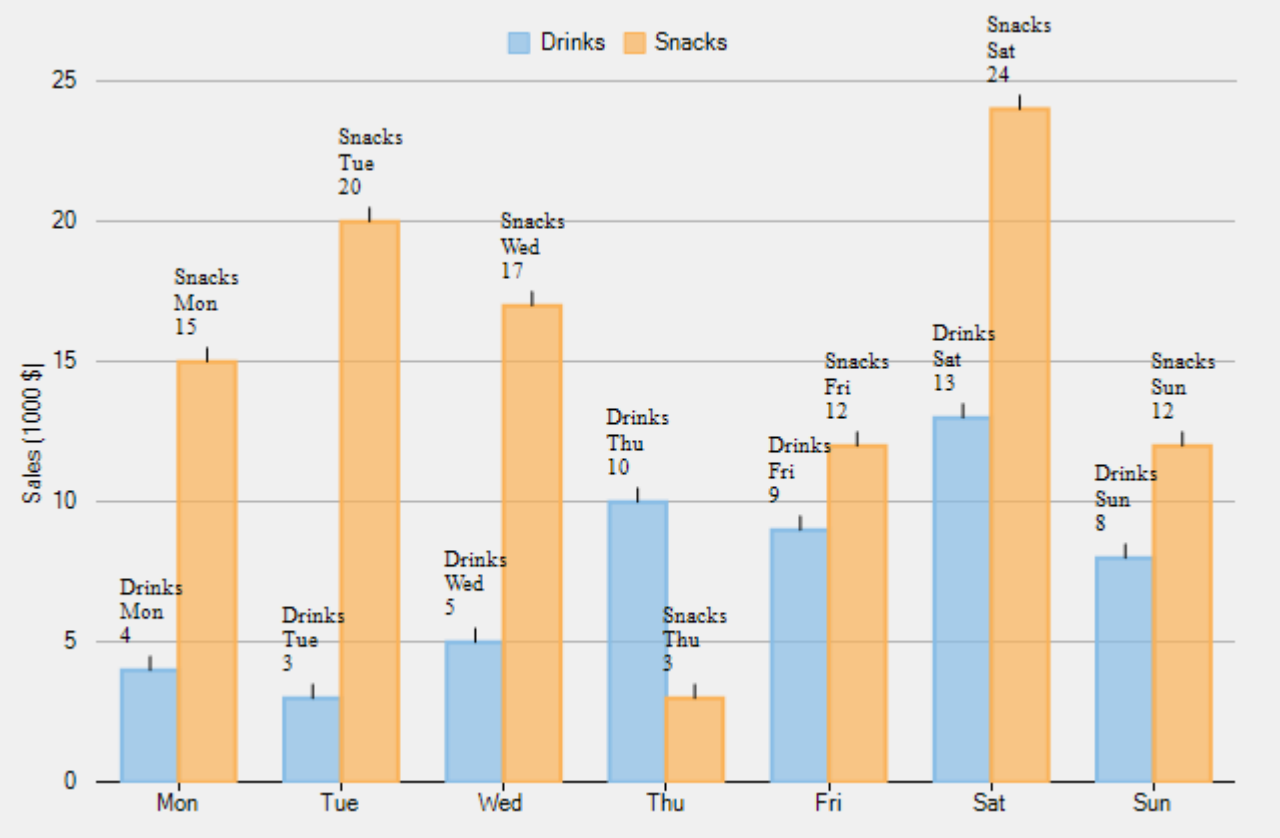
Bottom	Sets the labels below the data points.
Left	Sets the labels to the left of the data points.
Right	Sets the labels to the right of the data points.
Center	Sets the labels centered on the data points.
None	Hides the labels.

Using the Content property, you can customize the content of the data labels to further include series names, index values, or names of data points.

The following code shows how to set the Content property to include series names and data point values in data labels.

- **C#**

```
// set the Content property
flexChart1.DataLabel.Content = "{seriesName}\n{Days}\n{value}";
```



Formatting Data Labels

FlexChart provides a number of options to format data labels the way you want. You can set and style borders of data labels, connect them with their corresponding data points, and customize the way data labels appear.

Setting and Styling Borders of Data Labels

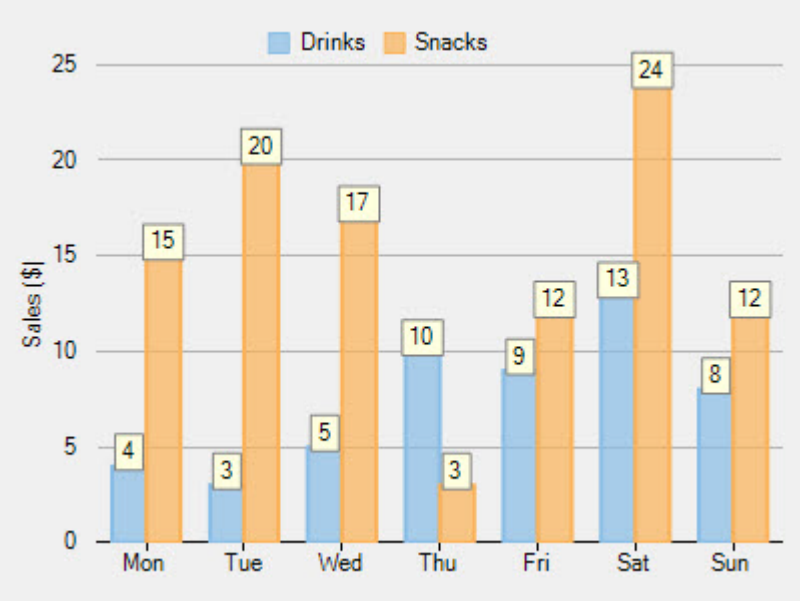
Borders add an extra appeal to data labels and make them more highlighted. This comes in handy to seamlessly highlight really crucial data in the chart, so that end users can focus on what is important.

In FlexChart, borders can be enabled and customized by using different properties, such as Border and BorderStyle.

Here is the code snippet illustrating the setting and the customization of borders.

- **C#**

```
// enable border
flexChart1.DataLabel.Border = true;

// set fill color
flexChart1.DataLabel.BorderStyle.FillColor = System.Drawing.Color.LightYellow;

// set stroke color
flexChart1.DataLabel.BorderStyle.StrokeColor = System.Drawing.Color.Gray;
```



Connecting Data Labels to Data Points

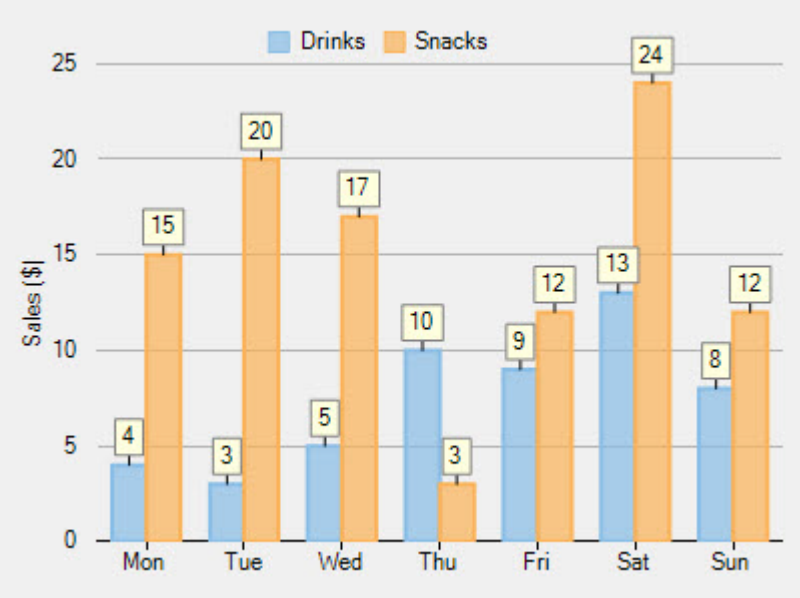
If you have placed data labels away from their corresponding data points, you can connect them using leader lines. A leader line is a line that connects a data label to its data point. Leader lines are beneficial to use, especially when you need to display a visual connection between data labels and their associated data points.

FlexChart does not display leader lines by default when you add data labels; nevertheless, you can enable leader lines and even set their appropriate length to create better visual connections for data labels in the chart. To enable leader lines, you need to use the ConnectingLine property. And to set the distance between data labels and their data points, you need to use the Offset property.

The following code snippet sets both the properties.

- **C#**

```
// enable leader line
flexChart1.DataLabel.ConnectingLine = true;

// set length of leader line
flexChart1.DataLabel.Offset = 7;
```



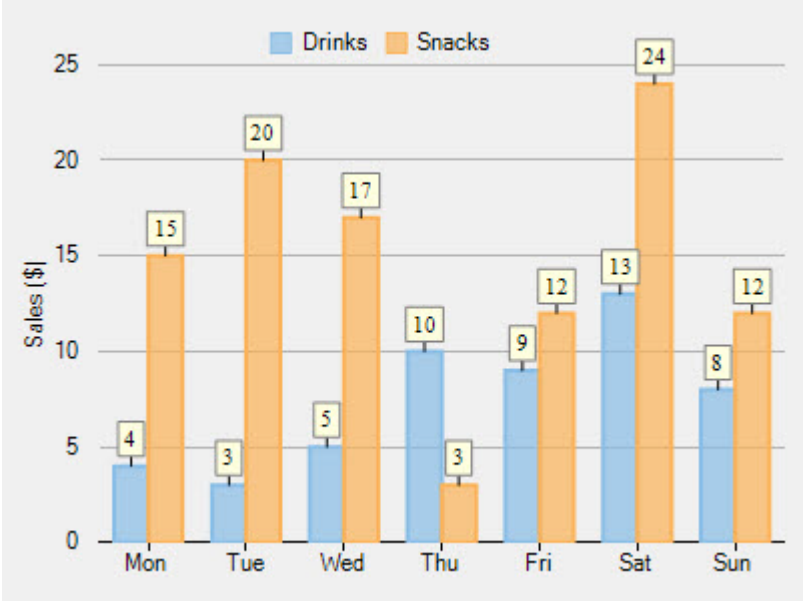
Changing the Appearance of Data Labels

You can make data visualization powerful and appealing by changing the way data labels appear in the chart. FlexChart contains various styling options, which you can use to enhance the clarity and look of data labels. You can use the Style property to change the appearance of data labels.

See the following code snippet for reference.

- **C#**

```
flexChart1.DataLabel.Style.Font = new System.Drawing.Font(FontFamily.GenericSerif, 8);
flexChart1.DataLabel.Style.StrokeWidth = 1;
```

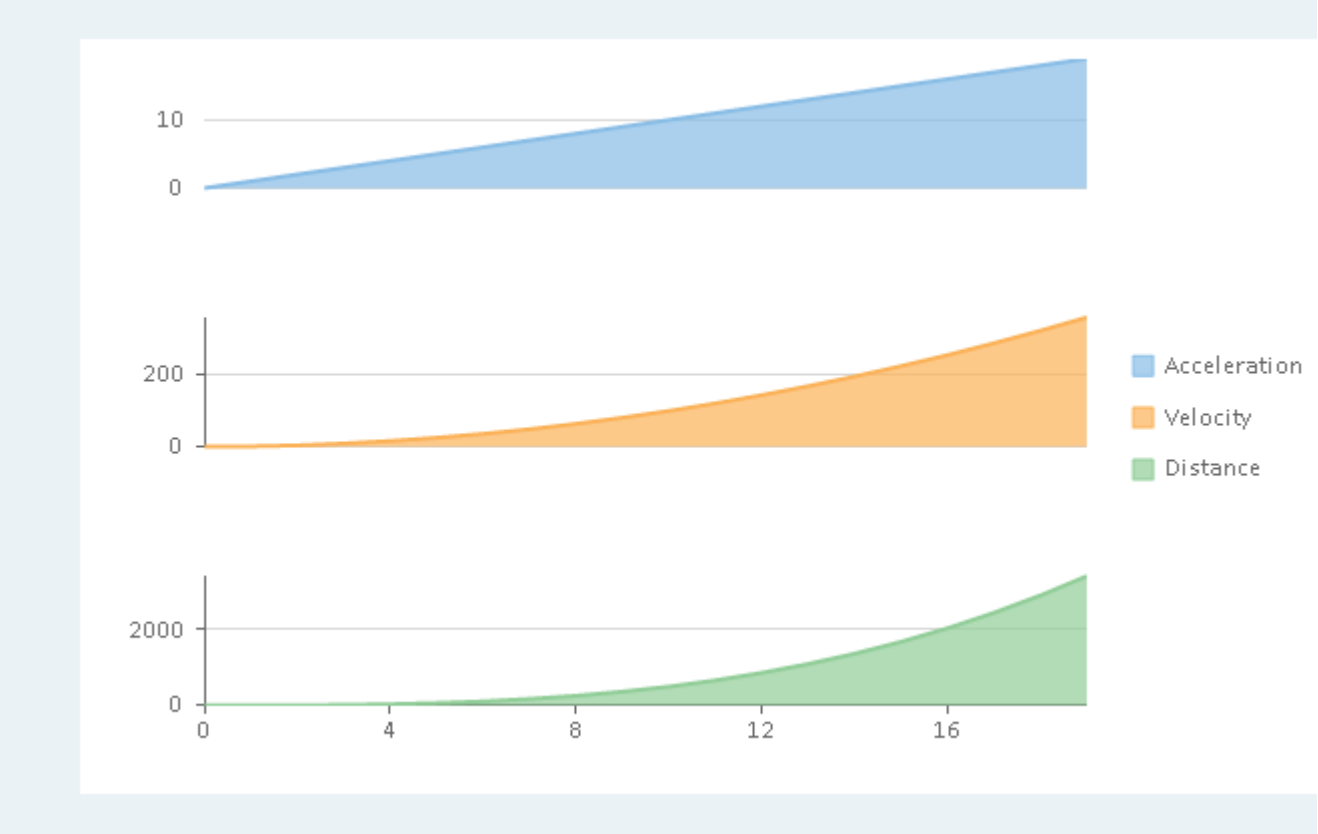


Multiple Plot Areas

Multiple plot areas allow you to increase the visibility of data by displaying each series in a separate plot area across one axis, keeping the other axis fixed.

FlexChart enables you to create multiple plot areas for different series within the same chart area. In FlexChart, create different plot areas and add them to the FlexChart.PlotAreas collection. In addition, you can customize the plot areas in terms of row index, column index, height, and width.

The following image displays multiple plot areas showing data for one series each in FlexChart.



The following code uses data regarding four metrics, namely, Acceleration, Velocity, Distance, and Time of a vehicle. The code demonstrates how to implement multiple plot areas in FlexChart.

- **Visual Basic**

```
' create and add multiple plot areas
FlexChart1.PlotAreas.Add(New PlotArea() With {
    .Name = "plot1",
    .Row = 0
})
FlexChart1.PlotAreas.Add(New PlotArea() With {
    .Name = "plot2",
    .Row = 2
})
FlexChart1.PlotAreas.Add(New PlotArea() With {
    .Name = "plot3",
    .Row = 4
})

' specify the chart type
FlexChart1.ChartType = C1.Chart.ChartType.Area

' create, add, and bind series
FlexChart1.Series.Add(New Series() With {
    .Name = "Acceleration",
    .Binding = "Acceleration"
})
```

```
FlexChart1.Series.Add(New Series() With {
    .Name = "Velocity",
    .Binding = "Velocity",
    .AxisY = New Axis() With {
        .Position = C1.Chart.Position.Left,
        .MajorGrid = True,
        .PlotAreaName = "plot2"
    }
})

FlexChart1.Series.Add(New Series() With {
    .Name = "Distance",
    .Binding = "Distance",
    .AxisY = New Axis() With {
        .Position = C1.Chart.Position.Left,
        .MajorGrid = True,
        .PlotAreaName = "plot3"
    }
})
```

- **C#**

```
// create and add multiple plot areas
flexChart1.PlotAreas.Add(new PlotArea { Name = "plot1", Row = 0 });
flexChart1.PlotAreas.Add(new PlotArea { Name = "plot2", Row = 2 });
flexChart1.PlotAreas.Add(new PlotArea { Name = "plot3", Row = 4 });

// specify the chart type
flexChart1.ChartType = C1.Chart.ChartType.Area;

// create, add, and bind series
flexChart1.Series.Add(new Series()
{
    Name = "Acceleration",
    Binding = "Acceleration",

});

flexChart1.Series.Add(new Series()
{
    Name = "Velocity",
    Binding = "Velocity",
    AxisY = new Axis()
    {
        Position = C1.Chart.Position.Left,
        MajorGrid = true,
        PlotAreaName = "plot2"
    },

});

flexChart1.Series.Add(new Series()
{
    Name = "Distance",
    Binding = "Distance",
    AxisY = new Axis()
    {
        Position = C1.Chart.Position.Left,
        MajorGrid = true,
        PlotAreaName = "plot3"
    }
```

```
});
```

Export

FlexChart supports exporting your charts as image file, so that they can be reused or shared. This can be easily accomplished with few lines of code.

To learn how to export a chart to various image formats, refer to the following topic:

- Export to Image

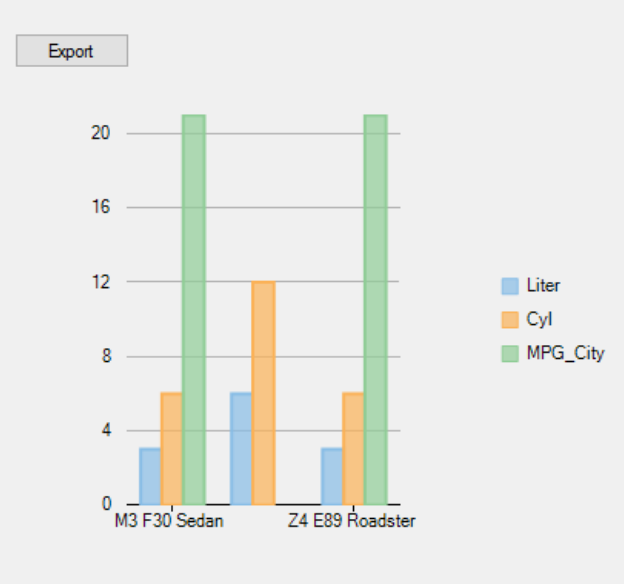
Export to Image

FlexChart for WinForms allows you to export the chart to multiple image formats. The supported formats are **PNG**, **JPEG**, and **SVG**.

To export a FlexChart to an image format, use SaveImage method. The method saves the chart as an image to the specified stream in the given ImageFormat. You can optionally specify the height, width, and back color of the image to be saved.

This topic uses the sample created in Quick Start topic to explain the implementation for exporting a FlexChart to an image on button click event.

The following image shows a chart with a button to be clicked to export chart to a desired image format.



To export a chart to image on button click use the following code.

- **Visual Basic**

```
Private Sub button1_Click(sender As Object, e As EventArgs)
    Dim dialog = New SaveFileDialog() With {
            Key.Filter = "PNG|*.png|JPEG |*.jpeg|SVG|*.svg"
    }
    If dialog.ShowDialog() = DialogResult.OK Then
        Using stream As Stream = dialog.OpenFile()
            Dim extension = dialog.FileName.Split("."c)(1)
            Dim fmt As ImageFormat = DirectCast([Enum].Parse(GetType(ImageFormat), extension, True), ImageFormat)
            FlexChart1.SaveImage(stream, fmt, 500, 800)
        End Using
    End If
End Sub
```

- **C#**

```
private void button1_Click(object sender, EventArgs e)
{
    var dialog = new SaveFileDialog()
    {
        Filter = "PNG|*.png|JPEG |*.jpeg|SVG|*.svg"
    };
    if (dialog.ShowDialog() == DialogResult.OK)
    {
```

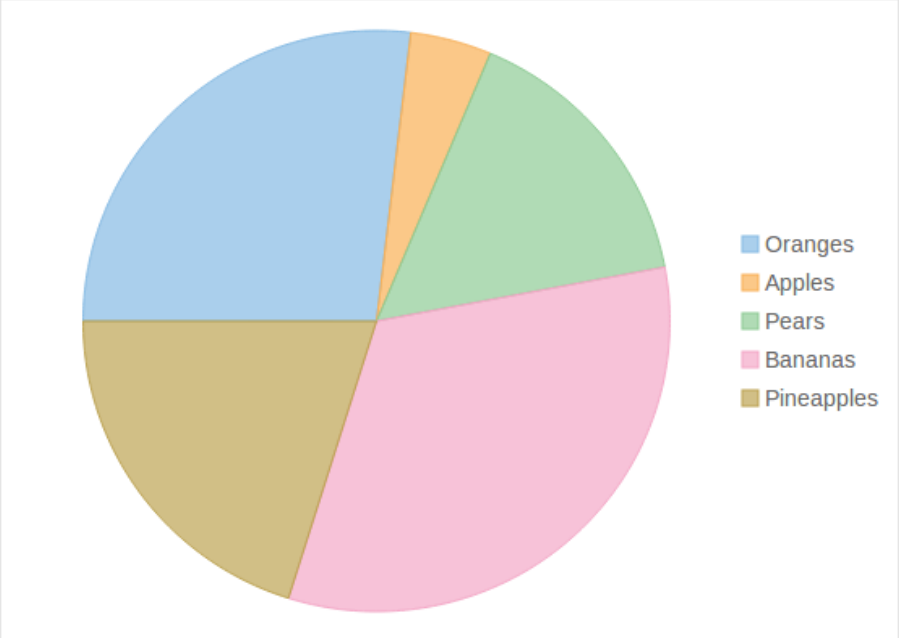
```
        using (Stream stream = dialog.OpenFile())
        {
            var extension = dialog.FileName.Split('.')[1];
            ImageFormat fmt = (ImageFormat)Enum.Parse(typeof(ImageFormat), extension, true);
            flexChart1.SaveImage(stream, fmt, 500, 800);
        }
    }
}
```

FlexPie

Pie charts are generally used to represent size of items in a series as a percentage of the entire pie. Ideally, the Pie Chart is to be used when you want to plot only one series comprising non-zero and positive values. And the number of categories are not more than seven.

The FlexPie control enables you to create customized pie charts that depict data points as slices of a pie. The arc length of each slice represents the value of that specific slice.

The multi-colored slices make pie charts easy to understand, and usually the value represented by each slice is displayed with the help of labels.



Key Features

- **Header and Footer**: Use simple properties to set a title and footer text.
- **Legend**: Change position of the legend as needed.
- **Selection**: Change the selection mode and customize the selected pie slice appearance.
- **Exploded and Doughnut Pie Charts**: Use simple properties to convert it into an exploding pie chart or a donut pie chart.

Quick Start

This quick start is intended to guide you through a step-by-step process of creating a simple FlexPie application and running the same in Visual Studio.

Complete the following steps to see how FlexPie appears on running the application:

- **Step 1: Adding FlexPie to the Application**
- **Step 2: Binding FlexPie to a Data Source**
- **Step 3: Running the Application**

Step 1: Adding FlexPie to the Application

1. Create a **Windows Forms Application** in Visual Studio.
2. Drag and drop the **FlexPie** control to the Form.
   The following dll is automatically added to the application:
   C1.Win.FlexChart.4.dll

Step 2: Binding FlexPie to a Data Source

Add the following code in the **Form_Load** event.

- **Visual Basic**

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    ' bind the field containing labels for pie slices and legend
    FlexPie1.BindingName = "Name"

    ' bind the field containing numeric values for pie slices
    FlexPie1.Binding = "Value"

    ' bind FlexPie to the data source
    FlexPie1.DataSource = New [Object]() {New With {
    Key .Value = 18,
    Key .Name = "US"
    }, New With {
    Key .Value = 16,
    Key .Name = "UK"
    }, New With {
    Key .Value = 18,
    Key .Name = "China"
    }, New With {
    Key .Value = 16,
    Key .Name = "France"
    }, New With {
    Key .Value = 17,
    Key .Name = "Germany"
    }, New With {
    Key .Value = 15,
    Key .Name = "Italy"
    }}

    ' set the Legend position
    FlexPie1.Legend.Position = C1.Chart.Position.Right
End Sub
```

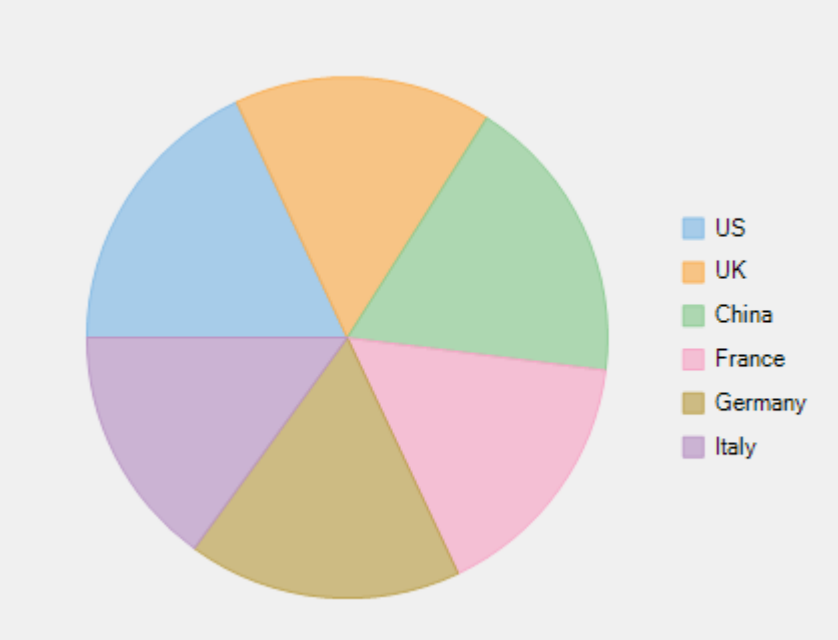
- **C#**

```
// bind the field containing labels for pie slices and legend
flexPie1.BindingName = "Name";

// bind the field containing numeric values for pie slices
flexPie1.Binding = "Value";

// bind FlexPie to the data source
```

```
flexPie1.DataSource = new Object[]
    {
      new {Value=18, Name="US"},
      new {Value=16, Name="UK"},
      new {Value=18, Name="China"},
      new {Value=16, Name="France"},
      new {Value=17, Name="Germany"},
      new {Value=15, Name="Italy"}
    };

// set the Legend position
flexPie1.Legend.Position = C1.Chart.Position.Right;
```

Step 3: Running the Application

Press F5 to run the application and observe the following output.



Doughnut Pie Chart

FlexPie allows you to create the doughnut Pie Chart by using the InnerRadius property.

The inner radius is measured as a fraction of the radius of the Pie Chart. The default value of the InnerRadius property is zero, which creates the Pie Chart. Setting this property to values greater than zero creates the Pie Chart with a hole in the middle, also known as the Doughnut Chart.

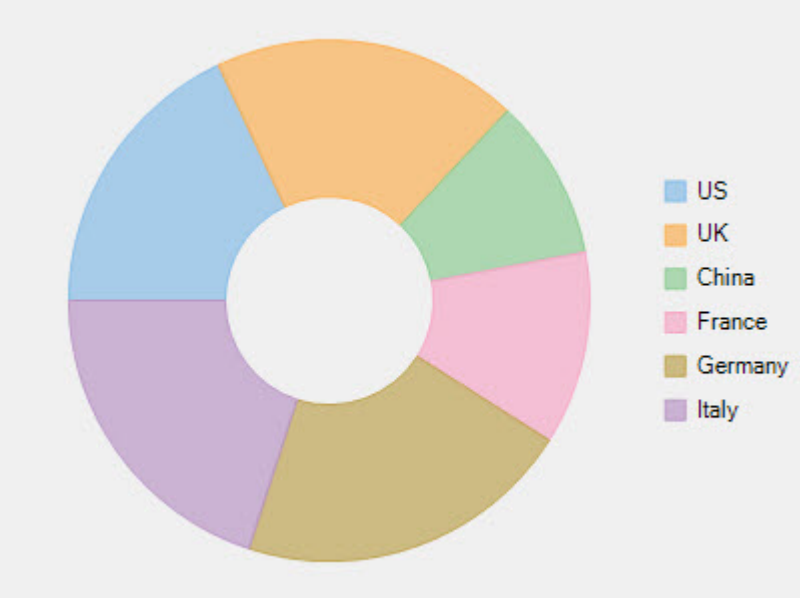
The following code snippets demonstrate how to set the InnerRadius property:

- **Visual Basic**

```
' set the InnerRadius property
flexPie1.InnerRadius = 0.4
```

- **C#**

```
// set the InnerRadius property
flexPie1.InnerRadius = 0.4D;
```



Exploded Pie Chart

The Offset property can be used to push the pie slices away from the center of FlexPie, producing the exploded Pie Chart. This property accepts a double value to determine how far the pie slices should be pushed from the center.

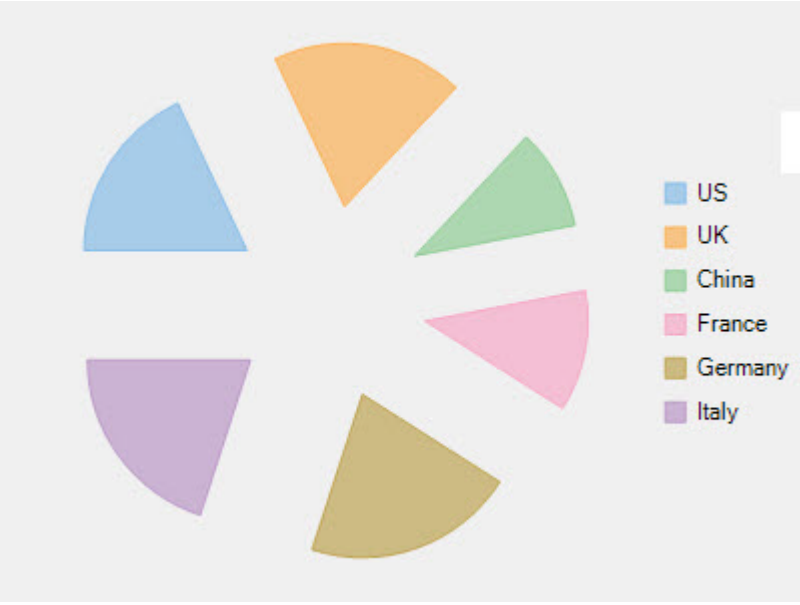
Here is the code snippet:

- **Visual Basic**

```
' set the Offset property
FlexPie1.Offset = 0.6
```

- **C#**

```
// set the Offset property
flexPie1.Offset = 0.6D;
```



Header and Footer

You can add a header to the FlexPie control by setting the Header property of FlexChartBase. Besides a header, you can add a footer to the control by setting the Footer property of FlexChartBase.

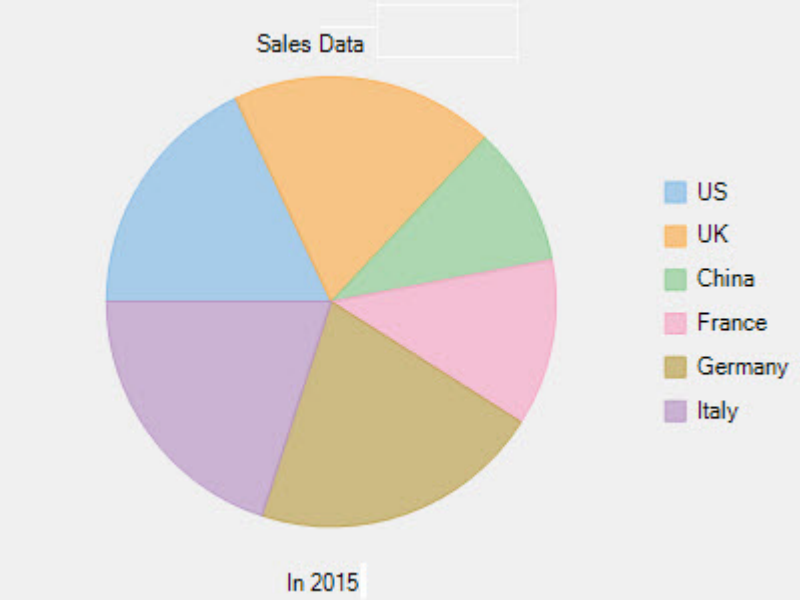
See the following code snippet for setting the two properties:

- **Visual Basic**

```
' set the Header
FlexPie1.Header.Content = "Sales Data"

' set the Footer
FlexPie1.Footer.Content = "In 2015"
```

- **C#**

```
// set the Header
flexPie1.Header.Content = "Sales Data";

// set the Footer
flexPie1.Footer.Content = "In 2015";
```



Legend

FlexPie enables you to specify the position where you want to display the Legend by using the Legend property of FlexChartBase.

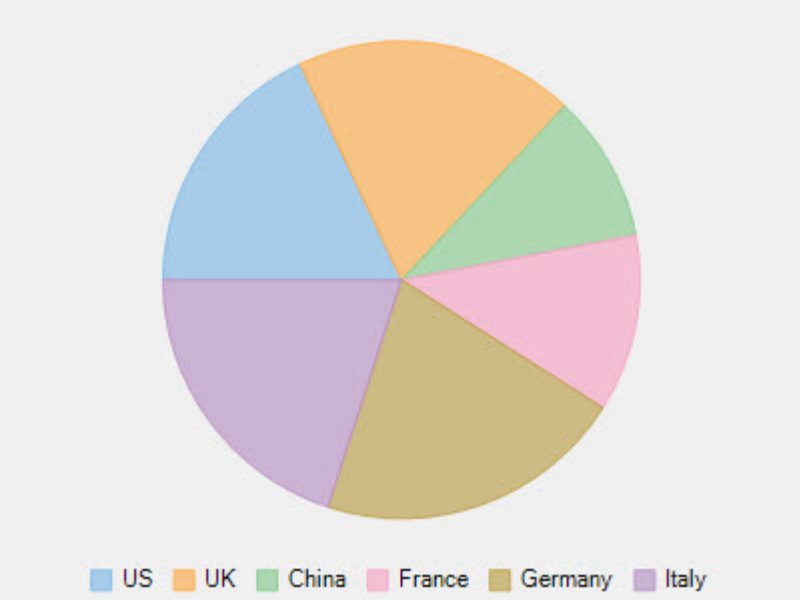
The below-mentioned code snippet shows how to set the property.

- **Visual Basic**

```
' set the Legend
FlexPie1.Legend.Position = C1.Chart.Position.Bottom
```

- **C#**

```
// set the Legend
flexPie1.Legend.Position = C1.Chart.Position.Bottom;
```



Selection

You can choose what element of FlexPie is selected when you click anywhere on the control by setting the SelectionMode property. This property provides three options:

- **None**: Does not select any element.
- **Point**: Highlights the pie slice that the user clicks.
- **Series**: Highlights the entire pie.

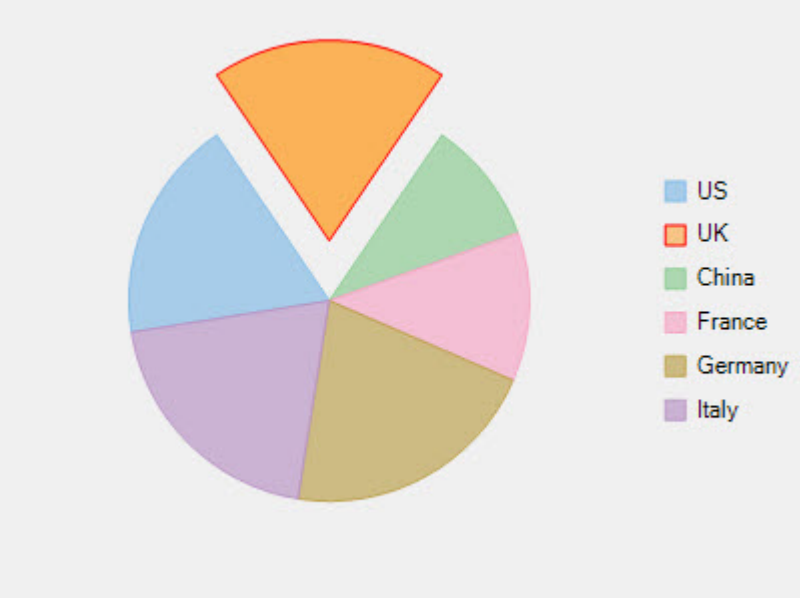
After setting the SelectionMode property to Point, you can change the position of the selected pie slice by setting the SelectedItemPosition property. And also, you can move the selected pie slice away from the center of FlexPie by setting the SelectedItemOffset property.

- **Visual Basic**

```
' set the SelectionMode property
FlexPie1.SelectionMode = C1.Chart.ChartSelectionMode.Point

' set the SelectedItemPosition property
FlexPie1.SelectedItemPosition = C1.Chart.Position.Top

' set the SelectedItemOffset property
FlexPie1.SelectedItemOffset = 0.3
```

- **C#**

```
// set the SelectionMode property
flexPie1.SelectionMode = C1.Chart.ChartSelectionMode.Point;

// set the SelectedItemPosition property
flexPie1.SelectedItemPosition = C1.Chart.Position.Top;

// set the SelectedItemOffset property
flexPie1.SelectedItemOffset = 0.3D;
```

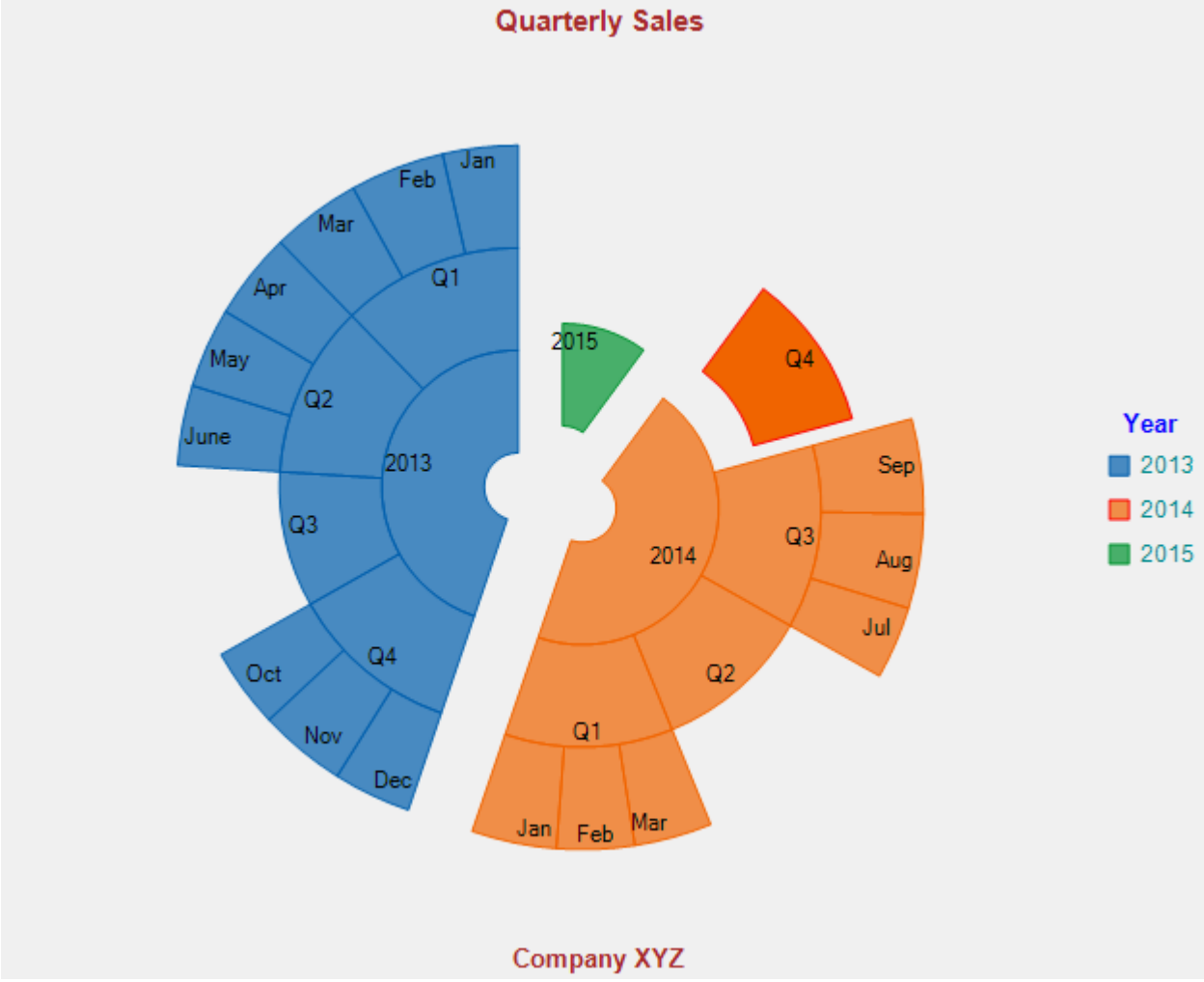


Sunburst Chart

Sunburst, also known as a multi-level pie chart, is ideal for visualizing multi-level hierarchical data depicted by concentric circles.The circle in the center represents the root node, with the data moving outside from the center. A section of the inner circle supports a hierarchical relationship to those sections of the outer circle which lie within the angular area of the parent section.

Using Sunburst chart helps the end user to visualize the relationship between outer rings and inner rings. For instance, you might want to display sales report for each quarter of three years. Using SunBurst chart, a sales report of a specific month can be highlighted, thereby depicting relationship with the respective quarter.

Both Sunburst chart and TreeMap chart are ideal for displaying and visualizing hierarchical data, but Sunburst chart is a better visualization tool to show hierarchical levels between the largest category in a data set and each respective data point as it paints a better picture of hierarchy. Whereas, Treemap is preferred when space is a constraint as it can show humongous data in a limited area.



To explore the features of the Sunburst chart, click the following links:

- Quick Start
- Key Features
- Legend and Titles
- Selection
- Drilldown

Key Features

The Sunburst control includes a number of features that make it efficient and useful for creating professional-looking applications.

- **Doughnut Sunburst chart**: Create a doughnut Sunburst chart by setting the InnerRadius property that has a default value of zero. Setting this property to a value greater than zero creates a hole in the middle, thereby creating the doughnut Sunburst chart.
- **Exploded Sunburst chart**: Create an exploded Sunburst chart by setting the Offset property that has a default value of zero. Setting the property pushes the slices away from the center of Sunburst chart, thereby producing the exploded Sunburst chart.
- **Header and Footer**: Use simple properties to set and customize Header and Footer of Sunburst chart. For more information, refer to Legend and Titles.
- **Legend**: Perform various customizations including setting orientation, position, or styling the legend of Sunburst chart. For more information, refer to Legend and Titles.
- **Palette**: Use different color palettes to make Sunburst chart more appealing and presentable. To specify the chart palette, set the Palette property that lets you specify an array of default colors to be used when rendering slices. The property accepts values from the Palette enumeration.
- **Reversed Sunburst chart**: Create a reversed Sunburst chart by setting the Reversed property that has False as the default value. Setting the property to True creates the reversed Sunburst chart that contains angles drawn in the counter-clockwise direction.
- **Start angle:** Set the start angle by setting the StartAngle property that accepts values of the double type. A start angle is the angle that is set in degrees to start drawing Sunburst slices in the clockwise direction. The default value is the 9 o'clock position.
- **Selection**: Change the selection mode and customize the position, appearance of the selected pie slice. For more information, refer to Selection.

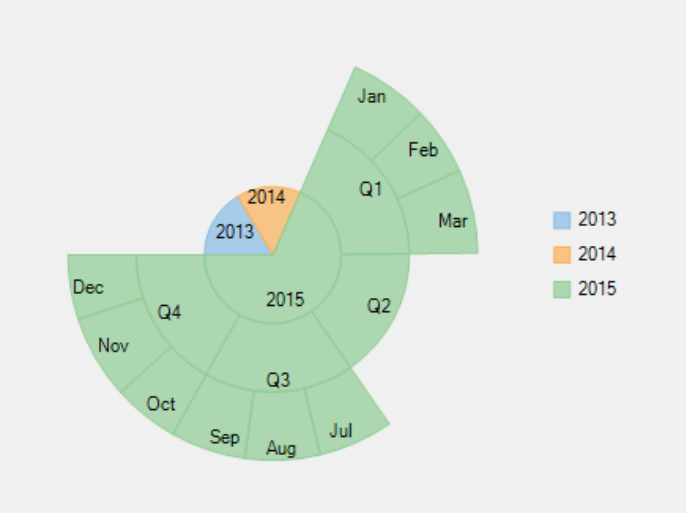
Quick Start

This quick start is intended to guide you through a step-by-step process of creating a simple Sunburst application and running the same in Visual Studio.

To quickly get started with Sunburst chart and observe how it appears on running the application, follow these steps:

1. **Add Sunburst Chart to the Application**
2. **Bind Sunburst Chart to a Data Source**
3. **Run the Application**

The following image displays how a basic Sunburst chart appears after completing the steps mentioned above.



Step 1: Add Sunburst Chart to the Application

1. Create a **Windows Forms Application** in Visual Studio.
2. Drag and drop the **Sunburst** control to the Form.
   The dlls which are automatically added to the application, are as follows:
   C1.Win.C1DX.4.dll
   C1.Win.C1Input.4.dll
   C1.Win.FlexChart.4.dll

Step 2: Bind Sunburst Chart to a Data Source

In this step, you first create a class DataService that generates random sales data for four quarters, namely Q1, Q2, Q3, and Q4 in 2013, 2014, and 2015. Next, you bind Sunburst chart to the created class using the DataSource property provided by the FlexPie class. Then, you specify numeric values and labels for the Sunburst slices using the Binding and the BindingName property, respectively of the FlexPie class.

1. Add a class, **DataService** and add the following code.
   - **Visual Basic**

```
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Threading.Tasks

Namespace SunburstQuickStart
    Class DataService
        Private rnd As New Random()
        Shared _default As DataService

        Public Shared ReadOnly Property Instance() As DataService
            Get
                If _default Is Nothing Then
                    _default = New DataService()
                End If

                Return _default
            End Get
        End Property
        Public Shared Function CreateFlatData() As List(Of FlatDataItem)
            Dim rnd As Random = Instance.rnd
            Dim years As New List(Of String)()
            Dim times As New List(Of List(Of String))() From {
                New List(Of String)() From {
                    "Jan",
                    "Feb",
                    "Mar"
                },
                New List(Of String)() From {
                    "Apr",
                    "May",
                    "June"
                },
                New List(Of String)() From {
                    "Jul",
                    "Aug",
                    "Sep"
                },
                New List(Of String)() From {
                    "Oct",
                    "Nov",
                    "Dec"
                }
            }

            Dim items As New List(Of FlatDataItem)()
            Dim yearLen = Math.Max(CInt(Math.Round(Math.Abs(5 - rnd.NextDouble() * 10))), 3)
            Dim currentYear As Integer = DateTime.Now.Year
            For i As Integer = yearLen To 1 Step -1
                years.Add((currentYear - i).ToString())
            Next
            Dim quarterAdded = False
            years.ForEach(Function(y)
                              Dim i = years.IndexOf(y)
                              Dim addQuarter = rnd.NextDouble() > 0.5
                              If Not quarterAdded AndAlso i = years.Count - 1 Then
                                  addQuarter = True
                              End If
                              If addQuarter Then
                                  quarterAdded = True
                                  times.ForEach(Function(q)
                                                    Dim addMonth = rnd.NextDouble() > 0.5
```

```
                                        Dim idx As Integer = times.IndexOf(q)
                                        Dim quar As String = "Q" + (idx + 1).ToString()
                                        If addMonth Then
                                            q.ForEach(Function(m)
                                                          items.Add(New FlatDataItem() With {
                                                              .Year = y,
                                                              .Quarter = quar,
                                                              .Month = m,
                                                              .Value = rnd.[Next](30, 40)
                                                          })

                                                      End Function)
                                        Else
                                            items.Add(New FlatDataItem() With {
                                               .Year = y,
                                               .Quarter = quar,
                                               .Value = rnd.[Next](80, 100)
                                            })
                                        End If

                                    End Function)
                                Else
                                    items.Add(New FlatDataItem() With {
                                     .Year = y.ToString(),
                                     .Value = rnd.[Next](80, 100)
                                    })
                                End If

                            End Function)

            Return items
        End Function
    End Class
End Namespace
```

- **C#**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SunburstQuickStart
{
    class DataService
    {
        Random rnd = new Random();
        static DataService _default;

        public static DataService Instance
        {
            get
            {
                if (_default == null)
                {
                    _default = new DataService();
                }

                return _default;
            }
        }
        public static List<FlatDataItem> CreateFlatData()
        {
            Random rnd = Instance.rnd;
            List<string> years = new List<string>();
            List<List<string>> times = new List<List<string>>()
            {
                new List<string>() { "Jan", "Feb", "Mar"},
                new List<string>() { "Apr", "May", "June"},
                new List<string>() { "Jul", "Aug", "Sep"},
                new List<string>() { "Oct", "Nov", "Dec" }
            };

            List<FlatDataItem> items = new List<FlatDataItem>();
            var yearLen = Math.Max((int)Math.Round(Math.Abs(5 - rnd.NextDouble() * 10)), 3);
            int currentYear = DateTime.Now.Year;
            for (int i = yearLen; i > 0; i--)
            {
                years.Add((currentYear - i).ToString());
```

```
            }
            var quarterAdded = false;
            years.ForEach(y =>
            {
                var i = years.IndexOf(y);
                var addQuarter = rnd.NextDouble() > 0.5;
                if (!quarterAdded && i == years.Count - 1)
                {
                    addQuarter = true;
                }
                if (addQuarter)
                {
                    quarterAdded = true;
                    times.ForEach(q =>
                    {
                        var addMonth = rnd.NextDouble() > 0.5;
                        int idx = times.IndexOf(q);
                        var quar = "Q" + (idx + 1);
                        if (addMonth)
                        {
                            q.ForEach(m =>
                            {
                                items.Add(new FlatDataItem()
                                {
                                    Year = y,
                                    Quarter = quar,
                                    Month = m,
                                    Value = rnd.Next(30, 40)
                                });
                            });
                        }
                        else
                        {
                            items.Add(new FlatDataItem()
                            {
                                Year = y,
                                Quarter = quar,
                                Value = rnd.Next(80, 100)
                            });
                        }
                    });
                }
                else
                {
                    items.Add(new FlatDataItem()
                    {
                        Year = y.ToString(),
                        Value = rnd.Next(80, 100)
                    });
                }
            });

            return items;
        }
    }
}
```

2. Switch to **Code View** (**Form1.cs**) and add the following code.
    - **Visual Basic**

```
Imports SunburstQuickStart.SunburstQuickStart

Partial Public Class Form1
    Inherits Form
    Public Sub New()
        InitializeComponent()

        ' specify the data source
        Sunburst1.DataSource = DataService.CreateFlatData()

        ' specify the field containing values for pie slices
        Sunburst1.Binding = "Value"

        ' specify the fields contaning labels for pie slices and legend
        Sunburst1.BindingName = "Year,Quarter,Month"

        ' set the data label content
        Sunburst1.DataLabel.Content = "{name}"

        ' set the data label position
```

```
        Sunburst1.DataLabel.Position = C1.Chart.PieLabelPosition.Inside
    End Sub
End Class

Public Class FlatDataItem
    Public Property Year() As String
        Get
            Return m_Year
        End Get
        Set
            m_Year = Value
        End Set
    End Property
    Private m_Year As String
    Public Property Quarter() As String
        Get
            Return m_Quarter
        End Get
        Set
            m_Quarter = Value
        End Set
    End Property
    Private m_Quarter As String
    Public Property Month() As String
        Get
            Return m_Month
        End Get
        Set
            m_Month = Value
        End Set
    End Property
    Private m_Month As String
    Public Property Value() As Double
        Get
            Return m_Value
        End Get
        Set
            m_Value = Value
        End Set
    End Property
    Private m_Value As Double
End Class
```

- **C#**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SunburstQuickStart
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            // specify the data source
            sunburst1.DataSource = DataService.CreateFlatData();

            // specify the field containing values for pie slices
            sunburst1.Binding = "Value";

            // specify the fields contaning labels for pie slices and legend
            sunburst1.BindingName = "Year,Quarter,Month";

            // set the data label content
            sunburst1.DataLabel.Content = "{name}";

            // set the data label position
            sunburst1.DataLabel.Position = C1.Chart.PieLabelPosition.Inside;
        }
    }
    public class FlatDataItem
```

```
        {
            public string Year { get; set; }
            public string Quarter { get; set; }
            public string Month { get; set; }
            public double Value { get; set; }
        }
    }
```

Step 3: Run the Application

Press F5 to run the application and observe how Sunburst chart appears.

Legend and Titles

Legend

The legend displays entries for series with their names and predefined symbols. In Sunburst, you can access the legend using the Legend property of the FlexChartBase class and perform various customizations, as follows:

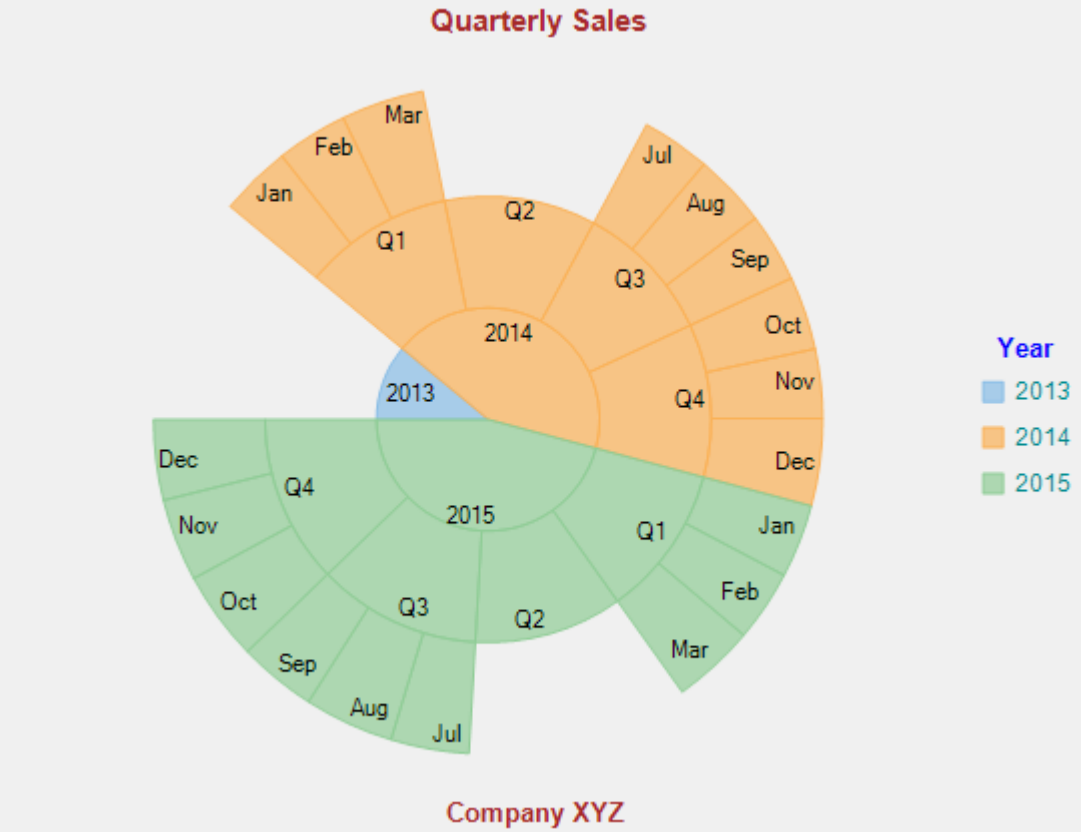
- **Orientation**: Set the orientation of the legend as horizontal, vertical, or automatic by using the Orientation property provided by the Legend class. The property can be set to any of the values from the Orientation enumeration.
- **Position**: Set the legend on top, bottom, left, right, or let it be positioned automatically by using the Position property that accepts values from the Position enumeration. Setting the Position property to None hides the legend.
- **Styling**: Customize the overall appearance of the legend, such as setting stroke color or changing font by using styling properties accessible through the Style property. The styling properties StrokeColor and Font are provided by the ChartStyle class.
- **Title and title styling**: Specify the legend title using the Title property that accepts a string. Once you have set the title, you can style it using the TitleStyle property that provides access to the customization properties of the ChartStyle class.

Header and Footer

Header and Footer are descriptive texts at the top and bottom of the chart that provide information about the overall chart data. You can access Header and Footer of Sunburst chart by using the Header and the Footer property respectively, of the FlexChartBase class. Possible customizations with Header and Footer are as follows:

- **Font**: Change the font family, font size, and font style of Header and Footer using the Font property of the ChartStyle class accessible through the Style property of the ChartTitle class.
- **Stroke color**: Select any stroke color of the titles using the StrokeColor property the ChartStyle class.
- **Stroke**: Set stroke of the titles for enhanced appeal by using the Stroke property.

The following image displays Sunburst chart with the legend and titles set.



The following code snippet illustrates how to set respective properties for the legend and titles customization. This code uses the sample created in Quick Start.

- **Visual Basic**

```
' set the legend orientation
Sunburst1.Legend.Orientation = C1.Chart.Orientation.Vertical

' set the legend position
Sunburst1.Legend.Position = C1.Chart.Position.Auto

' set the legend stroke color
Sunburst1.Legend.Style.StrokeColor = Color.DarkCyan

' Set the legend font
Sunburst1.Legend.Style.Font = New Font("Arial", 9, FontStyle.Regular)

' set the legend title
Sunburst1.Legend.Title = "Year"

' set the legend title stroke color
Sunburst1.Legend.TitleStyle.StrokeColor = Color.Blue

' set the legend title font
Sunburst1.Legend.TitleStyle.Font = New Font("Arial", 10, FontStyle.Bold)

' set the header
Sunburst1.Header.Content = "Quarterly Sales"

' set the header font
Sunburst1.Header.Style.Font = New Font("Arial", 11, FontStyle.Bold)

' set the header stroke
```

```
Sunburst1.Header.Style.Stroke = Brushes.Brown

' set the footer
Sunburst1.Footer.Content = "Company XYZ"

' set the footer font
Sunburst1.Footer.Style.Font = New Font("Arial", 10, FontStyle.Bold)

' set the footer stroke
Sunburst1.Footer.Style.Stroke = Brushes.Brown
```

- **C#**

```
// set the legend orientation
sunburst1.Legend.Orientation = C1.Chart.Orientation.Vertical;

// set the legend position
sunburst1.Legend.Position = C1.Chart.Position.Auto;

// set the legend stroke color
sunburst1.Legend.Style.StrokeColor = Color.DarkCyan;

// Set the legend font
sunburst1.Legend.Style.Font = new Font("Arial", 9, FontStyle.Regular);

// set the legend title
sunburst1.Legend.Title = "Year";

// set the legend title stroke color
sunburst1.Legend.TitleStyle.StrokeColor = Color.Blue;

// set the legend title font
sunburst1.Legend.TitleStyle.Font = new Font("Arial", 10, FontStyle.Bold);

// set the header
sunburst1.Header.Content = "Quarterly Sales";

// set the header font
sunburst1.Header.Style.Font = new Font("Arial", 11, FontStyle.Bold);

// set the header stroke
sunburst1.Header.Style.Stroke = Brushes.Brown;

// set the footer
sunburst1.Footer.Content = "Company XYZ";

// set the footer font
sunburst1.Footer.Style.Font = new Font("Arial", 10, FontStyle.Bold);

// set the footer stroke
sunburst1.Footer.Style.Stroke = Brushes.Brown;
```

Selection

The Sunburst chart lets you select data points by clicking a Sunburst slice. You can set the SelectionMode property provided by the FlexChartBase class to either of the following values in the ChartSelectionMode enumeration:

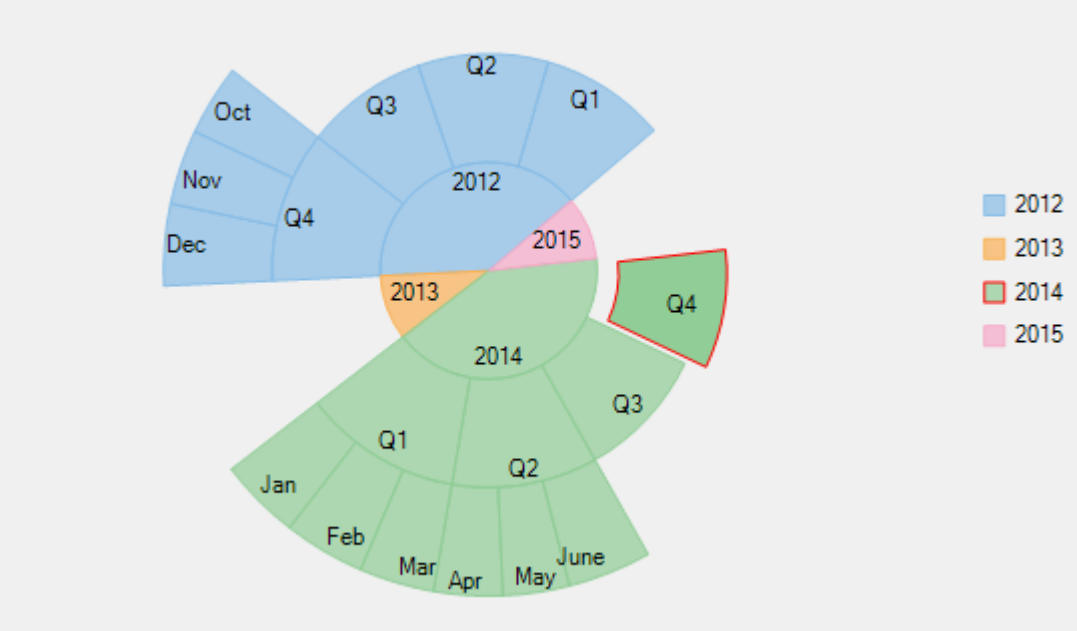
- **None (default)**: Selection is disabled.

- **Point**: A point is selected.

To customize the selection, you can use the SelectedItemOffset and the SelectedItemPosition property provided by the FlexPie class. The SelectedItemOffset property enables you to specify the offset of the selected Sunburst slice from the center of the control. And the SelectedItemPosition property allows you to specify the position of the selected Sunburst slice. The SelectedItemPosition property accepts values from the Position enumeration. Setting this property to a value other than 'None' causes the pie to rotate when an item is selected.

In addition, the FlexChartBase class provides the SelectionStyle property that can be used to access the properties provided by the ChartStyle class to style the Sunburst chart.

The following image displays Sunburst chart with a data point selected.



The following code snippet sets these properties:

- **Visual Basic**

```
' set the selection mode
Sunburst1.SelectionMode = C1.Chart.ChartSelectionMode.Point

' set the selected item position
Sunburst1.SelectedItemPosition = C1.Chart.Position.Auto

' set the selected item offset
Sunburst1.SelectedItemOffset = 0.2
```

- **C#**

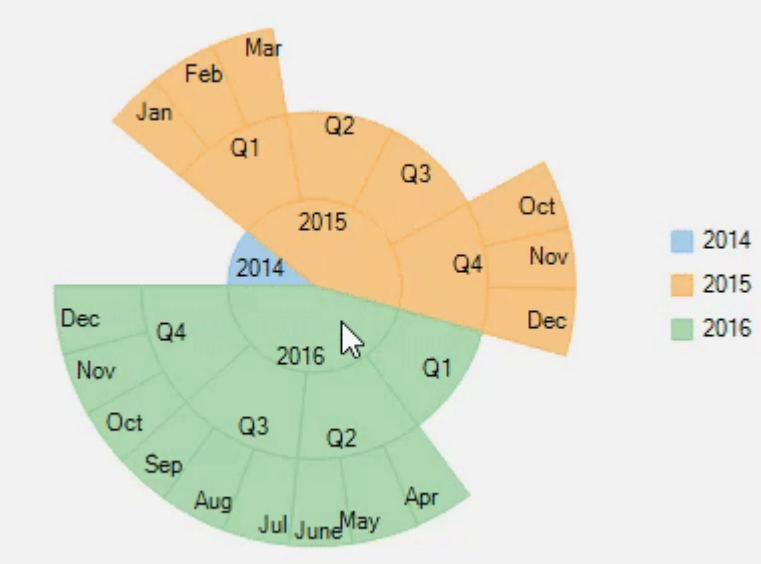
```
// set the selection mode
//sunburst1.SelectionMode = C1.Chart.ChartSelectionMode.Point;

// set the selected item position
//sunburst1.SelectedItemPosition = C1.Chart.Position.Auto;

// set the selected item offset
//sunburst1.SelectedItemOffset = 0.2;
```

Drilldown

Drilling down data to get in to the details and access lower levels in data hierarchy of Sunburst chart can be quite helpful while analysis. Sunburst chart provides **Drilldown** property to enable the functionality of drilling down and drilling back up the data at run-time.

End users can focus and drill down a data item in a Sunburst chart by simply clicking the desired slice. Whereas, to move up in the hierarchy, users simply need to right-click in the plot area.

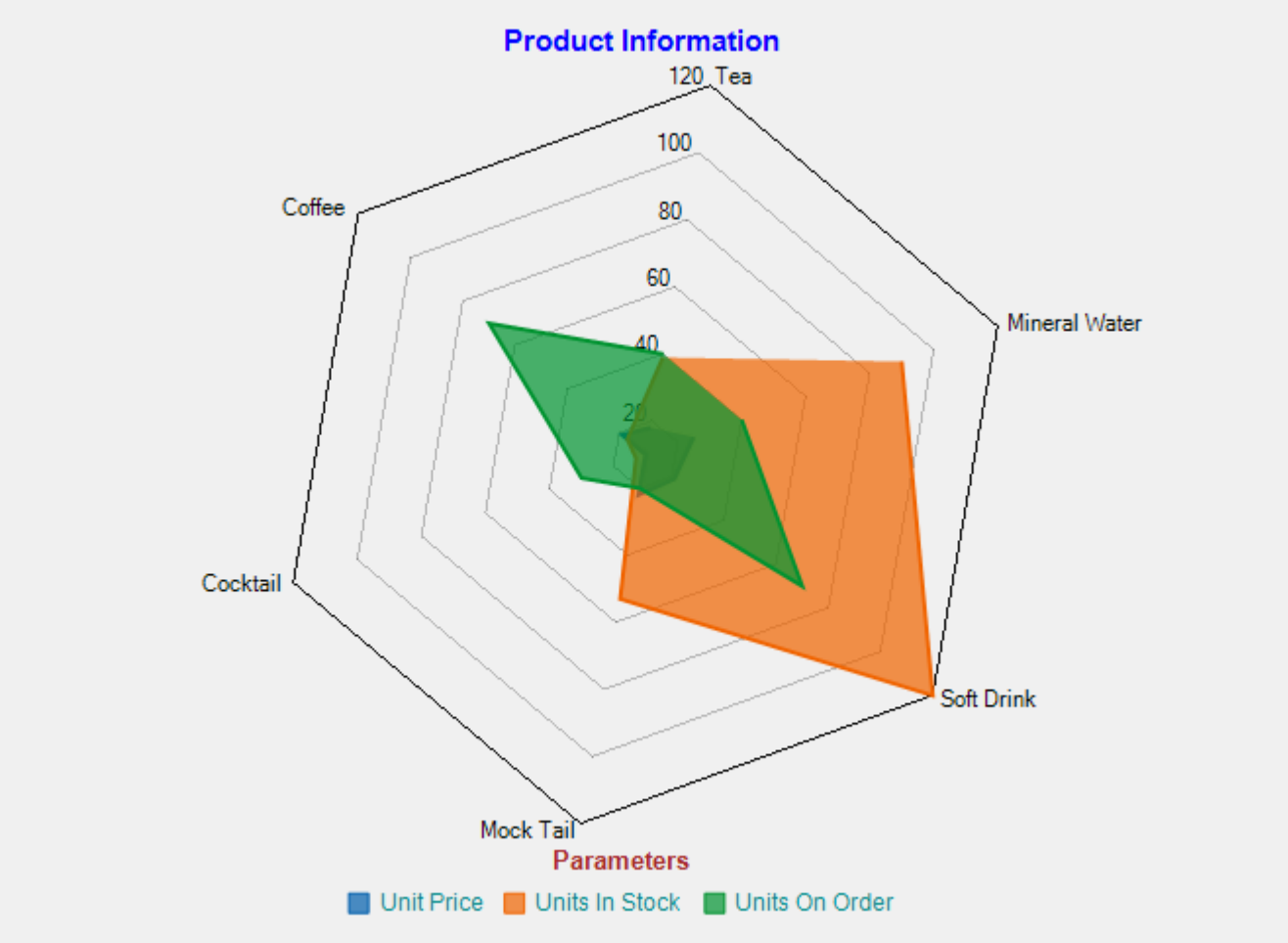
The following gif image demonstrates drilling-down by showing data points of the clicked Sunburst slice.



Note that drill down feature of Sunburst works only when selection of Sunburst slice is disabled, that is, SelectionMode property is set to **None**. For more information on selection, see Selection in Sunburst.

FlexRadar

FlexRadar is a radar chart that is also known as polar chart, star chart, web chart, or spider chart due to its appearance. The chart plots value of each category along a separate axis that starts from the center and ends on the outer ring. All axes are arranged radially, with equal distances between each other, while maintaining the same scale between all axes. Each category value is plotted along its individual axis and all the values are connected together to form a polygon. Common business applications of FlexRadar can include skill analysis of employees and product comparison. It is important to note that the FlexRadar control represents a polar chart when X values are numbers that specify angular values in degrees.



To know more about FlexRadar, click the following links:

- Quick Start
- Key Features
- Chart Types
- Legend and Titles

Key Features

Some of the key features of FlexRadar are as follows:

- **Chart types:** Visualize data using different chart types within FlexRadar. For more information, refer to Chart Types.
- **Header and Footer**: Use simple properties to set and customize Header and Footer of FlexRadar. For more

information, refer to Legend and Titles.

- **Legend**: Perform various customizations including setting orientation, position, or styling the legend of FlexRadar. For more information, refer to Legend and Titles.
- **Reversed FlexRadar**: Create a reversed FlexRadar by setting the Reversed property to True. A reversed FlexRadar is the one in which direction of plotting is reversed.
- **Start angle:** Set the start angle of FlexRadar by setting the StartAngle property provided by the FlexRadar class to double type values. A start angle is the angle that is set in degrees to start drawing radial axes in the clockwise direction.

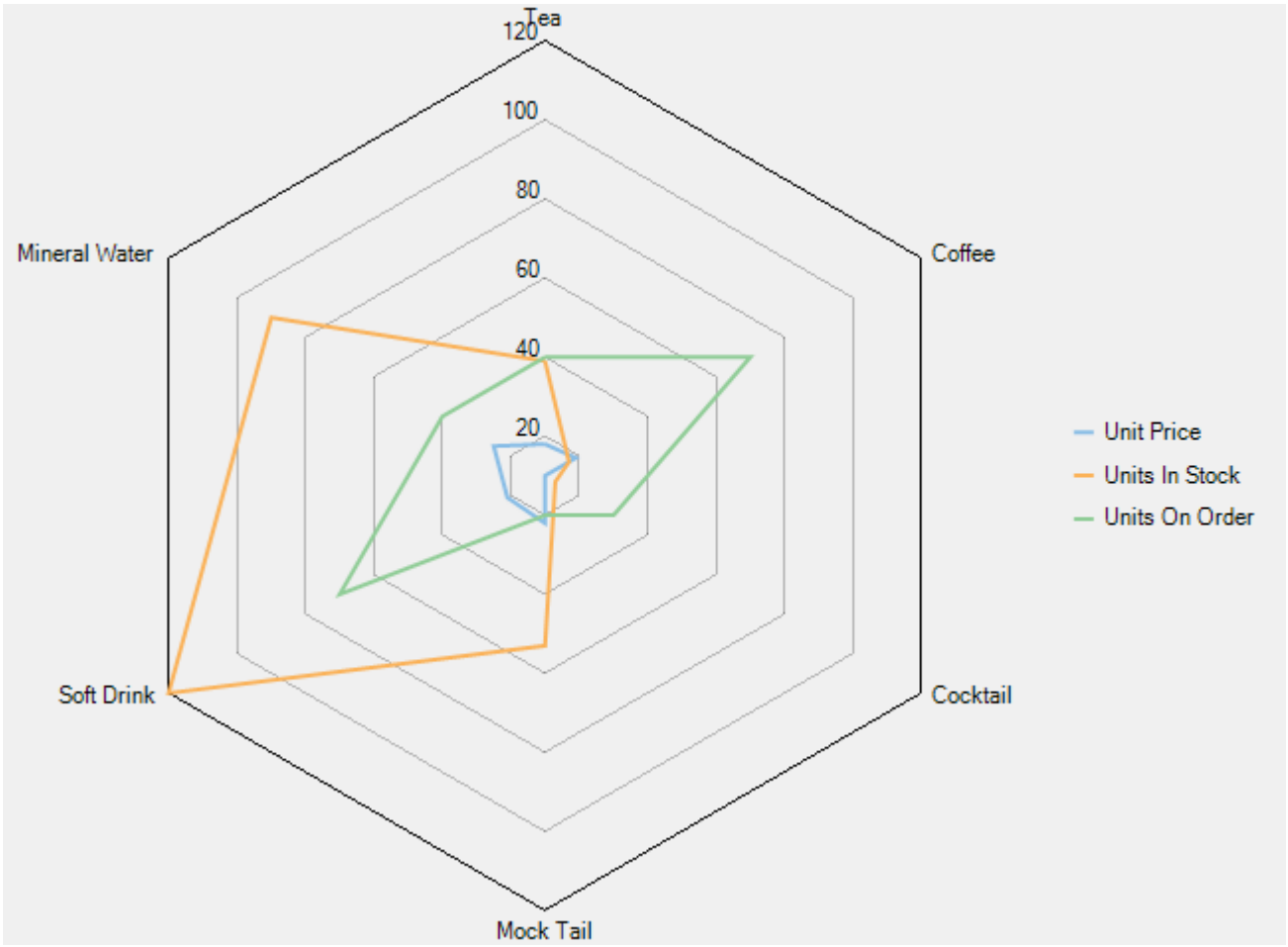
Quick Start

This quick start is intended to guide you through a step-by-step process of creating a simple FlexRadar application and running the same in Visual Studio.

To quickly get started with FlexRadar and observe how it appears on running the application, follow these steps:

1. **Add FlexRadar to the Application**
2. **Bind FlexRadar to the Data Source**
3. **Run the Application**

The following image displays how a basic FlexRadar appears after completing the steps mentioned above.



Step 1: Add FlexRadar to the Application

1. Create a **Windows Forms Application** in Visual Studio.
2. Drag and drop the **FlexRadar** control from the Toolbox to the application.

The .dll file which is automatically added to the application, is as follows:
C1.Win.FlexChart.4.dll

Step 2: Bind FlexRadar to the Data Source

In this step, you first create a data table that contains inventory data for different beverages. Next, you bind FlexRadar to the created data table using the DataSource property provided by the FlexChart class. Then, you specify fields containing X values and Y values for FlexRadar using the BindingX and the Binding property, respectively.

To bind FlexRadar to the data source, add the following code in the **Form_Load** event.

- **Visual Basic**

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    ' create a datatable
    Dim dt As New DataTable("Product Comparison")

    ' add columns to the datatable
    dt.Columns.Add("Beverages", GetType(String))
    dt.Columns.Add("Unit Price", GetType(Integer))
    dt.Columns.Add("Units In Stock", GetType(Integer))
    dt.Columns.Add("Units On Order", GetType(Integer))

    ' add rows to the datatable
    dt.Rows.Add("Tea", 18, 39, 40)
    dt.Rows.Add("Coffee", 19, 17, 70)
    dt.Rows.Add("Cocktail", 10, 13, 30)
    dt.Rows.Add("Mock Tail", 22, 53, 20)
    dt.Rows.Add("Soft Drink", 21, 120, 70)
    dt.Rows.Add("Mineral Water", 25, 90, 40)

    ' clear data series collection
    FlexRadar1.Series.Clear()

    ' create data series
    Dim series1 As New C1.Win.Chart.Series()
    Dim series2 As New C1.Win.Chart.Series()
    Dim series3 As New C1.Win.Chart.Series()

    ' add the data series to the data series collection
    FlexRadar1.Series.Add(series1)
    FlexRadar1.Series.Add(series2)
    FlexRadar1.Series.Add(series3)

    ' specify the datasource for the chart
    FlexRadar1.DataSource = dt

    ' bind the X-axis
    FlexRadar1.BindingX = "Beverages"

    ' bind the Y axes
    series1.Binding = "Unit Price"
    series2.Binding = "Units In Stock"
    series3.Binding = "Units On Order"

    ' name the series
    series1.Name = "Unit Price"
    series2.Name = "Units In Stock"
    series3.Name = "Units On Order"
```

```
End Sub
```

- **C#**

```
private void Form1_Load(object sender, EventArgs e)
{
    // create a datatable
    DataTable dt = new DataTable("Product Comparison");

    // add columns to the datatable
    dt.Columns.Add("Beverages", typeof(string));
    dt.Columns.Add("Unit Price", typeof(int));
    dt.Columns.Add("Units In Stock", typeof(int));
    dt.Columns.Add("Units On Order", typeof(int));

    // add rows to the datatable
    dt.Rows.Add("Tea", 18, 39, 40);
    dt.Rows.Add("Coffee", 19, 17, 70);
    dt.Rows.Add("Cocktail", 10, 13, 30);
    dt.Rows.Add("Mock Tail", 22, 53, 20);
    dt.Rows.Add("Soft Drink", 21, 120, 70);
    dt.Rows.Add("Mineral Water", 25, 90, 40);

    // clear data series collection
    flexRadar1.Series.Clear();

    // create data series
    C1.Win.Chart.Series series1 = new C1.Win.Chart.Series();
    C1.Win.Chart.Series series2 = new C1.Win.Chart.Series();
    C1.Win.Chart.Series series3 = new C1.Win.Chart.Series();

    // add the data series to the data series collection
    flexRadar1.Series.Add(series1);
    flexRadar1.Series.Add(series2);
    flexRadar1.Series.Add(series3);

    // specify the datasource for the chart
    flexRadar1.DataSource = dt;

    // bind the X-axis
    flexRadar1.BindingX = "Beverages";

    // bind the Y axes
    series1.Binding = "Unit Price";
    series2.Binding = "Units In Stock";
    series3.Binding = "Units On Order";

    // name the series
    series1.Name = "Unit Price";
    series2.Name = "Units In Stock";
    series3.Name = "Units On Order";
}
```

Step 3: Run the Application

Press F5 to run the application and observe the output.

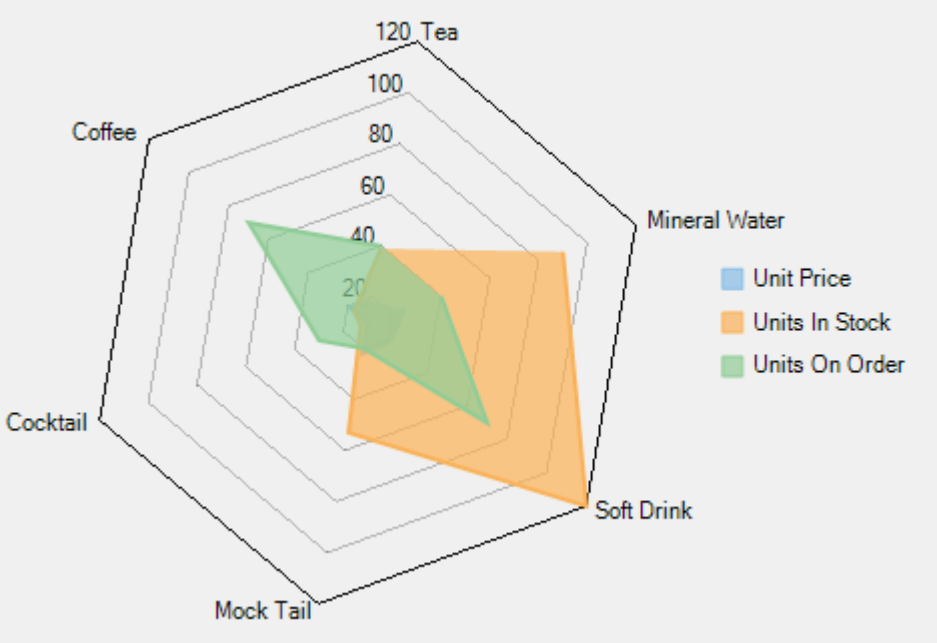
Chart Types

FlexRadar allows you to work with different chart types to meet your data visualization needs. You can use chart types

from Area to Scatter to display areas filled with colors or patterns within data depicted within FlexRadar. To work with different chart types in FlexRadar, set the ChartType property of FlexRadar to any of the following values in the RadarChartType enumeration:

- **Area**: Shows area below the line filled with color.
- **Line**: Shows trends over a period of time or across categories.
- **LineSymbols**: Shows line chart with a symbol on each data point.
- **Scatter**: Shows patterns within data using X and Y coordinates.

The following image displays FlexRadar with the chart type as Area.



The following code snippet sets the ChartType property in code using the sample created in Quick Start.

- **Visual Basic**

```
' set the FlexRadar chart type
FlexRadar1.ChartType = C1.Chart.RadarChartType.Area
```

- **C#**

```
// set the FlexRadar chart type
flexRadar1.ChartType = C1.Chart.RadarChartType.Area;
```

Legend and Titles

Legend

The legend displays entries for series with their names and predefined symbols. FlexRadar enables you to access the legend using the Legend property of the FlexChartBase class and perform various customizations, as follows:

- **Orientation**: Set the orientation of the legend as horizontal, vertical, or automatic by using the Orientation property provided by the Legend class. The property can be set to any of the values from the Orientation enumeration.
- **Position**: Set the legend on top, bottom, left, right, or let it be positioned automatically by using the Position property that accepts values from the Position enumeration. Setting the Position property to None hides the

legend.

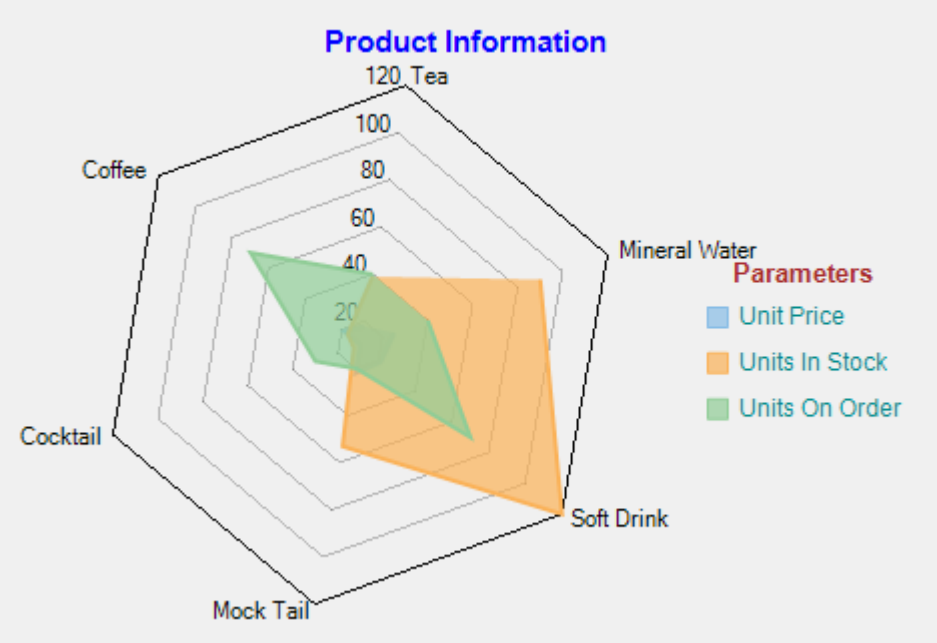
- **Styling**: Customize the overall appearance of the legend, such as setting stroke color or changing font by using styling properties accessible through the Style property. The styling properties StrokeColor and Font are provided by the ChartStyle class.
- **Title and title styling**: Specify the legend title using the Title property that accepts a string. Once you have set the title, you can style it using the TitleStyle property that provides access to the customization properties of the ChartStyle class.

Header and Footer

Header and Footer are descriptive texts at the top and bottom of the chart that provide information about the overall chart data. You can access Header and Footer of FlexRadar by using the Header and the Footer property, respectively of the FlexChartBase class. Possible customizations with Header and Footer are as follows:

- **Font**: Change the font family, font size, and font style of Header and Footer using the Font property of the ChartStyle class accessible through the Style property of the ChartTitle class.
- **Stroke color**: Choose any stroke color for the titles using the StrokeColor property of the ChartStyle class.
- **Stroke**: Set stroke of the titles for enhanced appeal by using the Stroke property.

The following image displays FlexRadar with the legend and titles set.



The following code snippet demonstrates how to set various properties.

- **Visual Basic**

```
' set the legend orientation
FlexRadar1.Legend.Orientation = C1.Chart.Orientation.Vertical

' set the legend position
FlexRadar1.Legend.Position = C1.Chart.Position.Right

' set the legend stroke color
FlexRadar1.Legend.Style.StrokeColor = Color.DarkCyan

' Set the legend font
FlexRadar1.Legend.Style.Font = New Font("Arial", 9, FontStyle.Regular)

' set the legend title
```

```
FlexRadar1.Legend.Title = "Parameters"

' set the legend title stroke color
FlexRadar1.Legend.TitleStyle.StrokeColor = Color.Brown

' set the legend title font
FlexRadar1.Legend.TitleStyle.Font = New Font("Arial", 10, FontStyle.Bold)

' set the header
FlexRadar1.Header.Content = "Product Information"

' set the header font
FlexRadar1.Header.Style.Font = New Font("Arial", 11, FontStyle.Bold)

' set the header stroke
FlexRadar1.Header.Style.Stroke = Brushes.Blue
```

- **C#**

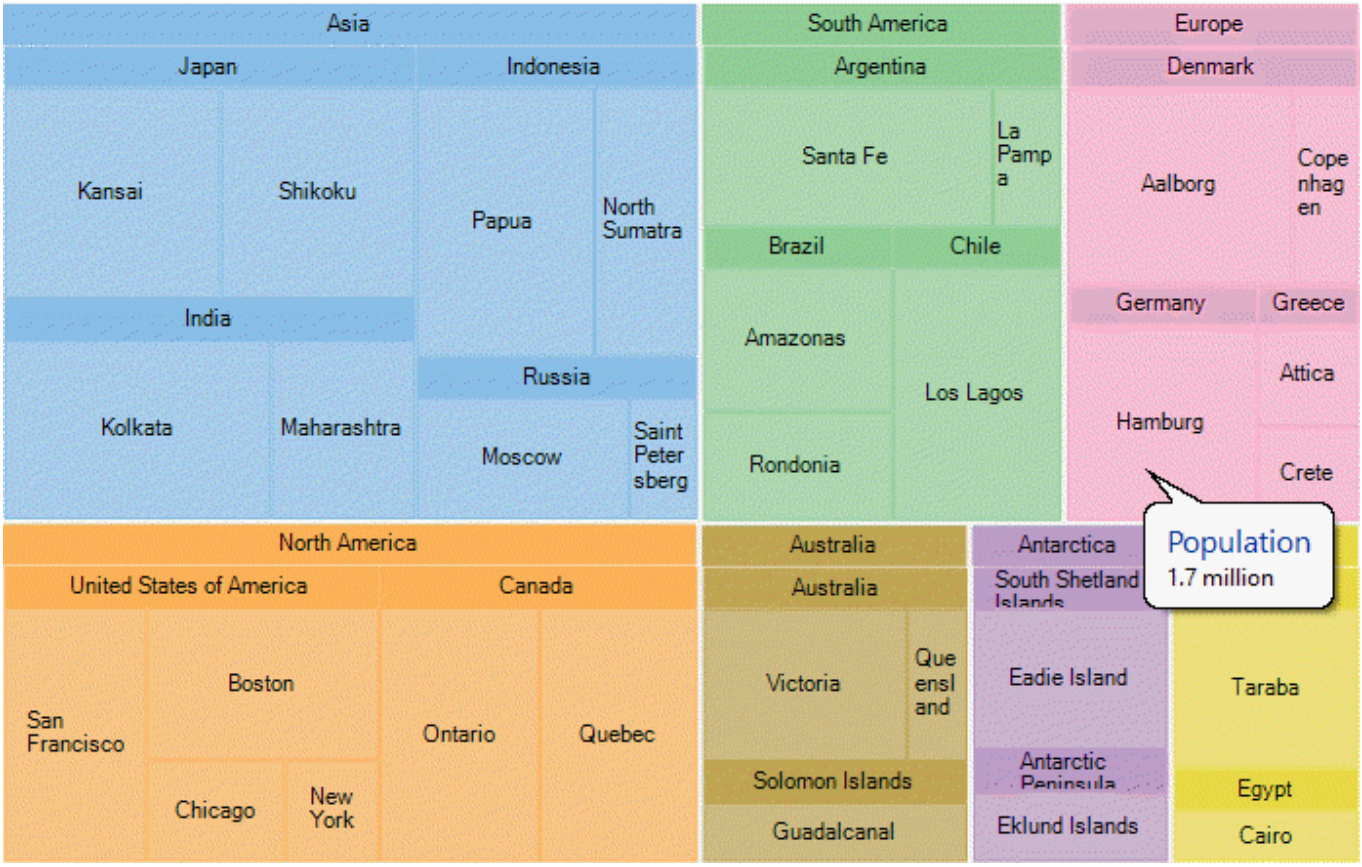
```
// set the legend orientation
flexRadar1.Legend.Orientation = C1.Chart.Orientation.Vertical;

// set the legend position
flexRadar1.Legend.Position = C1.Chart.Position.Right;

// set the legend stroke color
flexRadar1.Legend.Style.StrokeColor = Color.DarkCyan;

// Set the legend font
flexRadar1.Legend.Style.Font = new Font("Arial", 9, FontStyle.Regular);

// set the legend title
flexRadar1.Legend.Title = "Parameters";

// set the legend title stroke color
flexRadar1.Legend.TitleStyle.StrokeColor = Color.Brown;

// set the legend title font
flexRadar1.Legend.TitleStyle.Font = new Font("Arial", 10, FontStyle.Bold);

// set the header
flexRadar1.Header.Content = "Product Information";

// set the header font
flexRadar1.Header.Style.Font = new Font("Arial", 11, FontStyle.Bold);

// set the header stroke
flexRadar1.Header.Style.Stroke = Brushes.Blue;
```

TreeMap

Hierarchical information and data are useful in varied walks of life and setups, be it family tree, programming, organization structure, or directories. Visualizing such a data and spotting information in them is a difficult task, especially if the data is huge. Treemap charts enable visualization of hierarchical data as nested rectangles on a limited space. It is useful in having a quick glimpse of patterns in large data and comparing proportions.

TreeMap chart control supports binding to data to show hierarchy, and allows user to drill down the data further to numerous levels for detailed analysis. The control can be customized to display data in horizontal, vertical, and squarified layouts of constituting rectangles.

Both TreeMap and Sunburst charts are ideal for displaying and visualizing hierarchical data, but treemap is preferred when space is a constraint as it can show humongous data in a limited area.



The following topics help you get accustomed with the TreeMap control, and explore its advanced capabilities.

Key Features

TreeMap control has numerous features to enable users display hierarchical data in a limited area, and analyze data by comparing the size of tree nodes (or nested rectangles). These are as follows:

- **Hierarchical representation of data**
  TreeMap control is an ideal tool to help users visualize and compare proportions in data values within a hierarchy.
- **Layout**
  TreeMap supports multiple display arrangements, where the tree branches can be shown as squares, horizontal rectangles or vertical rectangles.

- **Customizable hierarchical levels**
  TreeMap control enables users to vary the depth of data to be visualized and further drill down (or reverse drill down) the data for analysis and comparison.
- **Customizable appearance**
  TreeMap enables users to stylize the control and vary its appearance as per their preference. A set of varied color palettes are available to clearly display categories in a tree map chart.
- **Data binding**
  TreeMap can be bound to different data sources that contain data of varied size, allowing you to display such a data in limited rectangular area.
- **Selection**
  TreeMap enables selecting tree nodes and group of nodes to draw focus on specific data items in the hierarchical data.
- **Optimum space utilization**
  TreeMap is ideal for compact display and visualization of huge data. The nested rectangles and groups constituting the treemap chart adjust their size to fit the display area.

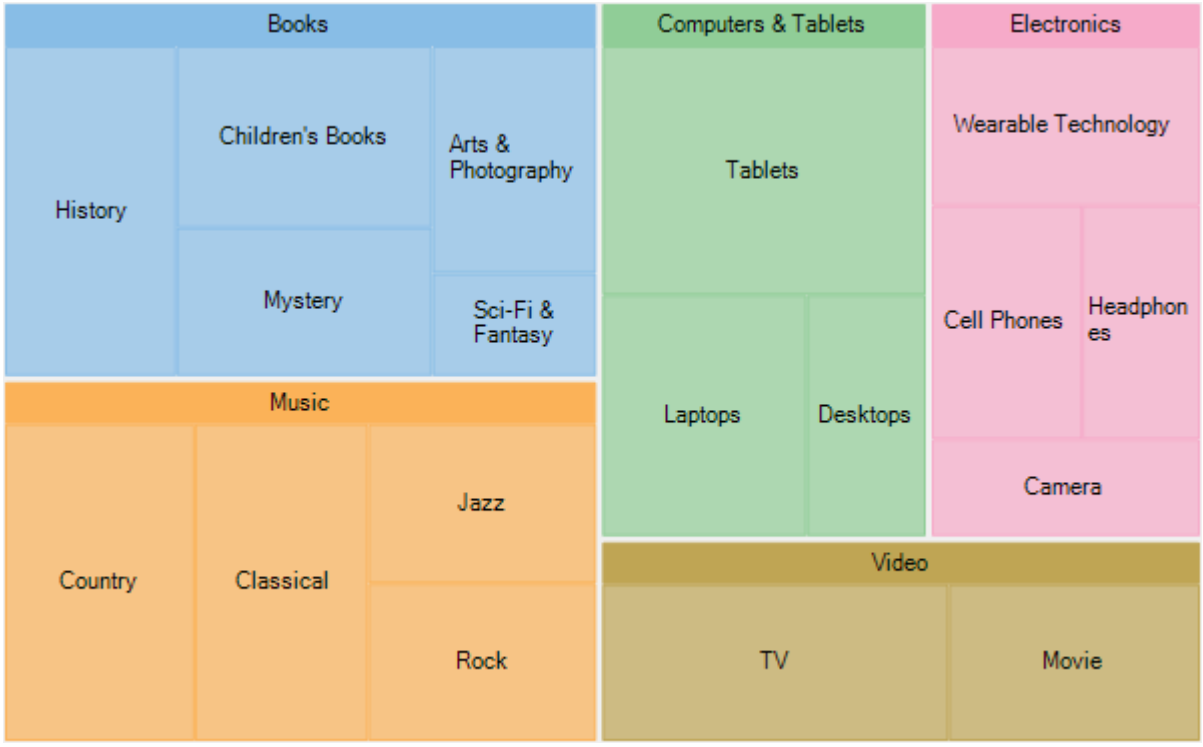
Quick Start

This quick start topic provides step-by-step instructions for adding a TreeMap chart to WinForms application, and show hierarchical data in it. In this topic, we consider an example where user wants to compare sales of books, music, videos, and gadgets (like computers and tablets) in XYZ city in a particular year.

The steps to display hierarchical data in TreeMap control are as follows:

- **Step 1: Add TreeMap to project**
- **Step 2: Create a hierarchical data source**
- **Step 3: Bind the TreeMap to data source**
- **Step 4: Build and run the project**

The following image exhibits and compares sales of different varieties of books, music, videos, and gadgets (like computers and tablets) in XYZ city in a particular year.



Back to Top

You need to set the DataSource property, of TreeMap class, to point to the collection of objects that contain data points to be plotted on the chart. To generate data items and display them in TreeMap chart, set BindingName and Binding properties. Set the **BindingName** property to the string value that specifies the name of the data item to display as chart rectangles, and **Binding** property to the string value that specifies the name of property of chart items that contain chart values (numeric values that help calculate the size of tree nodes).

To specify the level of hierarchical items to drill down and display in the chart set the MaxDepth property. Also, the display layout of the TreeMap is specified through its ChartType property. Additionally, color palette can be used to stylize the control and change its appearance.

Step 1: Add TreeMap to project

1. Create a **Windows Forms application** in Visual Studio.
2. Drag and drop **TreeMap** control from Toolbox to form.
   The following DLL gets added to your application:
   C1.Win.FlexChart.4.dll

Back to Top

Step 2: Create a hierarchical data source

Switch to the code view and add the following code to generate sales data of Books, Music, Electronic items, Videos, and Computers and tablets.

- **Visual Basic**

```
Private rnd As New Random()
Private Function rand() As Integer
    Return rnd.[Next](10, 100)
End Function
Private Function GetData() As Object
    Dim data = New Object() {New With {
          .type = "Music",
          .items = New () {New With {
              .type = "Country",
              .items = New () {New With {
                  .type = "Classic Country",
                  .sales = rand()
            }}
        }, New With {
              .type = "Rock",
              .items = New () {New With {
                  .type = "Funk Rock",
                  .sales = rand()
            }}
        }, New With {
              .type = "Classical",
              .items = New () {New With {
                  .type = "Symphonies",
                  .sales = rand()
            }}
        }}
    }, New With {
          .type = "Books",
          .items = New () {New With {
              .type = "Arts & Photography",
              .items = New () {New With {
                  .type = "Architecture",
                  .sales = rand()
            }}
```

```
        }, New With {
               .type = "Children's Books",
               .items = New () {New With {
                   .type = "Beginning Readers",
                   .sales = rand()
            }}
        }, New With {
               .type = "History",
               .items = New () {New With {
                   .type = "Ancient",
                   .sales = rand()
            }}
        }, New With {
               .type = "Mystery",
               .items = New () {New With {
                   .type = "Thriller & Suspense",
                   .sales = rand()
            }}
        }, New With {
               .type = "Sci-Fi & Fantasy",
               .items = New () {New With {
                   .type = "Fantasy",
                   .sales = rand()
            }}
        }}
    }, New With {
          .type = "Electronics",
          .items = New () {New With {
               .type = "Wearable Technology",
               .items = New () {New With {
                   .type = "Activity Trackers",
                   .sales = rand()
            }}
        }, New With {
               .type = "Cell Phones",
               .items = New () {New With {
                   .type = "Accessories",
                   .sales = rand()
            }}
        }, New With {
               .type = "Headphones",
               .items = New () {New With {
                   .type = "Earbud headphones",
                   .sales = rand()
            }}
        }, New With {
               .type = "Camera",
               .items = New () {New With {
                   .type = "Digital Cameras",
                   .sales = rand()
            }}
        }}
    }, New With {
          .type = "Video",
          .items = New () {New With {
               .type = "Movie",
               .items = New () {New With {
                   .type = "Children & Family",
                   .sales = rand()
            }}
        }, New With {
               .type = "TV",
```

```
                .items = New () {New With {
                    .type = "Comedy",
                    .sales = rand()
                }}
            }}
        }}
        Return data
End Function
```

- **C#**

```
    Random rnd = new Random();
    int rand()
    {
        return rnd.Next(10, 100);
    }
    object GetData()
    {
        var data = new object[] { new {
        type = "Music",
        items = new [] { new {
            type = "Country",
            items= new [] { new {
                type= "Classic Country",
                sales = rand()
            }}
        }, new {
            type= "Rock",
            items= new [] { new {
                type= "Funk Rock",
                sales= rand()
             } }
        }, new {
            type= "Classical",
            items= new [] { new {
                type= "Symphonies",
                sales= rand()
                } }
  }}
}, new {
        type= "Books",
  items= new [] { new {
            type= "Arts & Photography",
    items= new [] { new {
                type= "Architecture",
      sales= rand()
    }}
  }, new {
            type= "Children's Books",
    items= new [] { new {
                type= "Beginning Readers",
      sales= rand()
    } }
  }, new {
            type= "History",
    items= new [] { new {
                type= "Ancient",
      sales= rand()
    } }
  }, new {
            type= "Mystery",
    items= new [] { new {
```

```
                type= "Thriller & Suspense",
      sales= rand()
              } }
  }, new {
            type= "Sci-Fi & Fantasy",
    items= new [] { new {
                type= "Fantasy",
      sales= rand()
    }}
  } }
}, new {
        type= "Electronics",
  items= new [] { new {
            type= "Wearable Technology",
    items= new [] { new {
                type= "Activity Trackers",
      sales= rand()
    }}
  }, new {
            type= "Cell Phones",
    items= new [] { new {
                type= "Accessories",
      sales= rand()
    } }
  }, new {
            type= "Headphones",
    items= new [] { new {
                type= "Earbud headphones",
      sales= rand()
    } }
  }, new {
            type= "Camera",
    items= new [] { new {
                type= "Digital Cameras",
      sales= rand()
              } }
  } }
}, new {
        type= "Video",
  items= new [] { new {
            type= "Movie",
    items= new [] { new {
                type= "Children & Family",
      sales= rand()
    } }
  }, new {
            type= "TV",
    items= new [] { new {
                type= "Comedy",
      sales= rand()
    } }
  } }
} };
        return data;
    }
```

Back to Top

Step 3: Bind the TreeMap to data source

1. To set appropriate properties for displaying the tree map chart, define a method named SetupChart() and add the following code.

- **Visual Basic**

```
TreeMap1.BeginUpdate()
TreeMap1.DataSource = GetData()
TreeMap1.Binding = "sales"
TreeMap1.BindingName = "type"
TreeMap1.ChildItemsPath = "items"
TreeMap1.MaxDepth = 2
TreeMap1.EndUpdate()
```

- **C#**

```
treeMap1.BeginUpdate();
//DataSource property fetches random data
//generated in GetData() method of the type system.Object
treeMap1.DataSource = GetData();
treeMap1.Binding = "sales";
treeMap1.BindingName = "type";
treeMap1.ChildItemsPath = "items";
treeMap1.MaxDepth = 2;
treeMap1.EndUpdate();
```

2. Call the SetupChart() method in the Form class constructor below the InitializeComponent() method.

Back to Top

Step 4: Build and run the project

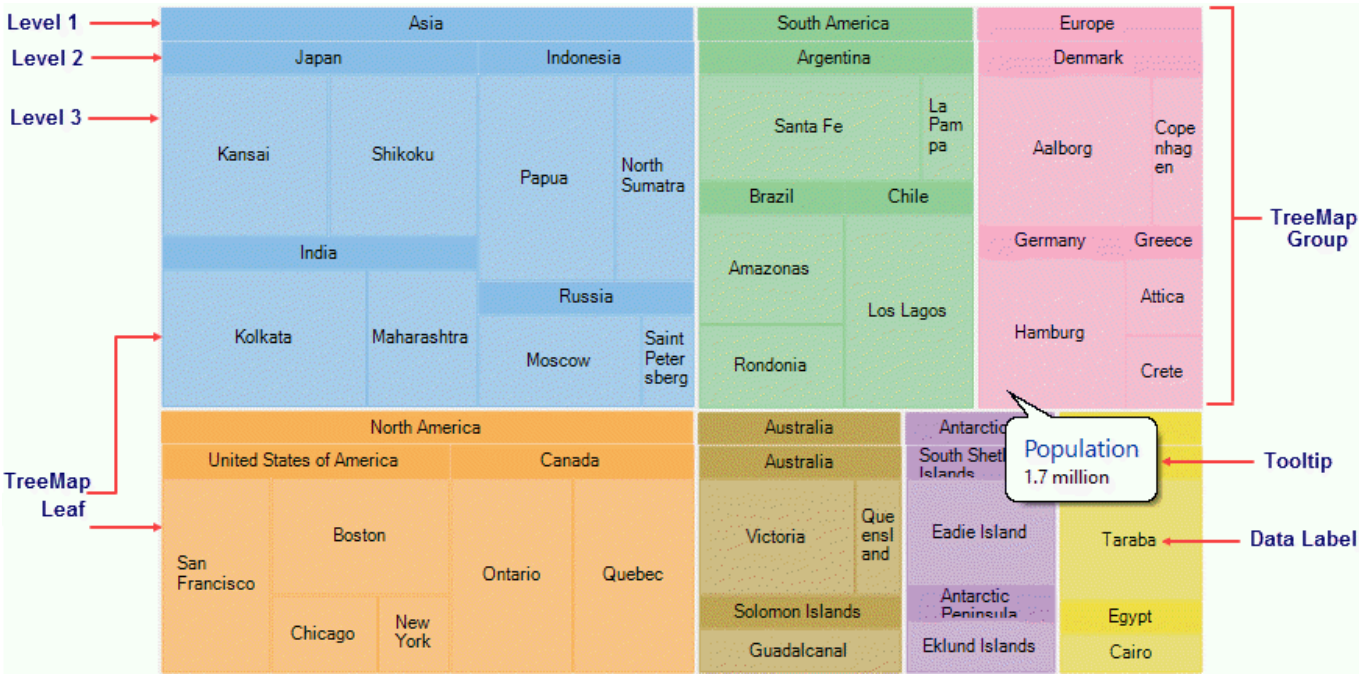
1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.

Back to Top

Elements

A tree map chart is composed of rectangles, representing individual data items, which are grouped into categories, to represent the hierarchical nature of data. The individual data items which make group are known as leaf nodes. The size of these nodes are proportional to the data they represent

The following image exhibits main elements of TreeMap Control.

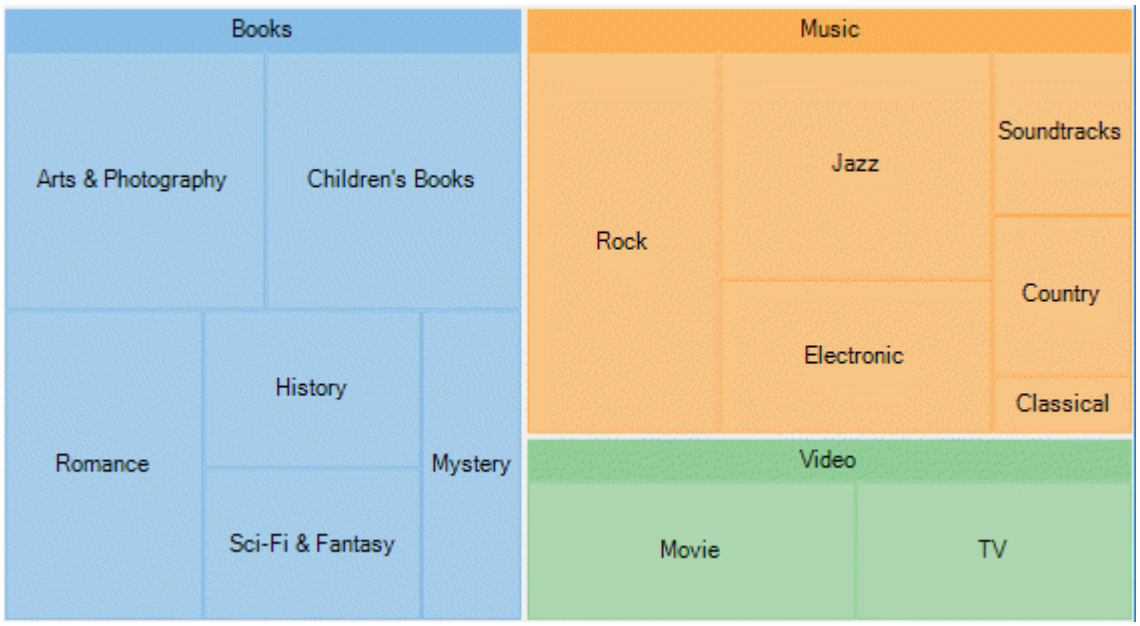


Layouts

TreeMap enables its data items and groups, represented as rectangles, to be displayed in a variety of arrangements. The tree map rectangles can be arranged into squarified, horizontal, and vertical layouts. To set the desired tree map layout, you need to use ChartType property of TreeMap class, which takes TreeMapType enum. The default layout of TreeMap chart control is squarified.

Squarified

The squarified layout tries to arrange the tree map rectangles (data items and groups) as approximate squares. This layout makes it easier to make comparisons and point patterns, as the accuracy of presentation is enhanced in squarified arrangement. This layout is very useful for large data sets.



Visual Basic

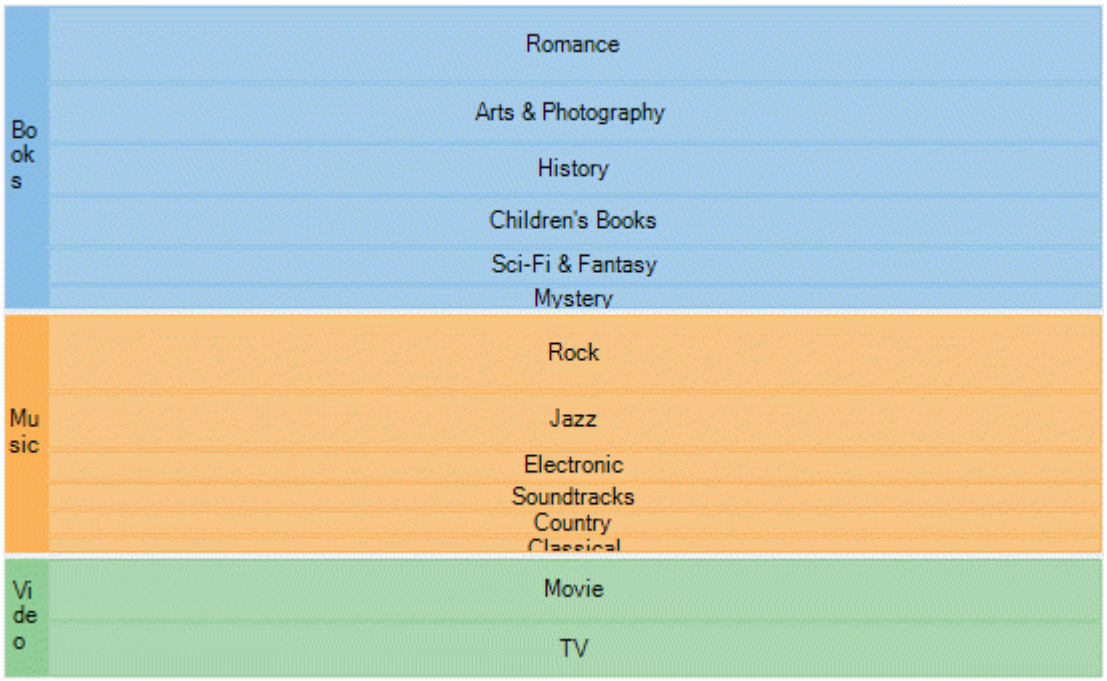
```
treeMap1.ChartType = C1.Chart.TreeMapType.Squarified
```

C#

```
treeMap1.ChartType = C1.Chart.TreeMapType.Squarified;
```

Horizontal

The horizontal layout stacks the tree map rectangles one over the other as rows. Here the width of the rectangles is greater than their height.



Visual Basic

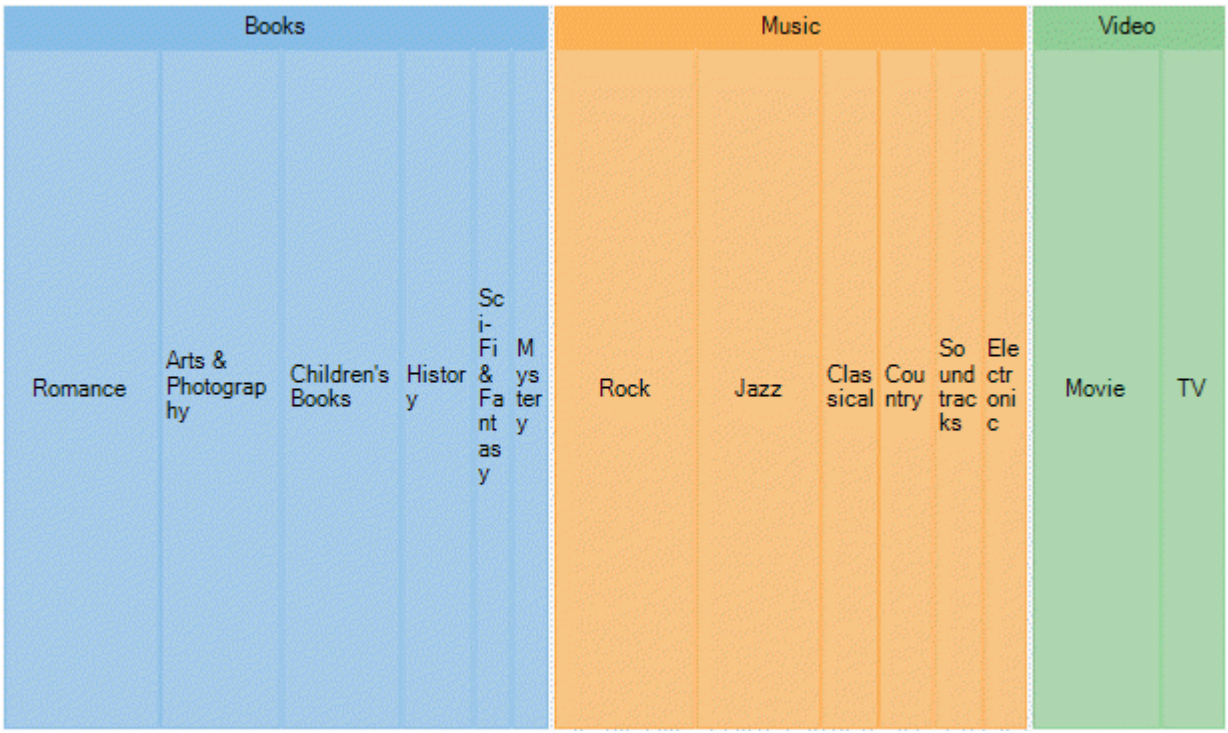
```
treeMap1.ChartType = C1.Chart.TreeMapType.Horizontal
```

C#

```
treeMap1.ChartType = C1.Chart.TreeMapType.Horizontal;
```

Vertical

The vertical layout arranges the tree map rectangles adjacent to each other as columns. Here the height of the rectangles is greater than their width.



Visual Basic

```
treeMap1.ChartType = C1.Chart.TreeMapType.Vertical
```

C#

```
treeMap1.ChartType = C1.Chart.TreeMapType.Vertical;
```

Horizontal and vertical treemaps are helpful in preserving and displaying the order of information.

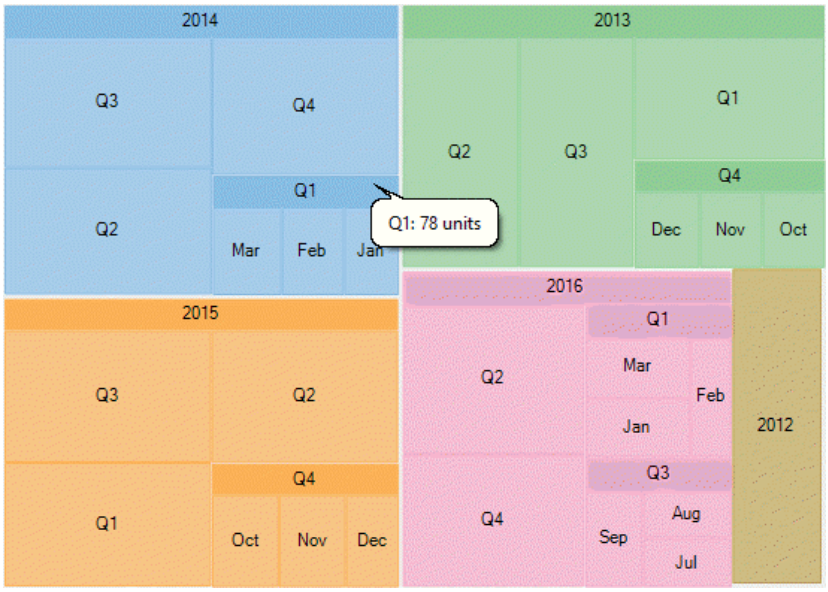
Data Binding

TreeMap chart control binds to hierarchical data, to represent the elements of tree-like data as nested rectangles. Once the control binds to the data source and displays data items as rectangles, the size and color of these constituting rectangles enable analysis and comparison of data items.

TreeMap class exposes DataSource property, which takes collection of the objects, that contain data, to populate in tree map chart. The Binding and BindingName properties are instrumental in generating rectangular nodes for data items and their respective categories or groups. While **Binding** property takes string value depicting the name of the property of data item that contains numeric data value, helpful in calculating the size of rectangular nodes, **BindingName** takes string value depicting the name of data items. **ChildItemPath** property ensures that a hierarchical structure of the provided data collection is maintained, by communicating to the control about the child items within the data.

To elaborate how data is populated in a tree map chart, let's consider a use case where we try to compare yearly sales (in units sold) of a multi-brand retail store. The analysis can then further be drilled down to quarters in a year and then to months in a quarter, by using Treemap chart. Here yearly sales are represented by the top level rectangles, quarterly sales in those years represent the subsequent level, and monthly sales form the next level that is leaf nodes in tree map.

The following image illustrates sales in a retail store, in terms of units sold, through TreeMap chart control. Note that the image shows hierarchical data up to third level, that is months in respective quarters of the years.



Back to Top

In this example, data generated in DataSource.cs class is serving as the source for tree map chart. **DataSource** property takes the hierarchical data collection generated in DataSource.cs class.

1. **Create a hierarchical data source**
    1. In the code view, create a DataService.cs class to generate hierarchical data, as shown in the following code.
        - **Visual Basic**

```
Public Class DataService
    Private rnd As New Random()
    Shared _default As DataService

    Public Shared ReadOnly Property Instance() As DataService
        Get
            If _default Is Nothing Then
                _default = New DataService()
            End If

            Return _default
        End Get
    End Property

    Public Shared Function CreateHierarchicalData() As List(Of DataItem)
        Dim rnd As Random = Instance.rnd

        Dim years As New List(Of String)()
        Dim times As New List(Of List(Of String))() From {
            New List(Of String)() From {
                "Jan",
                "Feb",
                "Mar"
            },
            New List(Of String)() From {
                "Apr",
                "May",
                "June"
            },
            New List(Of String)() From {
                "Jul",
                "Aug",
                "Sep"
            },
            New List(Of String)() From {
                "Oct",
                "Nov",
                "Dec"
            }
        }

        Dim items As New List(Of DataItem)()
        Dim yearLen = Math.Max(CInt(Math.Round(Math.Abs(5 - Instance.rnd.NextDouble() * 10))), 3)
        Dim currentYear As Integer = DateTime.Now.Year
        For i As Integer = yearLen To 1 Step -1
            years.Add((currentYear - i).ToString())
        Next
        Dim quarterAdded = False
```

```
        years.ForEach(Function(y)
                          Dim i = years.IndexOf(y)
                          Dim addQuarter = Instance.rnd.NextDouble() > 0.5
                          If Not quarterAdded AndAlso i = years.Count - 1 Then
                              addQuarter = True
                          End If
                          Dim year = New DataItem() With {
                              .Year = y
                          }
                          If addQuarter Then
                              quarterAdded = True
                              times.ForEach(Function(q)
                                                Dim addMonth = Instance.rnd.NextDouble() > 0.5
                                                Dim idx As Integer = times.IndexOf(q)
                                                Dim quar As String
                                                quar = "Q" + (idx + 1).ToString
                                                Dim quarters = New DataItem() With {
                                                    .Year = y,
                                                    .Quarter = quar
                                                }
                                                If addMonth Then
                                                    q.ForEach(Function(m)
                                                                  quarters.Items.Add(New DataItem() With {
                                                                      .Year = y,
                                                                      .Quarter = quar,
                                                                      .Month = m,
                                                                      .Value = rnd.[Next](20, 30)
                                                                  })

                                                              End Function)
                                                Else
                                                    quarters.Value = rnd.[Next](80, 100)
                                                End If
                                                year.Items.Add(quarters)

                                            End Function)
                          Else
                              year.Value = rnd.[Next](80, 100)
                          End If
                          items.Add(year)

                      End Function)

        Return items
    End Function

End Class
```

- **C#**

```
public class DataService
{
    Random rnd = new Random();
    static DataService _default;

    public static DataService Instance
    {
        get
        {
            if (_default == null)
            {
                _default = new DataService();
            }

            return _default;
        }
    }

    public static List<DataItem> CreateHierarchicalData()
    {
        Random rnd = Instance.rnd;

        List<string> years = new List<string>();
        List<List<string>> times = new List<List<string>>()
        {
            new List<string>() { "Jan", "Feb", "Mar"},
            new List<string>() { "Apr", "May", "June"},
            new List<string>() { "Jul", "Aug", "Sep"},
            new List<string>() { "Oct", "Nov", "Dec" }
        };

        List<DataItem> items = new List<DataItem>();
        var yearLen = Math.Max((int)Math.Round(Math.Abs(5 - Instance.rnd.NextDouble() * 10)), 3);
        int currentYear = DateTime.Now.Year;
```

```
            for (int i = yearLen; i > 0; i--)
            {
                years.Add((currentYear - i).ToString());
            }
            var quarterAdded = false;

            years.ForEach(y =>
            {
                var i = years.IndexOf(y);
                var addQuarter = Instance.rnd.NextDouble() > 0.5;
                if (!quarterAdded && i == years.Count - 1)
                {
                    addQuarter = true;
                }
                var year = new DataItem() { Year = y };
                if (addQuarter)
                {
                    quarterAdded = true;
                    times.ForEach(q =>
                    {
                        var addMonth = Instance.rnd.NextDouble() > 0.5;
                        int idx = times.IndexOf(q);
                        var quar = "Q" + (idx + 1);
                        var quarters = new DataItem() { Year = y, Quarter = quar };
                        if (addMonth)
                        {
                            q.ForEach(m =>
                            {
                                quarters.Items.Add(new DataItem()
                                {
                                    Year = y,
                                    Quarter = quar,
                                    Month = m,
                                    Value = rnd.Next(20, 30)
                                });
                            });
                        }
                        else
                        {
                            quarters.Value = rnd.Next(80, 100);
                        }
                        year.Items.Add(quarters);
                    });
                }
                else
                {
                    year.Value = rnd.Next(80, 100);
                }
                items.Add(year);
            });

            return items;
        }

    }
```

2. Create a DataItem class to define list of objects to represent data items and their categories.
    - **Visual Basic**

```
Public Class DataItem
    Private _items As List(Of DataItem)

    Public Property Year() As String
        Get
            Return m_Year
        End Get
        Set
            m_Year = Value
        End Set
    End Property
    Private m_Year As String
    Public Property Quarter() As String
        Get
            Return m_Quarter
        End Get
        Set
            m_Quarter = Value
        End Set
    End Property
    Private m_Quarter As String
    Public Property Month() As String
        Get
            Return m_Month
        End Get
        Set
```

```
            m_Month = Value
        End Set
    End Property
    Private m_Month As String
    Public Property Value() As Double
        Get
            Return m_Value
        End Get
        Set
            m_Value = Value
        End Set
    End Property
    Private m_Value As Double
    Public ReadOnly Property Items() As List(Of DataItem)
        Get
            If _items Is Nothing Then
                _items = New List(Of DataItem)()
            End If

            Return _items
        End Get
    End Property
End Class
```

- **C#**

```
public class DataItem
{
    List<DataItem> _items;

    public string Year { get; set; }
    public string Quarter { get; set; }
    public string Month { get; set; }
    public double Value { get; set; }
    public List<DataItem> Items
    {
        get
        {
            if (_items == null)
            {
                _items = new List<DataItem>();
            }

            return _items;
        }
    }
}
```

Back to Top

2. **Bind TreeMap to the data source**
    1. Define a method named SetupChart() as shown in the following code snippet, to set appropriate properties for displaying the tree map chart.
        - **Visual Basic**

```
Private Sub SetupChart()
    TreeMap1.BeginUpdate()
    TreeMap1.DataSource = DataService.CreateHierarchicalData()
    TreeMap1.Binding = "Value"
    TreeMap1.BindingName = "Year,Quarter,Month"
    TreeMap1.ChildItemsPath = "Items"
    TreeMap1.MaxDepth = 3
    TreeMap1.ToolTip.Content = "{name}: {value} units"
    TreeMap1.ToolTip.IsBalloon = True
    TreeMap1.EndUpdate()
End Sub
```

        - **C#**

```
void SetupChart()
{
    treeMap1.BeginUpdate();
    treeMap1.DataSource = DataService.CreateHierarchicalData();
    treeMap1.Binding = "Value";
    treeMap1.BindingName = "Year,Quarter,Month";
    treeMap1.ChildItemsPath = "Items";
    treeMap1.MaxDepth = 3;
    treeMap1.ToolTip.Content = "{name}: {value} units";
    treeMap1.ToolTip.IsBalloon = true;
    treeMap1.EndUpdate();
}
```

    2. Call the SetupChart() method in the Form class constructor below the InitializeComponent() method.

Back to Top

3. **Build and run the project**
    1. Click **Build | Build Solution** to build the project.
    2. Press **F5** to run the project.

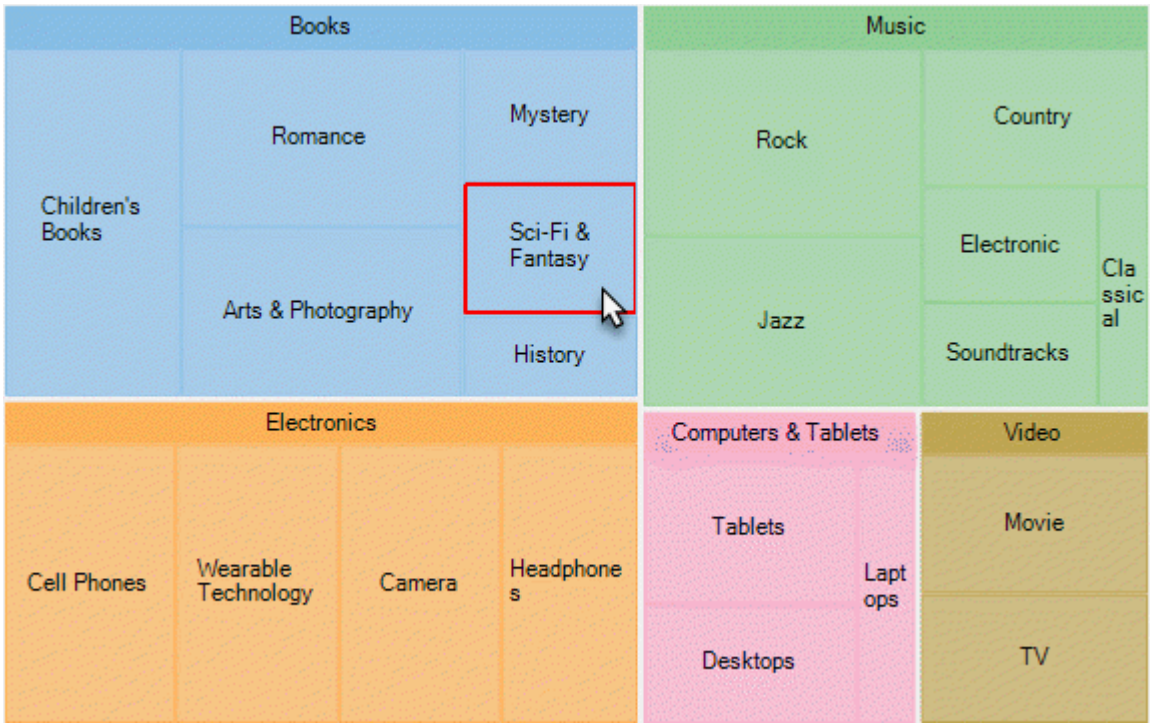
Back to Top

Selection

TreeMap chart lets you enable selection of its data items and groups. User can select a node and draw focus on it by simply clicking it. You need to set the SelectionMode property provided by the FlexChartBase class to either of the following values in the ChartSelectionMode enumeration:

- **None (default)**: Selection is disabled.
- **Point**: A point is selected.

The following image illustrates default selection of a data point along with its children nodes in the hierarchy.



The following code snippet shows how to set the **SelectionMode** property for a tree map chart.

- **Visual Basic**

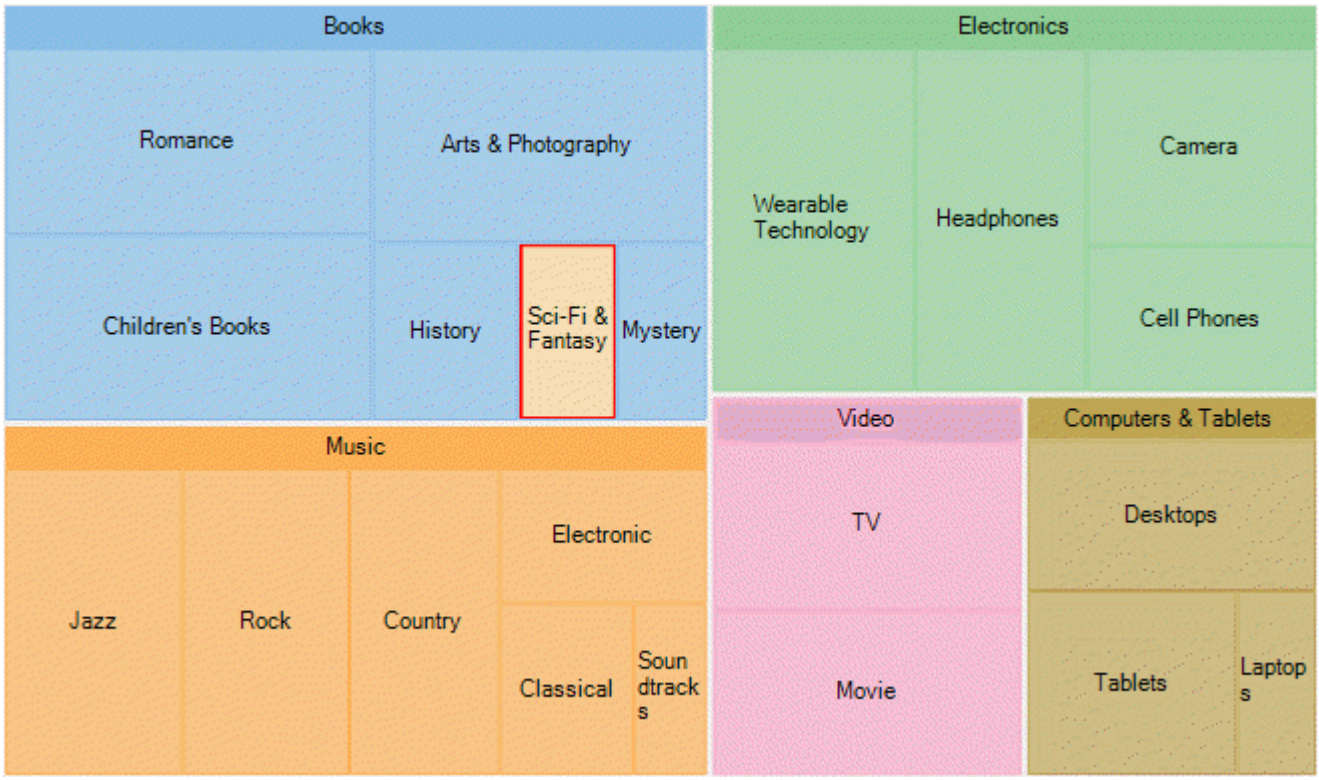
```
TreeMap1.SelectionMode = C1.Chart.ChartSelectionMode.Point
```

- **C#**

```
treeMap1.SelectionMode = C1.Chart.ChartSelectionMode.Point;
```

Customized TreeMap Selection

To customize the TreeMap selection, you can use SelectionStyle property and stylize the selected item as illustrated in the following image.



The following code snippet demonstrates utilizing **SelectionStyle** property to change fill color of the TreeMap node that is selected.

- **Visual Basic**

```
TreeMap1.SelectionStyle.Fill = Brushes.Wheat
```

- **C#**

```
treeMap1.SelectionStyle.Fill = Brushes.Wheat;
```

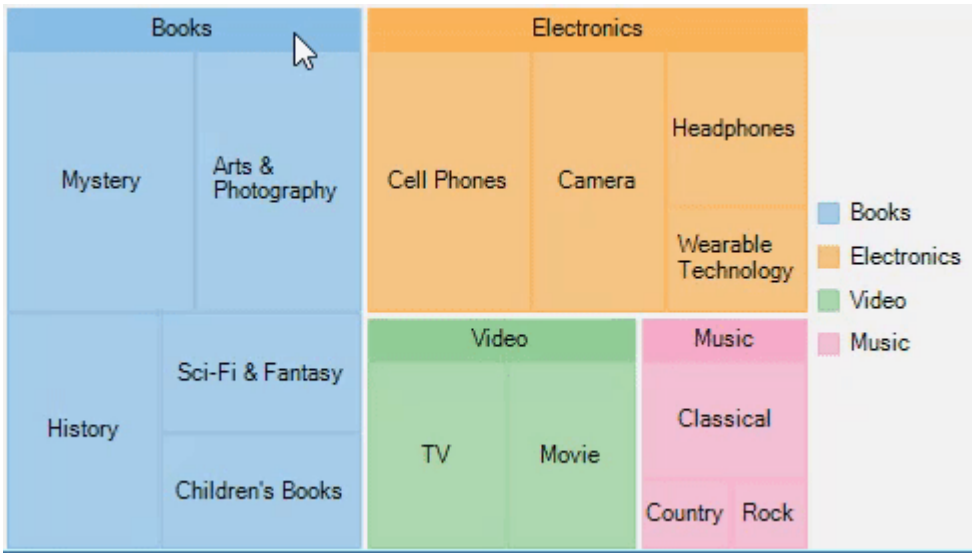
Additionally, you can customize the behavior of TreeMap selection by handling SelectionChanged event. Also, you can utilize SelectedIndex and SelectedItem properties, and reuse the obtained information in your application.

Drilldown

TreeMap allows drilling down the data items of its data further for detailed analysis. End users can access the lower levels in the data hierarchy by simply clicking the desired node. Whereas, to move back up in the hierarchy, users simply need to right-click in the plot area.

Note that, drilldown functionality in TreeMap is possible only if MaxDepth property is set to a value greater than 0. This property defines the levels of hierarchical data in the TreeMap chart.

The following gif image demonstrates drilling-down by showing data points of the clicked TreeMap node.



Note that drill down feature of TreeMap works only when selection of TreeMap nodes is disabled, that is, SelectionMode property is set to **None**. For more information on selection, see Selection in TreeMap.