

---

ComponentOne

# Chart2D for WinForms

**ComponentOne, a division of GrapeCity**

201 South Highland Avenue, Third Floor  
Pittsburgh, PA 15206 USA

**Website:** <http://www.componentone.com>

**Sales:** [sales@componentone.com](mailto:sales@componentone.com)

**Telephone:** 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

## Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

## 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 \$25 (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

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

## Table of Contents

2D Chart for WinForms Overview	11
Help with WinForms Edition	11
Key Features	12-16
Chart for WinForms Quick Start	17
Step 1 of 4: Create a Data Source for Chart	17-19
Step 2 of 4: Bind C1Chart to the Data Source	19-20
Step 3 of 4: Bind the ListBox to the DataSet	20-23
Step 4 of 4: Customize your Chart	23-26
Design-Time Support	27
C1Chart Smart Tag	27-28
C1Chart Context Menu	28-29
C1Chart Collection Editors	29
Action Collection Editor	29-30
AlarmZone Collection Editor	30-32
Axis Collection Editor	32-35
ChartDataSeries Collection Editor	35-36
ChartGroup Collection Editor	36-37
FunctionBase Collection Editor	37-39
Label Collection Editor	39-41
PointStyle Collection Editor	41-43
ScaleMenuItem Collection Editor	43-44
TrendLine Collection Editor	44-46
ValueLabel Collection Editor	46-47
VisualEffectsStyle Collection Editor	47-49
Chart Fundamentals	50
Defining the Header and Footer Elements	50-51
Defining the Legend Element	51-53
Defining the ChartArea Objects	53-54
Axes Object	54-56
PlotArea Object	56-57
Defining the ChartGroups Objects	57-58
ChartData Object	58
ChartDataSeries Object	58-60
Common Usage for Basic 2D Charts	61

Simple Bar Charts	61-63
Bar Charts with Two Y Axes	63-65
Pie Charts	65-67
Pie Charts with ChartLabels	67-69
XY-Plots (Scatter Plots)	69-72
Specific 2D Charts	73
Area Charts	73-74
Area Chart Programming Considerations	74
Area Chart 3D Effects	74
Bar Charts	74-75
Bar Chart Programming Considerations	75
Floating Bar Charts	75-76
Inverted Bar Charts	76
Stacking Bar Charts	76-79
Special Bar Chart Properties	79-81
Bar Chart 3D Effects	81-82
Variations of Bar Charts	82
Cylinder Charts	82-83
Cone Charts	83
Pyramid Charts	83-84
Bubble Charts	84
Bubble Chart Programming Considerations	84-85
Special Bubble Chart Properties	85
Candle Charts	85-86
Candle Chart Programming Considerations	86-87
Special Candle Chart Properties	87
Gantt Charts	87-88
Gantt Chart Programming Considerations	88-89
HiLo Charts	89
HiLo Chart Programming Considerations	89
HiLoOpenClose Charts	89
HiLoOpenClose Chart Programming Considerations	90
Special HiLoOpenClose Chart Properties	90
Histogram Charts	90-91
Histogram Chart Programming Considerations	91
Types of Histogram Graphs	91-92

Histogram Graph	92-96
Frequency Graph	96-97
Stepped Frequency Graph	97-98
Line and XY-Plot Charts	98-99
Plot Chart Programming Considerations	99
XY-Plot Chart 3D Effects	99
Pie and Doughnut Charts	99-100
Pie Chart Programming Considerations	100
Doughnut Charts	100-101
Special Pie Chart Properties	101-103
Pie Chart 3D Effects	103
Polar Charts	103-104
Polar Chart Programming Considerations	104
Special Polar Chart Properties	104-105
Radar Charts	105-106
Radar Chart Programming Considerations	106
Special Radar Chart Properties	106-108
Step Charts	108-109
Step Chart Programming Considerations	109
Step Chart 3D Effects	109-110
Design-Time Tools for Creating 2D Charts	111
Working with the Smart Designer	111
Primary Floating Toolbars	111
C1Chart Toolbar	111-114
ChartArea Toolbar	114-116
PlotArea Toolbar	116-118
Secondary Floating Toolbars	118
Axes Toolbars	118-121
Header and Footer Toolbars	121-124
Label Toolbar	124-125
Legend Toolbar	125-126
Working with the Chart Wizard	126-127
Step 1. Choose Chart Type	127-128
Step 2. Setup Chart	128-129
Step 3. Edit Chart Data	129-131
Working with the Chart Properties Designer	132

Gallery Element	132
Simple Types Tab	132-133
Complex Types Tab	133-134
Data Element	134-135
Chart Data Tab	135-136
Axis X , Axis Y, and Axis Y2 Elements	136-137
Axis X, AxisY, or Axis Y2 Tab	137-138
Scale Tab	138-139
Annotation Tab	139-140
Gridlines Tab	140-141
Appearance Element	141-142
Header Appearance Tab	142-143
Footer Appearance Tab	143-144
Legend Appearance Tab	144
ChartArea Appearance Tab	145
PlotArea Appearance Tab	145-146
Charting Data	147
Defining the Chart Data Objects	147-148
Defining the ChartGroup Object	148-149
Defining the ChartData Object	149-150
Defining the ChartDataSeries Object	150
Defining the ChartDataArray Object	150-151
Entering and Modifying Chart Data	151
Loading and Extracting Chart Data	151-153
Changing Data Elements in the Data Arrays	153-154
Displaying String Values as Annotations	154
Mixing ChartDataArray Inputs	154-155
Setting the X and Y Data Arrays Using Point Values	155
Customizing the ChartDataSeries	156
Showing, Excluding, or Hiding a Series	156
Excluding a Series from the Legend	156-157
Charting Irregular Data	157
Irregular Stacked Chart Data	157
Irregular X-Axis Data	157
Interpolating Y Value Data	157-158

Specifying Data Holes	158
Ignoring Data Holes	158-159
Plotting Functions	159-160
Using a Code String to Define a Function	160
Formula Code Types	160-161
Method Code Types	161-162
Unit Code Types	162-163
Calculating the Value for Functions	163
Calculating the Function Value Using Events	163-164
Calculating the Function Value Using the Calculate Method	164-165
Working with TrendLines	165-166
Creating TrendLines	166-167
Regression TrendLines	167
Regression Options	167-169
Regression Statistics	169-170
Custom TrendLine	170-173
Working with PointStyles	173-174
Creating PointStyles	174-176
Creating Custom PointStyles	176-177
Data Binding	178
Binding Data to C1Chart	178-182
Binding the Chart Directly to the Data Source	182-184
Functional Program Using Data Binding	184-189
Charting Labels	190-191
Attaching and Positioning Chart Labels	191-192
Attaching the Chart Label by Pixel Coordinate	192
Attaching the Chart Label by Data Coordinate	192-193
Attaching the Chart Label by Data Point	193
Attaching the Chart Label by Data Point and Y Value	193-194
Anchoring Chart Labels	194-195
Customizing Chart Labels	195-196
Setting a Default Label Style	196
Chart Area and Plot Area Objects	197
Axes	197
Axis Position	197-198
Axis Appearance	198-199

Axis Title and Rotation	199
Axis Tick Marks	199-201
Axis Grid Lines	201
Axis Bounds	201-202
Axis Scrolling and Scaling	202
Scrollbar Appearance	202-203
Scrollbar Scaling	203-205
Axis Scroll Events	205-209
Axis Logarithmic Scaling	209-210
Logarithmic scaling output for the annotations on the axis	210
Criteria used for Logarithmic Scaling	210-211
UnitMajor and Logarithmic Axes	211-212
Inverted and Reversed Chart Axes	212-213
Axes Annotation	213-214
Values Annotation	214-215
Value Labels Annotation	215-217
Mixed Annotation	217
Axis Annotation Location	217-218
Axis Annotation Rotation	219
Axis Annotation Overlap	219-220
Plot Area	220-221
Alarm Zones	221-223
Adding Alarm Zones	223-224
Defining Alarm Zones Properties	224
Customizing Chart Elements	225
Visual Effects Designer	225
Navigating the Visual Effects Designer	225-226
Visual Effects Elements	226
Visual Effects Designer's Tabs	226-228
Visual Effects Parameters	228
Scaling and Edge Effects	228-229
Light Effects	229-230
Shadow Effects	230-231
Chart Themes	231
Chart Titles	231
Title, Text, and Alignment	231

Title Position and Size	231
Title Border	231-232
Title Colors and Gradient Effects	232
Title Font	232
Chart Legend	232
Legend Position	232
Legend Title	232
Legend Border	232
Legend Colors	232-233
Legend Font	233
Line and Symbol Styles for the Series	233-234
Line FitType	234-235
Chart Borders	235-236
Chart Fonts	236-237
Chart Colors	237
Choosing Colors Interactively	237
Setting the Color Scheme for the Data Series	237-244
Specifying RGB Colors	244
Specifying Hue, Saturation, and Brightness	244
Using Transparent Colors	244
Chart Elements Position and Size	244
Changing Location	244-245
Changing Width and Height	245
Custom Brushes for Plotting Data	245
Creating a Hatched Brush	245-246
Creating a Gradient Brush	246-247
Loading and Saving Charts, Data, and Images	248
Loading and Saving to a String	248-249
Loading and Saving Chart to a File	249-250
Saving Chart Images	250-251
End-User Interaction	252
Coordinate Conversion Methods	252-253
Converting Coordinates to Series	253
Converting Pixel Coordinates into Data Points and Vice Versa	253-254
Converting Pixel Coordinates into Data Points	254-255

Converting Data Points into Pixel Coordinates	255-256
Rotating, Scaling, Translating, and Zooming	256-257
Controlling Transformation	257-258
Creating a Zoom Effect	258-259
Zooming, Panning, and Scaling with Multiple Y-axes	259
Chart for WinForms Tutorials	260
Bar Chart Tutorial	260-264
Line Chart Tutorial	264-269
Pie Chart Tutorial	269-273
Candle Chart Tutorial	273-277
Multiple Charts Tutorial	277-281
Chart for WinForms Task-Based Help	282
Rotating the Y-Axis Title	282
Rotating Data Labels	282
Displaying the Data Label as a Percent in Pie Charts	282-285
Setting the Font Style for Data Labels	285
Adding a Data Label on Top of Each Bar	285-286
Wrapping Labels	286
Adding a Transparent Label to Adjust the Gap Between the Values and the X-Axis	286
Displaying both the Chart Legend and Chart Header	286-288
Displaying the Legends Vertically	288
Getting the Slice of a Pie with a Click	288-289
Creating a Marker	289
Add Scrollbar to the X-Axis and Y-Axis	289-291
Add Symbols to Data Series	291-293
Add ToolTips to Chart Elements	293
Add ToolTips to Chart's Points in the Data Series	293-294
Add ToolTips to Chart's Header and Footer	294-295
Add ToolTips to Axes	295-296
Adding Visual Effects to Chart Elements	297
Access the Visual Effects Designer	297-298
Customize Header and Footer	298
Add a Light Pattern to the Chart Header and Footer	298-299
Add a Light Shape to the Chart Header and Footer	299-300
Add a Preset Style to the Chart Header and Footer	300-301
Add a Shadow to the Chart Header and Footer	301-303

Adjust the Focus of the Light to the Chart Header and Footer	303-304
Use the Color Sliders to Enhance an Existing Color for the Chart Header and Footer	304-305
Customize Data Series	305
Add a Light Pattern to the Chart Data Series	305-306
Add a Light Shape to the Chart Data Series	306
Add a Preset Style to the Chart Data Series	306-307
Add a Shadow to the Chart Data Series	307
Adjust the Focus of the Light for the Chart Data Series	307-308
Use the Color Sliders to Enhance an Existing Color for the Chart Data Series	308-309
Increase the Size of the Symbols in the Data Series	309
Creating and Formatting Chart Elements Using the Properties Window	309
Add a Chart Footer using the Properties Window	309-310
Add a Chart Header using the Properties Window	310-311
Add a Chart Legend using the Properties Window	311-312
Add Data Series and Data to the Chart using the Properties Window	312-314
Add Labels to the Chart using the Properties Window	314-315
Add Rotated Labels to the Chart using the Properties Window	315
Choose a Chart Type using the Properties Window	315-316
Modify the Appearance of the Chart Labels using the Properties Window	316-317
Modify the X and Y Axis Appearance using the Properties Window	317
Creating and Formatting Chart Elements Using the Smart Designer	317-318
Add a Chart Footer	318
Add a Chart Header	319
Add a Chart Legend	319-320
Add Data Series to the Chart	320-322
Add Data to the Data Series	322-323
Add Labels to the Chart	323-324
Choose a Chart Type	324-325
Choose a Chart sub-type	325-326
Edit the Chart Labels	326-327
Edit the X and Y Axis	327-328
Modify the Appearance of the Chart Footer	328-329
Modify the Appearance of the Chart Header	329
Modify the Appearance of the Chart Legend	329-330
Modify the Appearance of the Data Series	330-332
Modify the Color Theme of the Data Series	332-333

Attach Chart Labels	333
Attach By Coordinate	333-334
Attach By Data Coordinate	334-336
Attach By Data Index	336-337
Candle Chart Tasks	337
Increasing the Width Size of the Candle Body	337
Creating a Fill Color for Rising Candles	337
Frequently Asked Questions	338
How do I change chart type?	338
How do I change the way chart data is plotted?	338
How do I change colors displayed in the chart?	338-339
How do I change placement of the Legend?	339
How do I add or modify a Border?	339
How do I sync the X-axis of multiple charts?	339-340

## 2D Chart for WinForms Overview

Efficiently create professional looking 2D or 3D charts with **Chart for WinForms**. Using the latest technologies built into Visual Studio, **C1Chart** is fully compatible with the 2.0, 3.5 and 4.0 .NET Frameworks. ComponentOne's Chart tools completely manage the underlying complexities of a component chart, allowing developers to concentrate on important application-specific tasks.

**Chart for WinForms** includes two charting controls for creating 2D and 3D charts in Microsoft Visual Studio .NET: C1Chart and C1Chart3D. The **C1Chart** control is a two dimensional charting control that enables you create a variety of dynamic 2D charts for any type of charting application. The **C1Chart3D** control is three-dimensional charting control used to create 3D Surface, 3D Bar, 3D Scatter Plot charts, and 4D Bar and Surface charts.

**Chart for WinForms** features comprehensive and extensive documentation to help you get the full potential of the C1Chart and C1Chart3D controls. For your convenience, two separate help files are included with **Chart for WinForms**:

- C1Chart2D Help – Includes documentation relating to the C1Chart control.
- C1Chart3D Help – Includes documentation relating to the C1Chart3D control.

Create stunning, optimal performing charts in any type of charting application. Offering over 80 types of 2D and 3D charts, flexible and customizable charting elements, intuitive designers for code-free development, enhanced visual effects such as lighting effects and drop shadows, end-user interaction, and advanced mouse tracking capabilities, **Chart for WinForms** gives you the advantage.

### Getting Started

If you're new to **Chart for WinForms**, get started with the following topics:

- [Chart for WinForms Quick Start](#)
- [Specific 2D Charts](#)
- [Design Time Tools for Creating 2D Charts](#)
- [Chart Area and Plot Area Objects](#)
- [Chart for WinForms Tutorials](#)

## Help with WinForms Edition

### Getting Started

For information on installing **ComponentOne Studio WinForms Edition**, licensing, technical support, namespaces and creating a project with the control, please visit [Getting Started with WinForms Edition](#).

## Key Features

**Chart for WinForms** includes the following features for the **C1Chart** component:

- **Chart type can be changed simply by setting one property field**

Re-format any chart from line to bar to pie by simply changing one property. Get the exact chart representation you need for any application.

- **SmartDesigner provides highly-interactive chart building capabilities**

Save substantial time using Chart's SmartDesigner, which handles everyday tasks in chart placement. Accomplish tasks without leaving the design form; each chart element reveals built-in toolbars and editors with the click of your mouse pointer.

For more information on the **SmartDesigner** see, [Working with the Smart Designer](#).

- **Novice users can create a chart in three simple steps with the chart wizard**

The **Chart Wizard** walks beginners through the steps of creating a new chart from start to finish: choose the chart type; modify chart elements such as header, footer, and legend; and edit the chart's data.

For more information on the Chart Wizard, see [Working with the Chart Wizard](#).

- **You no longer have to tirelessly scroll through the Properties window to create a chart**

C1Chart places the chart elements in an organized Chart Properties designer so you can quickly address chart details. Create or modify existing charts: choose from simple to complex chart types, and modify the data, axis elements, and appearance settings.

For more information on the **Chart Properties** designer, see [Working with the Chart Properties designer](#).

- **Visually enhance chart elements with the intuitive Visual Effects designer**

Enhance the chart elements' appearance by applying angle, gradient, intensity, scaling, and shape to change the effects of the light source.

For more information on chart's visual effects, see [Visual Effects Designer](#). To see how to use the Visual Effects designer to apply visual effects to chart elements, see [Adding Visual Effects to Chart Elements](#).

- **Choose from over 20 built-in color schemes**

Apply color generation that mimics the Microsoft Office color themes to the data series with incredible ease.

For more information on how to access the color schemes, see [Setting the Color Scheme for the Data Series](#).

- **Industry leading stacking charts**

Line, Area, Bar, Radar, and Plot charts can be stacked to display more complex data in a smaller space.

For more information on stacking charts, see [Line and XY-Plot Charts](#).

- **Add visual appeal to your data analysis**

Add data highlighting, trend lines, and alarm zones to your charts to create a more effective and readable data chart.

For more information on trend lines, see [Working with TrendLines](#).

For more information on alarm zones, see [Alarm Zones](#).

- **Invert axes using one property**

Enables you to invert the X and Y axis using one simple property.

For more information on inverted axes, see [Inverted and Reversed Chart Axes](#).

- **Highly interactive behavior at run time drives up value in chart use**

C1Chart provides interactive built-in tools for rotation, scaling, and zooming. Using these tools, you can build highly-interactive charts for your users.

For more information on end-user interaction behaviors, see [Rotating, Scaling, Translating, and Zooming](#).

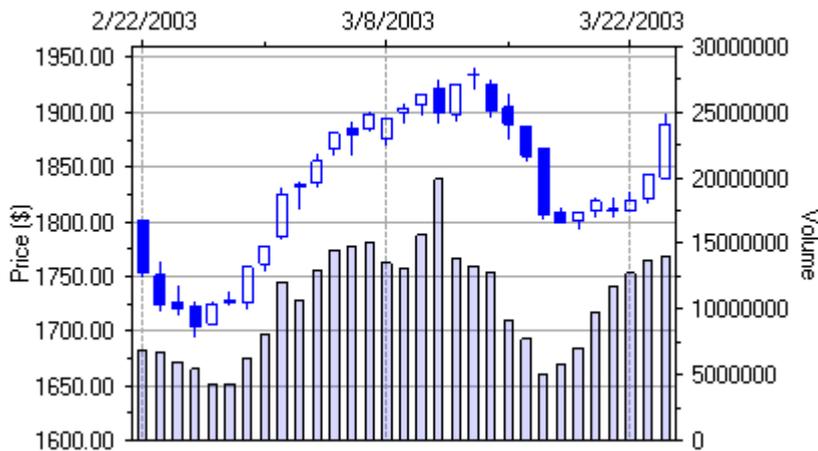
- **C1Chart provides flexible image formats for chart rendering**

Charts can be saved to any number of image formats (metafile, BMP, JPG, and more).

For more information on saving charts, see [Loading and Saving Charts, Data, and Images](#).

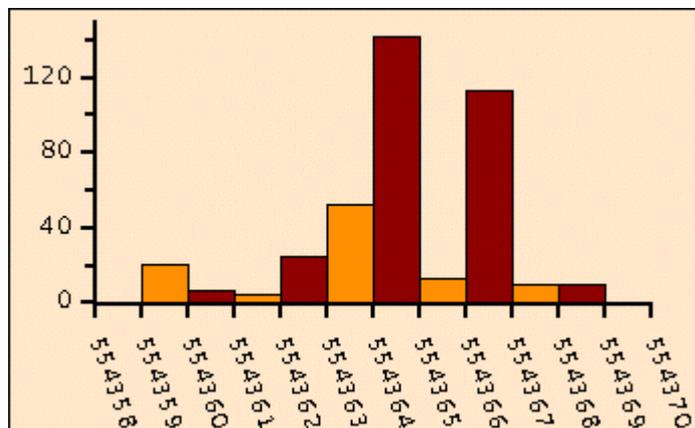
- **Create a dramatic data presentation by combining multiple chart types into a single chart**

Create an area-scatter combination chart or a bar-candle combination chart. Include a wealth of information in a single chart by combining multiple chart types.



- **Rotate annotations when working in a confined space**

When working with larger annotations, you can rotate the annotation to arbitrary angles so that text does not overlap.

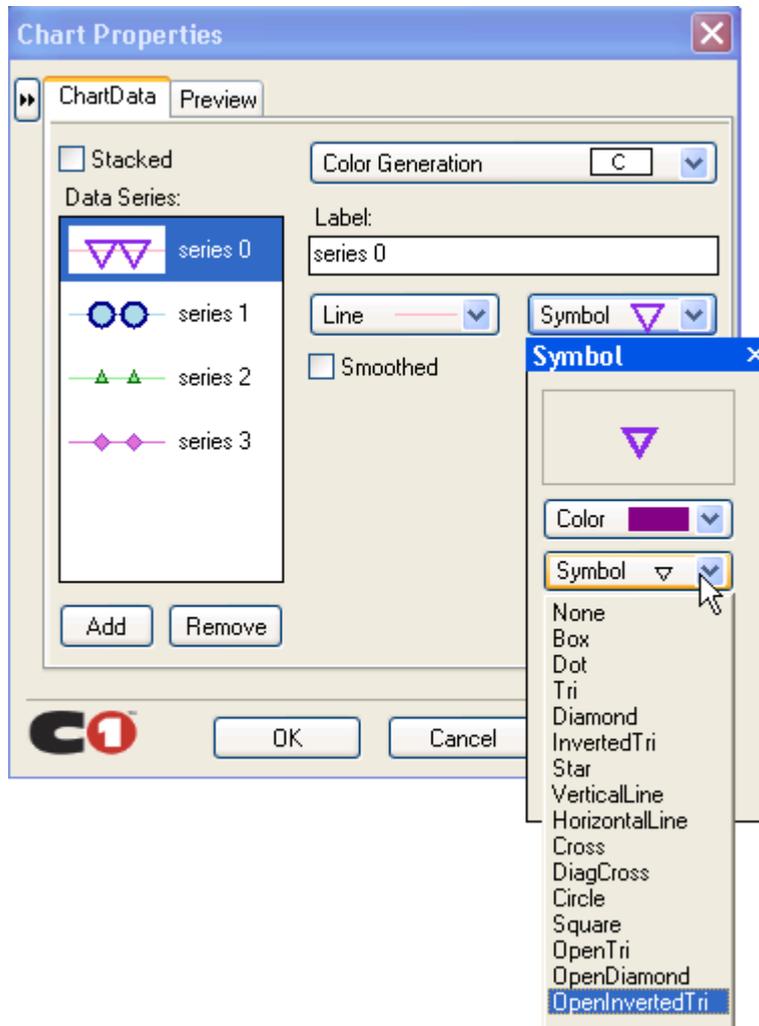


For more information on annotation rotation, see [Axis Annotation Rotation](#).

- **Add symbols such as stars, diamonds, squares and more to represent different series of data for Line,**

## Scatter, Step, and Polar/Radar charts

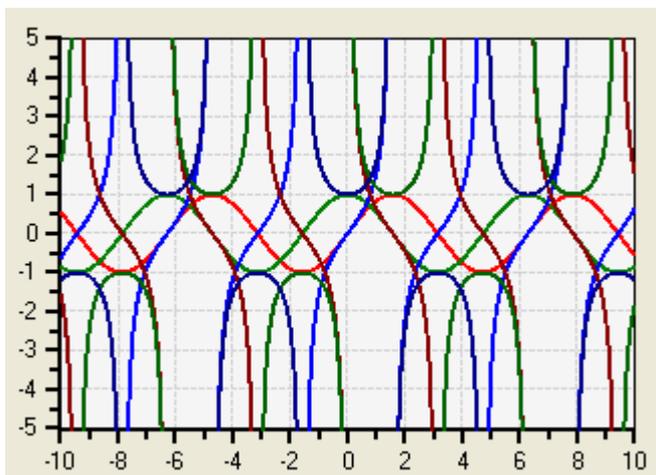
Choose from a large selection of symbol types such as star, diamond, square, and circle to represent different series of data. Customize the symbols color and size to make it truly unique from each series.



For more information on how to add symbols to a particular data series, see [Add Symbols to Data Series](#).

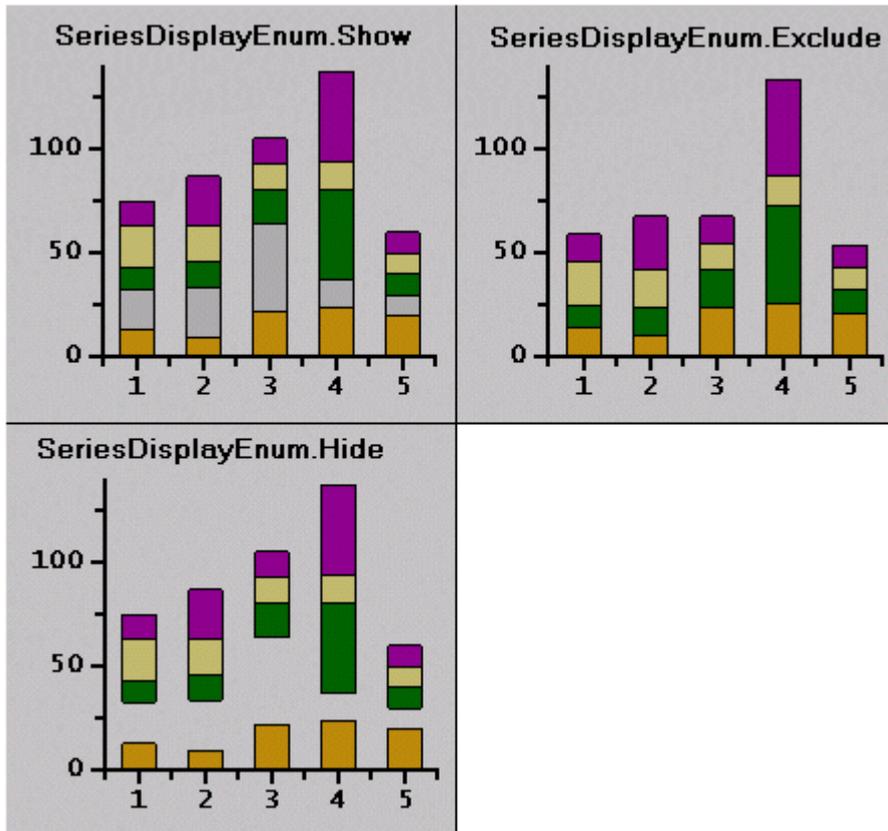
- **Built-in engine to help plot advanced functions**

The C1Chart control includes a FunctionBase Collection editor to help you create and edit functions for plotting explicit and parametric functions.



For more information on using functions for plotting data, see [Plotting Functions](#).

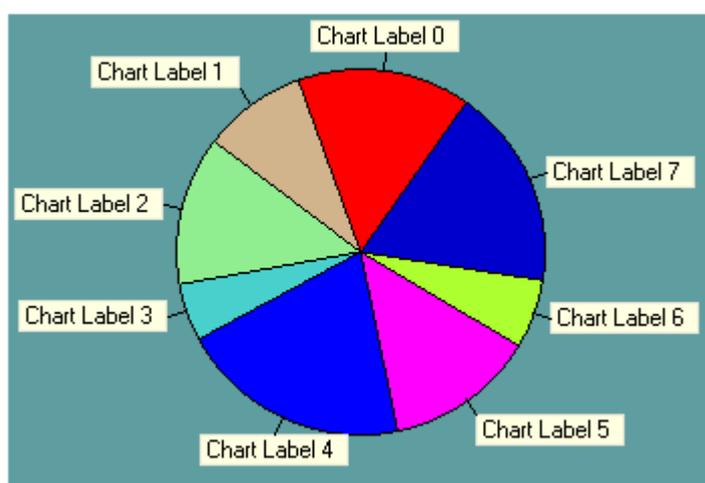
- **Hide and exclude series to create custom charts such as floating bars**



For more information on excluding and hiding data series, see [Showing, Excluding, or Hiding a Series](#).

- **Automatic creation of data labels**

C1Chart provides automatic creation for data labels through the DataLabel property. DataLabel supports a number of keywords (#TEXT, #XVAL, #YVAL, #YVAL1, #YVAL2, #YVAL3) which simplify common data labeling tasks.



For more information on adding data labels, see [Add Labels to the Chart](#).

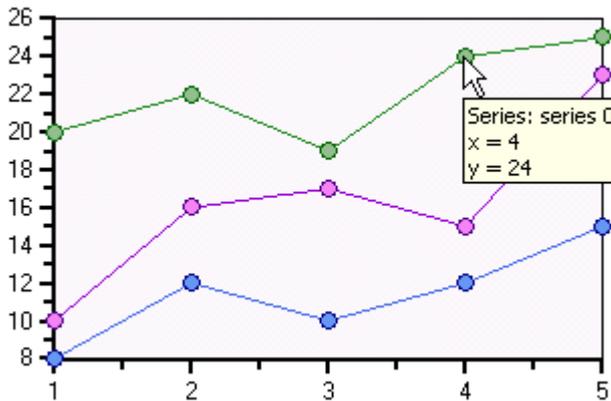
- **Flexible and interactive labels**

Chart labels and annotations can be positioned anywhere. Multi-line chart labels are unlimited and can be attached to data displayed in the graph or to graph or pixel coordinates.

For more information on using different attachment methods, see [Attach Chart Labels](#).

- **ToolTips for chart elements**

Highlight important information on C1Chart's elements using its ToolTipText property.



For more information on how to add ToolTips to chart elements, see [Add ToolTips to Chart Elements](#).

- **Custom brushes**

Use a brush for more unique appearance, including hatching, gradients, and textures.

For more information on creating custom hatched and gradient brushes, see [Custom Brushes for Plotting Data](#).

- **Advanced mouse tracking capabilities**

Provides a set of conversion methods that when used in conjunction with .NET's MouseMove event allow the programmer to keep track of the chart's region, series, or data point under the mouse pointer. This makes it easier to create interesting application specific features like handling a double-click in the legend, or chart tool tips.

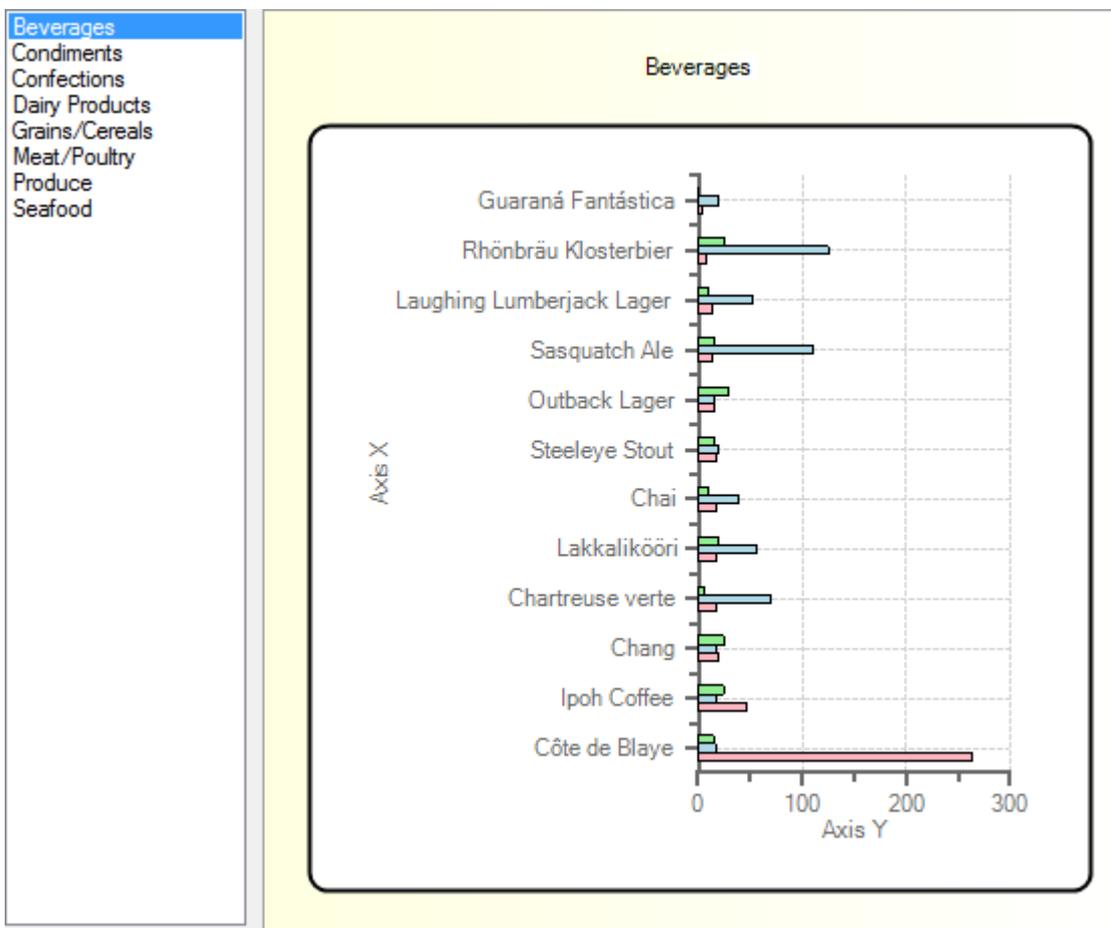
For more information on C1Chart's conversion methods, see [Coordinate Conversion Methods](#).

## Chart for WinForms Quick Start

This section provides step-by-step instructions for building a report that displays the products' unit price, units in stock, and reorder level organized by categories. The report shows the information as a simple bar chart with one y-axis that represents the names of the products and one x-axis that represents the numerical values for the products' unit price, units in stock, and reorder level. The Bar chart uses three series to draw the unit price, units in stock, and reorder level. A legend is used to show the color for each series.

The chart uses data from the sample Access database, C1NWind.mdb. This quick start assumes that the database file **C1NWind.mdb** is in the "**<PersonalDocumentsFolder>\ComponentOne Samples\Common**" where **<PersonalDocumentsFolder>** is the users **Documents** folder which varies from user to user and platform to platform.

Completing this quick start will produce a chart that resembles the following illustration:



## Step 1 of 4: Create a Data Source for Chart

In this step you will create a data source that you can later bind the chart to using the **Chart Properties** designer. Create a .NET project, and complete the following steps:

### Add a new data source

1. In the project toolbar, select **Add New Data Source** from the **PROJECT** menu. The **Data Source Configuration Wizard** dialog box appears.
2. Select **Database** and then click **Next**.

3. Select **Dataset** and then click **Next**.
4. Click **New Connection**
5. Select **Data source** as Microsoft Access Database File
6. Click browse for adding **Database file name** as C1NWind.mdb. Navigate to the default location of C1NWind.mdb database (located by default in **C:\Users\Documents\ComponentOne Samples\Common**), click **Open**, and then click **OK**.
7. Click the **Next** button to continue. A dialog box appears asking if you would like to add the data file to your project and modify the connection string. Since it is not necessary to copy the database to your project, click **No**.
8. Verify the **Yes, save the connection as** check box is checked and click **Next** to continue.

The connection string is saved as C1NWindConnectionString.

9. Expand the **Tables** node and select the **Categories** and **Products** objects.
10. Click **Finish**.

C1NWindDataSet.xsd is added to your project.

## Add an OleDbDataAdapter

11. From the Toolbox, double-click the **OleDbDataAdapter** component.

 **Note:** In Visual Studio, right-click the Toolbox, and then click **Choose Items**. On the **.NET Framework Components** tab in the dialog box, select **OleDbDataAdapter**.

The OleDbDataAdapter appears in the component tray and the **Data Adapter Configuration Wizard** appears.

12. In the **Data Adapter Configuration Wizard**, choose the connection you wish to use for the data adapter from the drop-down listbox (in this case, **C:\Users\Documents\ComponentOne Samples\Common\C1NWind.mdb**) and then click **Next**.
13. The **Use SQL statements** is selected by default, click **Next**.
14. Copy and paste the following SQL statement in the textbox of the Data Adapter Configuration Wizard:  

```
SELECT CategoryID, ProductName, UnitPrice, UnitsInStock, ReorderLevel FROM Products ORDER BY UnitPrice DESC
```
15. Click **Next** and then click **Yes** to add primary key columns to your query.
16. Click **Finish**.

Notice that the **OleDbConnection1** component is automatically inserted in the component tray.

Generate a DataSet

Generate a DataSet that is related to OleDbDataAdapter1 by completing the following steps:

17. Select **OleDbDataAdapter1** and click on its smart tag, then click **Generate DataSet**. The **Generate Dataset** dialog box appears.
18. Verify that the **Existing** radio button is selected, the **Products** table is selected, and the option **Add this dataset to the designer** is selected, and then click **OK**.

The C1NWindDataSet1 is added to the component tray.

## Add a second OleDbDataAdapter

19. From the Toolbox, double-click the **OleDbDataAdapter** component to add another **OleDbDataAdapter** component to the component tray.
20. Select the data connection that shows the directory, ACCESS. **C:\Users\Documents\ComponentOne Samples\Common\C1NWind.mdb** from the drop-down listbox.

21. Click **Next** to continue.
22. Verify the **Use SQL statements** is selected and then click **Next**.
23. Copy and paste the following SQL statement in the textbox of the Data Adapter Configuration Wizard:  

```
SELECT CategoryName, CategoryID FROM Categories
```
24. Click **Next** and then click **Finish**.

## Generate a DataSet for OleDbDataAdapter2

Generate a DataSet that is related to OleDbDataAdapter2 by completing the following steps:

25. Select **OleDbDataAdapter2** and click on its smart tag, and then click **Generate DataSet**.
26. In the **Generate DataSet** dialog box, click **New**, and then name it categoriesDataSet.
27. Verify that the Categories table is selected and that the option **Add this dataset to the designer** is selected, and then click **OK**.

The categoriesDataSet1 is added to the component tray.

## Fill the datasets

To fill the DataSets, add the following code in the Form1\_Load event:

### To write code in Visual Basic

Visual Basic

```
oleDbDataAdapter1.Fill(c1nwindDataSet1)
oleDbDataAdapter2.Fill(categoriesDataSet1)
```

### To write code in C#

C#

```
oleDbDataAdapter1.Fill(c1nwindDataSet1);
oleDbDataAdapter2.Fill(categoriesDataSet1);
```

## Add the DataView component

Return to Design view, and complete the following steps:

28. From the Toolbox, double-click the **DataView** component to add it to the component tray.



**Note:** In Visual Studio, right-click the Toolbox, and then click **Choose Items**. On the **.NET Framework Components** tab in the dialog box, select **DataView**.

29. In the **DataView** Properties window set the properties to the following:
  - **AllowDelete** = False
  - **AllowEdit** = False
  - **AllowNew** = False
  - **Table** = c1nwindDataSet1.Products

Congratulations! You have successfully created a data source. The next step will show you how to add a chart and bind it to the existing data source as well as how to easily customize your chart using the **Chart Properties** designer.

## Step 2 of 4: Bind C1Chart to the Data Source

To bind your chart to the data source that you created in the previous step, complete the following tasks:

1. From the Toolbox, double-click the **C1Chart** control to add it to the form. Resize the **C1Chart** control to be around 400x400.
2. Right-click on the chart and select **Chart Properties** from the menu. The **Chart Properties** designer appears.
3. Click **Gallery**, select the **Simple Types** tab, and then select **Bar**. In the right pane select the first bar image in the first row, and click **Apply**.
4. Click **Data** and select Series 3. Click **Remove** to remove Series 3 from the bar chart.
5. Label Series 0 as Unit Price, Series 1 as Units in Stock, and Series 2 as Reorder Level. Click **Apply**.
6. In the tree view, select **Data>Binding**, and select **dataView1** from the **Data source** drop-down list -box.
7. Click on **Data** and select the Unit Price from the **Data Series** listbox. In the Data binding, set the data field for the X axis to **ProductName** and the Y axis to **UnitPrice**.
8. Select the Units in Stock data series and set its X-axis data field to **ProductName** and Y-axis to **UnitsInStock**.
9. Select the Reorder Level data series and set its X-axis data field to **ProductName** and Y-axis data field to **ReorderLevel**.

This should bind the chart properly with DataSet. And after this if you check

**ChartGroups>Group0>ChartData>SeriesList** in the Properties Window, you see the proper data field name listed for Y Datafield.

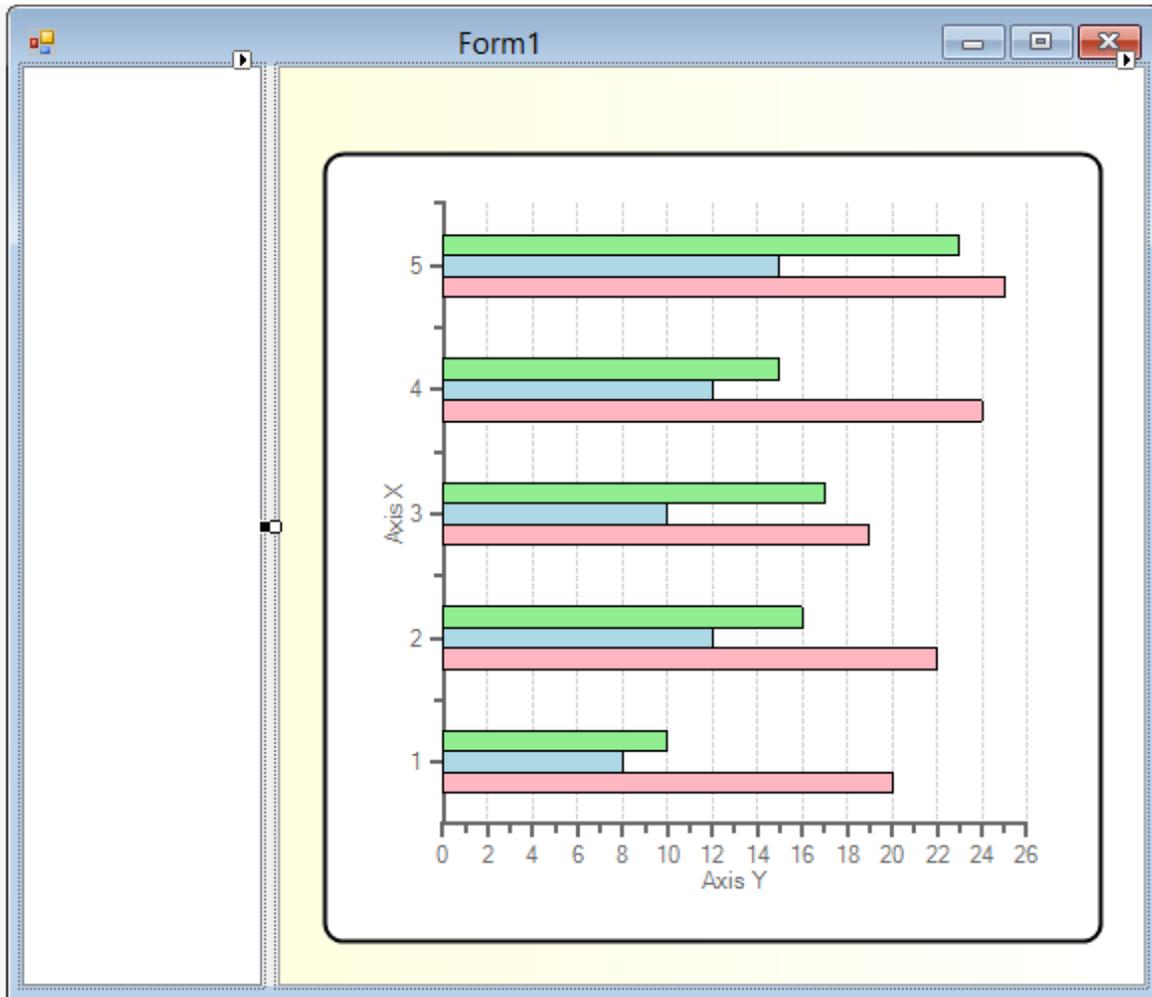
10. In the tree view, select **Appearance>Header**.
11. In the **Appearance [Header]** tab click the **Visible** checkbox to enable the chart header.
12. Select **OK** to close the **Chart Properties**.
13. Run the application.

You have successfully connected the chart to the data source and configured the chart's settings. To make our data more readable on the chart, in the next step we will add a listbox to filter the categories from the data.

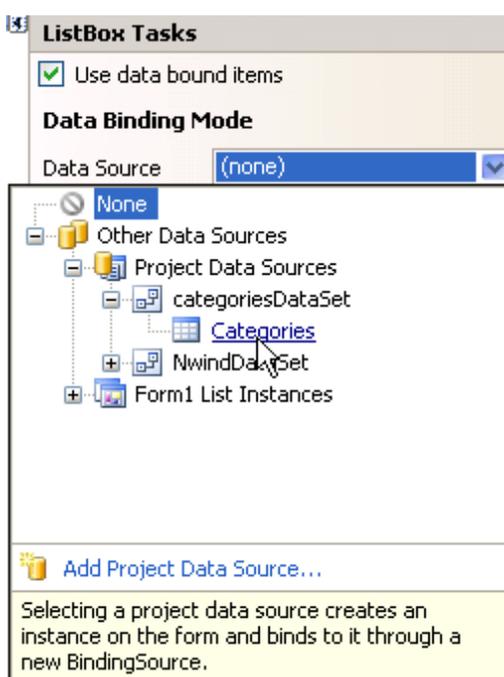
## Step 3 of 4: Bind the ListBox to the DataSet

To filter the categories from the data, complete the following steps:

1. From the Toolbox, double-click the **ListBox** control to add it to the form. Dock it to the left of the **C1Chart** control so it appears like the following:



2. Select the **ListBox** control and click on its smart tag to open the menu. Select **Use Data Bound items** and then in the **Data Source** drop-down listbox, select **Categories** from **Other Data Sources>Project Data Sources>categoriesDataSet**.



- Set the **DisplayMember** to **CategoryName**.
- Double-click on the **ListBox** to create a `listbox1_SelectedIndexChanged` event.
- Add the following code in the **listbox1\_SelectedIndexChanged** event to filter the `CategoryID` to the listbox when the user selects a category item:

**To write code in Visual Basic**

## Visual Basic

```
Private Sub listBox1_SelectedIndexChanged(sender As Object, e As
System.EventArgs) Handles listBox1.SelectedIndexChanged
    If listBox1.SelectedIndex >= 0 Then
        Dim categoryID As String =
Me.categoriesDataSet1.Categories(listBox1.SelectedIndex).CategoryID.ToString()
        Me.dataView1.RowFilter = "CategoryID = " + categoryID
        Me.c1Chart1.Header.Text = listBox1.Text
    End If
End Sub
```

**To write code in C#**

## C#

```
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    if (listBox1.SelectedIndex >= 0)
    {
        string categoryID =
this.categoriesDataSet1.Categories[listBox1.SelectedIndex].CategoryID.ToString();
        this.dataView1.RowFilter = "CategoryID = " + categoryID;
        this.c1Chart1.Header.Text = listBox1.Text;
    }
}
```

- In the **Form1\_Load** event add the following code to force the new calculation after the refill so the first category of product items, Beverages, appears rather than all of the unfiltered categories:

**To write code in Visual Basic**

## Visual Basic

```
'force the new calculation after the refill
listBox1_SelectedIndexChanged(Me.listBox1, New EventArgs())
```

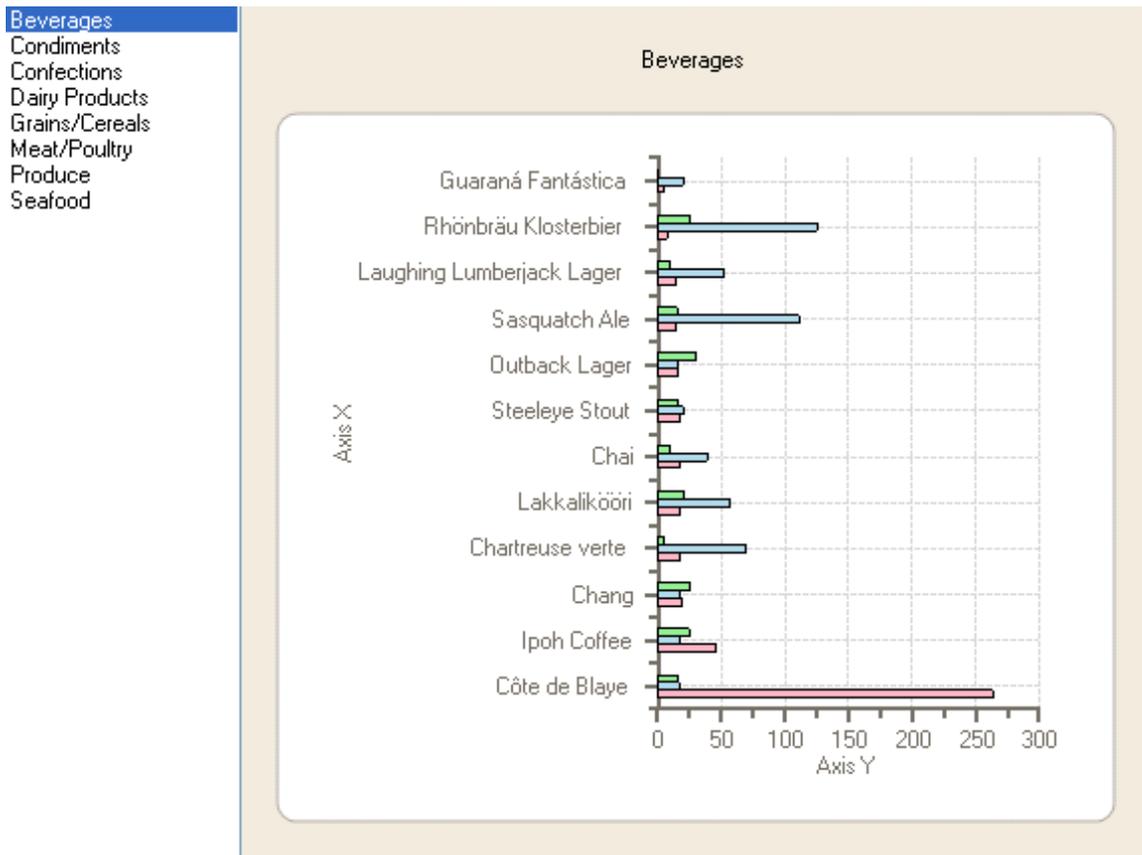
**To write code in C#**

## C#

```
//force the new calculation after the refill
listBox1_SelectedIndexChanged(this.listBox1, new EventArgs());
```

- Run the application and select a category from the listbox to observe the chart filter the data.

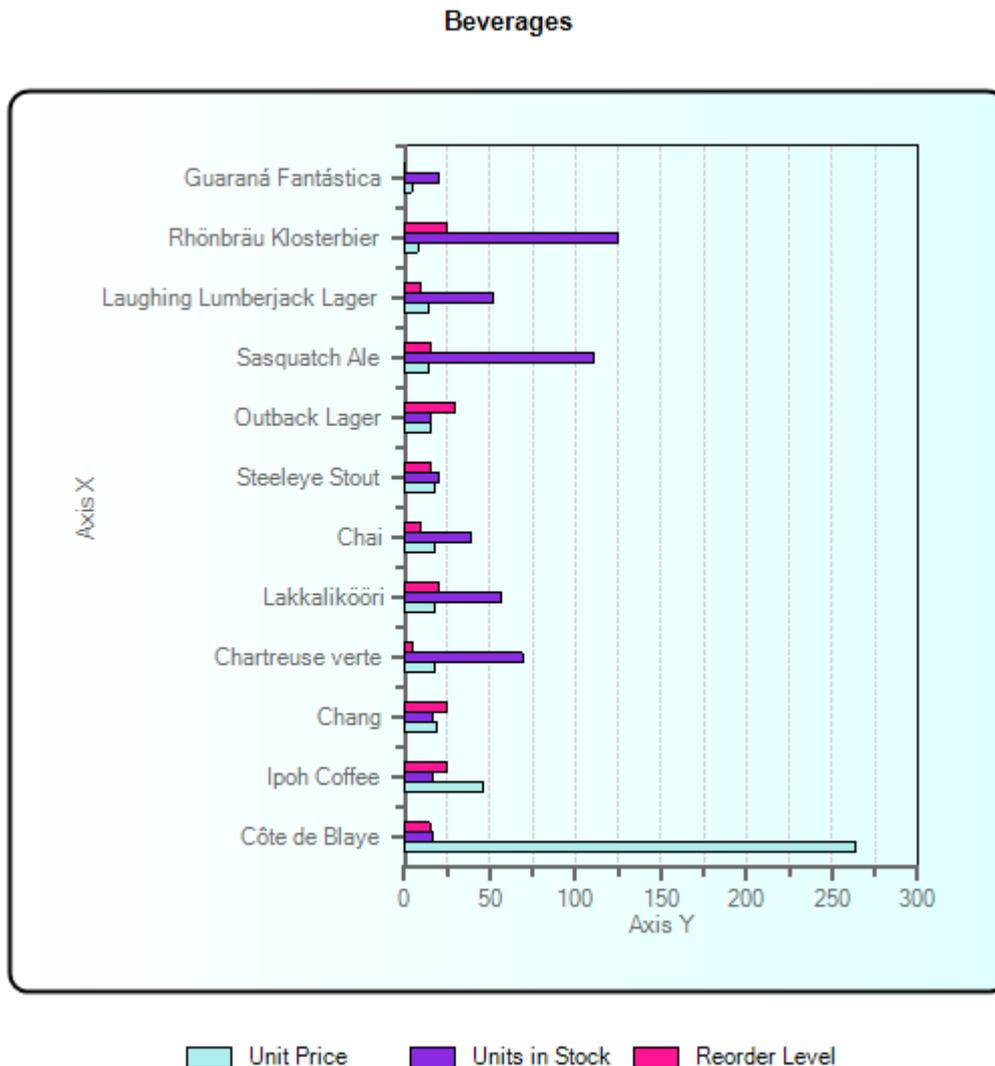
Congratulations! You successfully bound data to the chart. In the next step you will modify the appearance of the chart.



## Step 4 of 4: Customize your Chart

In this section you'll learn how to make your chart more visually appealing and professional looking by using the built-in formatting effects.

After finishing this tutorial your data bound chart will appear like the following:



**C1Chart** formats the chart's back color as a control color and the plot area as a white color, but you can choose another color, use no color at all, or be creative and use two colors to create gradient effects.

## Change the color for the data series

1. Right-click the **C1Chart** control and select **Chart Properties**.
2. In the **Chart Properties** designer, select **Data**.
3. In the **ChartData** tab, select each series from the **Data Series** group box and click on the Fill drop-down checkbox to choose a specific fill color.
4. Select **Simple** for the fill type and set the colors for the Unit Price to **PaleTurquoise**, the Units in Stock to **BlueViolet**, and the Reorder Level to **DeepPink**.
5. Click **OK**.

Notice how the contrast between the light and dark colors in the series makes it easier to see the difference between the data series.

To make more room for the data in the chart we can adjust the position of the chart legend.

## Modify the legend position

6. Right-click the **C1Chart** control and select **Chart Properties**.

7. In the **Chart Properties** designer, select **Appearance>Legend**.
8. Click on the **Position** drop-down checkbox and select **South**.
9. Click **OK**.

This helps free up horizontal space so the chart looks more appealing and easily readable.

To make the chart look less cluttered we can remove the horizontal gridlines since we only need the vertical gridlines to help us read the data.

## Remove the horizontal gridlines

10. Right-click the **C1Chart** control and select **Chart Properties**.
11. In the **Chart Properties** designer, select **AxisX>Gridlines**.
12. In the **Major grid** group, uncheck **Visible**.

The AxisX gridlines are removed from the chart.

13. Click **OK**.

To make the text stand out more in the chart you can change the chart's font style and color.

## Change the chart header's font style

14. Right-click the **C1Chart** control and select **Chart Properties**.
15. In the **Chart Properties** designer, select **Appearance>Header**.
16. Click on the **ForeColor** drop-down listbox and select **Black**.
17. Click on the **Font** button to open the **Font** dialog box. Change the font to **Arial**, its style to **Bold**, and its size to **10**.
18. Click **OK**.

Next, we should change the chart's border color from control to black to make it consistent with the chart's text color.

## Change the chart's border color

19. Right-click the **C1Chart** control and select **Chart Properties**.
20. In the **Chart Properties** designer, select **Appearance>ChartArea**.
21. In the **Appearance [ChartArea]** tab, click the **Border** drop-down checkbox and select **Solid** for the style, **Black** for the color, and **2** for the thickness.
22. In the **Appearance [PlotArea]** tab, check **Boxed**.
23. Click **OK**.

To add some style to the chart, we can add some gradient fill effects to the entire chart.

## Add gradient fill effects to the chart

23. Right-click the **C1Chart** control and select **Chart Properties**.
24. In the **Chart Properties** designer, select **Appearance>ChartArea**.
25. In the **Appearance [ChartArea]** tab, click the **Fill** drop-down checkbox and select the **Gradient** option. Several gradient options appear in the groupbox. Select the first gradient and choose **White** for Color1 and **LightCyan** for Color2, and then click **OK**.

Run the application and observe the chart's new appearance.

Congratulations, you have completed the C1Chart quick start! In this quick start you have learned how to:

- Bind C1Chart statically to a data table.
- Customize C1Chart's properties using the **Chart Properties** designer.

## Design-Time Support

C1Chart provides visual designers that offer rich design-time support and simplify working with the object model.

You have complete control to create a powerful and enhancing chart by using any of the following designers, menus, and collection editors in Visual Studio:

- Smart Tag
- Chart Wizard
- Chart Properties Designer
- Visual Effects Designer
- C1Chart's Collection Editors

This chapter describes how to use various tools available in C1Chart's design-time environment to configure the C1Chart control.

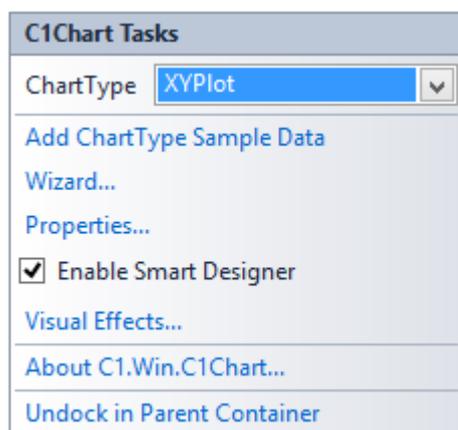
In addition to the Visual Studio design-time support that C1Chart provides, it also has special design-time tools that help you create a simple or complex chart in very little time. For additional information on the special design-time tools see, [Design-Time Tools for Creating 2D Charts](#).

## C1Chart Smart Tag

In Visual Studio, the **C1Chart** component includes a smart tag. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in each component/command.

The **C1Chart** component provides quick and easy access to the **Chart Wizard** editor, **Visual Effects** designer, and common properties through its smart tag.

To access the **C1Chart Tasks** menu, click the smart tag (☰) in the upper-right corner of the **C1Chart** control. This will open the **C1Chart Tasks** menu.



The **C1Chart Tasks** menu operates as follows:

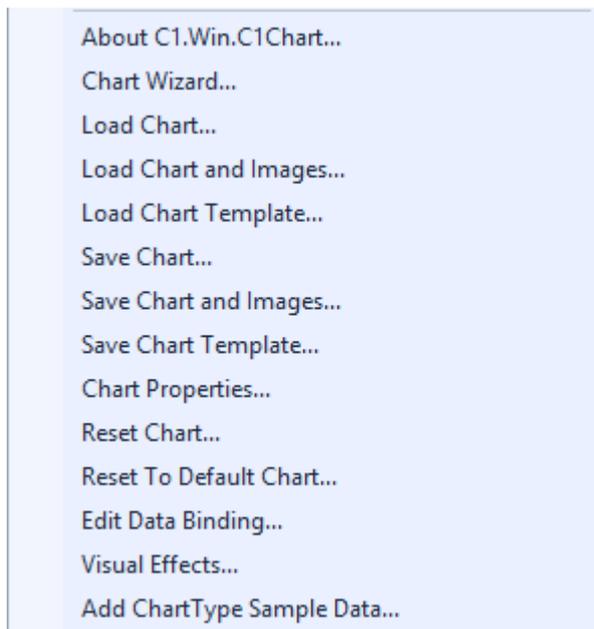
- **Add ChartType Sample Data**  
Clicking the **Add ChartType Sample Data** item opens a **Warning** dialog box that says the current data will be lost. Once you select **Yes** to continue it will change the current or default data to the sample data for the selected chart type.
- **Wizard**  
Clicking the **Wizard** item opens the **Chart Wizard** editor. For more information about the elements in the **Chart Wizard** dialog box and how to use them, see [Working with the Chart Wizard](#).
- **Properties**  
Clicking the **Properties** item opens the **Chart Properties** designer. For more information about the elements in the **Chart Properties** designer and how to use them, see [Working with the Chart Properties Designer](#).

- **Enable Smart Designer**  
Selecting the **Enable Smart Designer** check box enables the Smart Designer of the **C1Chart** control. The default value is **True** (checked). For more information about the Smart Designer's elements see, [Working with the Smart Designer](#).
- **Visual Effects**  
Clicking the **Visual Effects** item opens the **Visual Effects** designer. For more information about the elements in the **Visual Effects** designer and how to use them, see [Visual Effects Designer](#).
- **About C1.Win.C1Chart**  
Clicking the **About** item displays a dialog box, which is helpful in finding the version number of C1Chart and online resources.
- **Dock/Undock in Parent Container**  
Clicking the Dock in parent container docks the **C1Chart2D** control inside its parent container. Clicking Undock in Parent Container undocks the control.

## C1Chart Context Menu

The **C1Chart2D** control provides a context menu for additional functionality to use at design time.

To access C1Chart2D's context menu, right-click on the **C1Chart2D** component.



The **C1Chart** context menu operates as follows:

- **About C1.Win.C1Chart**  
Displays a dialog box, which is helpful in finding the version number of C1Chart and online resources.
- **Chart Wizard**  
Opens the **Chart Wizard**.
- **Load Chart**  
Loads the saved layout of the C1Chart2D control.
- **Save Chart**  
Saves the layout of the C1Chart2D control as a XML file.
- **Chart Properties** Opens the **Chart Properties** designer.
- **Reset Chart**  
Resets the Chart.
- **Reset To Default Chart**  
Resets the Chart back to its default settings.
- **Edit Data Binding**

- Opens the C1Chart Data Binding editor.
- **Visual Effects**  
Opens the **Visual Effects** designer.

## C1Chart Collection Editors

**C1Chart** provides the following collection editors that allow you to apply properties to the Chart elements at design time:

- [Action Collection Editor](#)
- [AlarmZone Collection Editor](#)
- [Axis Collection Editor](#)
- [ChartDataSeries Collection Editor](#)
- [ChartGroup Collection Editor](#)
- [FunctionBase Collection Editor](#)
- [Labels Collection Editor](#)
- [PointStyle Collection Editor](#)
- [ScaleMenuItem Collection Editor](#)
- [TrendLine Collection Editor](#)
- [ValueLabel Collection Editor](#)
- [VisualEffectsStyle Collection Editor](#)

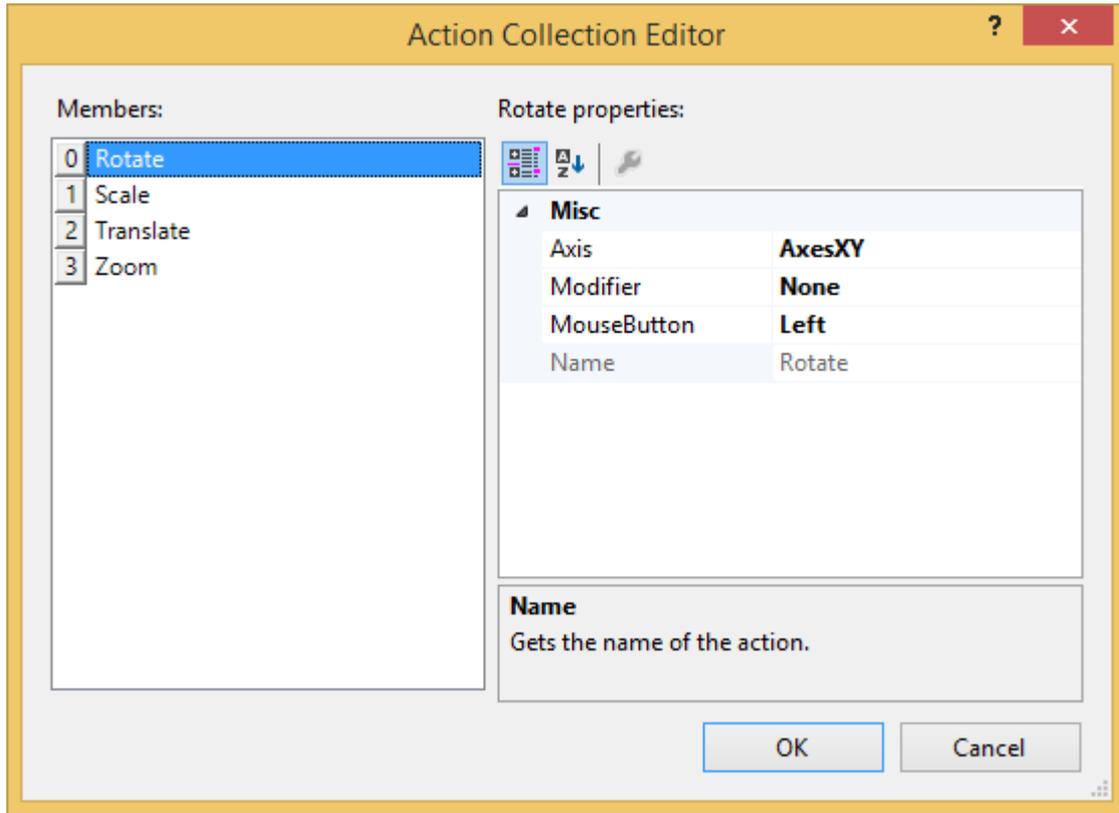
The main part for each of the editor's application consists of a windows form which conveniently allows the user to edit C1Chart's elements.

## Action Collection Editor

The **Action Collection Editor** is used for setting up the rotation, scaling, translation, and zooming interactions in the chart at design time. For more information on rotation, scaling, translation, and zooming interactions, see [Rotating, Scaling, Translating, and Zooming](#).

## To Access the Action Collection Editor

1. Right-click the **C1Chart** control and select Properties from its context menu.
2. In the Properties window, expand the **Interaction** node, then click on the **ellipsis** button next to the **Actions** property. The **Action Collection Editor** appears.



### Properties available in the Action Collection Editor

The following properties are available for the user in the **Action Collection Editor** at design time or they can be used in the Interaction class at run time:

Members	Description
<a href="#">Axis</a>	Gets or sets the axis or axes used in transformation action.
<a href="#">Modifier</a>	Gets or sets the key modifier.
<a href="#">MouseButton</a>	Gets or sets the mouse button that starts this action.
<a href="#">Name</a>	Gets the name of the action.

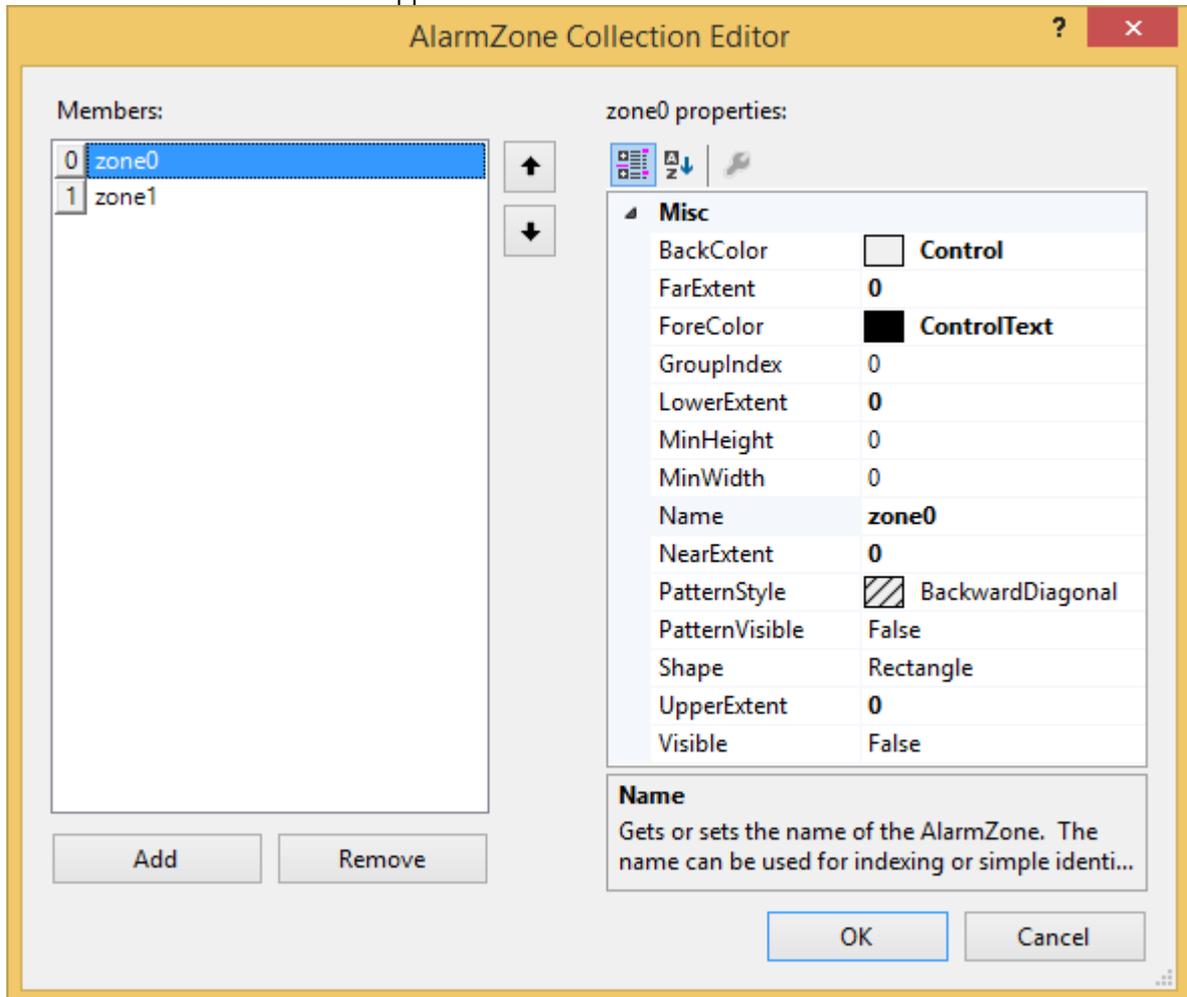
## AlarmZone Collection Editor

The **Alarm Zones Collection Editor** is used for adding or modifying alarms to the plot area at design time. For more information on Alarm Zones, see [Alarm Zones](#).

### To Access the AlarmZone Collection Editor

1. Right-click the **C1Chart** control and select Properties from its context menu.
2. In the Properties window, expand the **ChartArea** node in the **Properties** window, then expand the **PlotArea** node, and click the ellipsis next to the **AlarmZones** property.

The **AlarmZone Collection Editor** appears.



### Properties available in the AlarmZones Collection Editor

The following properties are available for the user in the **AlarmZone Collection Editor** at design time or they can be used in the AlarmZone class at run time:

Members	Description
<a href="#">BackColor</a>	Gets or sets the background color of the AlarmZone. Inherits from PlotArea.
<a href="#">FarExtent</a>	Gets or sets the far extent or the AlarmZone in the X-axis data coordinates.
<a href="#">ForeColor</a>	Gets or sets the foreground color of the AlarmZone. Inherits from the PlotArea.
<a href="#">GroupIndex</a>	Gets or sets the index of the data group of the AlarmZone.
<a href="#">LowerExtent</a>	Gets or set the lower extent of the AlarmZone in the Y-axis data coordinates.
<a href="#">MinHeight</a>	Gets or sets the AlarmZone minimum pixel height.
<a href="#">MinWidth</a>	Gets or sets the AlarmZone minimum pixel width.
<a href="#">Name</a>	Gets or sets the name of the AlarmZone. The name can be used for indexing or simple identification.
<a href="#">NearExtent</a>	Gets or sets the near extent of the AlarmZone in the X-axis data coordinates.

<a href="#">PatternStyle</a>	Gets or sets the pattern style to be used if PatternVisible is True. The PatternStyle is panned in the AlarmZone ForeColor.
<a href="#">PatternVisible</a>	Gets or sets whether the specified PatternStyle should be used to brush the background of the AlarmZone
<a href="#">Shape</a>	Gets or sets the shape of the AlarmZone.
<a href="#">UpperExtent</a>	Gets or sets the upper extent of the AlarmZone in the Y-axis data coordinates.
<a href="#">Visible</a>	Gets or sets the visibility of the AlarmZone.

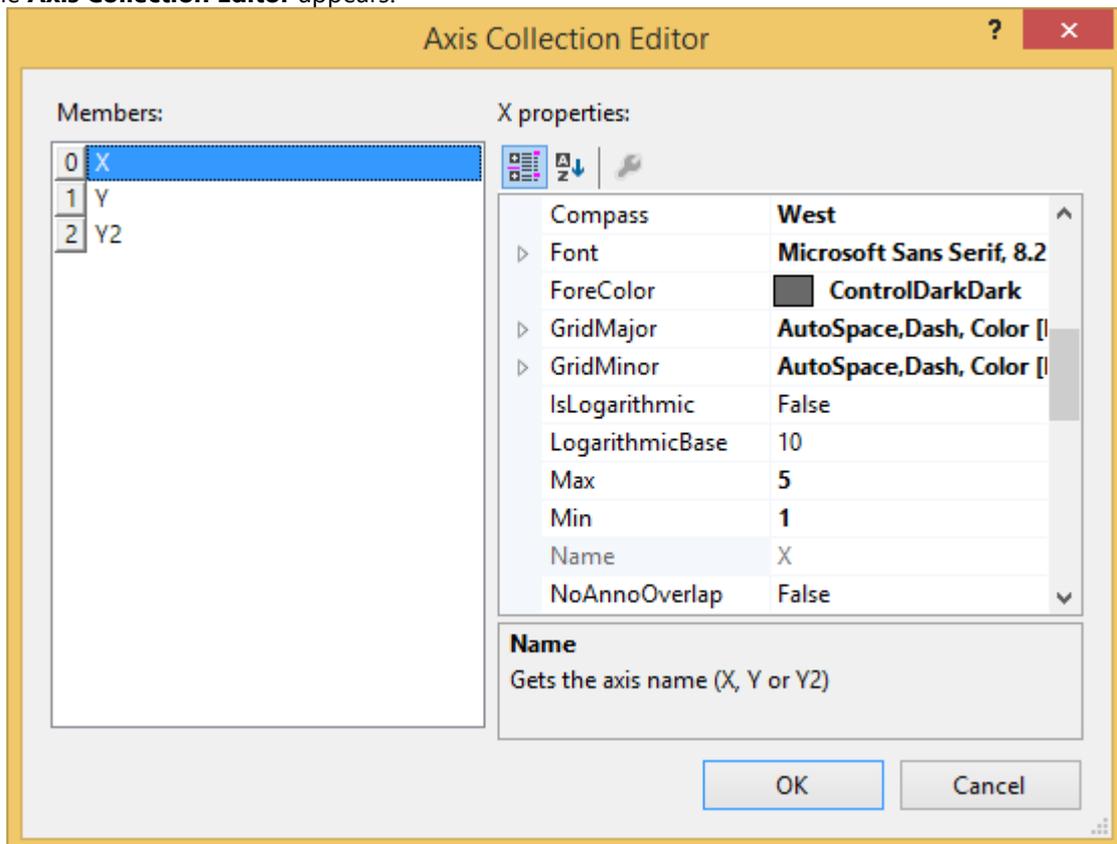
## Axis Collection Editor

The **Axis Collection Editor** is used for modifying the properties for the X, Y, or Y2 axes in C1Chart. For more information on the Axes properties see, [Axes](#).

### To Access the Axis Collection Editor

1. Right-click the **C1Chart** control and select Properties from its context menu.
2. In the **Properties** window, expand the **ChartArea** node, then click on the **ellipsis** button next to the **Axes** property.

The **Axis Collection Editor** appears.



### Properties available in the Axis Collection Editor

The following properties are available for the user in the **Axis Collection Editor** at design time or they can be used in the Axis class at run time:

#### Misc Properties of the Axis Collection Editor:

Members	Description
<a href="#">Alignment</a>	Gets or sets the text alignment within the axis display. Inherits from the ChartArea.
<a href="#">AnnoFormat</a>	Gets or sets the annotation format.
<a href="#">AnnoFormatString</a>	Gets or sets the annotation format string used with manual formats.
<a href="#">AnnoMethod</a>	Gets or sets the method of annotation.
<a href="#">AnnotationRotation</a>	Gets or sets the clockwise angle of rotation for the axis annotations.
<a href="#">AutoMajor</a>	Gets or sets whether major tickmark values are calculated automatically.
<a href="#">AutoMax</a>	Gets or sets whether the axis maximum value is calculated automatically.
<a href="#">AutoMin</a>	Gets or sets whether the axis minimum value is calculated automatically.
<a href="#">AutoMinor</a>	Gets or sets whether minor tickmark values are calculated automatically.
<a href="#">AutoOrigin</a>	Gets or sets whether the axis origin is calculated automatically.
<a href="#">Compass</a>	Gets or sets the general positioning of the axis. X may be set to North/South, Y and Y2 may be set to East/West.
<a href="#">Font</a>	Gets or sets the outline width of the data point symbol.
<a href="#">ForeColor</a>	Gets or sets the forecolor. Inherits from the ChartArea.
<a href="#">GridMajor.AutoSpace</a>	Gets or sets the automatic gridline spacing calculation.
<a href="#">GridMajor.Color</a>	Gets or sets the color of the line.
<a href="#">GridMajor.Pattern</a>	Gets or sets the pattern of the line.
<a href="#">GridMajor.Spacing</a>	Gets or sets the gridline spacing in data coordinate units.
<a href="#">GridMajor.Thickness</a>	Gets or sets the thickness of the line.
<a href="#">GridMinor.AutoSpace</a>	Gets or sets automatic gridline spacing calculation.
<a href="#">GridMinor.Color</a>	Gets or sets the color of the line.
<a href="#">GridMinor.Pattern</a>	Gets or sets the pattern of the line.
<a href="#">GridMinor.Spacing</a>	Gets or sets gridline spacing in data coordinate units.
<a href="#">GridMinor.Thickness</a>	Gets or sets the thickness of the line.
<a href="#">GridMinor.Visible</a>	Gets or sets the gridline visibility.
<a href="#">IsLogarithmic</a>	Gets or sets whether the axis is logarithmic.
<a href="#">LogarithmicBase</a>	Gets or sets the base of the logarithmic scale used. Les than or equal to 1 specifies natural logs.
<a href="#">Max</a>	Gets or sets the maximum value of the axis.
<a href="#">Min</a>	Gets or sets the minimum value of the axis.
<a href="#">Name</a>	Gets the axis name (X, Y or Y2)
<a href="#">NoAnnoOverlap</a>	Gets or sets whether axis annotations are permitted to overlap.
<a href="#">OnTop</a>	Gets or sets whether axis and gridlines appear on top of the chart image.

Origin	Gets or sets the axis origin.
Reversed	Gets or sets whether the axis is normal or reversed (ascending or descending).
Rotation	Gets or sets the rotational orientation of the textual caption of the axis.
ScrollBar	Gets the axis scroll bar.
ScrollBar.Alignment	Gets or sets the alignment of the scroll bar relative to the plot area.
ScrollBar.Anchored	Indicates whether or not axis scrollbars should be fixed to the PlotArea boundary or move with the axis origin.
ScrollBar.Appearance	Gets or sets the appearance of the scroll bar.
ScrollBar.Buttons	Gets or sets the buttons of the scroll bar.
ScrollBar.Max	Gets or sets the maximum value of the scroll bar position.
ScrollBar.Min	Gets or sets the minimum value of the scroll bar position.
ScrollBar.Scale	Gets or sets the scale of the scroll bar.
ScrollBar.ScaleMenu	Gets or sets the custom context menu that will be shown when the user clicks on the scale button.
ScrollBar.ScaleMenuItems	Gets the collection of scale menu items.
ScrollBar.ScrollKeys	Gets or sets the keys that the scroll bar responds to.
ScrollBar.Size	Gets or sets the size of the scroll bar.
ScrollBar.Step	Gets or sets the step of the scroll bar position changing.
ScrollBar.Value	Gets or sets a value that represents the current relative position of the scroll box on the scroll bar.
ScrollBar.Visible	Gets or sets the scroll bar visibility.
Text	Gets or sets the textual caption of the axis.
Thickness	Gets or sets the thickness of the axis in pixels.
TickFactorMajor	Gets or sets an integral factor for major tick mark length.
TickFactorMinor	Gets or sets an integral factor for minor tick mark length.
TickGauge	Gets or sets the approximate number of intervals delineated by gauge marks between major tick marks.
TickLabels	Gets or sets the placement of the annotation labels relative to the axis. (currently not implemented)
TickMajor	Gets or sets the type of major tickmark.
TickMinor	Gets or sets the type of minor tickmark.
TooltipText	Gets or sets the tooltip text.
UnitMajor	Gets or sets the units between major tick marks. Setting this value turns off AutoMajor.
UnitMinor	Gets or sets the units between minor tick marks. Setting this value turns off AutoMinor.
ValueLabels	Gets the ValueLabels collection for this axis.
VerticalText	Gets or sets whether the label text is displayed vertically.

Visible

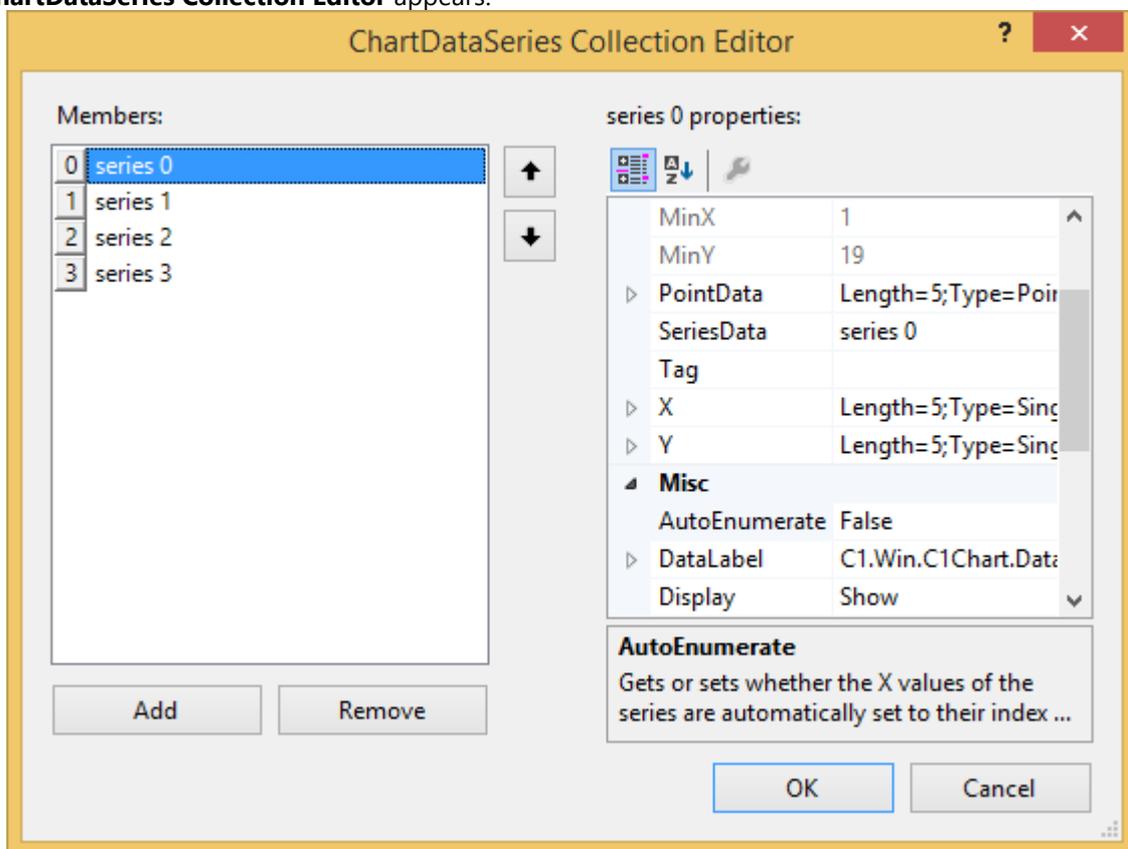
Gets or sets the Axis visibility.

## ChartDataSeries Collection Editor

The **ChartDataSeries Collection Editor** allows the user to add or remove data series and modify their properties. For more information on the ChartDataSeries, see [Defining the ChartDataSeries Object](#).

### To Access the ChartDataSeries Collection Editor

1. Right-click the **C1Chart** control and select Properties from its context menu.
2. In the **Properties** window, expand the **ChartGroups** node, then expand the Group0 or Group1 node.
3. Expand the **ChartData** node, then click the **ellipsis** button next to the **SeriesList** property. The **ChartDataSeries Collection Editor** appears.



### Properties available in the ChartDataSeries Collection Editor

The following table includes the name and description of the properties available for the user in the **ChartDataSeries Collection Editor** at design time or they can be used in the ChartDataSeries class at run time:

Members	Description
<a href="#">ChartDataSeries.Length</a>	Gets the number of data points in the series.
<a href="#">ChartDataSeries.MaxX</a>	Returns the maximum X value of the series data.
<a href="#">ChartDataSeries.MaxY</a>	Returns the maximum Y value of the series data.
<a href="#">ChartDataSeries.MinX</a>	Returns the minimum X value of the series point data array.
<a href="#">ChartDataSeries.MinY</a>	Returns the minimum Y value of the series point data array.

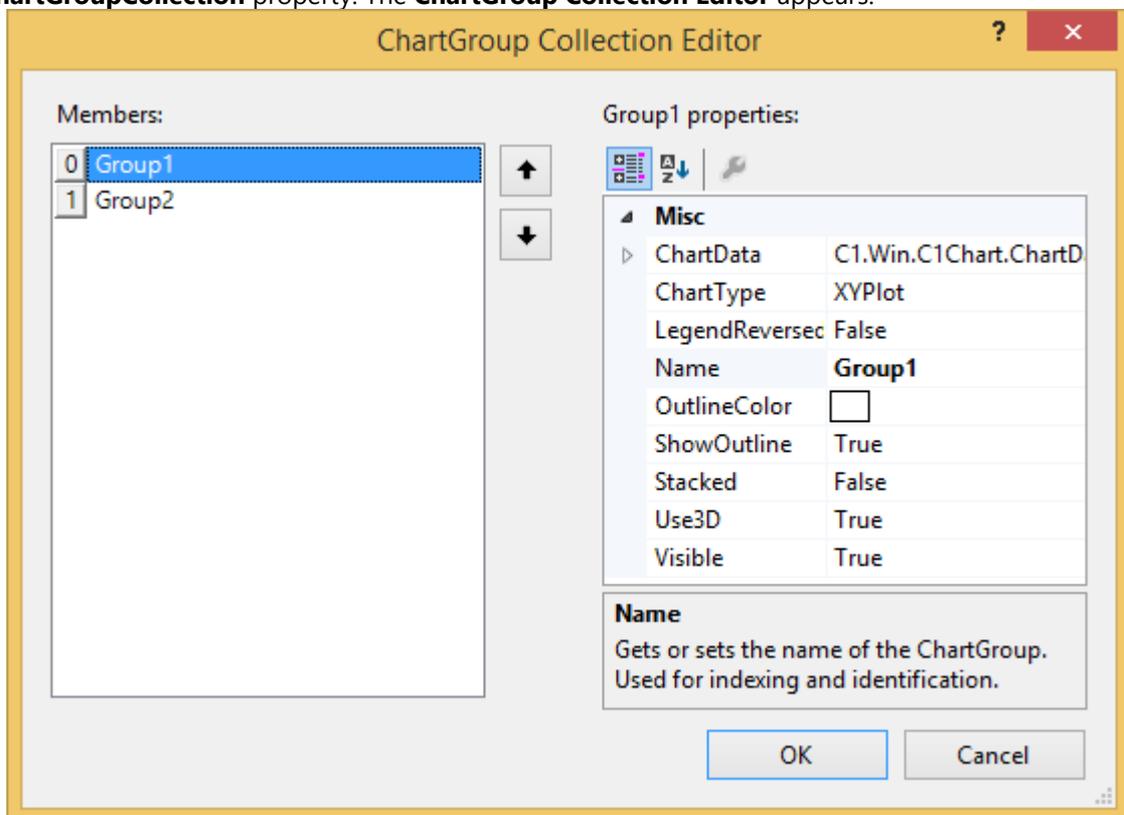
<a href="#">ChartDataSeries.PointData</a>	Gets the ChartDataArray object that manages point data (combined X and Y data).
<a href="#">StatisticalData.PropertyGridEnabled</a>	Gets or sets whether statistic calculations are reported in property grids and by reflection.
<a href="#">StatisticalData.DataStatus</a>	Gets a string indicating whether data is available for statistical calculations.
<a href="#">ChartDataSeries.Tag</a>	Gets or sets the user data for the series.
<a href="#">ChartDataArray.DataField</a>	The data field that is bound to.
<a href="#">ChartDataArray.DataType</a>	Gets or sets the external datatype.
<a href="#">ChartData.Hole</a>	Gets the data hole value.
<a href="#">ChartDataSeries.Length</a>	Gets or sets the number of elements of the ChartDataArray.

## ChartGroup Collection Editor

The **ChartGroup Collection Editor** allows the user to set or modify Group1 and Group2's properties. For more information on the ChartGroup, see [Defining the ChartGroup Object](#).

### To Access the ChartGroup Collection Editor

1. Right-click the **C1Chart** control and select Properties from its context menu.
2. In the Properties window, expand the **ChartGroups** node, click on the **ellipsis** button next to the **ChartGroupCollection** property. The **ChartGroup Collection Editor** appears.



### Properties available in the ChartGroup Collection Editor

The following table includes the name and description of the properties available for the user in the **ChartGroup Collection Editor** at design time or they can be used in the ChartGroup class at run time:

Members	Description
<a href="#">ChartData</a>	Gets the number of data points in the series.
<a href="#">ChartData.FunctionsList</a>	Returns the maximum X value of the series data.
<a href="#">ChartData.HighLight</a>	Returns the maximum Y value of the series data.
<a href="#">DataHighlight.Activation</a>	Returns the minimum X value of the series point data array.
<a href="#">DataHighlight.Appearance</a>	Returns the minimum Y value of the series point data array.
<a href="#">ChartDataSeries.PointData</a>	Gets the ChartDataArray object that manages point data (combined X and Y data).
<a href="#">StatisticalData.PropertyGridEnabled</a>	Gets or sets whether statistic calculations are reported in property grids and by reflection.
<a href="#">StatisticalData.DataStatus</a>	Gets a string indicating whether data is available for statistical calculations.
<a href="#">ChartDataSeries.Tag</a>	Gets or sets the user data for the series.
<a href="#">ChartDataArray.DataField</a>	The data field that is bound to.
<a href="#">ChartDataArray.DataType</a>	Gets or sets the external datatype.
<a href="#">ChartData.Hole</a>	Gets the data hole value.
<a href="#">ChartDataSeries.Length</a>	Gets or sets the number of elements of the ChartDataArray.

## FunctionBase Collection Editor

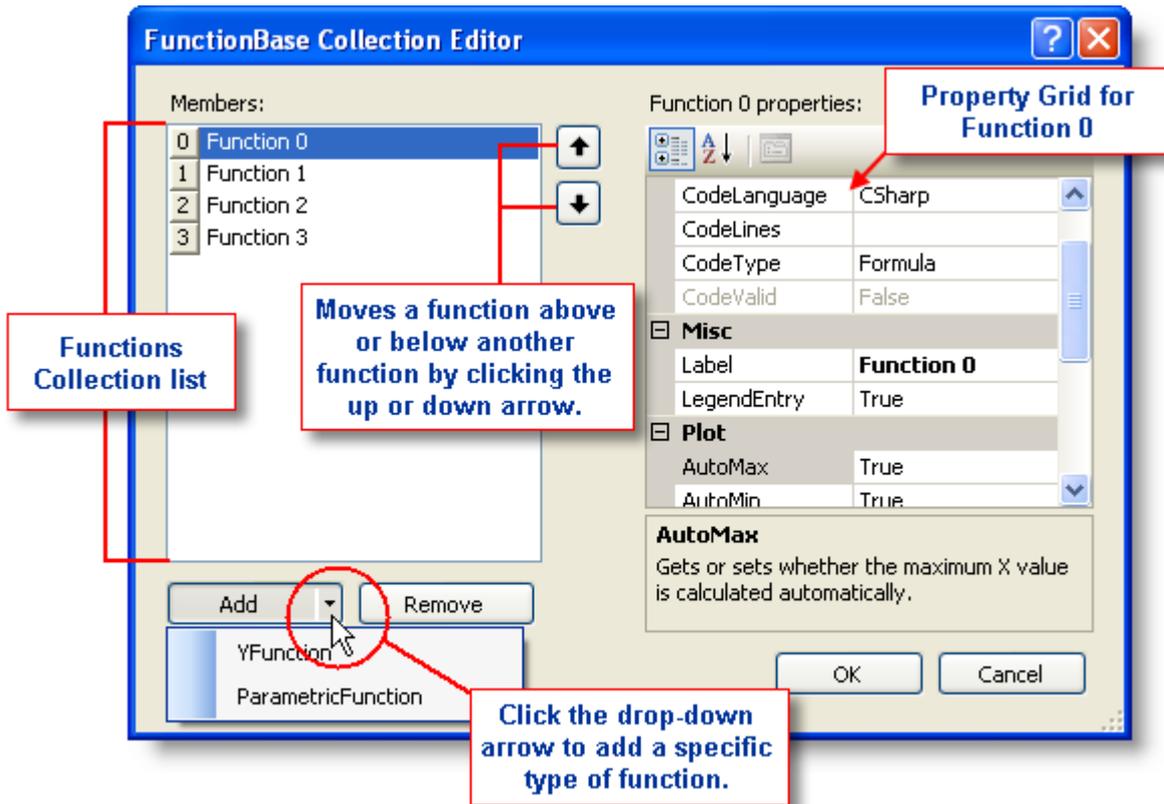
The **FunctionBase Collection Editor** is used for adding or modifying functions for plotting data at design time. For more information on using the functions for plotting data, see [Plotting Functions](#).

### To Access the FunctionBase Collection Editor

1. Right-click the **C1Chart** control and select **Properties** from its context menu.
2. Expand the **ChartGroups** node in the Properties window.
3. Expand the **Group0** node and then expand the **ChartData** node.
4. Select the **FunctionsList** property and click the **ellipsis** button. The **FunctionBase Collection Editor** appears.

#### Base parts of the FunctionBase Collection Editor

The following figure illustrates the elements in the **FunctionBase Collection Editor**:



The following properties are available for the user in the **FunctionBase Collection Editor** at design time or they can be used in the FunctionBase class at run time:

### Available Properties in the FunctionBase Collection Editor

The code properties specify the type and language of the function's code as well as the multi-line presentation for the code. The code properties and their descriptions are listed below:

Members	Description
<a href="#">CodeErrors</a>	Gets the string descriptions of any compiler errors.
<a href="#">CodeLanguage</a>	Gets or sets the programming language used for compiling VB or C#. The user can specify which language by clicking on the drop-down button in the Code Language textbox and selecting the language.
<a href="#">CodeLines</a>	Gets or sets the multi-line presentation of the function code. By clicking the ellipsis button you can enter code into the listbox of the collection string editor.
<a href="#">CodeType</a>	Gets or sets whether the code will be compiled as a formula, method, or as a full compile unit.
<a href="#">CodeValid</a>	Gets whether the function compiles correctly.

### Misc Properties of the FunctionBase Collection Editor:

The miscellaneous property and their functions are listed below:

Members	Description
<a href="#">Label</a>	Gets or sets the label of the function.
<a href="#">LegendEntry</a>	Gets or sets whether the function will be shown in the legend.

## Plot Properties of the FunctionBase Collection Editor:

The plot properties are used to modify the style or functionality of the chart data series plots used in your function. The plot properties and their descriptions are listed below:

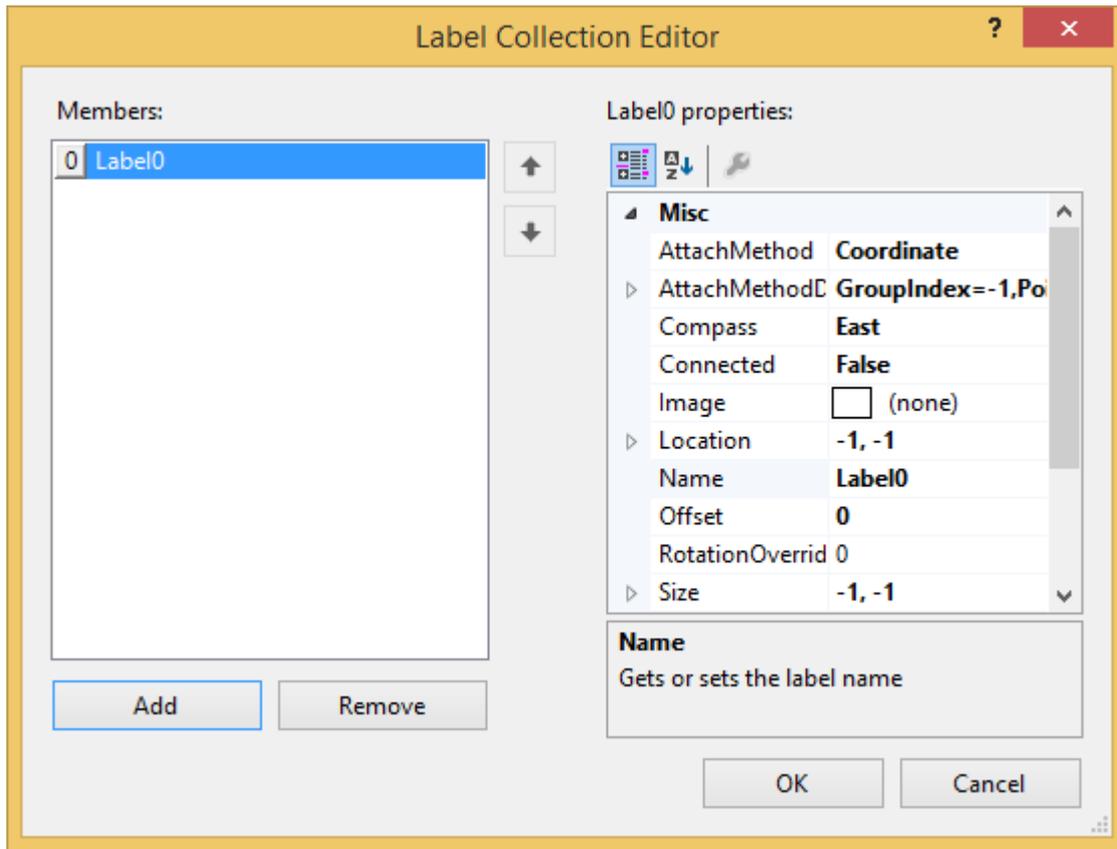
Members	Description
<a href="#">AutoMax</a>	Gets or sets whether the maximum X value is calculated automatically.
<a href="#">AutoMin</a>	Gets or sets whether the minimum X value is calculated automatically.
<a href="#">LineStyle</a>	Gets or sets the line style used for plotting the function.
<a href="#">MaxX</a>	Gets or sets the maximum X value of the plotted function.
<a href="#">MinX</a>	Gets or sets the minimum X value of the plotted function.
<a href="#">PlotLinesMethod</a>	Gets or sets the method used for plotting the function.
<a href="#">PlotNumPoints</a>	Gets or sets the number of points used when plotting the function.
<a href="#">PlotOverData</a>	Gets or sets whether the function is plotted over or behind the data series plots.
<a href="#">Visible</a>	Gets or sets whether the function plot is visible.

## Label Collection Editor

The **Label Collection Editor** is used for adding or modifying Chart labels on the data series. ChartLabels are used for labeling an important data point on the data series. For more information on using chart labels, see [Charting Labels](#).

## To Access the Label Collection Editor

1. Right-click the **C1Chart** control and select **Properties** from its context menu.
2. In the Properties window, expand the **ChartLabels** node, then click on the ellipsis button next to the LabelsCollection property. The **Label Collection Editor** opens.



## Properties available in the Label Collection Editor

The following properties are available for the user in the **Label Collection Editor** at design time or they can be used in the `ChartLabels` class at run time:

Members	Description
<a href="#">AttachMethod</a>	Gets or sets the label attachment method.
<a href="#">AttachMethodData</a>	Gets or sets the fill color.
<a href="#">GroupIndex</a>	Gets or sets the group index of the data point to attach a label when the label <code>AttachMethod</code> property specifies <code>DataIndex</code> attachment.
<a href="#">PointIndex</a>	Gets or sets the point index of the data point to attach a label when the label <code>AttachMethod</code> property specifies <code>DataIndex</code> attachment.
<a href="#">SeriesIndex</a>	Gets or sets the series index of the data point to attach a label when the label <code>AttachMethod</code> property specifies <code>Coordinate</code> or <code>DataCoordinate</code> attachment.
<a href="#">X</a>	Gets or sets the X coordinate (data or client) when the label <code>AttachMethod</code> property specifies <code>Coordinate</code> or <code>DataCoordinate</code> attachment.
<a href="#">Y</a>	Gets or sets the Y coordinate (data or client) when the label <code>AttachMethod</code> property species <code>Coordinate</code> or <code>DataCoordinate</code> attachment.
<a href="#">Compass</a>	Gets or sets the position of the label relative its specified location.
<a href="#">Connected</a>	Gets or sets whether a connecting line is drawn to an associated data point.
<a href="#">Image</a>	Gets or sets the image.
<a href="#">Location</a>	Gets the location of the label in chart control client coordinates.

Name	Gets or sets the label name.
Offset	Gets or sets the offset distance in pixels from an associated data point.
RotationOverride	This property allows the specification of the clockwise rotation angle around the connection point of the label. The property overrides the RotationEnum of the label Style object, and does NOT apply to Radial or RadialText compass values.
Size	Gets or sets the size of the label in chart control client coordinates.
SizeDefault	Gets or sets the default size of the label.
Style	Gets the Style object of the label.
AutoWrap	Gets or sets whether the text is automatically wrapped.
BackColor	Gets or sets the background color.
BackColor2	Gets or sets the gradient or hatch background.
Border	Gets the border object.
BorderStyle	Gets or sets the border style.
Border.Color	Gets or sets the border color.
Rounding	Gets the Rounding object that controls the rounding of corners.
Thickness	Gets or sets the border thickness.
Font	Gets or sets the font object.
ForeColor	Gets or sets the foreground color.
GradientStyle	Defines the style of the background gradient filling.
HatchStyle	Defines the style of the background hatch filling.
HorizontalAlignment	Gets or sets the text horizontal alignment.
ImageAlignment	Gets or sets the image alignment.
Opaque	Gets or sets the opaqueness of the background.
Rotation	Gets or sets the text orientation.
VerticalAlignment	Gets or sets the text vertical alignment.
Text	Gets or sets the text of the label.
TooltipText	Gets or sets the tooltip text.
Visible	Gets or sets the label visibility.

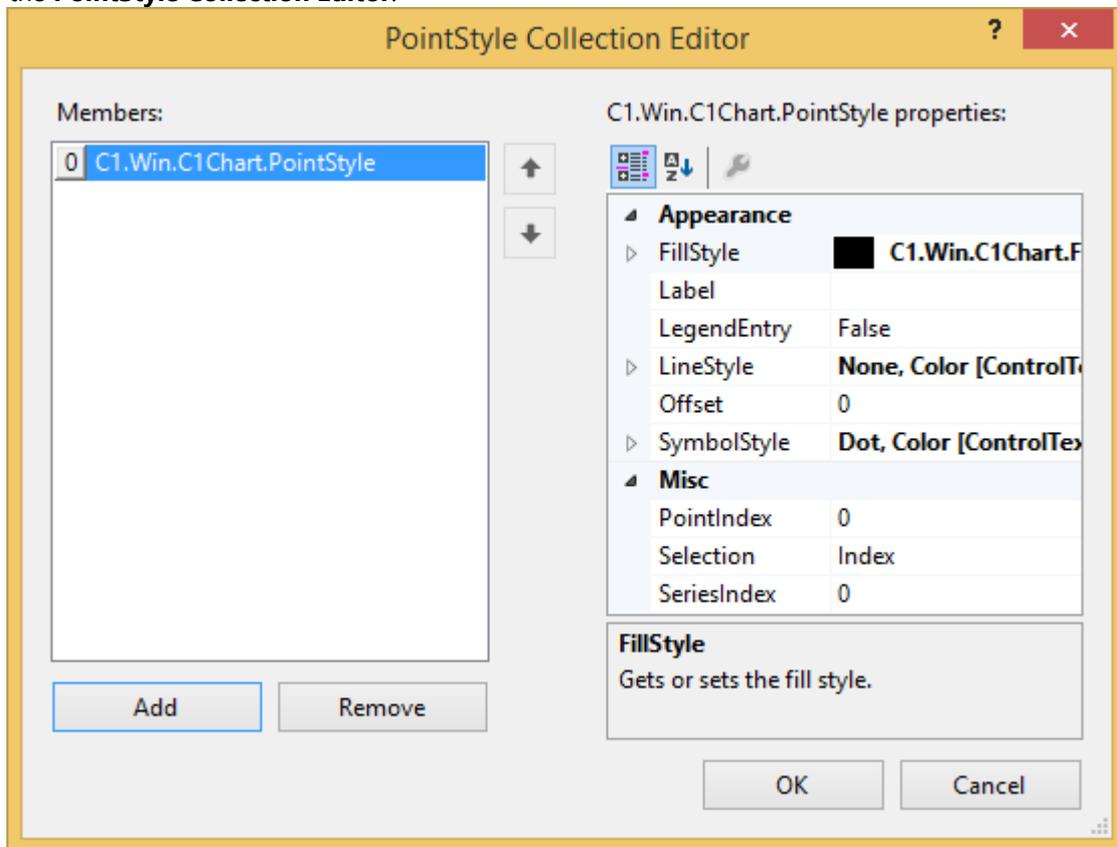
## PointStyle Collection Editor

The **PointStyle Collection Editor** is used for adding or modifying PointStyles for specific data points on the chart at design time. For more information on using the PointStyles see, [Working with PointStyles](#).

## To Access the PointStyle Collection Editor

1. Right-click the **C1Chart** control and select **Properties** from its context menu.

2. In the Properties window, expand the **ChartGroups** node, then expand the **Group0** or **Group1** node.
3. Expand the **ChartData** node, then click on the **ellipsis** button next to the **PointStylesList** property.
4. Click on the **Add** button to add a PointStyle to the collection editor. Its properties appear in the Property grid of the **PointStyle Collection Editor**.



### Properties available in the PointStyle Collection Editor

The following properties are available for the user in the **PointStyle Collection Editor** at design time or they can be used in the PointStyle class at run time:

#### Appearance Properties of the PointStyle Collection Editor:

Members	Description
<a href="#">FillType</a>	Gets or sets the fill type.
<a href="#">Color1</a>	Gets or sets the fill color.
<a href="#">Alpha</a>	Gets or sets the fill alpha value (transparency).
<a href="#">Label</a>	Gets or sets the point style label.
<a href="#">LegendEntry</a>	Gets or sets whether the point style appears in the legend.
<a href="#">Color</a>	Gets or sets the color of the line.
<a href="#">Pattern</a>	Gets or sets the pattern of the line.
<a href="#">Thickness</a>	Gets or sets the thickness of the line.
<a href="#">Offset</a>	Gets or sets the offset for appropriate charts.
<a href="#">Color</a>	Gets or sets the color of the data point symbol.

<a href="#">OutlineColor</a>	Gets or sets the outline color of the data point symbol.
<a href="#">OutlineWidth</a>	Gets or sets the outline width of the data point symbol.
<a href="#">Shape</a>	Gets or sets the shape of the data point symbol.
<a href="#">Size</a>	Gets or sets the size of the data point symbol.

## Misc Properties of the `PointStyle` Collection Editor:

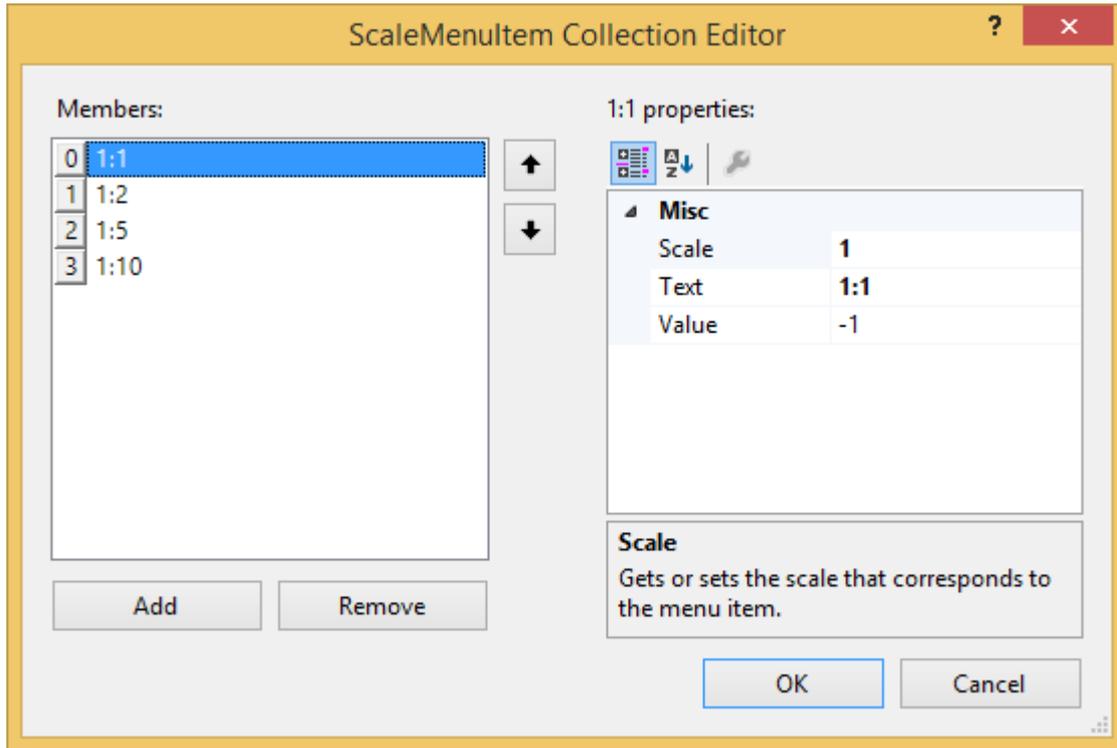
Members	Description
<a href="#">PointIndex</a>	Gets or sets the index of data point associated with the point style.
<a href="#">Selection</a>	Gets or sets the selection method for the point style.
<a href="#">SeriesIndex</a>	Gets or sets the index of data series associated with the point style.

## ScaleMenuItem Collection Editor

The **ScaleMenuItem Collection Editor** is used for modifying the properties in the `ScaleMenuItem` collection.

### To Access the ScaleMenuItem Collection Editor

1. Right-click the **C1Chart** control and select **Properties** from its context menu.
2. In the Properties window, expand the **ChartArea** node, expand the `AxisX`, `AxisY`, or `AxisY2` node, and then expand the **ScrollBar**.
3. Click the **ellipsis** button next to the **ScaleMenuItems** property. The **ScaleMenuItem Collection Editor** appears.



## Properties available in the ScaleMenuItem Collection Editor

The following properties are available for the user in the **ScaleMenuItem Collection Editor** at design time or they can be used in the [ScaleMenuItem](#) class at run time:

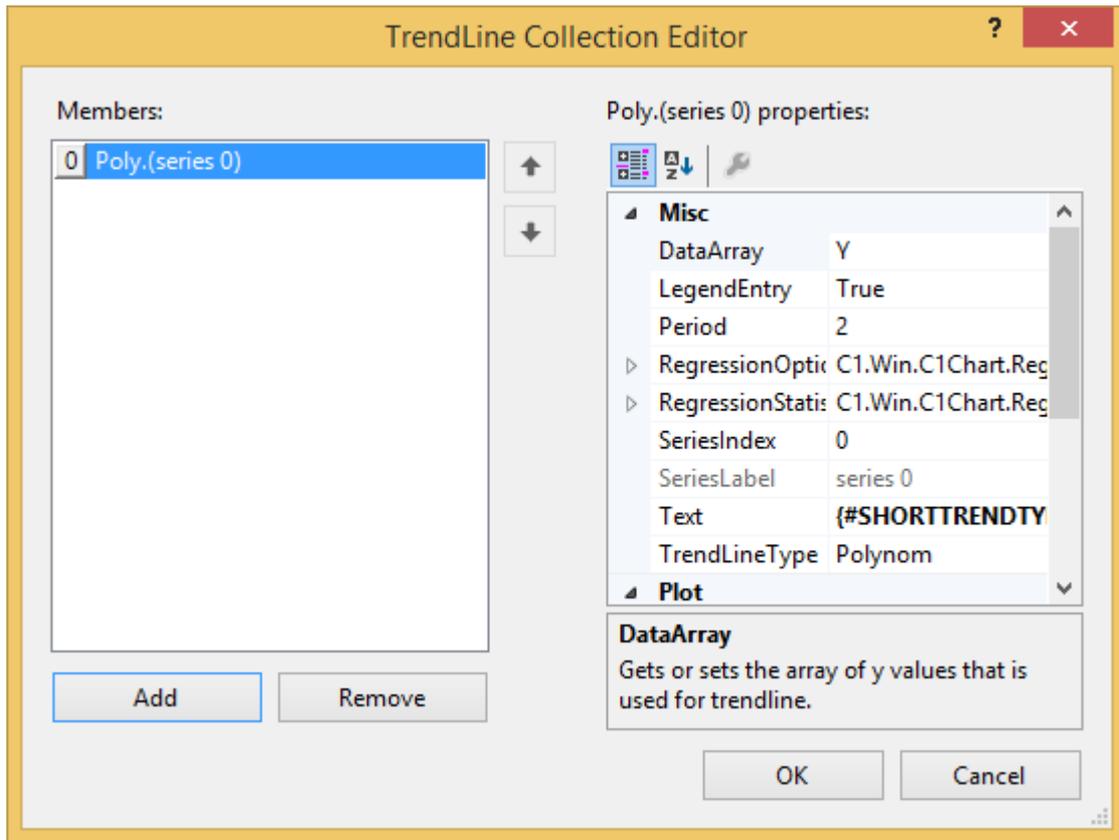
Members	Description
<a href="#">Scale</a>	Gets or sets the scale that corresponds to the menu item.
<a href="#">Text</a>	Gets or sets the text of the menu item.
<a href="#">Value</a>	Gets or sets the value that corresponds to the menu item.

## TrendLine Collection Editor

The **TrendLine Collection Editor** is used for adding or modifying TrendLines that approximate the data which the functions trend at design time. For more information on TrendLines see, [Working with TrendLines](#).

## To Access the TrendLine Collection Editor

1. Right-click the **C1Chart** control and select **Properties** from its context menu.
2. In the Properties window, expand the **ChartGroups** node, then expand the **Group0** or **Group1** node.
3. Expand the **ChartData** node, then click on the **ellipsis** button next to the **TrendsList** property. The **TrendLine Collection Editor** appears.



## Properties available in the TrendLine Collection Editor

The following properties are available for the user in the **TrendLine Collection Editor** at design time or they can be used in the [TrendLine](#) class at run time:

### Misc Properties of the TrendLine Collection Editor:

Members	Description
<a href="#">DataArray</a>	Gets or sets the array of y values that is used for the trendline.
<a href="#">LegendEntry</a>	Gets or sets whether the trendline appears in the legend.
<a href="#">Period</a>	Gets or sets period for Moving Average trendline.
<a href="#">RegressionOptions</a>	Gets the options for regression calculation.
<a href="#">NumTerms</a>	Gets or sets the number of terms in regression equation. Only for polynomial and fourier trend lines.
<a href="#">UseYIntercept</a>	Gets or sets whether the trend line uses Yintercept property. Only for polynomial trend line.
<a href="#">YIntercept</a>	Gets or sets the y value at which trend line intercepts x =0 line. Only for polynomial trend line.
<a href="#">Coeffs</a>	Gets the coefficients of regression equation.
<a href="#">DF</a>	Gets the degrees of freedom.
<a href="#">F</a>	Gets the F-observed(F-statistic) value.
<a href="#">Rsqr</a>	Gets the coefficient of determination(R-squared).
<a href="#">Sey</a>	Gets the standard error for the y estimate.
<a href="#">Sse</a>	Gets the sum of squares due to error.

<a href="#">Ssr</a>	Gets the sum of squares due to regression.
<a href="#">SeriesIndex</a>	Gets or sets the data series index that is used for trendline plotting.
<a href="#">SeriesLabel</a>	Gets the data series label that is used for trendline plotting.
<a href="#">Text</a>	Gets or sets the text that will be shown in legend.
<a href="#">TrendLineType</a>	Gets or sets the type of trendline.

## Plot Properties of the TrendLine Collection Editor:

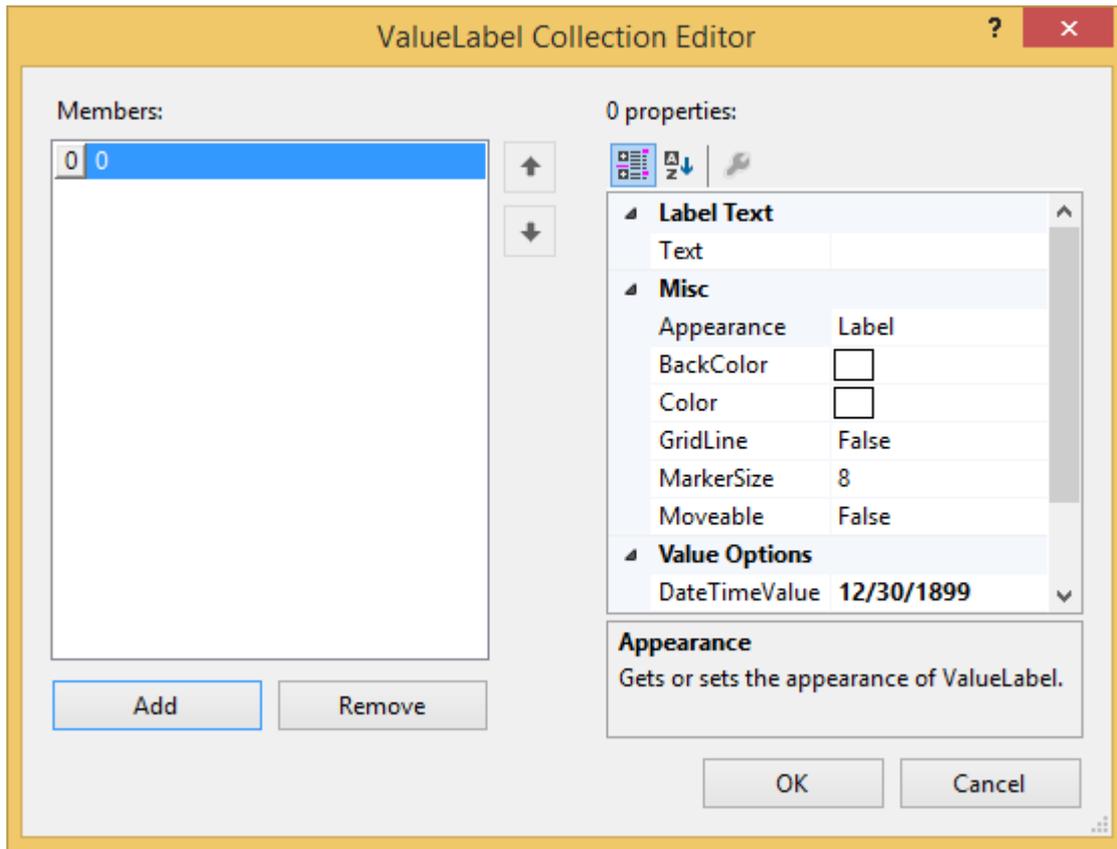
Members	Description
<a href="#">ForecastBackward</a>	Gets or sets how long the trendline extends comparing series data minimum.
<a href="#">ForecastForward</a>	Gets or sets how long the trendline extends comparing series data maximum.
<a href="#">LineStyle</a>	Gets or sets the line used to plot the TrendLine.
<a href="#">Color</a>	Gets or sets the color of the line.
<a href="#">Pattern</a>	Gets or sets the pattern of the line.
<a href="#">Thickness</a>	Gets or sets the thickness of the line.
<a href="#">PlotLinesMethod</a>	Gets or sets the method of trendline plotting.
<a href="#">PlotNumPoints</a>	Gets or sets the number of points of trendline plotting.
<a href="#">PlotOverData</a>	Gets or sets whether the trendline is plotted over or behind data.
<a href="#">Visible</a>	Gets or sets whether the trendline is visible.

## ValueLabel Collection Editor

The **ValueLabel Collection Editor** is used for adding or modifying the properties in the ValueLabels class. ValueLabels displays text defined at a specific axis coordinate. For more information on ValueLabels, see [Value Labels Annotation](#).

## To Access the ValueLabel Collection Editor

1. Right-click on the C1Chart control and select **Properties** from its context menu.
2. In the Properties window, expand the ChartArea node, then expand the **AxisX**, **AxisY**, or **AxisY2** node and click on the **ellipsis** button next to the **ValueLabels** property. The **ValueLabel Collection Editor** appears.



## Properties of the ValueLabel Collection Editor

The following properties are available for the user in the **ValueLabel Collection Editor** at design time or they can be used in the [ValueLabel](#) class at run time:

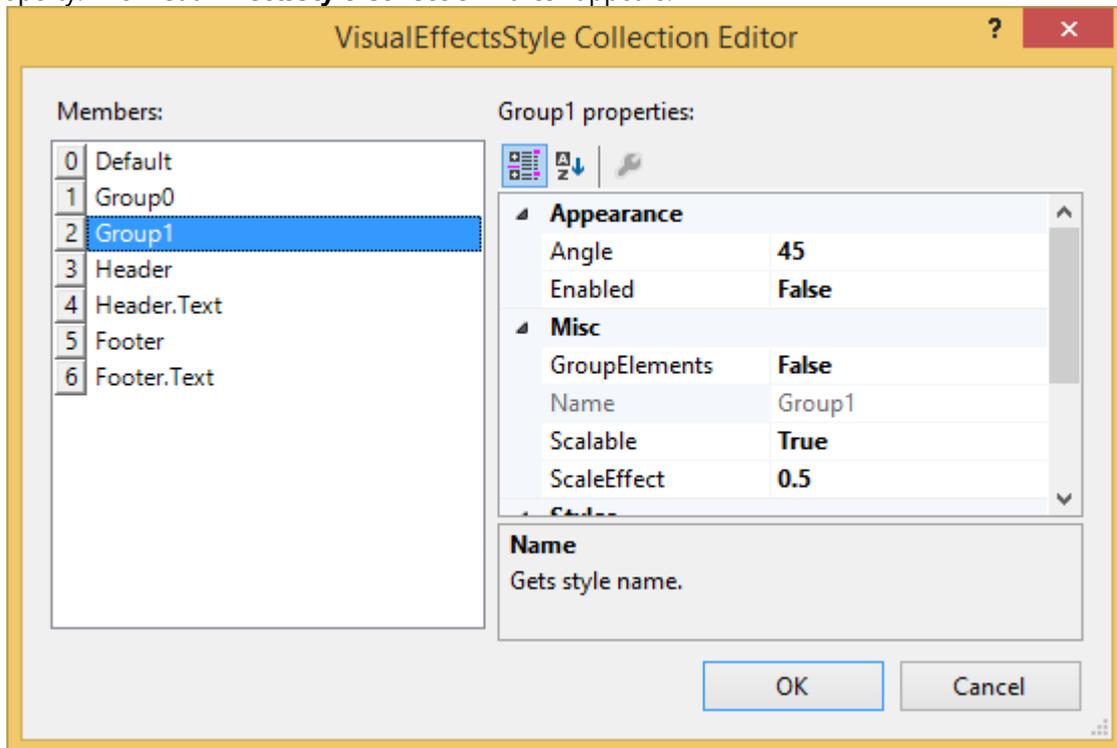
Members	Description
<a href="#">Text</a>	Gets or sets the displayed text value for the ValueLabel.
<a href="#">Appearance</a>	Gets or sets the appearance of ValueLabel.
<a href="#">BackColor</a>	Get or sets the background color of ValueLabel.
<a href="#">Color</a>	Gets or sets the color of the ValueLabel.
<a href="#">Gridline</a>	Gets or sets a value indicating whether a grid line is drawn at the ValueLabel.
<a href="#">MarkerSize</a>	Gets or sets the marker size of ValueLabel.
<a href="#">Moveable</a>	Gets or sets a value indicating whether a ValueLabel can be dragged by the user.
<a href="#">DateTimeValue</a>	Gets or sets the axis value to be replaced by the ValueLabel using a DateTime value.
<a href="#">NumericValue</a>	Gets or sets the axis value to be replaced by the ValueLabel using a double value.

## VisualEffectsStyle Collection Editor

The **VisualEffectsStyle Collection Editor** is used for editing the **VisualEffects** properties. Another way to modify the VisualEffectsStyle properties is through the **Visual Effects** designer. For more information on **VisualEffects**, see [Visual Effects Designer](#).

## To Access the VisualEffectsStyle Collection Editor

1. Right-click the **C1Chart** control and select Properties from its context menu.
2. In the Properties window, expand the **VisualEffects** node, and then click the **ellipsis** button next to the **Styles** property. The **VisualEffectsStyle Collection Editor** appears.



## Properties of the VisualEffectsStyle Collection Editor

The following properties are available for the user in the **VisualEffectsStyle Collection Editor** at design time or they can be used in the [VisualEffectsStyle](#), [ColorOptions](#), [EdgeStyle](#), [LightStyle](#) classes at run time:

Members	Description
<a href="#">Angle</a>	Gets or sets the angle which is used during rendering.
<a href="#">Enabled</a>	Enables effects.
<a href="#">GroupElements</a>	Gets or sets a value indicating whether the graphic elements can be grouped when rendering.
<a href="#">Scalable</a>	Gets or sets a value indicating whether the size of elements depends of the control size.
<a href="#">ScaleEffect</a>	Gets or sets a scale factor which is used to scale elements when Scalable property is true.
<a href="#">Colors</a>	Gets color options for the style.
<a href="#">Brightness</a>	Gets or sets the brightness correction value.
<a href="#">HueShift</a>	Gets or sets hue shift from 0 to 359.
<a href="#">Saturation</a>	Gets or sets the saturation adjustment from -100 to 100.
<a href="#">Edge</a>	Gets the edge settings for the style.
<a href="#">Rounding</a>	Gets or sets the radius rounding for rectangular elements. When Scalable property is true it is measured in relative units, otherwise - in pixels.
<a href="#">Width</a>	Gets or sets the width of the element edge. When Scalable property is true it is measured in relative units, otherwise – in pixels.

Light	Gets the light settings for the style.
Focus	Gets or sets the position of light focus.
Gradient	Gets or sets the type of light gradient.
Intensity	Gets or sets the light intensity.
Scale	Gets or sets the light scale. When scale is less than one the light pattern it is repeated through the element.
Shape	Gets or sets the light shape.
Shift	Gets or sets the light shift.
Size	Gets or sets the size of light spot.

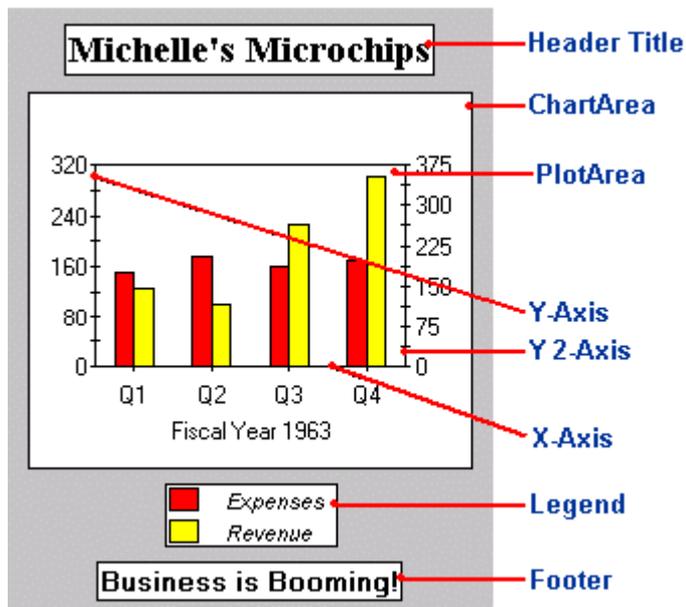
## Chart Fundamentals

The **C1Chart** control has a rich object model that allows you to create a variety of chart types and control their appearance and behavior.

This chapter introduces the **C1Chart** object model by describing the main elements in the chart and the main properties related to each one. By the end of this chapter, you will have a good understanding of the main elements of the **C1Chart** object model which will help you when you use **C1Chart** to create your chart. Later sections drill down into greater detail, covering the remaining parts of the programming model.

## 2D Chart Terminology

The following diagram shows the terms used to describe chart elements:

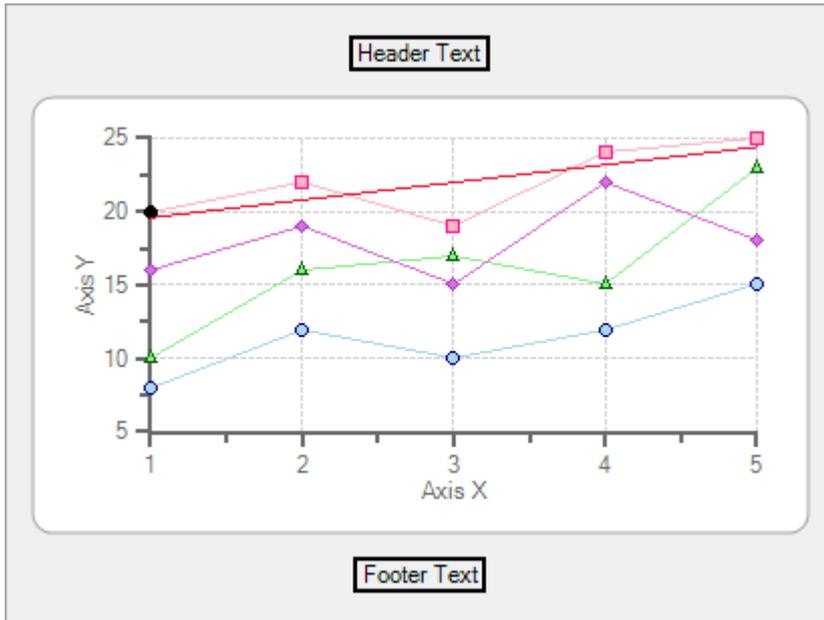


**Note:** Pie charts contain the same elements except for the axes.

The following topics describe the elements listed in the preceding graphic and the main properties associated with each element.

## Defining the Header and Footer Elements

Here is how the Header and Footer elements appear in **C1Chart**:



The header and footer elements are used to display descriptive information about the chart. They are controlled by the Header and Footer properties. Both properties return a [Title](#) object that contains the following main properties:

Property	Description
<a href="#">Text</a>	Contains text displayed in the title (header or footer).
<a href="#">Style</a>	Contains properties that set the font, orientation, colors, and border of the title.
<a href="#">Compass</a>	Determines the position of the title.
<a href="#">Visible</a>	Determines whether the title is visible.

**C1Chart** sizes and positions the titles automatically, based on their contents and how the [Compass](#) property is set. Title object positions are customized using the [SizeDefault](#) and [LocationDefault](#) properties (negative values activate the automatic sizing and positioning).

## Creating header and footer elements

The Chart header and footer elements can be created programmatically through its [Title](#) object or they can be created at design time through the Chart's Properties window, **Chart Properties** designer, or by Chart's SmartDesigner.

The simplest way of creating them is through the Chart's Smart Designer. For more information on creating header or footers through the Chart's Smart Designer, see [Add a Chart Header](#) or [Add a Chart Footer](#).

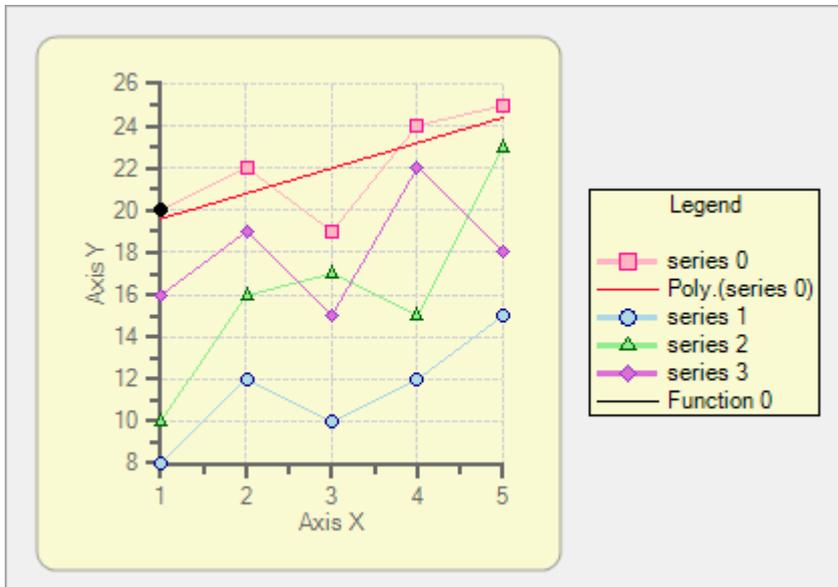
## Customizing header and footer elements

The header and footer elements' text and alignment, position, border, colors, and font can be customized using the Title's properties. Additionally, light patterns, shadows, and custom colors can be added to the header and footer elements using the **Visual Effects** designer.

Please refer to [Customizing Chart Elements](#) for more information on the properties and tools used to enhance 2D Chart's elements. For more information on positioning the header element see [Displaying both the Chart Legend and Chart Header](#).

## Defining the Legend Element

The following graphic illustrates the Legend element in **C1Chart**:



The legend element displays information about each data series of the chart. The chart legend displays the mapping between the physical colors and the data series. The legend is controlled by the [Legend](#) property, which returns a Legend object with the following main properties:

Property	Description
<a href="#">Text</a>	Contains text displayed in the legend title.
<a href="#">Style</a>	Contains properties that set the font, orientation, colors, and border of the legend.
<a href="#">Compass</a>	Determines the position of the legend.
<a href="#">Visible</a>	Determines whether the legend is visible.
<a href="#">Orientation</a>	Determines whether the legend items should be displayed in the horizontal or vertical direction.
<a href="#">Reversed</a>	Controls the order that the ChartGroups appear in the legend. The order in which items of a ChartGroup appear in the legend is controlled by the LegendReversed property of each ChartGroup.

**C1Chart** sizes and positions the legend automatically, based on its contents and the [Compass](#) and [Orientation](#) properties. To customize the legend position, the [SizeDefault](#) and [LocationDefault](#) properties can be modified (negative values activate the automatic sizing and positioning).

Note that the [Legend](#) property does not control the text for each legend item. It is determined by the label attached to each series. For example, [Label](#) contains the text and [LegendEntry](#) determines whether the series label should be

displayed in the legend.

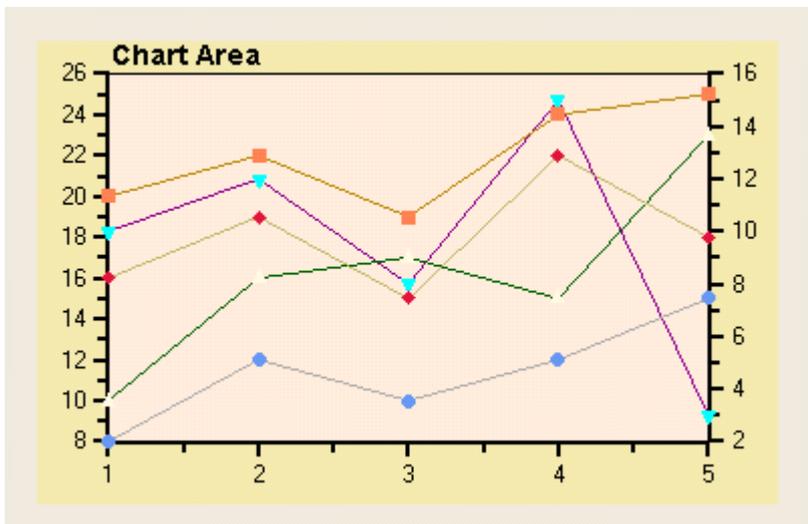
## Creating the legend element

The legend element can be created programmatically through its Legend object or they can be created at design time through the Chart's Properties window, **Chart Properties** designer, or by Chart's Smart Designer.

The simplest way of creating them is through the Chart's Smart Designer. For more information on creating a chart legend through the Chart's Smart Designer, see [Add a Chart Legend](#). For more information on positioning the chart legend programmatically see [Displaying both the Chart Legend and Chart Header](#).

## Defining the ChartArea Objects

The following graphic illustrates the ChartArea element in **C1Chart**:



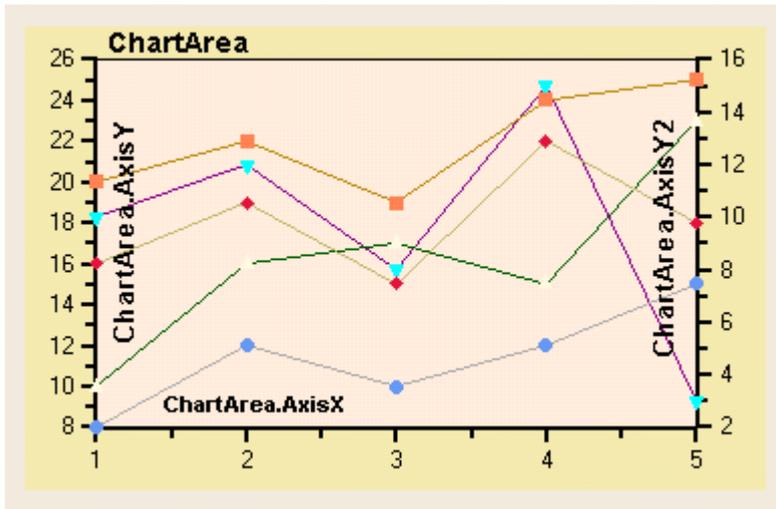
The ChartArea object represents the area of the chart that contains data (excluding the titles and legend, but including the axes). The [ChartArea](#) property returns an [Area](#) object with the following main properties:

Property	Description
<a href="#">AxisX</a> , <a href="#">AxisY</a> , <a href="#">AxisY2</a>	Each of these properties returns Axis objects that allow you to customize the appearance of the chart axes. The Axis objects are introduced in the next topic, <a href="#">Axes Object</a> and they are further discussed in the <a href="#">Axes</a> topic.
<a href="#">Inverted</a>	Allows you to rotate the chart by 90 degrees (this is especially useful to create horizontal bar charts). For more information see the <a href="#">Inverted and Reversed Chart Axes</a> topic.
<a href="#">Margins</a>	Returns a Margin object that allows you to specify the distance between the chart area and the plot area. The axes labels are displayed in this space.
<a href="#">PlotArea</a>	Returns a PlotArea object that controls the appearance of the area inside the axes. For more information see the <a href="#">Plot Area</a> topic.

Style	Contains properties that set the color and border of the chart area.
-------	--

## Axes Object

The following graphic illustrates the X and Y axes in **C1Chart**:



Most charts have two axes, X and Y. The exceptions are pie charts (which have no axes) and charts with a secondary Y axis (Y2, which have three axes).

The axes are represented by sub-properties of the **ChartArea** property: `ChartArea.AxisX`, `ChartArea.AxisY`, and `AxisY2`. Each of these properties returns an Axis object with the following main properties:

### Layout and Style properties

The following properties below represent the layout and style of the axes in **C1Chart**:

Property	Description
<a href="#">Compass</a>	Allows you to set the position of the axis. For example, you may want to display the X-axis above the data instead of below. For more information see <a href="#">Axis Position</a> .
<a href="#">Font</a>	Sets the font used to display the values along the axis. For more information see <a href="#">Axis Appearance</a> .
<a href="#">ForeColor</a>	Sets the color used to display the axis, tickmarks, and values. For more information see <a href="#">Axis Appearance</a> .
<a href="#">Reversed</a>	Allows you to reverse the direction of the axis. For example, you can show Y values going down instead of up. For more information see <a href="#">Inverted and Reversed Chart Axes</a> .
<a href="#">Text</a>	Sets a string to display next to the axis (this is typically used to describe the variable and units being depicted by the axis). For more information see <a href="#">Axis Title and Rotation</a> .

<a href="#">Rotation</a>	Sets the orientation of the Text string.
--------------------------	--

## Annotation properties

The following properties below represent the format for the annotation of the axes in **C1Chart**:

Property	Description
<a href="#">AnnoFormat</a>	A set of predefined formats used to format the values displayed next to the axis.
<a href="#">AnnoFormatString</a>	The .NET formatting string used to format the values displayed next to the axis when AnnoFormat is set to "NumericManual" or "DateManual". If the AnnoFormat is set to either NumericManual or DateManual and the AnnoFormatString is empty, then the chart uses an algorithm to find the "best" formatting available.
<a href="#">AnnoMethod</a>	Determines what values are displayed next to the axis. Options are "Values", which displays the actual series values, or "ValueLabels", which displays the elements in the ValueLabels collection.
<a href="#">ValueLabels</a>	A collection of text/value pairs to display next to the axis when AnnoMethod is set to "ValueLabels". This property is useful when you want to display strings along an axis instead of numeric values (for example, you may be charting product prices and want to display the product names along the X-axis). For more information see <a href="#">Value Labels Annotation</a> .
<a href="#">AnnotationRotation</a>	Allows you to rotate the values so they take up less space along the axis. For more information see <a href="#">Axis Annotation Rotation</a> .

## Scaling Tickmark and Gridline properties

The following properties represent the scaling, tickmarks, and gridline styles and function for the axes in **C1Chart**:

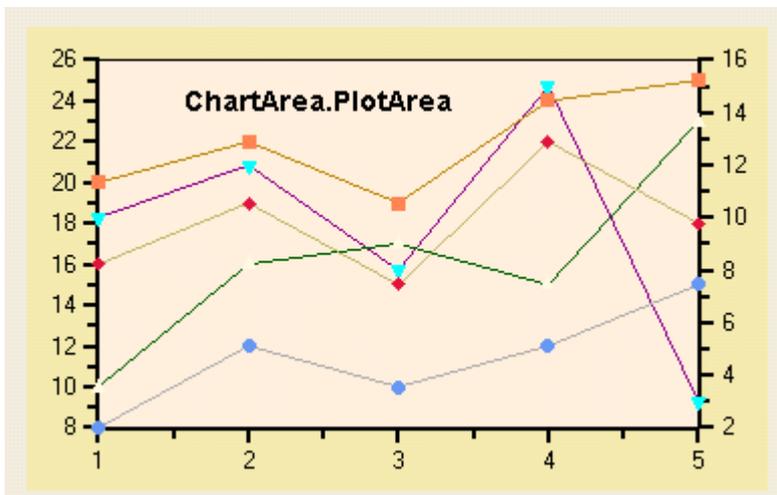
Property	Description
<a href="#">AutoMin</a> , <a href="#">AutoMax</a>	Determine whether the minimum and maximum values for the axis should be calculated automatically. For more information see <a href="#">Axis Bounds</a> .
<a href="#">Min</a> , <a href="#">Max</a> .	Set the minimum and maximum values for the axis (when AutoMin and AutoMax are set to False). For more information see <a href="#">Axis Bounds</a> .
<a href="#">AutoMajor</a> , <a href="#">AutoMinor</a>	Determine whether the spacing between the major and minor tickmarks should be calculated automatically.
<a href="#">UnitMajor</a> , <a href="#">UnitMinor</a>	Set the spacing between the major and minor tickmarks (when the AutoMajor and AutoMinor properties are set to False).
<a href="#">GridMajor</a> , <a href="#">GridMinor</a>	Returns a ChartGridStyle object with properties that

	control the appearance of the grid lines drawn perpendicular to major and minor tickmarks. For more information see <a href="#">Axis Grid Lines</a> .
<a href="#">AutoOrigin</a>	Determines whether the X-axis should be automatically positioned with respect to the Y-axis.
<a href="#">Origin</a>	Sets the position of the X-axis with respect to the Y-axis. The X-axis is usually placed at the bottom of the chart. This property allows you to place it so that it crosses the Y-axis at a given coordinate.
<a href="#">TickFactorMajor</a>	Gets or sets an integral factor for major tick mark length.
<a href="#">TickFactorMinor</a>	Gets or sets an integral factor for minor tick mark length.
<a href="#">TickGauge</a>	Gets or sets the approximate number of intervals delineated by gauge marks between major tick marks.

For more information about the Axes object, see [Axes](#).

## PlotArea Object

The following illustration shows the PlotArea object on **C1Chart**:



The **PlotArea** object represents the part of the chart area that is used to display the data series. The most common properties used to customize the **PlotArea** include the following:

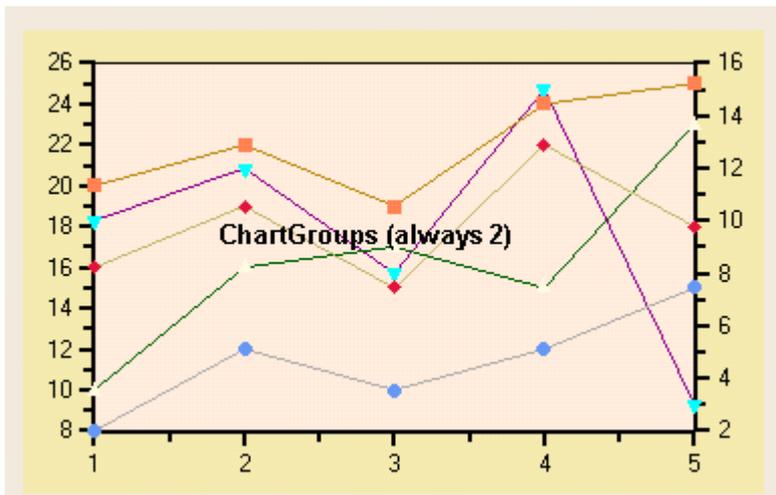
Property	Description
<a href="#">BackColor</a>	Sets the color of the background of the PlotArea.
<a href="#">Boxed</a>	Determines whether the PlotArea has a solid border around it.
<a href="#">ForeColor</a>	Sets the color of the border around the PlotArea (when

	Boxed is set to True).
<a href="#">View3D</a>	Returns a View3D object that allows you to add 3D effects to 2D charts.

For more information on customizing the plot area, see [Plot Area](#).

## Defining the ChartGroups Objects

The following illustration shows the ChartGroups elements on **C1Chart**:



This property returns a collection of two ChartGroup objects (always Group0 and Group1, you can't add or remove elements from this collection). These objects determine the chart type and contain the data being charted.

The first group (Group0) contains series that are charted against the primary Y-axis. The second group (Group1) may be empty or it may contain series that are charted against the secondary Y-axis.

Because there are two ChartGroup objects, and the chart type is associated with the ChartGroup, you can mix up to two chart types in the same chart. For example, you can create charts that show some series as bars and some as lines.

The ChartGroup objects have the following main properties:

Property	Description
<a href="#">ChartData</a>	Returns a ChartData object that contains a SeriesList property that holds the data for all series in the group. This property is described in the <a href="#">ChartDataSeries Object</a> topic.
<a href="#">ChartType</a>	Determines the type of chart to use when rendering this group (there are many chart types available, and you can select different types for each ChartGroup). For information on the available 2D chart types, see <a href="#">Specific 2D Charts</a> .
<a href="#">ShowOutline</a>	Determines whether graphical elements used to display the data (bars, areas, pie slices) should be outlined using the color specified by the <a href="#">ForeColor</a>

	property.
<a href="#">Stacked</a>	Determines whether the data should be stacked (by adding Y values) when the series are rendered.

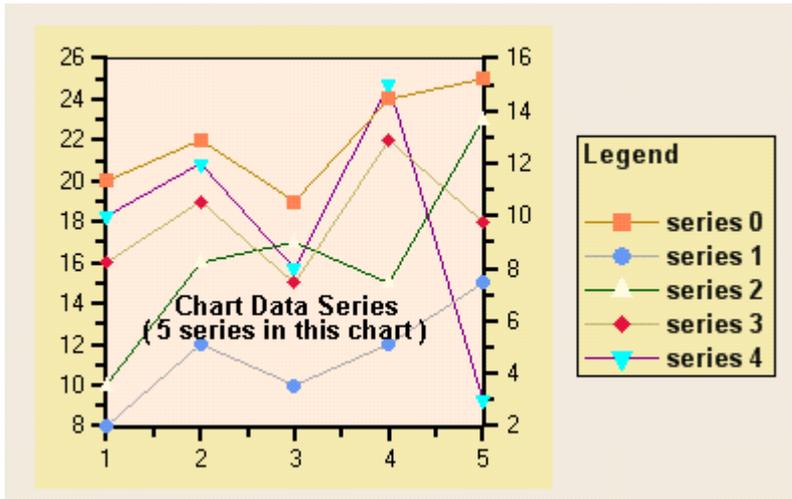
## ChartData Object

The ChartData objects have the following main properties:

Property	Description
FunctionsList	The FunctionsList property gets the FunctionsCollection object associated with the current ChartData object. For more information on the FunctionsList property see <a href="#">Defining the ChartData Object</a> .
Hole	Gets or sets the data hole value. For more information see <a href="#">Specifying Data Holes</a> .
PointStylesList	Gets the PointStylesCollection object associated with the current ChartData object. For more information on point styles, see <a href="#">Working with PointStyles</a> .
SeriesList	Gets the ChartDataSeriesCollection object associated with the current ChartData object. The next topic, <a href="#">ChartDataSeries Object</a> , provides more detail on the ChartData.SeriesList property.
TrendsList	Gets the TrendLinesCollection object associated with the current ChartData object. For more information on trend lines, see <a href="#">Working with TrendLines</a> .

## ChartDataSeries Object

The following illustration shows five Chart Data Series on **C1Chart**:



From a developer's perspective, this is one of the most important properties in **C1Chart**. All other properties can be set at design time using the **Chart Wizard**, **Chart Properties** designer, or loaded from predefined chart layout files. In most cases, however, the data being plotted is added using code, and to do that you need to use the `SeriesList` property.

The `SeriesList` property returns a `ChartDataSeriesCollection` object that allows you to add and remove series from the chart, and to retrieve individual series. For an example that shows how to programmatically add data series see, [Adding Data Series](#).

The `ChartDataSeries` objects have the following main properties:

Property	Description
Display	Determines whether the series is visible and how missing values ("data holes") should be displayed.
Label	Contains the text that is displayed in the legend (if <code>LegendEntry</code> is set to <code>True</code> ).
LegendEntry	Determines whether the series <code>Label</code> should be displayed in the legend.
LineStyle	Contains properties that determine the color, thickness, and pattern used to display the series (the color is used for lines, areas, bars, and pie slices). For more informatio see <a href="#">Line and Symbol Styles for the Series</a> .
SymbolStyle	Contains properties that determine the shape, size, and color of the symbols used to mark the data points in the series. For more information see <a href="#">Line and Symbol Styles for the Series</a> .
PointData	Returns a <code>ChartDataArray</code> object used to get or set the X, Y coordinates of each data point in the series.
X, Y	Return <code>ChartDataArray</code> objects used to get or set individual coordinates of each data point in the series. Some chart types have additional data arrays (for

example, "HiLoOpenClose" also has Y1, Y2, and Y3).

The `PointData`, `X`, and `Y` properties allow you to set and retrieve the data used to display each individual series.

Most chart types require you to provide X and Y values for each point. The exceptions are pie charts (which do not require X values) and specialized charts such as **Bubble**, **HiLo**, **Gantt**, and **HiLoClose**, which require additional Y values.

## Common Usage for Basic 2D Charts

This chapter describes the common usage of the basic chart types such as **Bar**, **Pie**, and **X-Y Plot** charts. It also provides a sample code for each chart type. The samples are simple and concise, and focus on the main aspects of each common chart type. The chart data in most of the samples included in this topic are from the C1NWind.mdb provided in the Chart for Winforms installation. The distribution package includes a lot of sophisticated samples that show details and advanced features not discussed in this quick walkthrough.

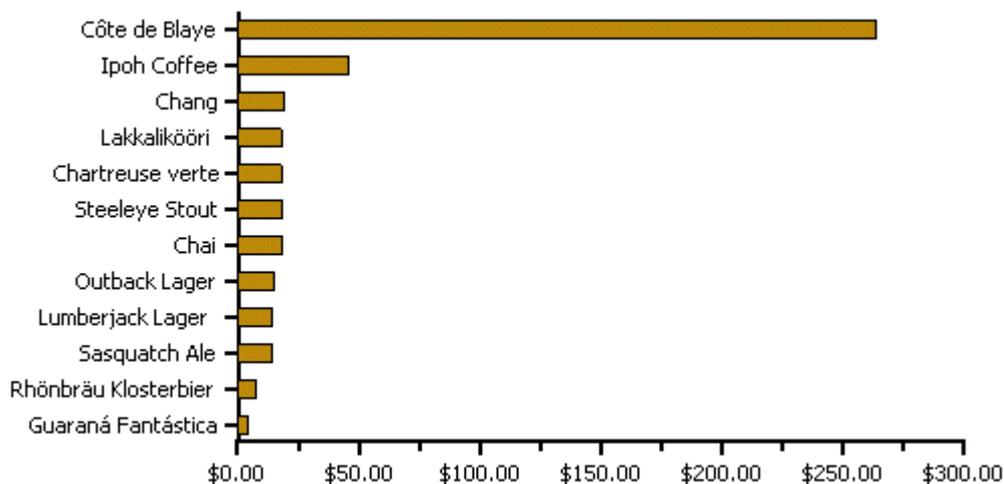
## Simple Bar Charts

It is common to use charts to display a series of simple values, like product prices. This type of chart is useful because it shows the relative magnitude of each value in a quick and effective way.

The main characteristic of this type of chart is that each point conveys only one piece of information. When creating these charts, the value being displayed is assigned to the Y values of the series. The X values only provide constant spacing between the points, and usually display labels attached to each data point.

The recommended chart types for this type of information are **Bar** and **Pie** charts.

The following illustration shows a simple bar chart with product prices (Beverages from the NorthWind product list).



Note that the bars are displayed horizontally, in order to provide room for the product names along the vertical axis. This is done using the [Inverted](#) property.

The following code creates the bar chart in the preceding graphic. The data is used from the Beverages table in the NorthWind product list which can be found in the **Documents\ComponentOne Samples\Common**. In actual applications, the code would be even simpler because some properties would be set at design time.

### To write code in Visual Basic

#### Visual Basic

```
' get chart data
Dim data As DataView = _dataSet.Tables("Products").DefaultView
data.Sort = "UnitPrice"
data.RowFilter = "CategoryID = 1" ' beverages

' configure the chart
C1Chart1.Reset()
```

```

C1Chart1.ChartArea.Inverted = true
C1Chart1.ChartGroups(0).ChartType = Chart2DTypeEnum.Bar

' create single series for product price
Dim dscoll As ChartDataSeriesCollection =
C1Chart1.ChartGroups(0).ChartData.SeriesList
dscoll.Clear()
Dim series As ChartDataSeries = dscoll.AddNewSeries()
series.Label = "Product Prices"

' populate the series
series.PointData.Length = data.Count
Dim i As Integer
For i = 0 To data.Count - 1
series.X(i) = I
series.Y(i) = data(i) ("UnitPrice")
Next I

' attach product names to x-axis
Dim ax As Axis = C1Chart1.ChartArea.AxisX
ax.AnnoMethod = AnnotationMethodEnum.ValueLabels
For i = 0 To data.Count - 1
ax.ValueLabels.Add(i, CType(data(i) ("ProductName"), String))
Next I

' configure y-axis
Dim ay As Axis = C1Chart1.ChartArea.AxisY
ay.AnnoFormat = FormatEnum.NumericCurrency

```

**To write code in C#**

```

C#
// get chart data
DataGridView data = _dataSet.Tables["Products"].DefaultView;
data.Sort = "UnitPrice";
data.RowFilter = "CategoryID = 1"; // beverages

// configure the chart
c1Chart1.Reset();
c1Chart1.ChartArea.Inverted = true;
c1Chart1.ChartGroups[0].ChartType = Chart2DTypeEnum.Bar;

// create single series for product price
ChartDataSeriesCollection dscoll = c1Chart1.ChartGroups[0].ChartData.SeriesList;
dscoll.Clear();
ChartDataSeries series = dscoll.AddNewSeries();
series.Label = "Product Prices";

// populate the series
series.PointData.Length = data.Count;
for (int i = 0; i < data.Count; i++)

```

```

{
    series.X[i] = i;
    series.Y[i] = data[i]["UnitPrice"];
}

// attach product names to x-axis
Axis ax = c1Chart1.ChartArea.AxisX;
ax.AnnoMethod = AnnotationMethodEnum.ValueLabels;
for (int i = 0; i < data.Count; i++)
ax.ValueLabels.Add(i, (string)data[i]["ProductName"]);

// configure y-axis
Axis ay = c1Chart1.ChartArea.AxisY;
ay.AnnoFormat = FormatEnum.NumericCurrency;
    
```

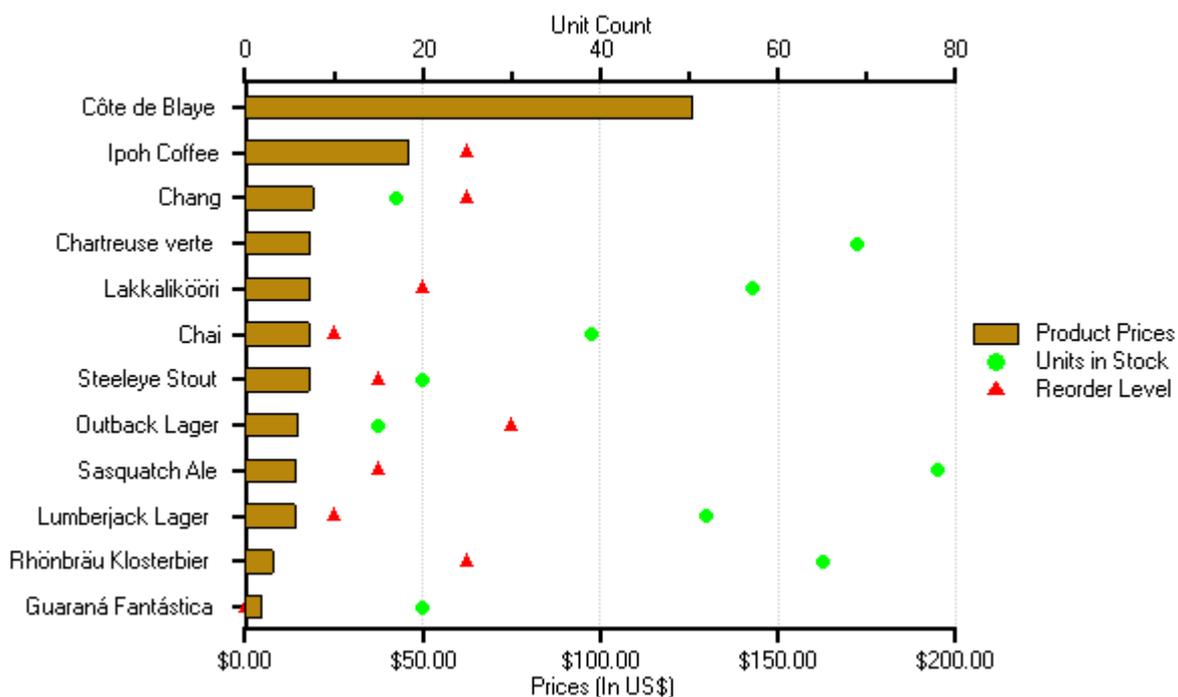
It is common to show more than one data series on a chart, and that is easy to do with C1Chart. The code would look similar to the above, except you would add additional lists to the [ChartDataSeriesCollection](#).

## Bar Charts with Two Y Axes

Although not as common, it is possible to use a single chart to show series that are measured using different scales and units. For example, you could show product prices and inventory levels on a single chart. To do this, you need to use a second Y-axis. The primary axis would show prices (in currency units) and the secondary axis would show counts.

To plot data against the secondary Y-axis, add new series to the second chart group (remember every C1Chart has two chart groups). You can use a different data type for the second chart group in order to make the chart clearer.

For example, the following chart contains three series. The first shows product unit prices (as bars, plotted against the primary Y-axis), and the others show the number of units in stock and the reorder level (as symbols, plotted against the secondary Y-axis).



The code used to create this second chart starts with the exact same steps used to create the previous chart. The two

additional series, **Units in Stock** and **Reorder Level**, are created using the following code.

#### To write code in Visual Basic

##### Visual Basic

```
' label Y-axis and show legend
C1Chart1.Legend.Visible = True
C1Chart1.ChartArea.AxisY.Text = "Prices (in US$)"
C1Chart1.ChartArea.AxisY2.Text = "Unit Count"

' create two series for units in stock and reorder level
' (these are plotted against the secondary Y axis)
dscoll = C1Chart1.ChartGroups(1).ChartData.SeriesList
dscoll.Clear()

' units in stock
series = dscoll.AddNewSeries()
series.Label = "Units In Stock"
series.SymbolStyle.Color = Color.Green
series.LineStyle.Pattern = LinePatternEnum.None
series.PointData.Length = data.Count
For i = 0 To data.Count - 1
    series.X(i) = I
    series.Y(i) = data(i) ("UnitsInStock")
Next I

' reorder level
series = dscoll.AddNewSeries()
series.Label = "Reorder Level"
series.SymbolStyle.Color = Color.Red
series.LineStyle.Pattern = LinePatternEnum.None
series.PointData.Length = data.Count
For i = 0 To data.Count - 1
    series.X(i) = I
    series.Y(i) = data(i) ("ReorderLevel")
Next I

' show gridlines for secondary Y-axis
C1Chart1.ChartArea.AxisY2.GridMajor.Visible = True
```

#### To write code in C#

##### C#

```
// label Y-axis and show legend
_c1c.Legend.Visible = true;
_c1c.ChartArea.AxisY.Text = "Prices (in US$)";
_c1c.ChartArea.AxisY2.Text = "Unit Count";

// create two series for units in stock and reorder level
// (these are plotted against the secondary Y axis)
dscoll = _c1c.ChartGroups[1].ChartData.SeriesList;
```

```
dscoll.Clear();

// units in stock
series = dscoll.AddNewSeries();
series.Label = "Units In Stock";
series.SymbolStyle.Color = Color.Green;
series.LineStyle.Pattern = LinePatternEnum.None;
series.PointData.Length = data.Count;
for (int i = 0; i < data.Count; i++)
{
    series.X[i] = i;
    series.Y[i] = data[i]["UnitsInStock"];
}

// reorder level
series = dscoll.AddNewSeries();
series.Label = "Reorder Level";
series.SymbolStyle.Color = Color.Red;
series.LineStyle.Pattern = LinePatternEnum.None;
series.PointData.Length = data.Count;
for (int i = 0; i < data.Count; i++)
{
    series.X[i] = i;
    series.Y[i] = data[i]["ReorderLevel"];
}

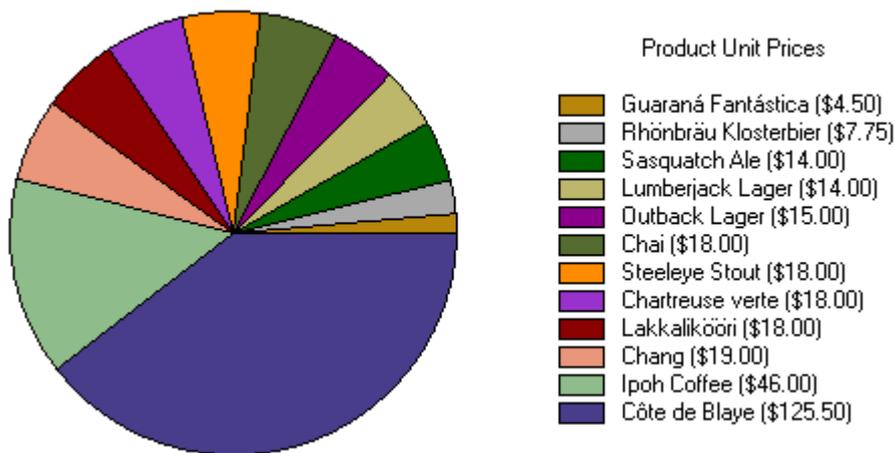
// show gridlines for secondary Y-axis
_c1c.ChartArea.AxisY2.GridMajor.Visible = true;
```

## Pie Charts

Pie charts are commonly used to display simple values. They are visually appealing and often displayed with 3D effects such as shading and rotation. **C1Chart** allows you to add 3D effects to charts, but you should use them carefully because they distort the data.

Pie charts have one significant difference when compared to other **C1Chart** chart types: in Pie charts, each series represents one slice of the pie. Therefore, you will never have Pie charts with a single series (they would be just circles). In most cases, Pie charts have multiple series (one per slice) with a single data point in each series. **C1Chart** represents series with multiple data points as multiple pies within the chart.

This arrangement makes sense when you think of the labels used to identify each series and how they are displayed in the chart legend. The following chart illustrates the same sales data as a pie chart.



The code used to create this chart is significantly different from the bar chart code. It creates one series per value, each with a single data point. The following example creates the chart shown in the preceding graphic:

#### To write code in Visual Basic

##### Visual Basic

```
' get chart
Dim data As DataView = _dataSet.Tables["Products"].DefaultView
data.Sort = "UnitPrice"
data.RowFilter = "CategoryID = 1"      ' beverages

' configure chart
C1Chart1.Reset()
C1Chart1.BackColor = Color.White
C1Chart1.ChartArea.Style.Font = new Font("Tahoma", 8)
C1Chart1.ChartGroups(0).ChartType = Chart2DTypeEnum.Pie

' get series collection (pies have one series per slice)
Dim dscoll As ChartDataSeriesCollection =
C1Chart1.ChartGroups(0).ChartData.SeriesList
dscoll.Clear()

' populate the series
Dim i As Integer
For i = 0 To data.Count - 1
Dim series As ChartDataSeries = dscoll.AddNewSeries()
series.PointData.Length = 1
series.Y(0) = data(i)("UnitPrice")
series.Label = String.Format("{0} ({1:c})", _
    data(i)("ProductName"), data(i)("UnitPrice"))
Next i

' show pie legend
C1Chart1.Legend.Visible = True
C1Chart1.Legend.Text = "Product Unit Prices"
```

## To write code in C#

```
C#  
  
// get chart data  
DataView data = _dataSet.Tables["Products"].DefaultView;  
data.Sort = "UnitPrice";  
data.RowFilter = "CategoryID = 1"; // beverages  
  
// configure chart  
c1Chart1.Reset();  
c1Chart1.BackColor = Color.White;  
c1Chart1.ChartArea.Style.Font = new Font("Tahoma", 8);  
c1Chart1.ChartGroups[0].ChartType = Chart2DTypeEnum.Pie;  
  
// get series collection (pies have one series per slice)  
ChartDataSeriesCollection dscoll = c1Chart1.ChartGroups[0].ChartData.SeriesList;  
dscoll.Clear();  
// populate the series  
for (int i = 0; i < data.Count; i++)  
{  
    ChartDataSeries series = dscoll.AddNewSeries();  
    series.PointData.Length = 1;  
    series.Y[0] = data[i]["UnitPrice"];  
    series.Label = string.Format("{0} ({1:c})",  
        data[i]["ProductName"], data[i]["UnitPrice"]);  
}  
  
// show pie legend  
c1Chart1.Legend.Visible = true;  
c1Chart1.Legend.Text = "Product Unit Prices";
```

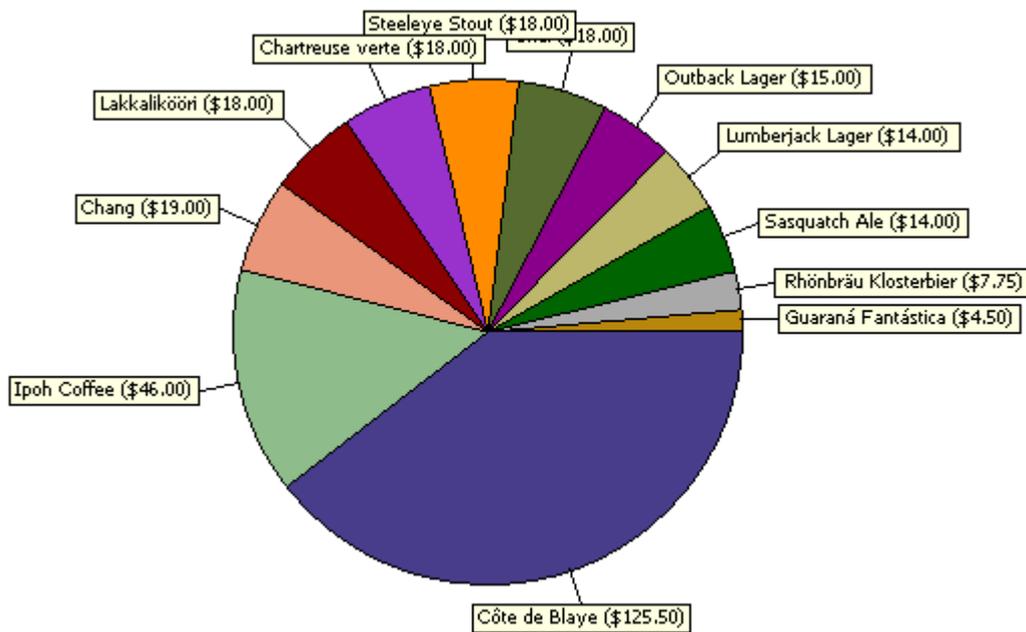
## Pie Charts with ChartLabels

A common way to display data series in a pie chart is through a legend, such as the previous example. This method works well because the information is laid out in a single region. Regardless of how many labels you have, they are laid out neatly within the legend.

In some cases, however, you may want to position the labels near the data points to make their association more obvious. You may also want to add labels containing information on data points or other chart elements, and position those labels at specific locations or near certain chart elements.

For this type of task, **C1Chart** provides the [ChartLabels](#) property, which allows you to create Label objects and attach them to any chart elements.

For example, the following picture shows the same pie chart as the previous example, this time with Labels attached to each slice instead of a Legend.



The chart illustrates a common difficulty associated with using too many chart labels: they may overlap and can be difficult to position correctly if there are too many of them (in this example the labels were automatically positioned by **C1Chart**).

The code used to create this second chart starts with the exact same steps used to create the first pie chart. The labels are created using the following additional code.

#### To write code in Visual Basic

##### Visual Basic

```
' hide legend, configure labels
C1Chart1.Legend.Visible = false
Dim s As Style = C1Chart1.ChartLabels.DefaultLabelStyle
s.Font = new Font("Tahoma", 7)
s.BackColor = SystemColors.Info
s.Opaque = true
s.Border.BorderStyle = BorderStyleEnum.Solid

' attach labels to each slice
Dim i As Integer
For i = 0 To data.Count - 1
    Dim lbl As C1.Win.C1Chart.Label = _
    C1Chart1.ChartLabels.LabelsCollection.AddNewLabel()
    lbl.Text = string.Format("{0} ({1:c})", _
    data(i)("ProductName"), data(i)("UnitPrice"))
    lbl.Compass = LabelCompassEnum.Radial
    lbl.Offset = 20
    lbl.Connected = True
    lbl.Visible = True
    lbl.AttachMethod = AttachMethodEnum.DataIndex

Dim am As AttachMethodData = lbl.AttachMethodData
```

```
am.GroupIndex = 0
am.SeriesIndex = i
am.PointIndex = 0
Next i
```

### To write code in C#

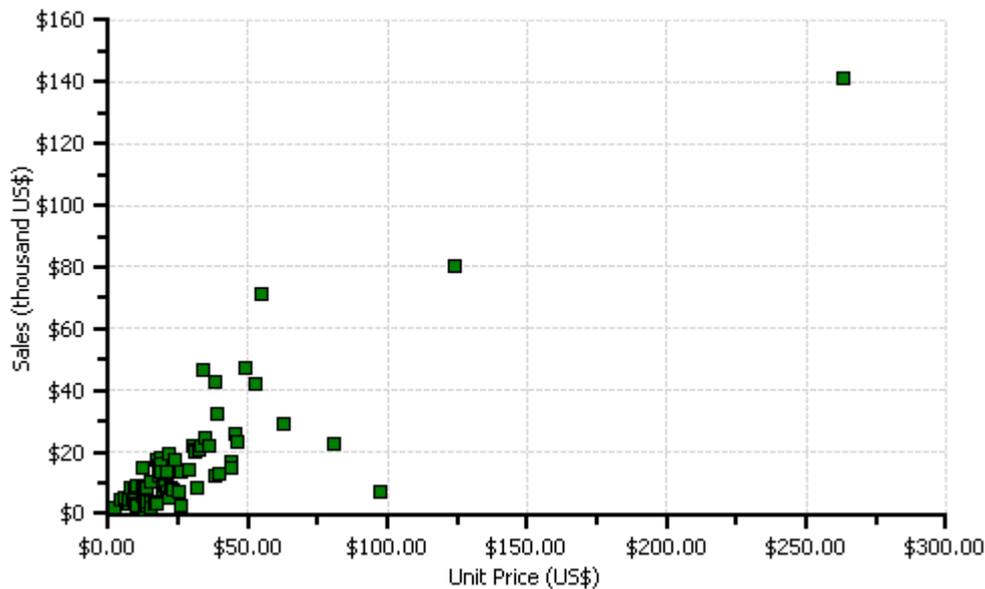
```
C#
// hide legend, configure labels
c1Chart1.Legend.Visible =
false;
Style s = c1Chart1.ChartLabels.DefaultLabelStyle;
s.Font = new Font("Tahoma", 7);
s.BackColor = SystemColors.Info;
s.Opaque = true;
s.Border.BorderStyle = BorderStyleEnum.Solid;

// attach labels to each slice
for (int i = 0; i < data.Count; i++)
{
    C1.Win.C1Chart.Label lbl =
c1Chart1.ChartLabels.LabelsCollection.AddNewLabel();
    lbl.Text = string.Format("{0} ({1:c})",
    data[i]["ProductName"], data[i]["UnitPrice"]);
    lbl.Compass = LabelCompassEnum.Radial;
    lbl.Offset = 20;
    lbl.Connected = true;
    lbl.Visible = true;
    lbl.AttachMethod = AttachMethodEnum.DataIndex;
    AttachMethodData am = lbl.AttachMethodData;
    am.GroupIndex = 0;
    am.SeriesIndex = i;
    am.PointIndex = 0;
}
```

## XY-Plots (Scatter Plots)

Bar and pie charts use a single value to represent each data point (price, units in stock, or reorder level). **XYPlot** charts, by contrast, use two values to represent each data point. They are useful to depict relationships between two dimensions of the data, and are often used in statistical analysis of data.

For example, the following chart illustrates an XY-Plot depicting the relationship between product unit prices and sales.



This type of chart is often used to support statistical techniques that quantify the relationship between the variables (typically Linear Regression Analysis). For example, the preceding chart illustrates a point with unit price above \$250 that is relatively distant from the others. In a linear regression, this point would be called an "outlier", and would have a much greater influence on the result of the analysis than the other points in the series.

XY-Plot charts are also useful on their own, to show qualitative aspects of relationships. For example, are sales related to unit prices? In this example, the data seems to suggest there is indeed a relationship. Perhaps the customers prefer to buy expensive products from NorthWind and cheaper products from their local supermarkets.

The following code is used to create the chart in the preceding graphic. It is very similar to the code used to create the bar chart, except it sets the X and Y values to data, instead of using a simple sequence for the X values.

#### To write code in Visual Basic

##### Visual Basic

```
' get chart data
Dim data As DataView = _dataSet.Tables("Sales").DefaultView
data.Sort = "UnitPrice"

' configure chart
C1Chart1.Reset()
C1Chart1.ChartGroups(0).ChartType = Chart2DTypeEnum.XYPlot

' create single series for product price vs sales
Dim dscoll As ChartDataSeriesCollection = _
C1Chart1.ChartGroups(0).ChartData.SeriesList;
dscoll.Clear();

Dim series As ChartDataSeries = dscoll.AddNewSeries()

' show symbols only (no lines)
series.SymbolStyle.Color = Color.Green
series.SymbolStyle.OutlineColor = Color.Black
series.LineStyle.Pattern = LinePatternEnum.None
```

```
' populate the series
series.PointData.Length = data.Count
Dim i As Integer
For i = 0 To data.Count - 1
    series.X(i) = data(i) ("UnitPrice")
    series.Y(i) = data(i) ("ProductSales")
Next I

' attach product names to x-axis
Dim ax As Axis = C1Chart1.ChartArea.AxisX
ax.Text = "Unit Price (US$)"
ax.AnnoFormat = FormatEnum.NumericCurrency
ax.GridMajor.Visible = True

' configure y-axis
Dim ay As Axis = C1Chart1.ChartArea.AxisY
ay.Text = "Sales (thousand US$)"
ay.AnnoFormat = FormatEnum.NumericManual
ay.AnnoFormatString = "$#,##0,"
ay.GridMajor.Visible = True
```

#### To write code in C#

```
C#
// get chart data
DataGridView data = _dataSet.Tables["Sales"].DefaultView;
data.Sort = "UnitPrice";

// configure chart
c1Chart1.Reset();
c1Chart1.ChartGroups[0].ChartType = Chart2DTypeEnum.XYPlot;

// create single series for product price vs sales
ChartDataSeriesCollection
dscoll = c1Chart1.ChartGroups[0].ChartData.SeriesList;
dscoll.Clear();
ChartDataSeries series = dscoll.AddNewSeries();

// show symbols only (no lines)
series.SymbolStyle.Color = Color.Green;
series.SymbolStyle.OutlineColor = Color.Black;
series.LineStyle.Pattern = LinePatternEnum.None;

// populate the series
series.PointData.Length = data.Count;
for (int i = 0; i < data.Count; i++)
{
    series.X[i] = data[i]["UnitPrice"];
    series.Y[i] = data[i]["ProductSales"];
}
```

```
// attach product names to x-axis
Axis ax = clChart1.ChartArea.AxisX;
ax.Text = "Unit Price (US$)";
ax.AnnoFormat = FormatEnum.NumericCurrency;
ax.GridMajor.Visible = true;

// configure y-axis
Axis ay = clChart1.ChartArea.AxisY;
ay.Text = "Sales (thousand US$)";
ay.AnnoFormat = FormatEnum.NumericManual;
ay.AnnoFormatString = "$#,##0,";
ay.GridMajor.Visible = true;
```

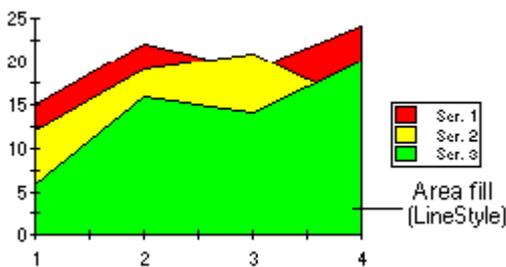
## Specific 2D Charts

The 2D Chart can display data as one of several basic chart types or sub-types. Sub-types are variations on a basic chart type, such as Bubble charts. More specialized chart types, such as **Gantt** charts, can also be created. In addition, many of the charts have the ability to change to other chart types independent of the properties, as long as the data requirements for both charts are equivalent. For example, the same data can be displayed as a **XY-Plot** and then later as a **Bar** chart.

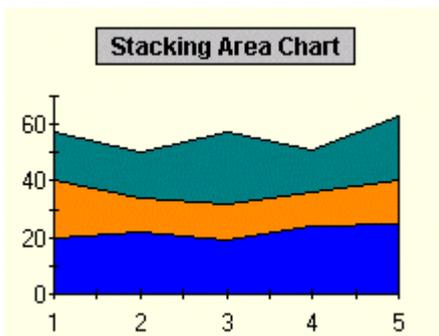
This chapter introduces the 2D Chart types available in **C1Chart**, shows how to select specific chart types, and shows how to add 3D effects to applicable chart types.

## Area Charts

An Area chart draws each series as connected points of data, filled below the points. Each series is drawn on top of the preceding series. The series can be drawn independently or stacked. Using the [LineStyle](#), the fill properties of each series can be customized. For more information, see [Line and Symbol Styles for the Series](#).



Use the ChartGroup object's [Stacked](#) property to create a stacking Area chart. Stacking charts represent the data by stacking the values for each series on top of the values from the previous series.



## To set the chart type to Area at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the [ChartType](#) property to **Area**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From

- the Gallery, select Chart type as **Area**.
- Another alternate method is to select **Chart Properties**, from the Properties pane. From the **Gallery**, select Chart type as **Area**.

## Area Chart Programming Considerations

The following table describes the Data Arrays that are used for Area charts. Each data series requires X and Y array values to be charted. Adding values to the other arrays will not have an effect for this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays:

Property	Description
X	Holds the position on the X-axis.
Y	Holds the position on the X-axis.
Y1	No effect for Area charts.
Y2	No effect for Area charts.
Y3	No effect for Area charts.

## Area Chart 3D Effects

C1Chart's 3D effects can be used with area charts or stacking area charts to create the illusion of depth with each data series. By using the depth, elevation, rotation and shading properties, you can enhance your area charts, making them stand out by creating visual depth.

To access the 3D view for an area chart, adjust the properties in the View3D object. The View3D object is a member of the PlotArea, which in turn is a member of the ChartArea. By adjusting the View3D object properties, [Depth](#), [Elevation](#), [Rotation](#) and [Shading](#) you can customize the 3D view.

Note that the [Depth](#) property is the key to all 3D type chart logic. While the [Elevation](#) and [Rotation](#) properties modify the way a user views the chart, it is the [Depth](#) property that actually dictates whether a chart is 3D. By using a non-zero value for the [Depth](#) property and setting the [Elevation](#) and [Rotation](#) property values to zero, you have created a 3D chart even though nothing seems to have changed. In effect you are looking at the "front" surface of the chart, which is visually represented in the same way as a standard area chart.

Note also, that there may be times when it is desirable to chart some data with a 3D view, and other data in the 2D plane. For these instances, adjust the [Use3D](#) property associated with each ChartGroup.

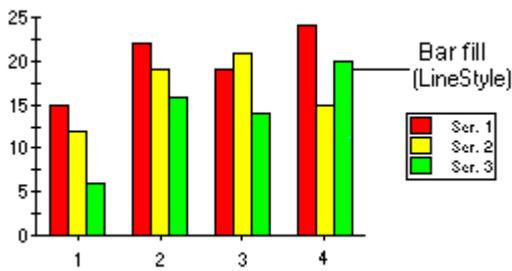


Axes origin settings are not supported when Chart2D is used to represent 3D effects with Area charts.

## Bar Charts

A **Bar** chart is an inverted column chart where the category axis is the vertical axis. A Bar/Column chart draws each series as a bar in a cluster. The number of clusters is the number of points in the data. Each cluster displays the nth data point in each series. Using the [LineStyle](#), the fill properties of each series can be customized. For more information, see [Line and Symbol Styles for the Series](#).

The following image represents a **Bar** chart:



## To set the chart type to Bar at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the **ChartType** property to **Bar**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the Gallery, select **ChartType** as **Bar**.
- Another alternate method is to select **Chart Properties**, from the Properties pane. From the Gallery, select **ChartType** as **Bar**.

## Bar Chart Programming Considerations

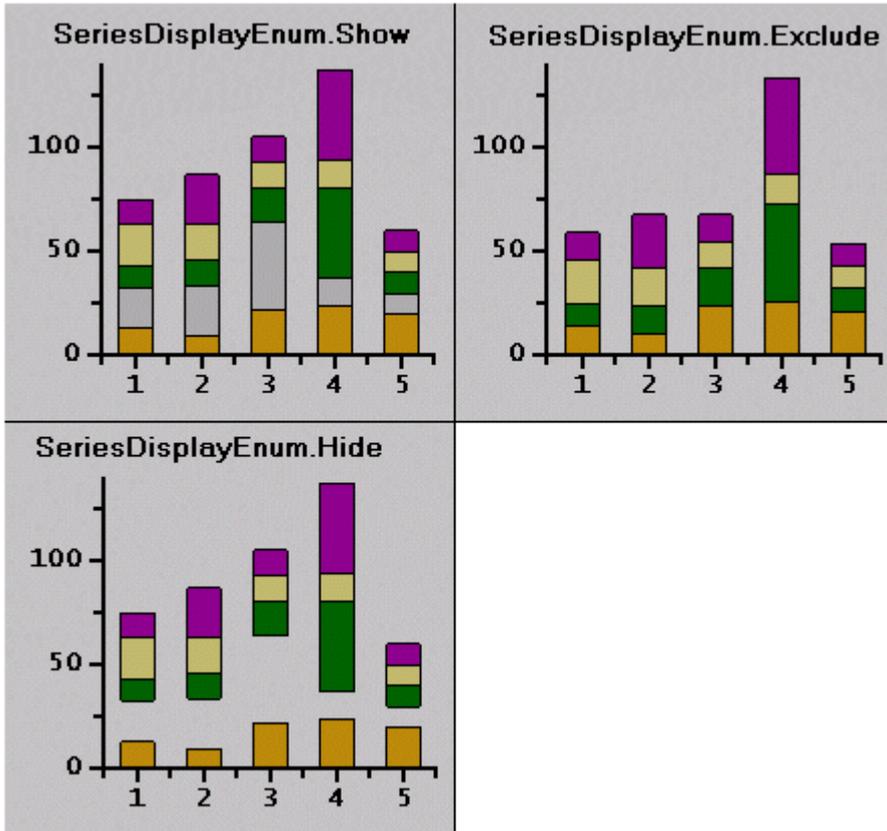
The following tables describes the Data Arrays that are used for Bar charts. Each data series requires X and Y array values to be charted. Adding values to the other arrays will not have an effect for this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays.

Property	Description
X	Holds the position on the X-axis.
Y	Holds the position on the Y-axis.
Y1	No effect for Bar charts.
Y2	No effect for Bar charts.
Y3	No effect for Bar charts.

## Floating Bar Charts

Each series in a Stacking Bar chart can be either hidden (where the series is charted as a blank set) or excluded (where the series is not considered part of the chart data). Use the **Display** property of the ChartDataSeries object to determine if each series in the chart is shown, hidden, or excluded.

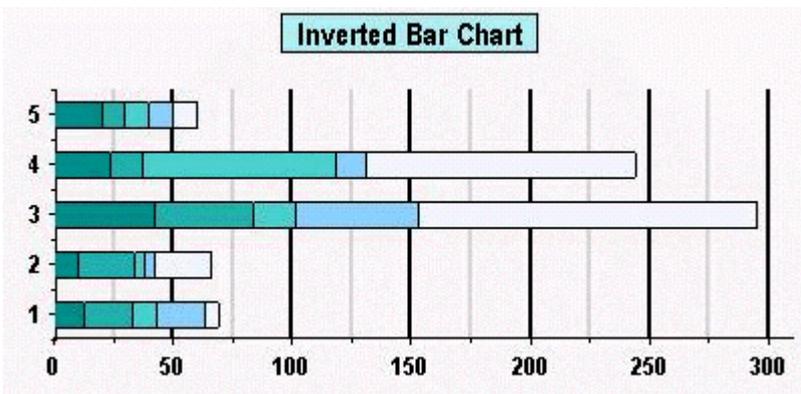
In the following examples, the second series is shown, hidden, and then excluded.



The **Display** property can be accessed at design time in the **SeriesList Collection Editor**. This editor can be accessed by opening up the **ChartGroupsCollection Editor**, expanding the **ChartData** node, then clicking the **ellipsis** next to the **SeriesList** property.

## Inverted Bar Charts

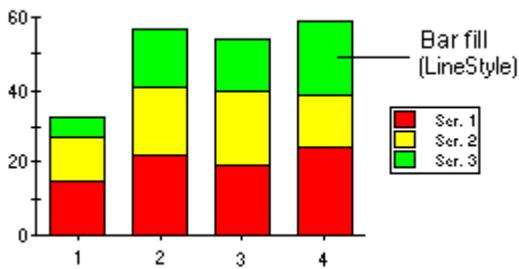
A Bar chart is an inverted Column chart in the sense that the X and Y axes are reversed. For more information, see [Inverted and Reversed Chart Axes](#).



## Stacking Bar Charts

A Stacking Bar chart draws each series as a portion of a stacked bar cluster, the number of clusters being the number of points in the data. Each bar displays the *n*th data point in each series. Cylinder, Pyramid, and Cone Bar charts can also be stacked by setting the **Stacked** property to **True**. Using the **LineStyle**, the fill properties of each series can be

customized. For more information, see [Line and Symbol Styles for the Series](#).



## Code Example

The following example creates a Stacked Bar chart. Just add the **C1Chart** control to your Visual Studio project, and add the following code to see a running example of the Stacked Bar chart:

### To write code in Visual Basic

#### Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load

    ' Clear previous data
    C1Chart1.ChartGroups(0).ChartData.SeriesList.Clear()

    ' Data
    Dim items As String() = New String() {"Item1", "Item2", "Item3"}
    Dim sales2005 As Integer() = New Integer() {800, 1500, 2000}
    Dim sales2006 As Integer() = New Integer() {1000, 1200, 1400}

    ' Create first series
    Dim ds2005 As C1.Win.C1Chart.ChartDataSeries =
    C1Chart1.ChartGroups(0).ChartData.SeriesList.AddNewSeries()
    ds2005.Label = "2005"
    ds2005.LineStyle.Color = Color.Yellow
    ds2005.X.CopyDataIn(items)
    ds2005.Y.CopyDataIn(sales2005)

    ' Create second series
    Dim ds2006 As C1.Win.C1Chart.ChartDataSeries =
    C1Chart1.ChartGroups(0).ChartData.SeriesList.AddNewSeries()
    ds2006.Label = "2006"
    ds2006.LineStyle.Color = Color.Red
    ds2006.AutoEnumerate = True
    ds2006.Y.CopyDataIn(sales2006)

    ' Set chart type
    C1Chart1.ChartGroups(0).ChartType =
    C1.Win.C1Chart.Chart2DTypeEnum.Bar
    C1Chart1.ChartGroups(0).Stacked = True

    ' Set y-axis minimum
    C1Chart1.ChartArea.AxisY.Min = 0

```

```
        c1Chart1.Legend.Visible = True

        ' Remove the Axes caption
            c1Chart1.ChartArea.AxisX.Text = ""
        c1Chart1.ChartArea.AxisY.Text = ""
    End Sub
```

### To write code in C#

```
C#
private void Form1_Load(object sender, EventArgs e)
{
    // Clear previous data
        c1Chart1.ChartGroups[0].ChartData.SeriesList.Clear();
    // Data
        string[] items = new string[] { "Item1", "Item2", "Item3"};
    int[] sales2005 = new int[] { 800, 1500, 2000};
    int[] sales2006 = new int[] { 1000, 1200, 1400};

    // Create first series
        C1.Win.C1Chart.ChartDataSeries ds2005 =
    c1Chart1.ChartGroups[0].ChartData.SeriesList.AddNewSeries();
    ds2005.Label = "2005";
    ds2005.LineStyle.Color = Color.Yellow;
    ds2005.X.CopyDataIn( items);
    ds2005.Y.CopyDataIn( sales2005);

    // Create second series
        C1.Win.C1Chart.ChartDataSeries ds2006 =
    c1Chart1.ChartGroups[0].ChartData.SeriesList.AddNewSeries();
    ds2006.Label = "2006";
    ds2006.LineStyle.Color = Color.Red;
    ds2006.AutoEnumerate = true;
    ds2006.Y.CopyDataIn( sales2006);

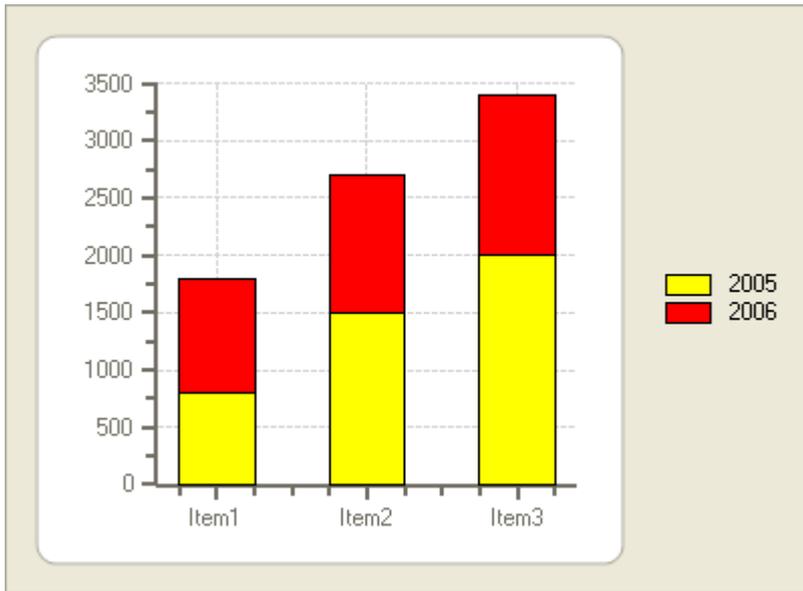
    // Set chart type
        c1Chart1.ChartGroups[0].ChartType =
    C1.Win.C1Chart.Chart2DTypeEnum.Bar;
    c1Chart1.ChartGroups[0].Stacked = true;

    // Set y-axis minimum
        c1Chart1.ChartArea.AxisY.Min = 0;

    c1Chart1.Legend.Visible = true;

    // Remove the Axes caption
        c1Chart1.ChartArea.AxisX.Text = "";
    c1Chart1.ChartArea.AxisY.Text = "";
}
```

The code example above produces the following chart:



## Special Bar Chart Properties

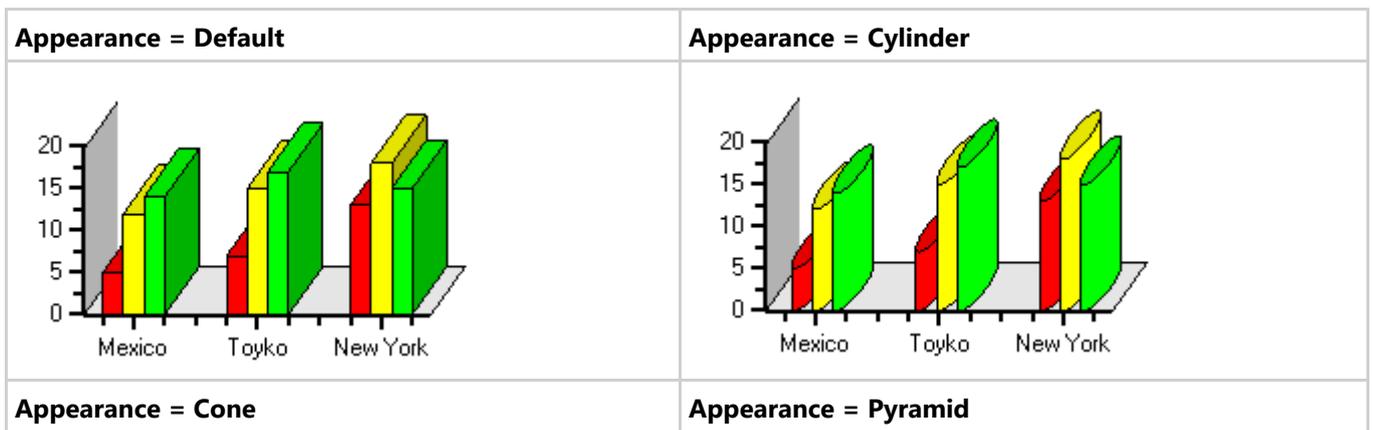
A Bar chart draws each series as a bar in a cluster. You can display each series on a single row with 2D Bar charts, or display each series in multiple rows with 3D Bar charts. The 3D Bar chart provides an interesting alternative view; you can view the front side of the 3D Bar or Column charts rather than the typical side view. The sizing and spacing of the clusters for Bar and Stacking Bar charts can be customized. Additionally, you can change the Bar chart's appearance to any of the following shapes: cylinder, cone, or pyramid.

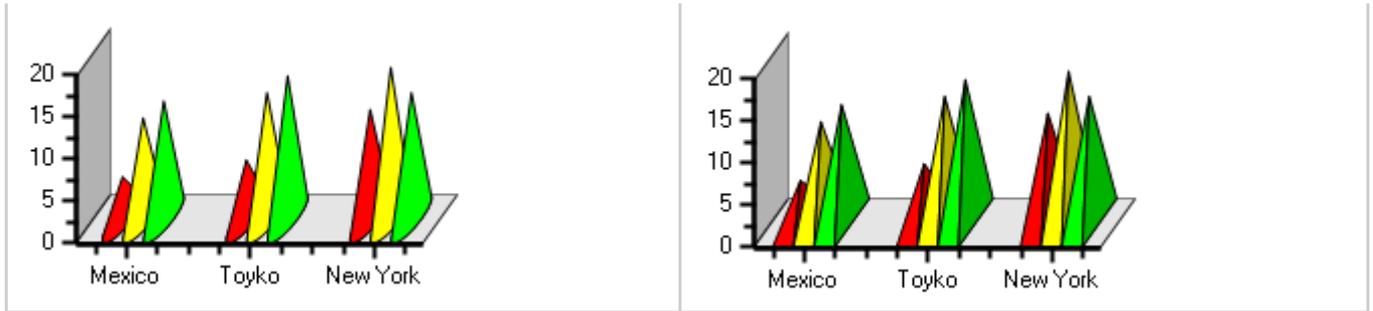
### Appearance

While Bar charts are commonly represented as rectangular bars (default), you can alternatively represent the chart with cylinders, cones, or pyramids for a different effect. To change the bar's shape from its default setting to cylinder, cone, or pyramid, use the [Appearance](#) property.

**Note:** The following Cylinder and Cone Bar charts appear more elliptical rather than circular due to the depth of the charts. To make them appear more circular, you can decrease the [Depth](#) property. For more information on 3D Bar Chart effects, see [Bar Chart 3D Effects](#).

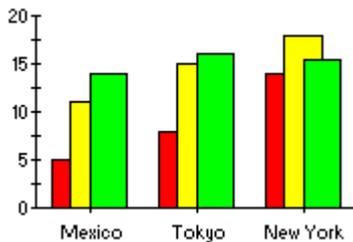
The following figures illustrate the values of the [Appearance](#) property:





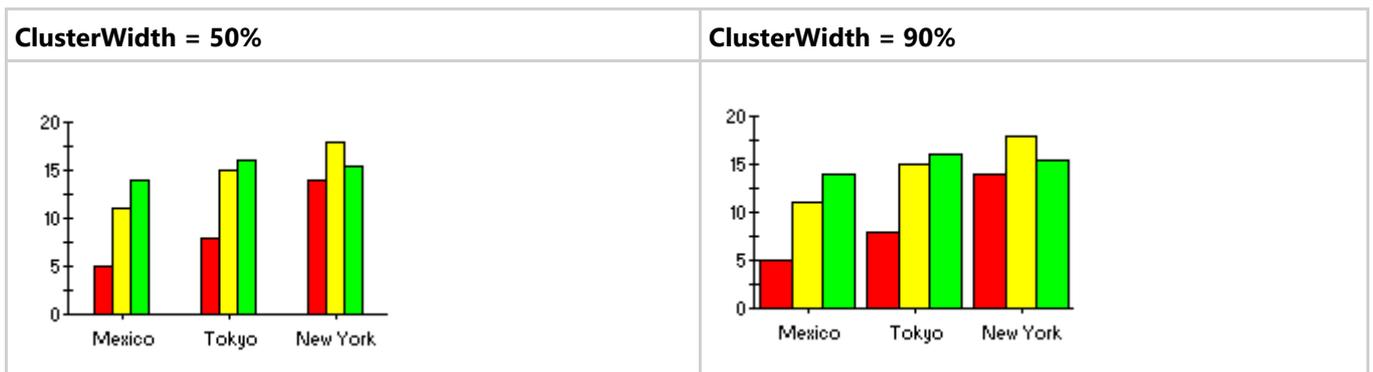
## Cluster Overlap

Use the [ClusterOverlap](#) property to set the amount that bars overlap each other in a cluster. The value represents the percentage of bar overlap, with valid values between 0 and 100. The following figure illustrates a bar chart with a [ClusterOverlap](#) of 50 percent:



## Cluster Width

Use the [ClusterWidth](#) property to set the space used by each bar cluster. The value represents the percentage of available space, with valid values between 0 and 100.



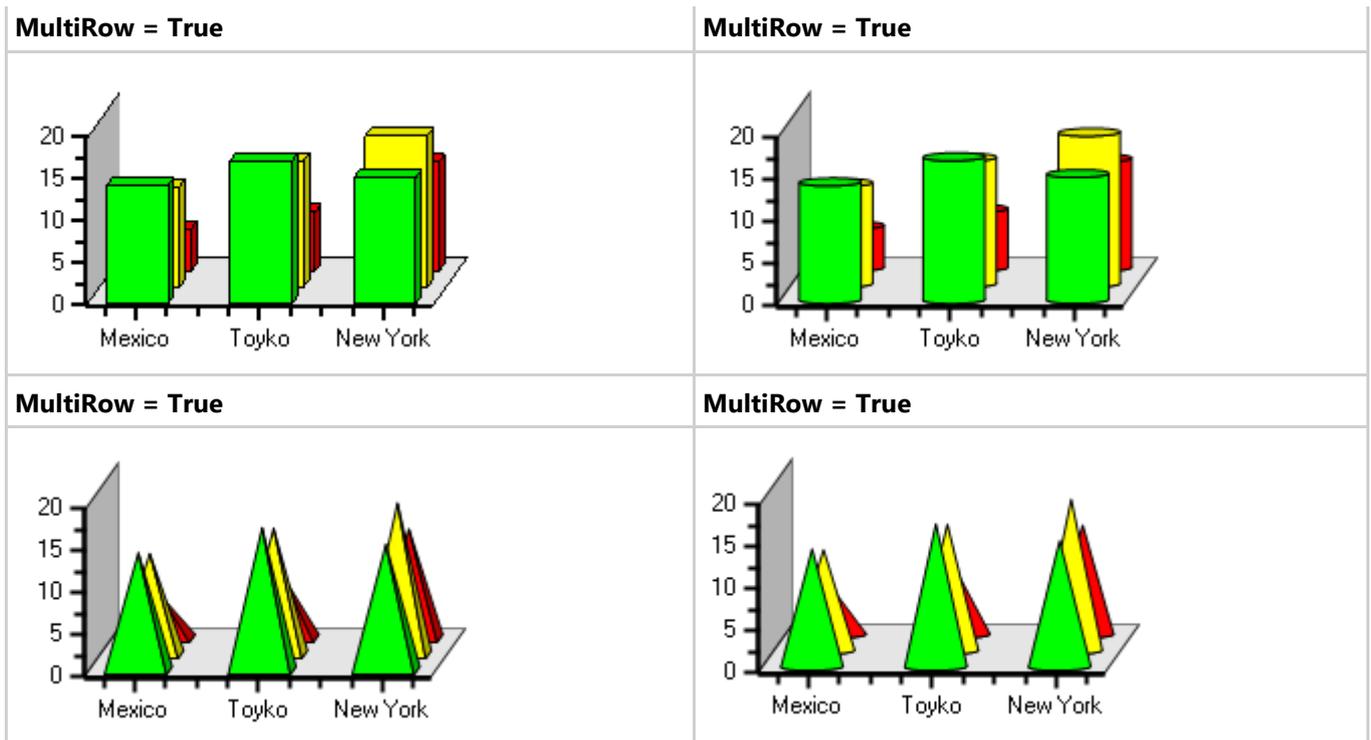
The Bar class properties, [ClusterWidth](#) and [ClusterOverlap](#), can be accessed at design time under the Bar node in the **ChartGroupsCollection Editor**.

## MultiRow

When the [Use3D](#) property is set to True to enable 3D Charts you can use the [MultiRow](#) property to display a new row for each bar or column in the cluster.

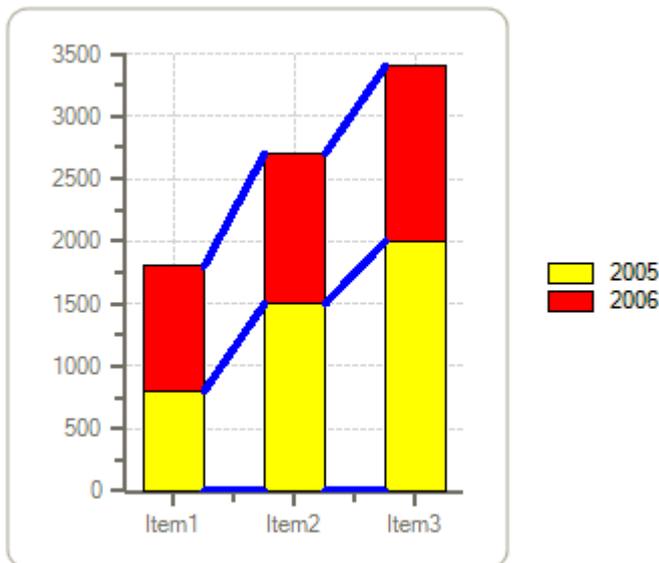
The following figures illustrate the effect of the [MultiRow](#) property for each Bar chart type:





## Lines for Stacked 2D Bar and Column Charts

Set the [BarLines](#) property to True when you want to show the lines between the data series rectangles from point to point in stacked 2D bar and column charts. When you set the [BarLines](#) property to True you can use the [BarLineColor](#) and [BarLineThickness](#) properties to specify the line color and line thickness for the bar lines as depicted in the following figure:



Using bar lines in Stacked 2D Bar or Column charts makes it easier to see the quantitative information by using the bar lines to compare the values across the X-axis (Column charts) or the Y-axis (Bar charts).

## Bar Chart 3D Effects

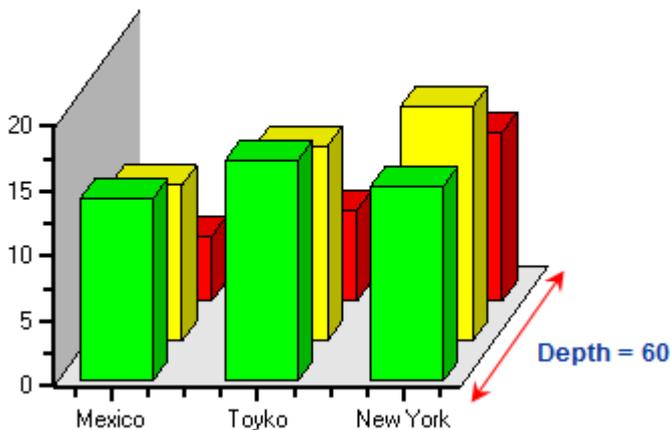
You can use C1Chart's 3D effects with Bar charts or Stacking Bar charts to enhance the appearance of your charts. By

using the depth, elevation, rotation, and shading properties, you can create the illusion of depth with each data series.

When the [Depth](#) property is used to modify the depth of the bar chart it does not affect the shape of the columns, however, it affects the shape of the cones, pyramids, and cylinders. The width of the of the cylinders automatically changes with the number of bars on the X axis, but the depth of the bar depends on the [Depth](#) property.

The following image represents such a chart that stands out from applying the 3D effects:

**Elevation = 30, Rotation = 25, Shading = ColorDark**



To access the 3D view for a bar chart, adjust the properties in the View3D object. The View3D object is a member of the PlotArea, which in turn is a member of the ChartArea. By adjusting the View3D object properties, [Depth](#), [Elevation](#), [Rotation](#) and [Shading](#) you can customize the 3D view.

Note that the [Depth](#) property is the key to all 3D type chart logic. While the [Elevation](#) and [Rotation](#) properties modify the way a user views the chart, it is the [Depth](#) property that actually dictates whether a chart is 3D. By using a non-zero value for the [Depth](#) property and setting the [Elevation](#) and [Rotation](#) property values to zero, you have created a 3D chart even though nothing seems to have changed. In effect you are looking at the front surface of the chart, which is visually represented in the same way as a standard area chart.

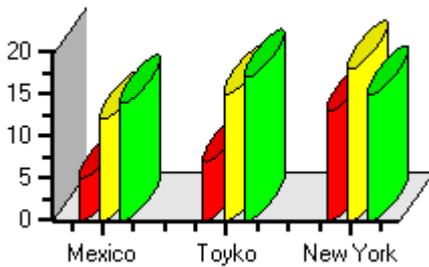
Note also, that there may be times when it is desirable to chart some data with a 3D view, and other data in the 2D plane. For these instances, adjust the [Use3D](#) property associated with each ChartGroup.

## Variations of Bar Charts

For variations of the Bar chart, you can use Cylinder, Cone, and Pyramid charts to represent series of data. The charts function the same as 3D Bar and Column charts: comparing series of data. They only differ in appearance; instead of using rectangles to represent bars, they use cylinders, cones, or pyramid shapes. The following topics provide further information on the Cylinder, Cone, and Pyramid charts.

## Cylinder Charts

A Cylinder chart is a variation of the Bar and Column charts. It represents the bars or columns as cylinders. The Cylinder chart creates long circular boxes of the same base on both ends. Like all bar and column charts, the Cylinder bar chart is appropriate for comparing individual items or groups of items.

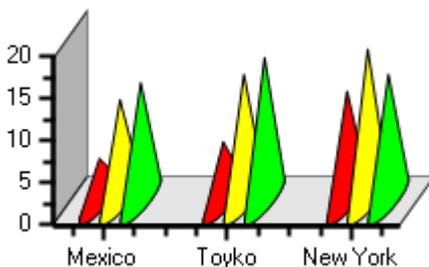


## To Change the Chart Type

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the **ChartType** property to **Bar**, then set the **Appearance** property to **Cylinder**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the **Gallery**, select **Cylinder**.
- Another alternate method is to select **Cylinder**, from the **C1Chart** toolbar.

## Cone Charts

A Cone chart is a variation of the 3D Bar and Column charts. It represents the bars or columns as cones. The cone chart essentially is a rotated triangle. It has a flat circular base and one curved side topped by a higher point.

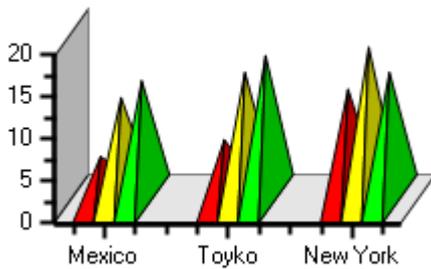


## To set the bar chart to Cone at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the **ChartType** property to **Cone**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the **Gallery**, select **Cone**.
- Another alternate method is to select **Cone**, from the **C1Chart** toolbar.

## Pyramid Charts

A Pyramid chart is a variation the 3D-Bar and Column charts. It represents the bars or columns as pyramids. The Pyramid chart is similar to the cone chart except for their base. Pyramid charts are often used for geographical purposes.

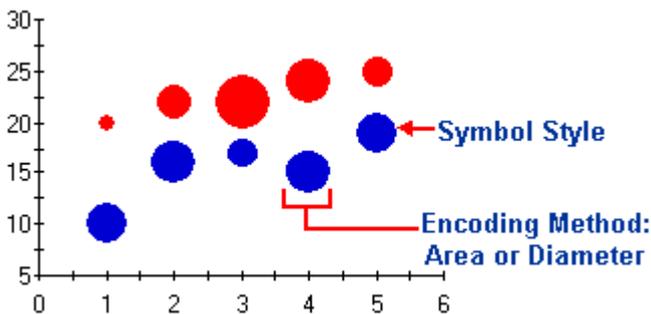


## To set the bar chart to Pyramid at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the **ChartType** property to **Bar**, then set the **Appearance** property to **Pyramid**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the Gallery, select **Pyramid**.
- Another alternate method is to select **Pyramid**, from the **C1Chart** toolbar.

## Bubble Charts

A Bubble chart combines two independent values to supply both the point y value and the point sizes. Bubble charts are used to represent an additional data value at each point by changing its size. The Y array elements determine the Cartesian position (as in a XY-Plot chart), and the Y1 element values determine the size of the bubble at each point. The size of the points can be encoded according to area or diameter. Using the **LineStyle** and **SymbolStyle** properties, the symbol style and color, and the appearance of connecting lines appear can be specified. For more information, see [Line and Symbol Styles for the Series](#).



## To set the chart type to Bubble at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the **ChartType** property to **Bubble**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the Gallery, select **ChartType** as XY-Plot and Chart sub-type as **Bubble**.
- Another alternate method is to select **Chart Properties** from the Properties pane. From the Gallery, select **ChartType** as XY-Plot and Chart sub-type as **Bubble**.

## Bubble Chart Programming Considerations

The following table lists the Data Arrays used for Bubble charts. Each data series requires X, Y, and Y1 array values to

be charted. Adding values to the other arrays will not affect this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays:

Property	Description
X	Holds the position of the bubble relative the X-axis.
Y	Holds the position of the bubble relative to the Y-axis.
Y1	Holds the relative size of the bubble.
Y2	No effect for Bubble charts.
Y3	No effect for Bubble charts.

## Special Bubble Chart Properties

A Bubble chart combines two series to draw a plot chart with varying point sizes. The encoding method for the size of the bubbles, as well as their minimum and maximum size can be specified.

### Encoding Method

Use the [EncodingMethod](#) property to set whether to size the bubbles according to diameter or area. When specifying the size of the bubble, both the diameter and area are measured as a percentage of the total diameter or area of the PlotArea. Inverting these values (making the Minimum value larger than the Maximum) draws large bubbles for small values, and small bubbles for large values.

The [EncodingMethod](#) is a property of the Bubble object and can be accessed through the Bubble node in the **ChartGroupsCollection Editor**.

### Maximum and Minimum Size

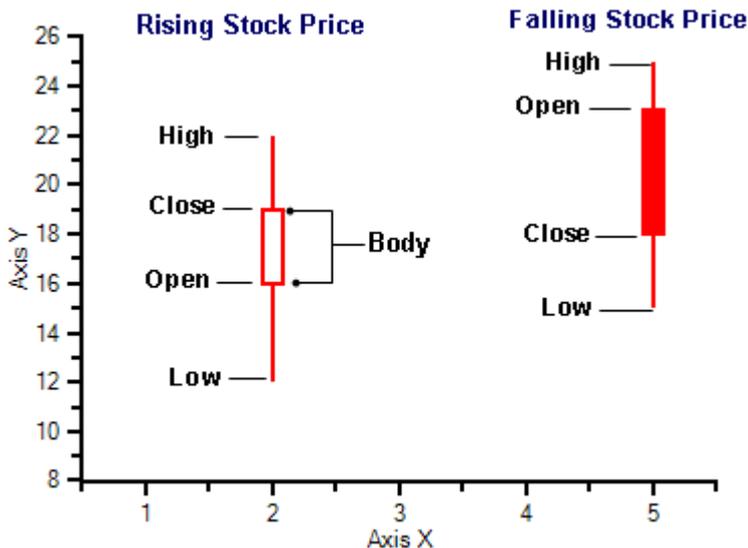
The maximum and minimum allowable size for bubbles can be set using the [MaximumSize](#) and [MinimumSize](#) properties respectively. These are properties of the Bubble object and can be accessed through the Bubble node in the **ChartGroups Collection Editor** as well.

## Candle Charts

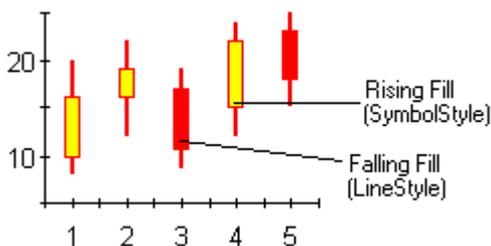
**Candle, HiLo, HiLoOpenClose** charts are all types of Stock charts used in financial applications to show the opening, closing, high and low prices of a given stock. A **Candle** chart is a special type of **HiLoOpenClose** chart that is used to show the relationship between the open and close as well as the high and low. Like, **HiLoOpenClose** charts, **Candle** charts use the same price data (high, low, open, and close values) except they include a thick candle-like body that uses the color and size of the body to reveal additional information about the relationship between the open and close values. For example, long transparent candles show buying pressure and long filled candles show selling pressure.

The Candle chart is made up of the following elements: candle, wick, and tail. The candle or the body (the solid bar between the opening and closing values) represents the change in stock price from opening to closing. The thin lines, wick and tail, above and below the candle depict the high/low range. A hollow candle or transparent candle indicates a rising stock price (close was higher than open). In a hollow candle, the bottom of the body represents the opening price and the top of the body represents the closing price. A filled candle indicates a falling stock price (open was higher than close). In a filled candle the top of the body represents the opening price and the bottom of the body represents the closing price.

C1Chart creates the Candle chart with using the Y value for the High, Y1 for the low, Y2 for the open, and Y3 for the close. C1Chart automatically fills the falling candle with the value of the line color.



Using the [LineStyle](#) and [SymbolStyle](#) properties and the [HiLoData](#) class, the fill and line properties of each series can be customized. For more information, see [Special Candle Chart Properties](#). To see how to create a candle chart from start to finish using the designer or programmatically, see [Candle Chart Tutorial](#).



## To set the chart type to Candle at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the [ChartType](#) property to **Candle**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the Gallery, select [ChartType](#) as Stock and Chart sub-type as Candle.
- Another alternate method is to select **Chart Properties** from the Properties pane. From the Gallery, select [ChartType](#) as Stock and Chart sub-type as Candle.

## Candle Chart Programming Considerations

The following table lists the Data Arrays used for Candle charts. Each data series requires the use of the X array and four Y array values including Y, Y1, Y2, and Y3 to be charted.

Property	Description
X	Holds the position on the X-axis.

Y	Holds the high value for the Candle chart.
Y1	Holds the low value for the Candle chart.
Y2	Holds the open value for the Candle chart.
Y3	Holds the close value for the Candle chart.

## Special Candle Chart Properties

When the [ChartType](#) property is set to **Candle**, you can use the following properties to indicate a rising or falling stock price:

- [FillFalling](#)
- [FillTransparent](#)

### To show falling prices

Set the [FillFalling](#) property to True. The color of the fill is determined by the value of the [ChartDataSeries.LineStyle](#).

### To show raising prices

Set the [FillTransparent](#) property to True. This will make the body of the open candles transparent or hollow. If you would like to specify a color for the raising values, then set the [FillTransparent](#) property to False and specify a color for the [SymbolStyle](#) property. Note that it is recommended to use a different color so there is one color for the line style and a different color for the symbol style. For more information, see [Creating a Fill Color for Rising Candles](#).

## Gantt Charts

A **Gantt** chart is used to illustrate a timeline of various tasks and outline the critical activities to the project's completion.

The **Gantt** chart has the following similarities to the **Bar** and the **HiLo** charts:

- Like the Bar chart, the **Gantt** chart uses bars, but it is commonly displayed as an inverted and reversed bar chart.
- Similar to the **HiLo** chart, where the elements of the [Y](#) and [Y1](#) arrays in each series represent the "high" value and the "low" value, the **Gantt** chart uses the [Y](#) and [Y1](#) elements to represent the start and finish time of a task.

A **Gantt** chart clearly illustrates a timeline in the following ways:

- **Activities/Tasks**

The activities/tasks are displayed along the left side of the chart and a timeline is shown at the top or bottom of the chart.

- **Task Duration**

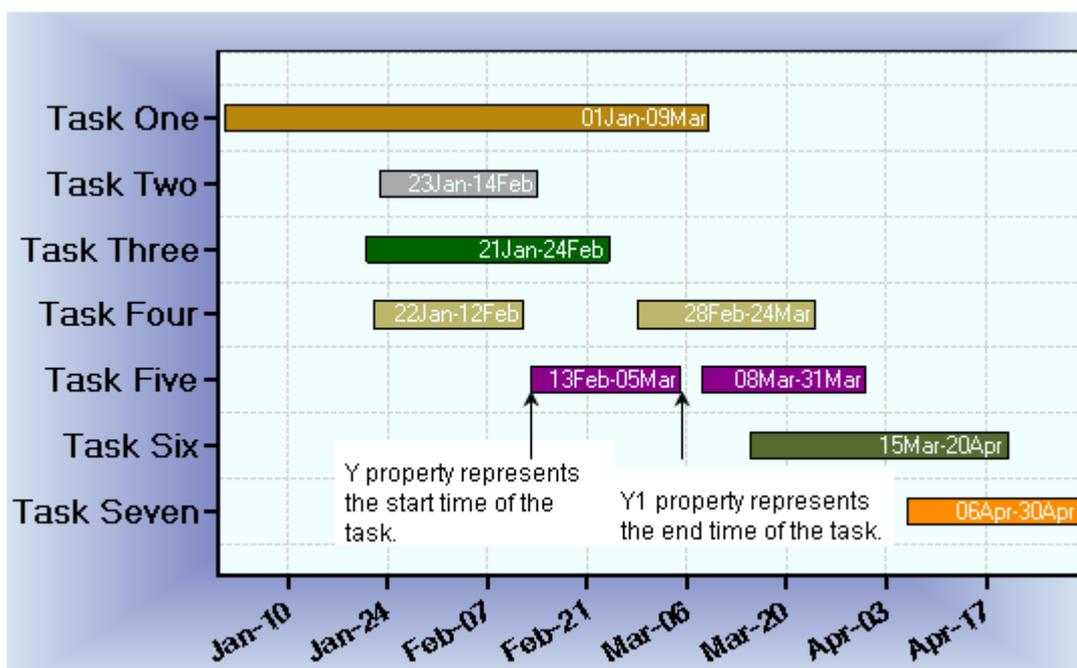
The duration of each project's task is represented as a bar. The beginning of the bar indicates the start time of the activity or task. The end of the bar indicates the finish or completion time of the activity or task.

- **Critical Activities or Accomplishments**

Typically, in **Gantt** charts, special characters or colors are used to represent critical activities or accomplishments through the duration of each task.

In a **Gantt** chart you can create a new task by adding a new series to the [ChartDataSeries](#). In each series there are several properties that you will use to create the information for your data in the task. For instance, you can use the **Y** property to fill in the data for the start time of your task and the **Y1** property to record the data for the end time of your task. You can choose the type of data that you want to use for your **Gantt** chart from the [DataType](#) property of the [ChartDataArray](#).

The **Gantt** chart below shows the start and finish time of each task. Gantt charts can display single or multiple projects in one task. In some cases, a task is assigned multiple subtasks. Each bar represents one task or subtask. Having a few bars in one axis can span a large range of x and y values. The data for a Gantt chart is more effective when it is displayed horizontally rather than vertically. As a result, the chart is inverted when the x-axis is vertical and the y axes are horizontal. C1Chart provides the [Inverted](#) property of the [ChartArea](#) and the [Reversed](#) property of the [Axis](#). Notice that each horizontal bar is filled in with a solid color. Often, this represents a completed task. Also, each horizontal bar contains a label indicating the start date and finish date. The labels can be added to the horizontal bars at run time or at design time by using the [Label](#) property of the [ChartDataSeries](#) class.



## To set the chart type to Gantt at design time

- Expand the **ChartGroups** node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the [ChartType](#) property to **Gantt**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the Gallery, select [ChartType](#) as **Gantt**.
- Another alternate method is to select **Chart Properties** from the Properties pane. From the Gallery, select [ChartType](#) as **Gantt**.

## Gantt Chart Programming Considerations

The following table lists the Data Arrays and describes the effect each one has on Gantt charts. Each data series requires the use of the X array and two Y array values including Y and Y1 to be charted. Adding values to the other arrays will not affect this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays:

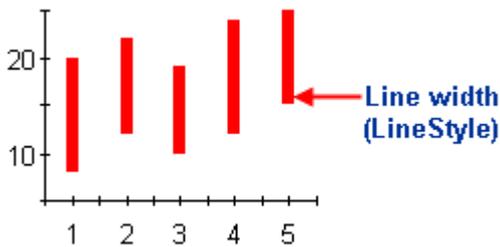
Property	Description
----------	-------------

X	Holds the position on the X-axis.
Y	Holds the start value for the start time of the project.
Y1	Holds the end value for the finish date of the project.
Y2	No effect for Gantt charts.
Y3	No effect for Gantt charts.

## HiLo Charts

A **HiLo** chart combines two independent values to supply high and low data for each point in a series. HiLo charts are used primarily in financial applications to show the high and low price for a given stock. The elements of the Y and Y1 arrays in each series of a HiLo chart represent the "high" value, and the "low" value.

Using the [LineStyle](#), the fill and line properties of each series can be customized. For more information, see [Line and Symbol Styles for the Series](#).



## To set the chart type to HiLo at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the [ChartType](#) property to **HiLo**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the Gallery, select [ChartType](#) as Stock and Chart sub-type as Hi-low.
- Another alternate method is to select **Chart Properties** from the Properties pane. From the Gallery, select [ChartType](#) as Stock and Chart sub-type as Hi-low.

## HiLo Chart Programming Considerations

The following table lists the Data Arrays used for HiLo charts. Each data series requires the use of the X array and two Y array values including Y and Y1 to be charted. Adding values to the other arrays will not affect this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays.

Property	Description
X	Holds the position on the X-axis.
Y	Holds the high value for the chart.
Y1	Holds the low value for the chart.
Y2	Not in effect for HiLo charts.
Y3	Not in effect for HiLo charts.

## HiLoOpenClose Chart Programming Considerations

The following table lists the Data Arrays used for HiLoOpenClose charts. Each data series requires four Y array values including Y, Y1, Y2, and Y3 to be charted.

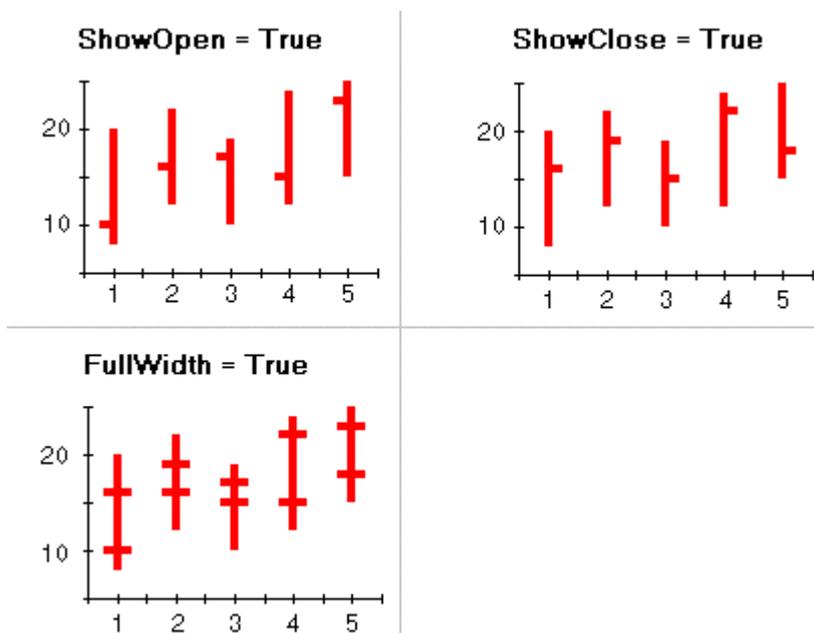
Property	Description
X	Holds the position on the X-axis.
Y	Holds the high value for the chart.
Y1	Holds the low value for the chart.
Y2	Holds the open value for the chart.
Y3	Holds the close value for the chart.

## Special HiLoOpenClose Chart Properties

The developer can specify how the open and close tick marks are displayed on the chart.

### To Display Open and Close Ticks

- Use the [ShowOpen](#), [ShowClose](#) and [FullWidth](#) properties to determine how the open and close tick marks are displayed.
- Enable [ShowOpen](#) to display the open tick marks and enable [ShowClose](#) to display the close tick marks.
- Enable [FullWidth](#) to draw the open and close tick marks on both sides of the vertical range.



### To format the Lines and Ticks

- Look at the sample, [HLCandle](#)

## Histogram Charts

A histogram is a chart that takes a collection of raw data values and plots the frequency distribution. It is frequently used with grouped data, which is generated by measuring a collection of raw data and plotting the number of data values that fall within defined intervals. Note that raw values are not used to generate data for a histogram, but are used to generate a frequency instead. While showing similarities to bar charts, it is important to note that histograms are used with quantitative variables whereas bar charts are commonly used with qualitative variables.

A histogram is beneficial for pinpointing prominent features of the distribution of data for a quantitative variable. The important features for a quantitative variable include the following:

- It reveals the typical average value.
- The data yields a general shape. The data values can be distributed symmetrically around the middle or they can be skewed.
- If there are distant values from the group of data it shows them as outlier values.
- The data values can be near or far to the typical value.
- The distribution may result in a single peak or multiple peaks and valleys.

 **Note:** The interval values represent the frequency of data elements. There are many different techniques used for displaying frequency. Frequency is commonly displayed in the shape of a polygon, column, or bar graph.

### To set the chart type to Histogram at design time

- Expand the ChartGroups node in the .NET Properties window. In the right pane of the editor, set the [ChartType](#) property to Histogram.
- Another alternate method is to expand the ChartGroups node in the .NET Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the [ChartType](#) property of the appropriate ChartGroup to **Histogram**.

## Histogram Chart Programming Considerations

Unlike other Cartesian charts, a histogram input is not specified by X and Y coordinates. Instead, a single array of raw data values is specified in the Y ChartDataArray of each ChartDataSeries. Depending upon the IntervalCreationMethod selected, histogram input may also use the X ChartDataArray of a ChartDataSeries to specify interval boundaries.

The following table lists the Data Arrays and describes the effect each one has on Histogram charts.

Property	Description
X	Holds the interval boundaries for histograms with XDataBoundaries as the IntervalCreationMethod.
Y	Holds the raw data values.
Y1	No effect for Histogram charts.
Y2	No effect for Histogram charts.
Y3	No effect for Histogram charts.

## Types of Histogram Graphs

There are several types of histogram charts. **C1Chart** uses the following types of histograms:

- Histogram

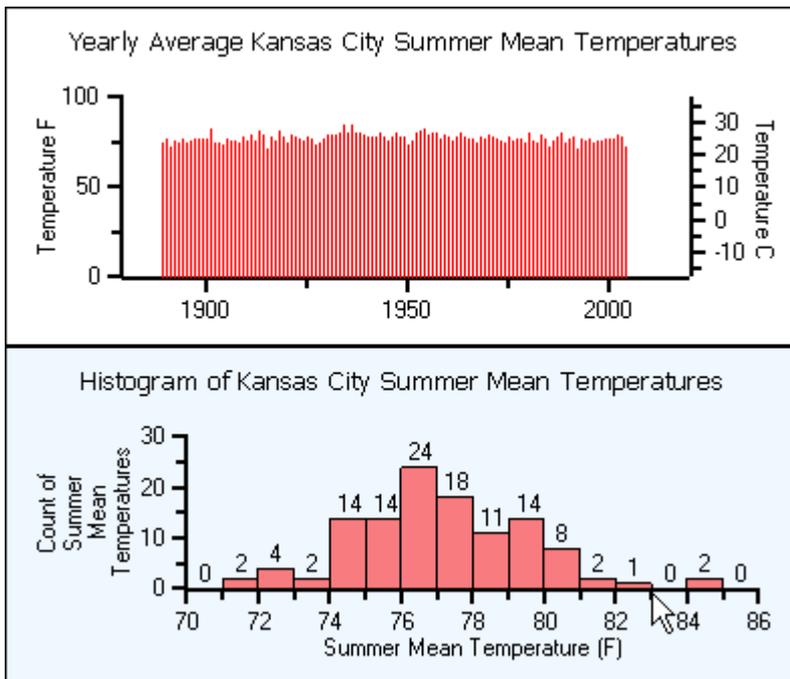
- Frequency Graph
- Stepped Frequency Graph

## Histogram Graph

A **Histogram** chart's appearance is similar to a bar chart. A minor difference in appearance is that the rectangles in a bar chart are commonly separated by a space, whereas the rectangles in a histogram are not separated by a space. Histogram's columns are not separated because the column boundaries represent the actual X axis values bounding the interval represented by each column.

While the histogram and bar charts' appearances relate, their functionality does not. A bar chart is created from data points whereas a Histogram is created from the frequency distribution of the data.

The charts following illustrate the difference between a bar chart and a histogram chart. Both of the charts use exactly the same Y data. The bar chart (top) shows each average mean temperature for each year in which it occurred. The histogram chart (bottom) using the same input temperature data automatically tabulates the number of temperatures that fall within each interval and draws the resulting histogram. For convenience, chart labels with the count in each interval have been added at the top of each interval.



The chart below shows the details and properties of a histogram. It is a scatter chart of one hundred random data coordinates in one ChartGroup, and in the second ChartGroup, a histogram based on the distribution of distances from each scatter data point to the intersection of the markers.



requires **N+1** unique **X** values.

To access the [IntervalCreationMethod](#) property at design time, expand the ChartData node and click the **ellipsis** button next to the SeriesList node. The **ChartDataSeries Collection Editor** will appear. Locate the **Histogram** node and expand it. Select the [IntervalCreationMethod](#) property and choose the type of method by selecting from one of the three methods in the [IntervalCreationMethod](#) property's drop-down box. The [IntervalCreationMethod](#) property can also be accessed through code by entering the following:

#### To write code in Visual Basic

Visual Basic

```
cds.Histogram.IntervalCreationMethod = IntervalMethodEnum.SemiAutomatic
```

#### To write code in C#

C#

```
cds.Histogram.IntervalCreationMethod = IntervalMethodEnum.SemiAutomatic;
```

Note that the text, "cds," shown above, is the variable name for the [ChartDataSeries](#) object. In the code example above, the SemiAutomatic method is used. You could also use the Automatic or XdataBoundaries method by assigning either one to the [IntervalMethodEnum](#).

You can display the intervals and counts as a Histogram, Frequency, or a Stepped Frequency by using the [DisplayType](#) property of the [ChartHistogram](#) object as shown in the following example:

#### To write code in Visual Basic

Visual Basic

```
'Displays the chart histogram as a Histogram  
ch.DisplayType = DisplayTypeEnum.Histogram  
  
'Displays the chart histogram as a frequency graph  
ch.DisplayType = DisplayTypeEnum.FrequencyGraph  
  
'Displays the chart histogram as a stepped frequency graph  
ch.DisplayType = DisplayTypeEnum.SteppedFrequencyGraph
```

#### To write code in C#

C#

```
//Displays the chart histogram as a Histogram  
ch.DisplayType = DisplayTypeEnum.Histogram;  
  
//Displays the chart histogram as a frequency graph  
ch.DisplayType = DisplayTypeEnum.FrequencyGraph;  
  
//Displays the chart histogram as a stepped frequency graph  
ch.DisplayType = DisplayTypeEnum.SteppedFrequencyGraph;
```

 **Note:** The text, "ch," in the above example, is the variable name for the [ChartHistogram](#) object.

Sometimes a few data elements appear far from the rest of the data elements. In this circumstance, the distant value is referred to an outlier value. In most cases, the outlier will not fall into an interval since it's so far from the rest of the

data, therefore, the value of the data element will be missed.

To prevent outlier values from exclusion, C1Chart's [ChartHistogram](#) object provides the following two properties:

- [AboveIntervalCount](#)

The [AboveIntervalCount](#) returns the number of values greater than the ending value of the last interval.

- [BelowIntervalCount](#)

The [BelowIntervalCount](#) returns the number of values numerically less than the starting value of the first interval.

In the designer you can find the [AboveIntervalCount](#) and [BelowIntervalCount](#) properties located in the Histogram node of the **ChartDataSeries Collection Editor**. The following code shows how to use the [AboveIntervalCount](#) property to see if there are any outlier values:

#### To write code in Visual Basic

##### Visual Basic

```
Dim overflow As Integer = CInt(cds.Histogram.AboveIntervalCount)
Dim msg As String = ""
' this tests to see if there are any outlier values that fall after the last
interval.
If overflow > 0 Then
    msg = "Number > " + carea.AxisX.Max.ToString() + " = " + overflow.ToString()
End If
c1Chart1.ChartLabels("overflow").Text = msg
```

#### To write code in C#

##### C#

```
int overflow = (int)cds.Histogram.AboveIntervalCount;
string msg = "";
// this tests to see if there are any outlier values that fall after the last
interval.
if(overflow > 0)
{
    msg = "Number > " + carea.AxisX.Max.ToString() + " = " + overflow.ToString();
}
c1Chart1.ChartLabels["overflow"].Text = msg;
```

In histograms the area of a column or interval is proportional to the value it represents. When the interval width is the same for all intervals then each interval height represents the same frequency per unit width. In some cases the widths of the columns vary in size. When this occurs, the height of the columns must be adjusted to keep the areas proportional. For example, if you had a few columns that were non-uniform you could utilize the [Normalized](#) property in C1Chart's Histogram object to make the column or interval proportional to the value it represents.

The histogram object contains three properties: [NormalDisplay](#), [NormalizationInterval](#), and [Normalized](#). The [Normalized](#) property can be used when you want non-uniform intervals to be normalized so each interval height represents the same frequency per unit width. A non-uniform interval is an interval that is not in uniform with the remaining intervals. When you set the [Normalized](#) property to **True** you have the flexibility of setting the [NormalizationInterval](#) property to a specific width size. Let's say that your [IntervalWidth](#) is 10 and you want to normalize it to 20. This will cause the size of each interval to double. As a result it will cause each frequency count (y-value) to double. If you wanted to normalize your [IntervalWidth](#) to 2 from 10 so you can normalize the interval width

1/5 of the size then this would make each frequency count (y-value) 1/5 smaller.

## Frequency Graph

A frequency chart is a histogram without the columns. The midpoint of each column is connected by a straight or curved (spline) line. As a result, the columns are eliminated. In **C1Chart**, the connecting lines from one midpoint of a column to another midpoint of a column can be lines or spline fits. A frequency chart is beneficial when you are comparing multiple data series since it makes the data easier to read.

The frequency graph below illustrates the distribution (in distance) from all the data coordinates from each of the scatter data points to the marker intersection. The numbers associated with the scatter data points are point index values.

You can set the connected lines in the Frequency chart to straight by setting the `FitType` property to `Line`. Alternatively, curved lines can be obtained with a `FitType` of `Spline`. The advantage of a frequency graphic is that it shows the important features of the distribution without the distraction at the corners. The following code shows the `FitType` property set to `Spline` for the Frequency graph:

### To write code in Visual Basic

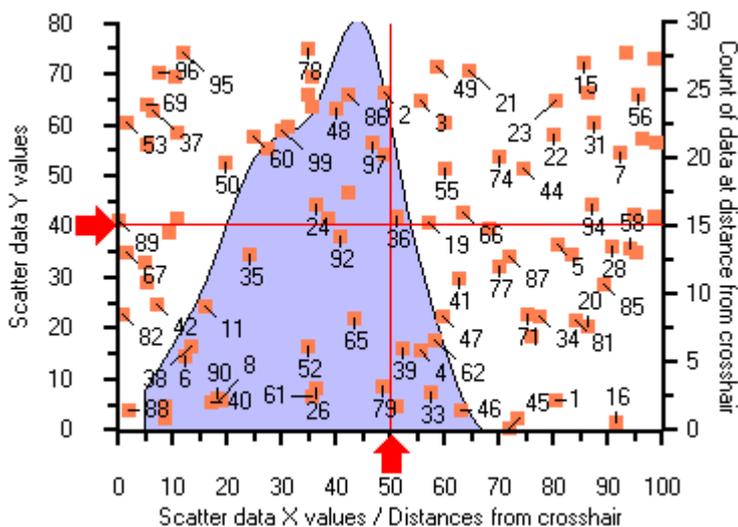
Visual Basic

```
cds.FitType = FitTypeEnum.Spline
```

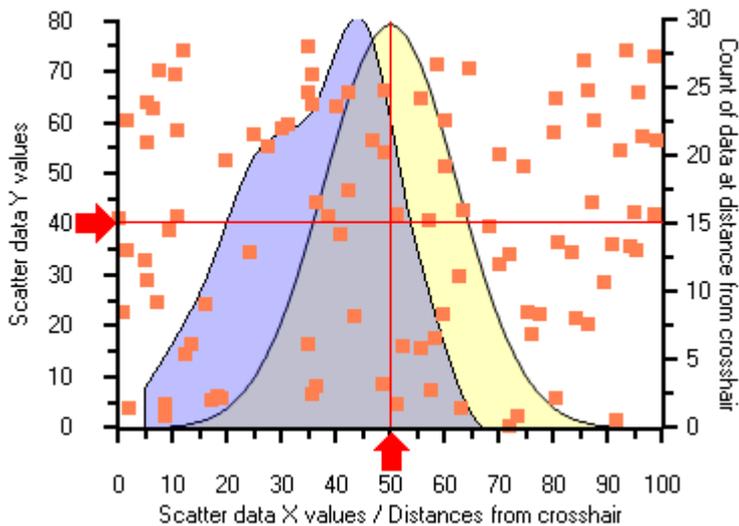
### To write code in C#

C#

```
cds.FitType = FitTypeEnum.Spline;
```



The following **Frequency** graph is the same as the preceding graph except it includes a normal curve to compare the distance distribution with the normal distance distribution. With the normal curve shown we can evaluate whether the raw data is a reasonable approximation to the normal distribution. The raw data's labels are hidden to provide a clearer picture of both curves.



The **Histogram** object in Chart provides a [NormalDisplay](#) property to get the (Gaussian) Normal curve to display. You can access this property at design time in the C1Chart's properties grid by expanding the Histogram node and setting the [NormalDisplay](#) property to **C1.Win.C1Chart.NormalCurve**. Another alternative is to access the [NormalDisplay](#) property by using the following code:

**To write code in Visual Basic**

```
Visual Basic
Dim nc As NormalCurve = cg.Histogram.NormalDisplay
```

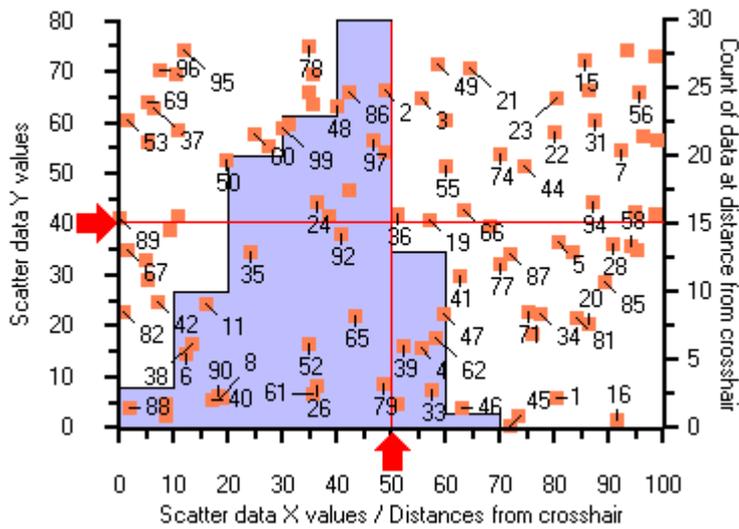
**To write code in C#**

```
C#
NormalCurve nc = cg.Histogram.NormalDisplay;
```

**Note:** The text, nc, is the variable name for the NormalCurve object.

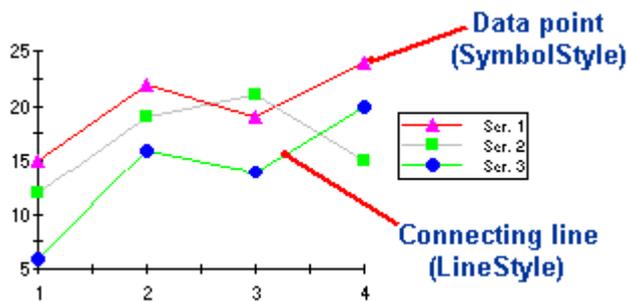
## Stepped Frequency Graph

A **stepped frequency** is the same as a histogram chart in terms of function. The charts only differ in appearance. In the stepped frequency chart the lines are eliminated from the columns whereas in a histogram the lines are kept in the columns.

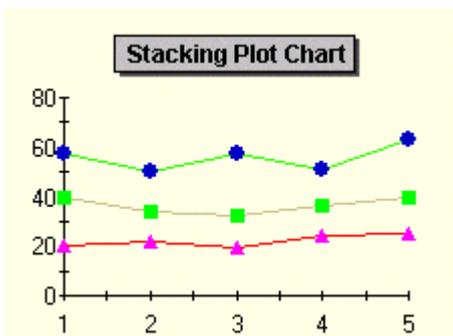


## Line and XY-Plot Charts

**Line and XY-Plot** charts draw each series as connected points of data. By customizing Line and Symbol Styles for each series, the connecting lines can be removed to emphasize the data values themselves, or the points can be removed to emphasize the relationship between points. The series can be drawn independently or stacked. The line and symbol properties for each series can also be customized. For more information, see [Line and Symbol Styles for the Series](#).



Use the ChartGroup object's **Stacked** property to create a stacking Plot chart. Stacking charts represent the data by stacking the values for each series on top of the values from the previous series.



## To set the chart type to Line or X-Y Plot at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the ChartType property to **XY-Plot**. The Line Chart option is not available using this method.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the **Gallery**, select Chart type as either **Line or XY-Plot**.
- Another alternate method is to select **Chart Properties** from the bottom of the Properties pane. From the **Gallery**, select Chart type as either **Line or XY-Plot**.

## Plot Chart Programming Considerations

The following table describes the Data Arrays used for Line and XY-Plot charts. Each data series requires X and Y array values to be charted. Adding values to the other arrays will not affect this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays.

Property	Description
X	Holds the position on the X-axis.
Y	Holds the position on the Y-axis.
Y1	No effect for Line and XY-Plot charts.
Y2	No effect for Line and XY-Plot charts.
Y3	No effect for Line and XY-Plot charts.

## XY-Plot Chart 3D Effects

C1Chart's 3D effects can be used with XY-Plot charts or stacking XY-Plot charts to create the illusion of depth with each data series. Sometimes these charts are called **ribbon charts**. By using the depth, elevation, rotation and shading properties, you can enhance your area charts, making them stand out by creating visual depth.

To access the 3D view for a XY-Plot chart, adjust the properties in the View3D object. The View3D object is a member of the PlotArea, which in turn is a member of the ChartArea. By adjusting the View3D object properties, [Depth](#), [Elevation](#), [Rotation](#) and [Shading](#) you can customize the 3D view.

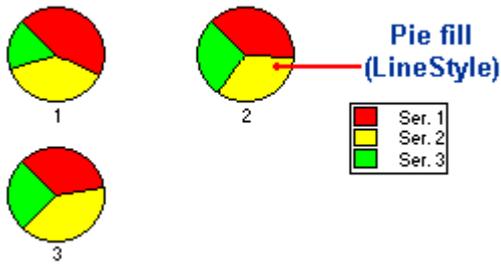
Note that the [Depth](#) property is the key to all 3D type chart logic. While the [Elevation](#) and [Rotation](#) properties modify the way a user views the chart, it is the [Depth](#) property that actually dictates whether a chart is 3D. By using a non-zero value for the [Depth](#) property and setting the [Elevation](#) and [Rotation](#) property values to zero, you have created a 3D chart even though nothing seems to have changed. In effect you are looking at the "front" surface of the chart, which is visually represented in the same way as a standard area chart.

Note also, that there may be times when it is desirable to chart some data with a 3D view, and other data in the 2D plane. For these instances, adjust the [Use3D](#) property associated with each ChartGroup.

 Axes origin settings are not supported when Chart2D is used to represent 3D effects with XY-Plot Charts (Ribbon Charts).

## Pie and Doughnut Charts

A **Pie** chart draws each series as a slice in a pie. The number of pies is the number of points in the data. Each pie displays the nth data point in each series. Using the [LineStyle](#), the fill properties of each series can be customized. For more information, see [Line and Symbol Styles for the Series](#).



## To set the chart type to Pie at design time

- Expand the **ChartGroups** node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the **ChartType** property to **Pie**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the Gallery, select **ChartType** as **Pie**.
- Another alternate method is to select **Chart Properties**, from the Properties pane. From the **Gallery**, select **ChartType** as **Pie**.

## Pie Chart Programming Considerations

The following table describes the Data Arrays used for pie charts. Each data series requires X and Y array values to be charted. Adding values to the other arrays will not affect this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays.

Property	Description
X	Holds 0-based slice index for Pie chart.
Y	Holds pie slice value indicated by the X value.
Y1	No effect for Pie charts.
Y2	No effect for Pie charts.
Y3	No effect for Pie charts.

 **Note:** Entering data in more than one series will display multiple Pie charts within one ChartGroup.

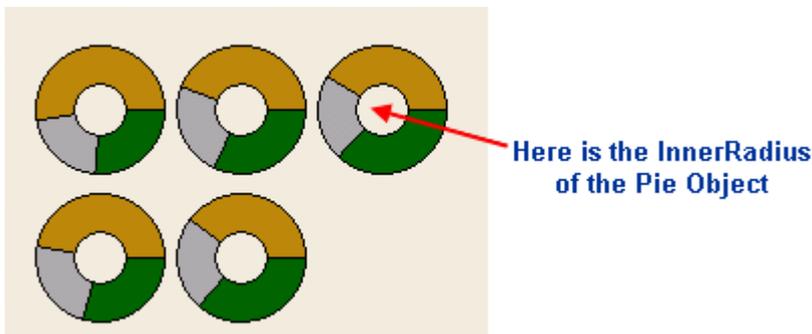
## Doughnut Charts

A chart is a pie chart with a non-zero radius and is identical in function to a pie chart, but can be used to increase aesthetic appeal, particularly when shown with 3D effects. As with all pie charts, each doughnut shows each series as a fraction of the whole at each data point. If multiple data points are specified, then multiple doughnuts appear in the chart.

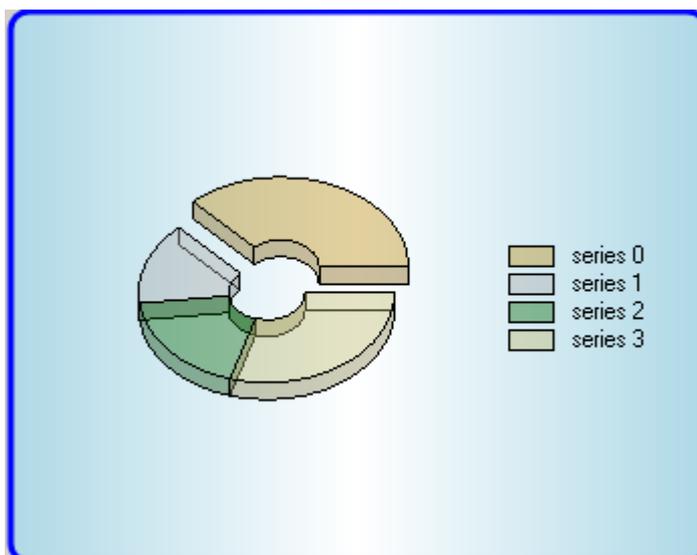
A doughnut chart can be created by setting the **InnerRadius** property of a pie chart to a non-zero value. The **InnerRadius** value represents the percentage of the full pie radius. The **InnerRadius** property can be accessed in the pie object of each Chart group. The chart property sheet below shows the **InnerRadius** property. In this example the **InnerRadius** property is set to 40 percent.

Properties	
c1Chart1 C1.Win.C1Chart.C1Chart	
Group0 C1.Win.C1Chart.ChartGroup	
ChartData C1.Win.C1Chart.ChartData	
ChartType	<b>Pie</b>
LegendReversed	False
Name	<b>Group1</b>
Pie <b>InnerRadius=40,OtherOffset=0,Start=0</b>	
InnerRadius	<b>40</b>
OtherOffset	<b>0</b>
Start	<b>0</b>

The picture below illustrates the effect on the pie charts by setting the `InnerRadius` property to 40 percent:



The following illustration is another example of a doughnut chart. In this example, the `Offset` property of the `ChartDataSeries` object is set to 30.



## Special Pie Chart Properties

Pie charts are quite different from the other chart types since they do not follow the concept of a two-dimensional grid or axes. Altering the diameter of the pie or the properties of the exploding slices can be accomplished with the properties of the `Pie` class.

## Clockwise or Counter-Clockwise Direction

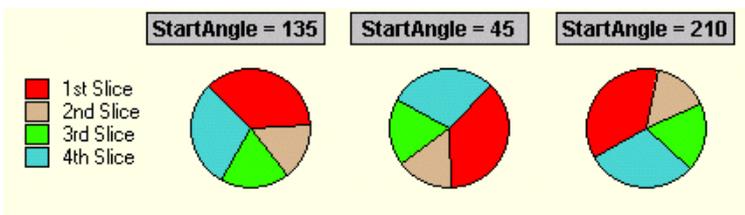
Setting the property `Clockwise` to `True` draws each series in the pie in clockwise direction and setting it to `False` draws each series in the pie in counter-clockwise direction.

The following images display two pie charts with one drawn in counter-clockwise and the other in clockwise direction.



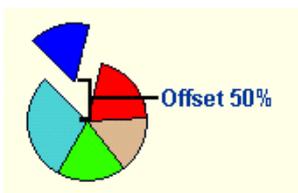
## Starting Angle

Use the `Start` property to specify the angle at which the slices for the first series start. The default angle is 0 degrees. The angle represents the arc between the most clockwise edge of the first slice and the right horizontal radius of the pie, as measured in the counter-clockwise direction (see image below). `Start` is a property of the `Pie` class and can be accessed at design time through the `Pie` node of the **ChartGroupsCollection Editor**.



## Exploding Pies

A slice of a Pie chart can be emphasized by exploding it, which extrudes the slice from the rest of the pie. Use the `Offset` property of the series to set the exploded slice's offset from the center of the pie. The offset is measured as a percentage of the radius of the pie.



Exploding slices can be set programmatically, and can be set for the series only:

### To write code in Visual Basic

Visual Basic

```
'Get the appropriate ChartData object.
Dim cd As ChartData = C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData

'Sets the offset for the first series to 10% of the pie's radius
cd.SeriesList(0).Offset = 10

'Resets the exploded slices.
cd.SeriesList(0).Offset = 0
```

### To write code in C#

C#

```
//Get the appropriate ChartData object.
ChartData cd = c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData;

//Sets the offset for the first series to 10% of the pie's radius
cd.SeriesList[0].Offset = 10;

//Resets the exploded slices.
cd.SeriesList[0].Offset = 0;
```

## Pie Chart 3D Effects

C1Chart's 3D effects can be used with pie charts to create the illusion of depth with each data series. By using the depth, elevation, rotation and shading properties, you can enhance your area charts, making them stand out by creating visual depth.

To access the 3D view for a pie chart, adjust the properties in the View3D object. The View3D object is a member of the PlotArea, which in turn is a member of the ChartArea. By adjusting the View3D object properties, [Depth](#), [Elevation](#), [Rotation](#) and [Shading](#) you can customize the 3D view.

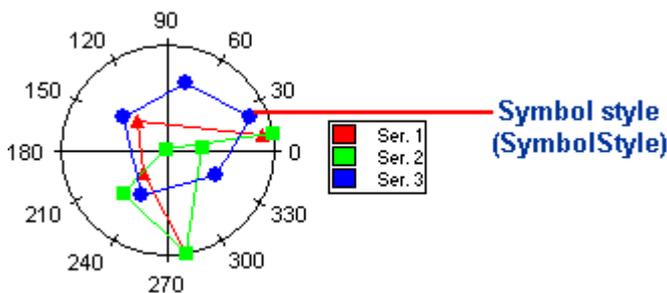
Note that the [Depth](#) property is the key to all 3D type chart logic. While the [Elevation](#) and [Rotation](#) properties modify the way a user views the chart, it is the [Depth](#) property that actually dictates whether a chart is 3D. By using a non-zero value for the [Depth](#) property and setting the [Elevation](#) and [Rotation](#) property values to zero, you have created a 3D chart even though nothing seems to have changed. In effect you are looking at the "front" surface of the chart, which is visually represented in the same way as a standard area chart.

Note also, that there may be times when it is desirable to chart some data with a 3D view, and other data in the 2D plane. For these instances, adjust the [Use3D](#) property associated with each [ChartGroup](#).

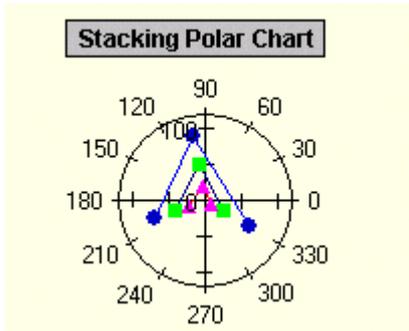
## Polar Charts

A Polar chart draws the x and y coordinates in each series as (theta,r), where theta is amount of rotation from the origin and r is the distance from the origin. Theta may be specified in either degrees (default) or radians. Since the X-axis is a circle, the X-axis maximum and minimum values are fixed. The series can be drawn independently, or stacked.

Using the [LineStyle](#) and [SymbolStyle](#), the line and symbol properties of each series can be customized. For more information, see [Line and Symbol Styles for the Series](#).



Use the ChartGroup object's [Stacked](#) property to create a stacking Polar chart. Stacking charts represent the data by stacking the values for each series on top of the values from the previous series.



## To set the chart type to Polar/Radar at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the **ChartType** property to **Polar**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the Gallery, select **ChartType** as **Polar**.
- Another alternate method is to select **Chart Properties** from the Properties pane. From the Gallery, select **ChartType** as **Polar**.

## Polar Chart Programming Considerations

The following table lists the Data Arrays that are used for Polar charts. Each data series requires X and Y array values to be charted. Adding values to the other arrays will not affect this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays.

Property	Description
X	Holds the X-axis position in either radians or degrees.
Y	Holds the position on the Y-axis.
Y1	No effect for Polar charts.
Y2	No effect for Polar charts.
Y3	No effect for Polar charts.

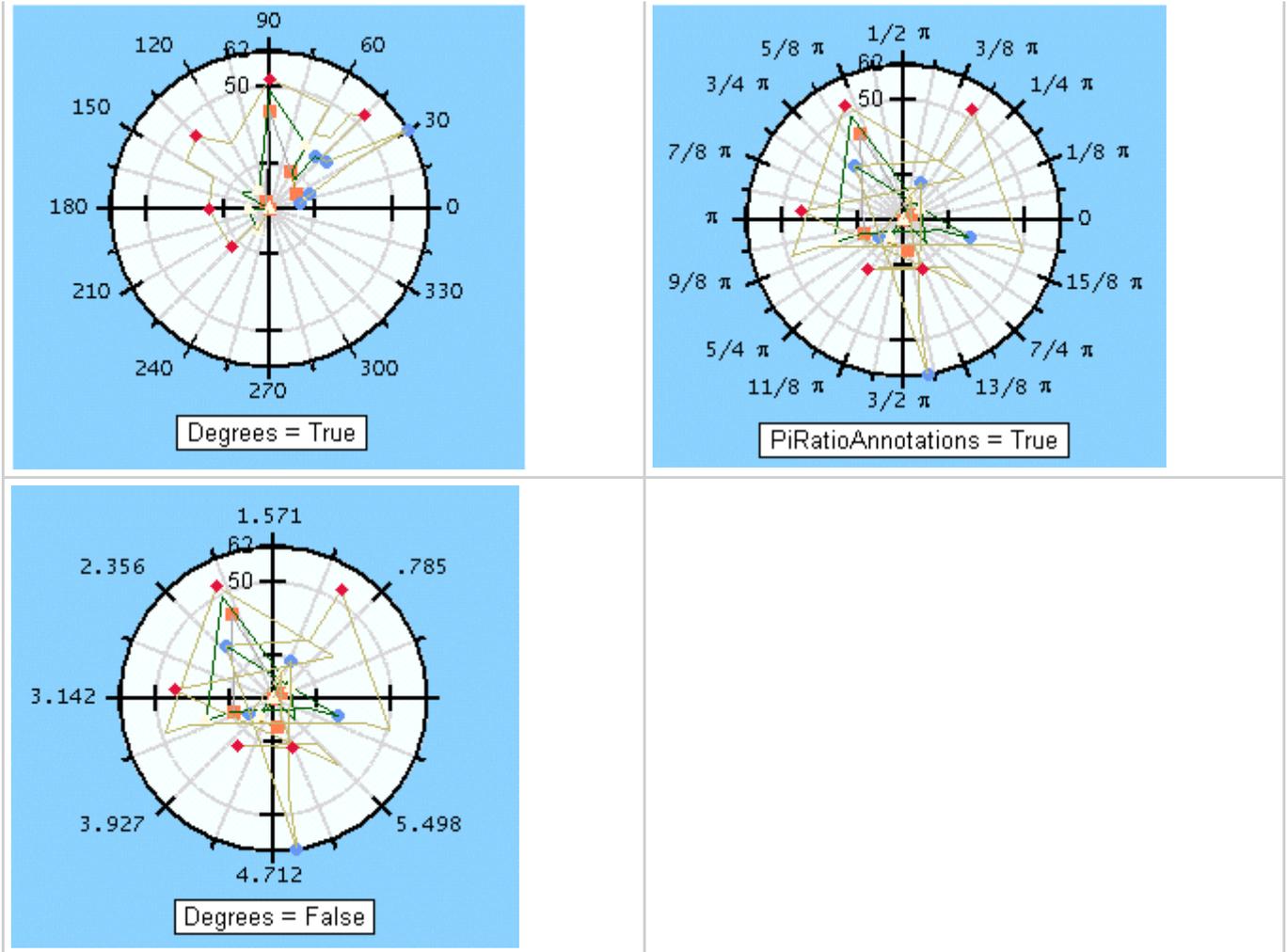
## Special Polar Chart Properties

The properties below can be accessed at design time through the Polar node in the **ChartGroupsCollection Editor**.

### Charting Degrees or Radians

The **Degrees** property of the **Polar** class sets whether the X data values for polar charts reflect angles in degrees (True) or radians (False). The **Degrees** property can be accessed at design time through the Polar node of the **ChartGroups Collection Editor**.

If **Degrees** is set to **False** and the chart reflects Radian values, then C1Chart provides the option of having the chart annotated with ratios of Pi rather than radians. Setting the **PiRatioAnnotations** property of the **Polar** class to **True** will annotate the X values in ratios of Pi.



## Setting the Starting Angle

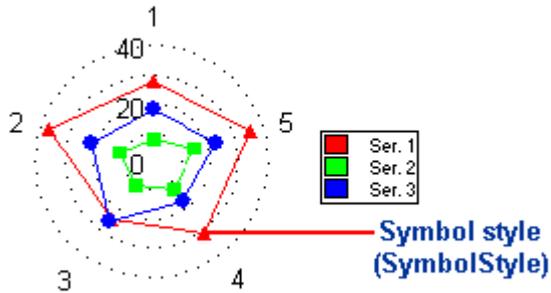
The `Start` property of the `Polar` class sets the starting angle for Polar charts. The default setting for this property is 0. Setting this property to a value other than 0 will move the origin of the chart counter-clockwise by the specified amount. For instance, setting the `Start` property to 90 rotates the Polar chart 90 degrees in the counter-clockwise direction.

## Radar Charts

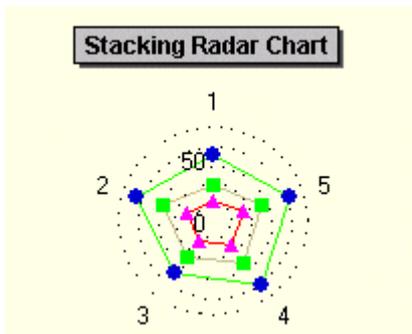
A Radar chart draws the y value in each data set along a radar line (the x value is ignored except for labels). If the data has n unique points, then the chart plane is divided into n equal angle segments, and a radar line is drawn (representing each point) at n/360 degree increments. By default, the radar line representing the first point is drawn vertically (at 90 degrees). The series can be drawn independently or stacked.

Radar charts can display tickmarks in the radial direction along X axis major gridlines. The tickmarks are controlled by the Y axis tickmark properties. For more information about the tickmark properties see, [Axis Tick Marks](#).

Using the `LineStyle` and `SymbolStyle`, the line and symbol properties of each series can be customized. For more information, see [Line and Symbol Styles for the Series](#).



Use the ChartGroup object's Stacked property to create a stacking Radar chart. Stacking charts represent the data by stacking the values for each series on top of the values from the previous series.



## To set the chart type to Radar at design time

- Expand the ChartGroups node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the **ChartType** property to **Radar**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the Gallery, select **ChartType** as **Radar**.
- Another alternate method is to select **Chart Properties** from the Properties pane. From the Gallery, select **ChartType** as **Radar**.

## Radar Chart Programming Considerations

The following table lists the Data Arrays used for Radar charts. Each data series requires only a Y array value to be charted. Adding values to the other arrays will not affect this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays:

Property	Description
X	Ignored except for labels.
Y	Holds the Y-axis data values for the Radar chart. The number of points in the Y Data Array determines the amount of radar lines in the chart.
Y1	No effect for Radar charts.
Y2	No effect for Radar charts.
Y3	No effect for Radar charts.

## Special Radar Chart Properties

The Radar chart has special properties to chart the degrees of the Radar, set the starting angle, create filled radar chart, and whether or not to use flat Y coordinate gridlines for radar charts.

### Charting Degrees or Radians

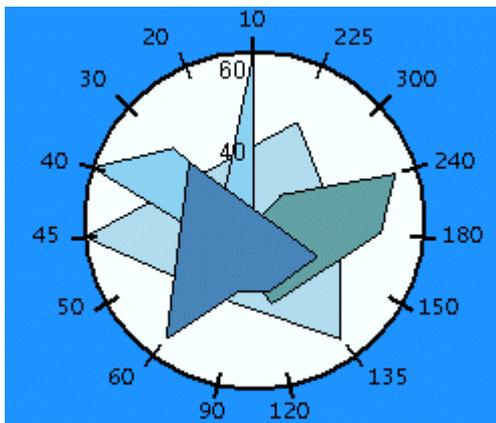
The `Start` property of the `Radar` class sets whether the `Start` property value for radar charts reflects angles in degrees (True) or radians (False). The `Degrees` property can be accessed at design time through the Radar node of the **ChartGroups Collection Editor**.

### Setting the Starting Angle

The `Start` property of the `Radar` class sets the starting angle for radar charts. The default setting for this property is 0. Setting this property to a value other than 0 will move the origin of the chart counter-clockwise by the specified degrees. For instance, setting the `Start` property to 90, the Radar chart rotates 90 degrees in the counter-clockwise direction.

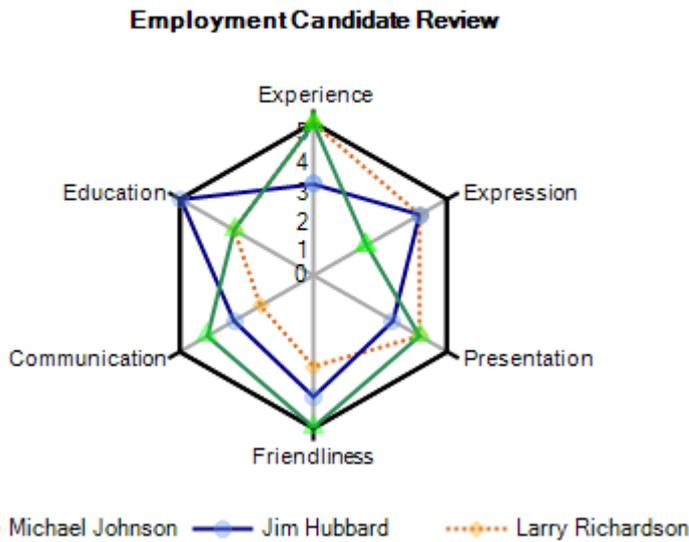
### Filled Radar Charts

A Filled Radar chart draws the y value in each series along a radar line (the x value is ignored except for labels). If the data has n unique points, the chart plane is divided into n equal angle segments, and a radar line is drawn (representing each point) at n/360 degree increments. Each series is drawn "on top" of the preceding series. The series can be drawn independently or stacked. Filled Radar charts are the same as Radar charts, except that the area between the origin and the points is filled and symbols are not present. To create a Filled Radar chart, set the `Filled` property of the `Radar` class to **True**.



### Flat Gridlines

You can show flat Y coordinate gridlines in radar charts by setting the `FlatGridLines` property to true.



To programmatically set the `FlatGridLines` property to true:

### To write code in Visual Basic

Visual Basic

```
C1Chart1.ChartGroups(0).Radar.FlatGridLines = True
```

### To write code in C#

C#

```
c1Chart1.ChartGroups[0].Radar.FlatGridLines = true;
```

## Step Charts

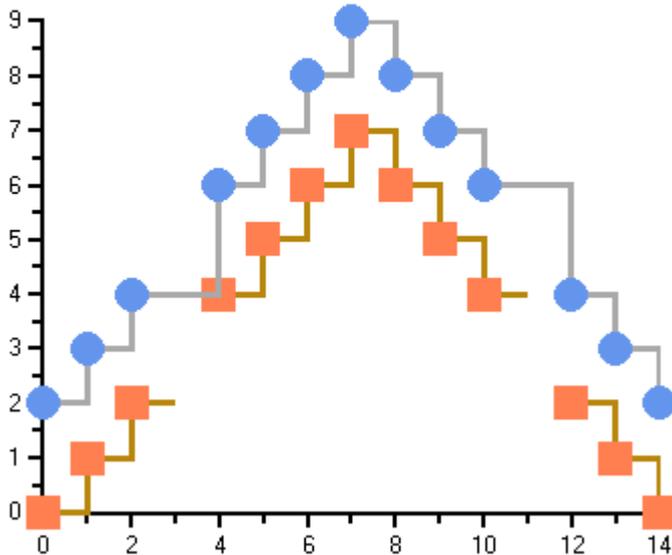
A **Step** chart is a form of a **XY Plot** chart. **Step** charts are often used when Y values change by discrete amounts, at specific values of X with a sudden change of value. A simple, everyday example would be a plot of a checkbook balance with time. As each deposit is made, and each check is written, the balance (Y value) of the check register changes suddenly, rather than gradually, as time passes (X value). During the time that no deposits are made, or checks written, the balance (Y value) remains constant as time passes.

Similar to Line and XY plots, the appearance of the step chart can be customized by using the Line and Symbol Styles for each series by changing colors, symbol size, and line thickness. Symbols can be removed entirely to emphasize the relationship between points or included to indicate actual data values. If data holes are present, the step chart behaves as expected, with series lines demonstrating known information up to the X value of the data hole. Symbols and lines resume once non-hole data is again encountered.

As with most XY style plots, step charts can be stacked when appropriate.

The following chart illustrates the 2D Step Chart:

## 2D Step Chart



## To set the chart type to Step at design time

- Expand the **ChartGroups** node in the Properties window. Open the **ChartGroups Collection Editor** by clicking the **ellipsis** button. In the right pane of the editor, set the **ChartType** property to **Step**.
- An alternate method to change chart type is to right-click the existing chart and select **Chart Properties**. From the **Gallery**, select Chart type as **Step**.
- Another alternate method is to select **Chart Properties** from the Properties pane. From the Gallery, select Chart type as **Step**.

## Step Chart Programming Considerations

The following table describes the Data Arrays used for Step charts. Each data series requires the use of the X array and Y array values to be charted. Adding values to the other arrays will not affect this chart, but filling the arrays with data could make it easier to switch to another chart type that does use those arrays.

Property	Description
X	Holds the position on the X-axis.
Y	Holds the position on the Y-axis.
Y1	No effect for Step charts.
Y2	No effect for Step charts.
Y3	No effect for Step charts.

## Step Chart 3D Effects

C1Chart's 3D effects can be used with Step charts to create the illusion of depth with each data series. By using the depth, elevation, rotation and shading properties, you can enhance your area charts, making them stand out by

creating visual depth.

To access the 3D view for an area chart, adjust the properties in the [View3D](#) object. The View3D object is a member of the PlotArea, which in turn is a member of the ChartArea. By adjusting the [View3D](#) object properties, [Depth](#), [Elevation](#), [Rotation](#) and [Shading](#) you can customize the 3D view.

Note that the [Depth](#) property is the key to all 3D type chart logic. While the [Elevation](#) and [Rotation](#) properties modify the way a user views the chart, it is the [Depth](#) property that actually dictates whether a chart is 3D. By using a non-zero value for the [Depth](#) property and setting the [Elevation](#) and [Rotation](#) property values to zero, you have created a 3D chart even though nothing seems to have changed. In effect you are looking at the "front" surface of the chart, which is visually represented in the same way as a standard area chart.

Note also, that there may be times when it is desirable to chart some data with a 3D view, and other data in the 2D plane. For these instances, adjust the **Use3D** property associated with each ChartGroup.

## Design-Time Tools for Creating 2D Charts

This chapter includes three options for creating 2D charts at design time. You can create charts using the Smart Designer, **Chart Wizard**, or the **Chart Properties** designer. After reading this chapter you can decide which method for creating a chart is easiest for you.

## Working with the Smart Designer

The Smart Designer allows you to quickly set Chart properties without leaving the design form. This solves the earlier problem of having to drill down through Chart's properties in the Properties window. You can use the Smart Designer feature to create a functional 2D Chart at design time.

The Smart Designer provides the following features:

- Built-in toolbars and editors.
- Ability to add/edit labels through the **Edit labels** editor.
- Drag-and-drop labels to any series on the plot area of the chart.
- Labels for each chart element are provided for better user interaction with the chart elements.

This chapter describes the functionality of the Smart Designer and provides labeled figures for each of the elements in the Smart Designer.

For information on how to achieve specific chart tasks using the Smart Designer, see [Creating and Formatting Chart Elements Using the Smart Designer](#).

## Primary Floating Toolbars

The Smart Designer has three primary floating toolbars for the **C1Chart** control. The primary floating toolbars include the following:

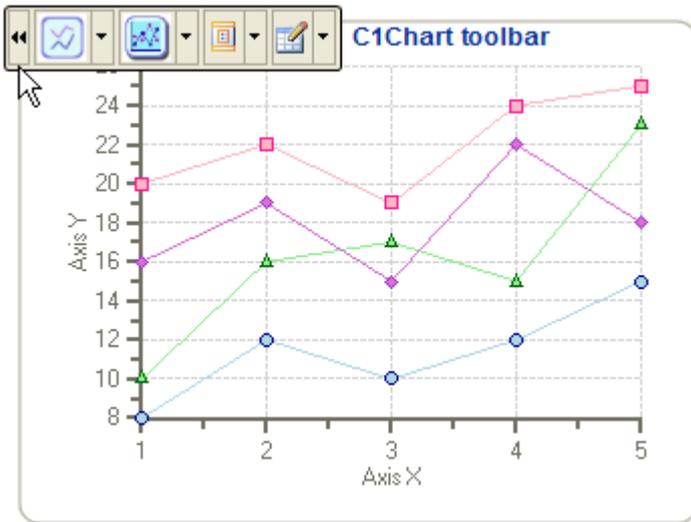
- C1Chart toolbar
- ChartArea toolbar
- PlotArea toolbar

The Smart Designer's toolbars are referred as floating toolbars since their behavior is slightly different from typical toolbars. The Smart Designer's toolbars appear only when the control is active and they can't be docked to other controls.

This section describes the functionality of the buttons in C1Chart's primary toolbars.

## C1Chart Toolbar

The primary **C1Chart** floating toolbar for the **C1Chart** control includes the following command buttons: a **Close**, **Chart type**, **Chart sub-type**, **Layout**, and a **Data** button.

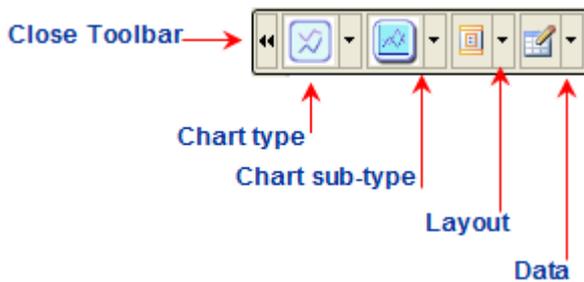


## Exposing the C1Chart Floating Toolbar

To expose the **C1Chart** floating toolbar slide your cursor anywhere on the **C1Chart** control.

## C1Chart Toolbar's Command Buttons

The following figure provides a label for each of the command buttons in the **C1Chart** toolbar.



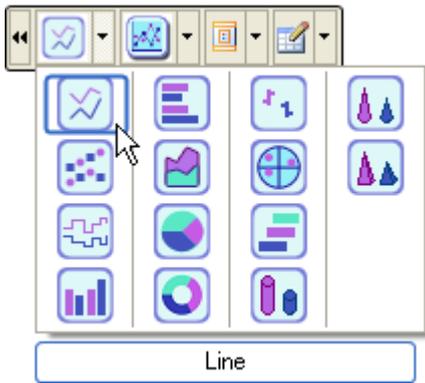
The following section lists all of the command buttons available in the **C1Chart** toolbar and describes the functionality of each one.

## Close Toolbar button

The close command button  closes the toolbar once it is clicked.

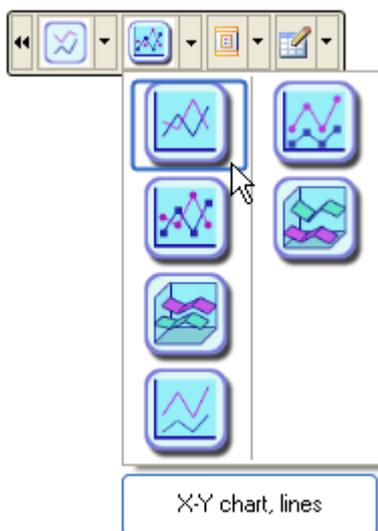
## Chart type button

The **Chart type** command has a drop-down menu that contains a selection of all the chart types provided by the **C1Chart** control. Mousing over each chart image exposes a label with the name of the selected chart type. You can choose from one of the following simple chart types: Line, X-Y, Step, Column, Bar, Area, Pie, Doughnut, Financial/Stock, Polar/Radar, and Gantt, Cylinder, Cone, and Pyramid.



## Chart sub-type button

The **Chart sub-type** command also has a drop-down menu which contains a selection of all the chart sub-types provided by the **C1Chart** control. Mousing over each chart image exposes a label with the name of the selected chart sub-type. For example, when the **Line** chart type is selected from the chart-type, the following complex chart sub-types are available: X-Y chart lines, X-Y chart/lines/symbols, X-Y chart/lines/3D, X-Y chart/lines/stacked, X-Y chart/lines/symbols/stacked, X-Y chart/lines/symbols/stacked, and X-Y chart/lines/stacked/3D. There are unique sub-type charts for each chart type.



## Layout button

The **Layout** command button has a drop-down menu which contains the Header, Footer, and Legend elements. Selecting one of the elements exposes either an editable Header, Footer, or Legend element directly on the Chart Area.



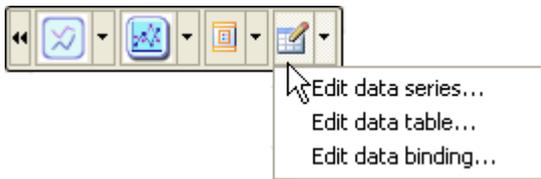
Selecting either the Header or Footer element exposes an editable textbox along with a toolbar that provides formatting commands. If the toolbar does not appear instantly then you can click the left mouse button and slide it

over the textbox to expose the toolbar. The image below illustrates the **Header** text box automatically added to the Chart Area of the **C1Chart** control.



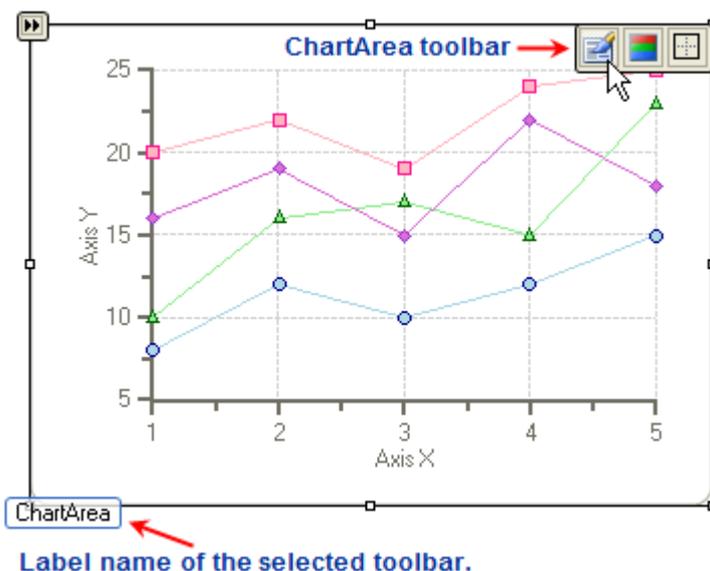
## Data button

The **Data** button has a drop-down menu which includes three options for the user to select from: Edit data series, Edit data table, and Edit data binding. The **Edit data series** item exposes the **Chart Properties** designer for the data series. The Edit data table item exposes the **Chart Properties** designer for the data table. The **Edit data binding** item exposes the **Chart Properties** designer for the data binding. If you would like to edit another element other than the data series, data table, and data binding you can click on the  button to navigate to a different chart element in the tree view of the **Chart Properties** designer.



## ChartArea Toolbar

Another primary floating toolbar, the **ChartArea** floating toolbar for the **C1Chart** control includes a **Properties**, **Background**, and a **Border** command button. The figure below shows how the **ChartArea** toolbar appears when the user selects the ChartArea on the **C1Chart** control. When a user selects a toolbar, a label name is provided for the user's convenience.



## Exposing the ChartArea toolbar

To expose the ChartArea toolbar slide your cursor in the ChartArea.

## ChartArea Toolbar's Command Buttons

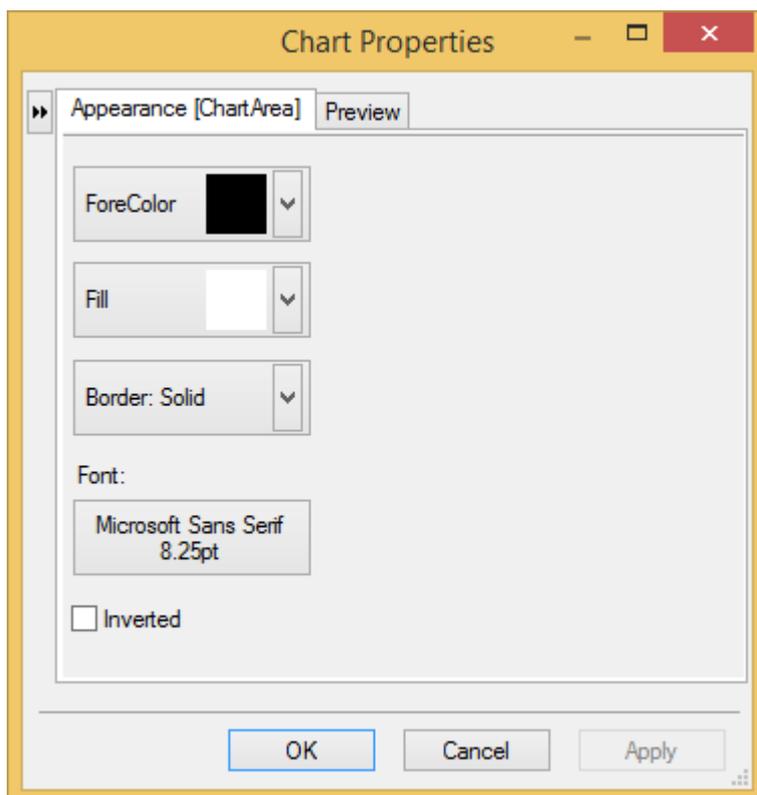
The figure below illustrates each element in the **ChartArea** toolbar.



The section below lists all of the command buttons available in the **ChartArea** toolbar and describes the functionality of each one.

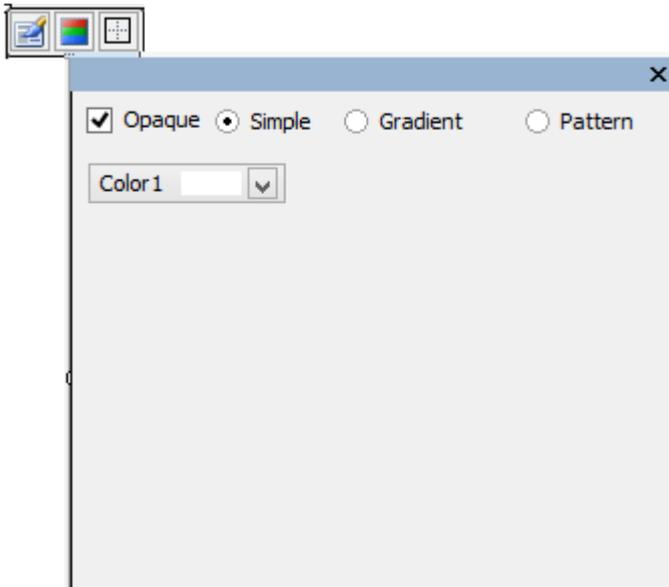
## Properties

The **Properties** command button exposes the **Chart Properties** editor for the **ChartArea** element when the user clicks on the properties command button.



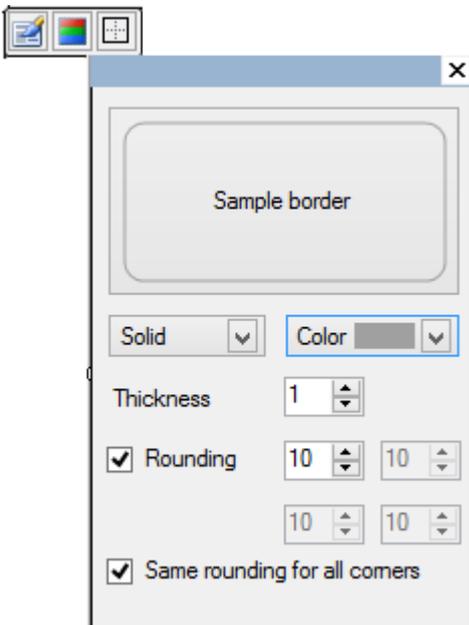
## Background button

The **Background** command button has a drop-down box that contains three different types of styles for the background and a color drop-down list box for the user to specify a color for the ChartArea's background.



## Border button

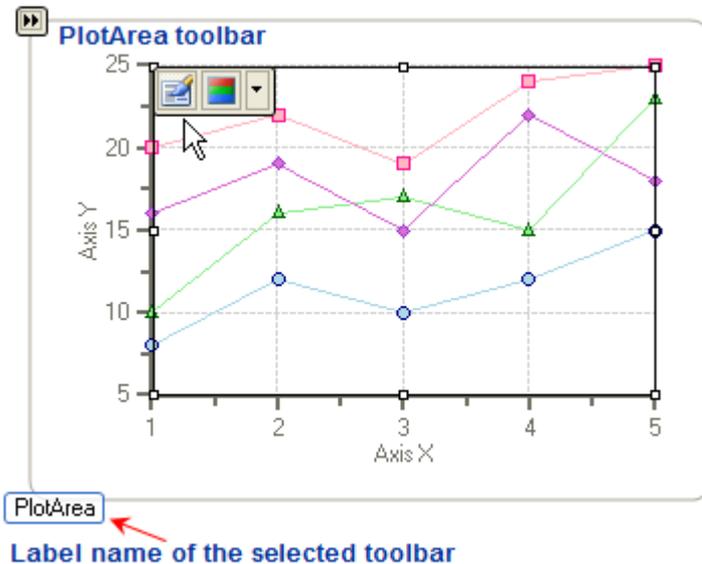
The **Border** command button includes a drop-down box that contains editable Border styles and colors for the ChartArea's border.



## PlotArea Toolbar

Another primary floating toolbar, the **PlotArea** toolbar for the [C1Chart](#) control includes a **Properties**, **Background**, and a drop-down menu which contains an **Add/Edit** labels command item to add or edit chart labels and an **Inverted chart** command item to make the chart inverted.

The figure below shows how the **PlotArea** toolbar appears when the user selects the **PlotArea** on the [C1Chart](#) control. When a user selects a toolbar, a label name is provided for the user's convenience.

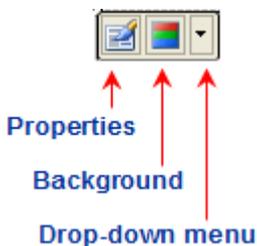


## Exposing the PlotArea Floating toolbar

To expose the **PlotArea** floating toolbar slide your cursor in the PlotArea of the chart.

## PlotArea Floating Toolbar's Command Buttons

The following figure illustrates each button in the **PlotArea** toolbar:



The section below lists all of the command buttons available in the **PlotArea** toolbar and describes the functionality of each one.

## Properties Button

The **Properties** button for the **PlotArea** toolbar functions exactly like the **Properties** button for the **ChartArea** toolbar.

## Background Button

The **Background** button for the **PlotArea** toolbar functions exactly like the **Background** button for the **ChartArea** toolbar.

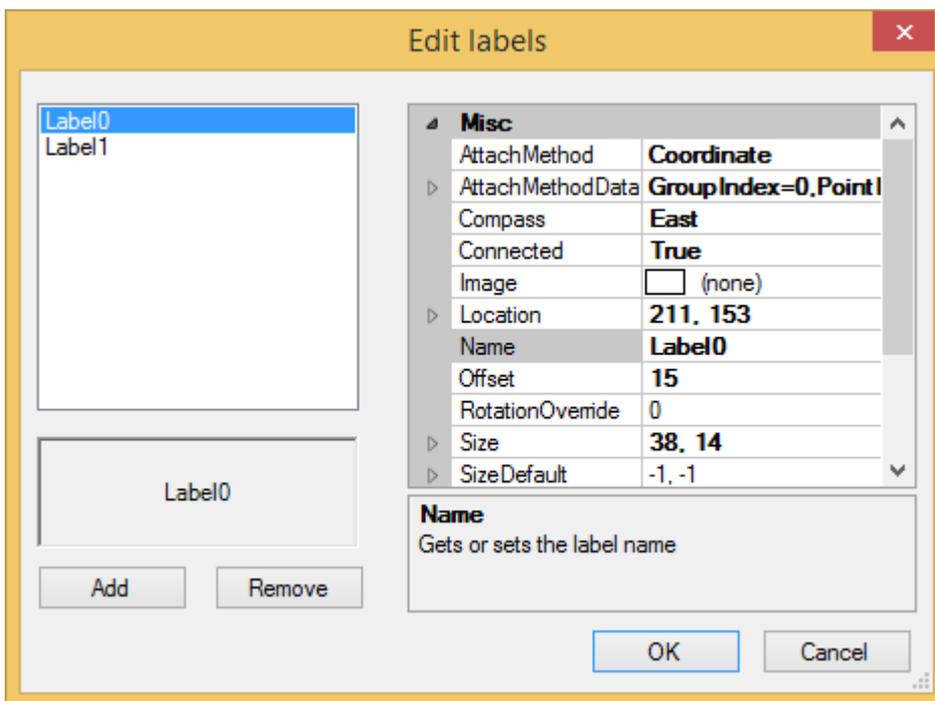
## Drop-down menu

The drop-down menu for the **PlotArea** toolbar contains two menu items: **Inverted** and **Add/Edit** labels. The **Inverted**

command item inverts the Y and X-axis. For more information about Inverted charts, please see [Inverted and Reversed Chart Axes](#).



The **Add/Edit** labels command item exposes the **Edit labels** editor. In the **Edit labels** editor you can add or edit existing labels.



For more information on using the Edit labels editor to add labels see, [Add Labels to the Chart](#).

## Secondary Floating Toolbars

The Smart Designer has four secondary floating toolbars for C1Chart's additional elements such as the header, footer, label, and legend. The secondary floating toolbars include the following:

- Axes toolbars
- Header toolbar
- Footer toolbar
- Label toolbar
- Legend toolbar

This section describes the functionality of the buttons in C1Chart's secondary floating toolbars.

## Axes Toolbars

The floating toolbar for the Axis X and Axis Y elements include the following command buttons: **Properties**, **Font**, and a **Axis X** and **Axis Y** position for the Axis X and Axis Y elements of the C1Chart control. The figure below illustrates the

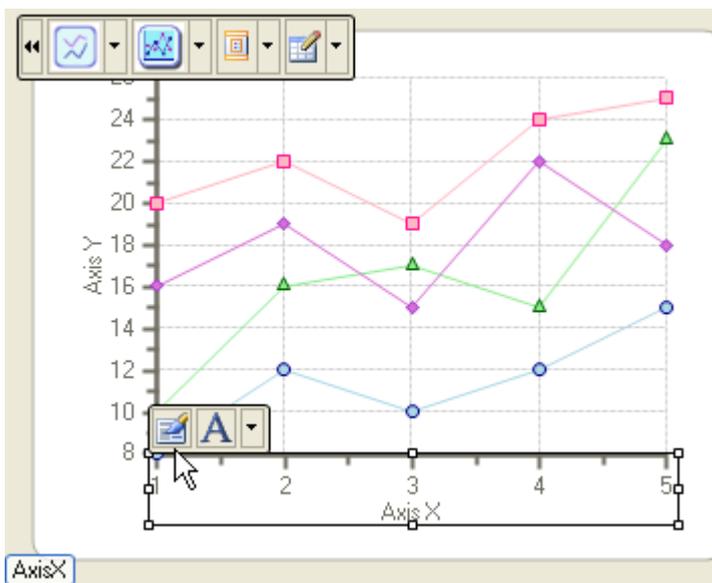
command buttons on the Axis X or Axis Y toolbars.



## Exposing the AxisX or AxisY Floating Toolbar

To expose the **AxisX** or **AxisY** floating toolbar slide your cursor in the AxisX or AxisY area.

The following example illustrates the **AxisX** toolbar exposed from sliding the cursor over the Axis X area.



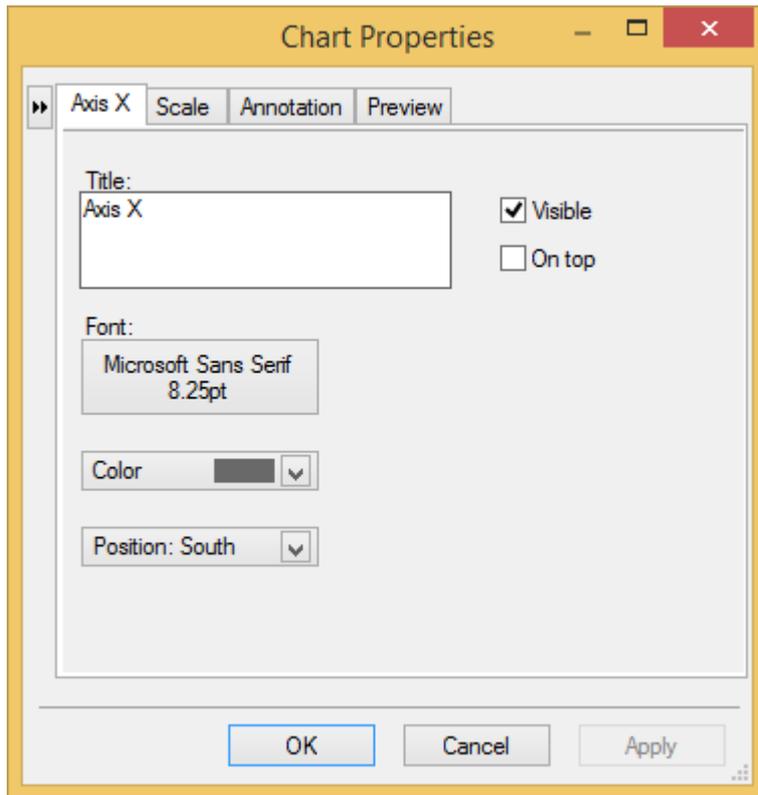
## AxisX/AxisY Toolbar's Command Buttons

The section below lists all of the command buttons available in the **AxisX/AxisY** toolbar and describes the functionality of each one.

### Properties button

The **Properties** command button exposes the **Chart Properties** dialog box for the **Axis X** and/or **Axis Y** elements when the user clicks on the **Properties** command button in the AxisX or AxisY toolbars.

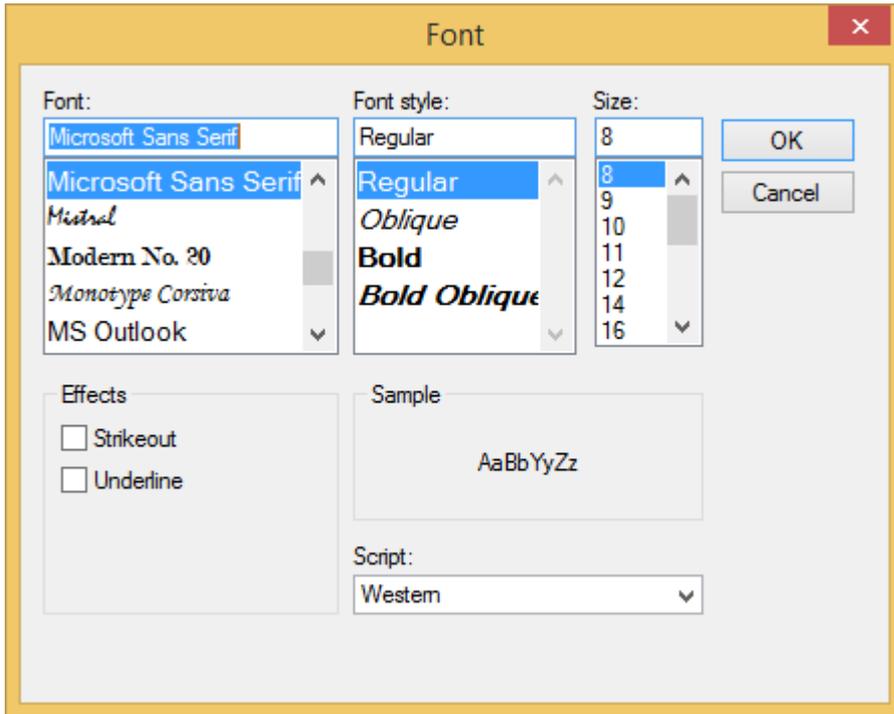
The following image represents the properties for the AxisX when the **Properties** command button is selected from the Axis X toolbar:



Clicking on the  button opens the navigational tree view of the chart elements in the left pane of the **Chart Properties** designer.

## Font button

The **Font** command button exposes the **Font** dialog box for the **AxisX** and **AxisY** elements. Here the Font style can be modified for the Axes' text.

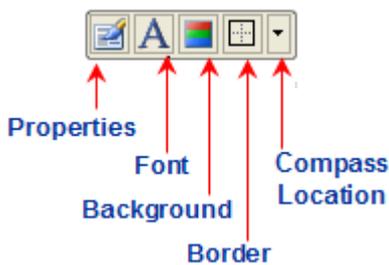


## Position button

The **Position** command button has a drop-down list box which includes a list of different compass directions (West, East, North, or South) for the user to choose from. The **AxisY** compass directions will position the AxisY to the West position which is to the left of the Chart or to the East position which is to the right of the Chart. The AxisX compass directions will position the AxisX to the South which is below the chart or to the North which is above the chart. The default compass position for the AxisY is West and the default position for the AxisX is South.

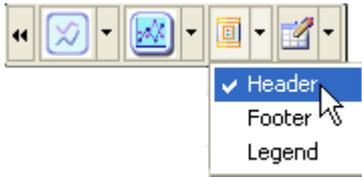
## Header and Footer Toolbars

The floating toolbar for the **Header** and **Footer** element includes the following command buttons: **Properties**, **Font**, **Background**, **Border**, and a **Location** for the **Header** and **Footer** elements of the **C1Chart** control. The figure below provides a label for each of the command buttons for the Header and Footer toolbar.



## Exposing the Header or Footer Floating Toolbar

In order for the **Header** or **Footer** floating toolbar to appear you have to select either the **Header** or **Footer** from **C1Chart** toolbar's drop-down menu.

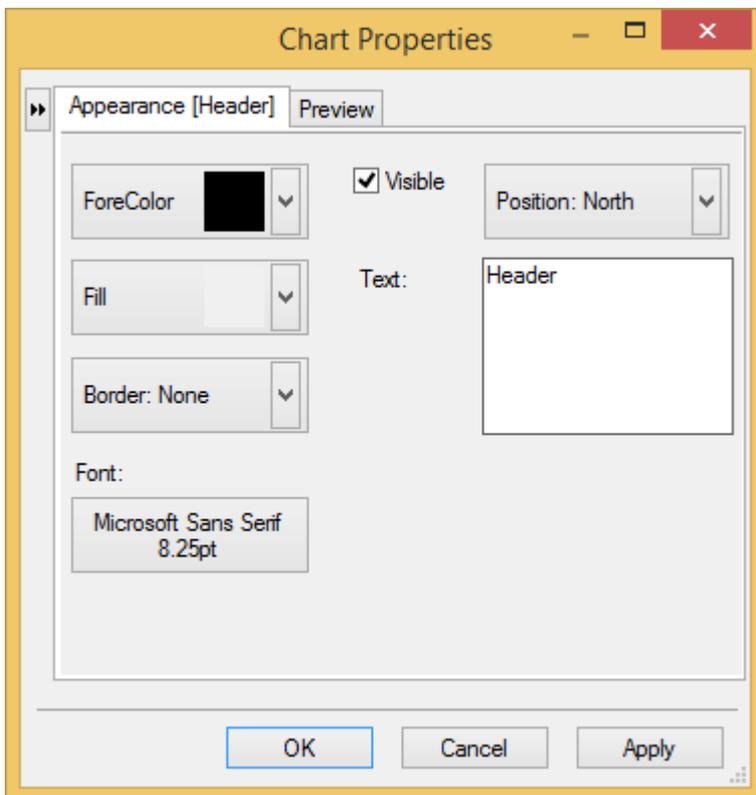


## Header/Footer Floating Toolbar's Command Buttons

The section below lists all of the command buttons available in the **Header/Footer** toolbar and describes the functionality of each one.

### Properties button

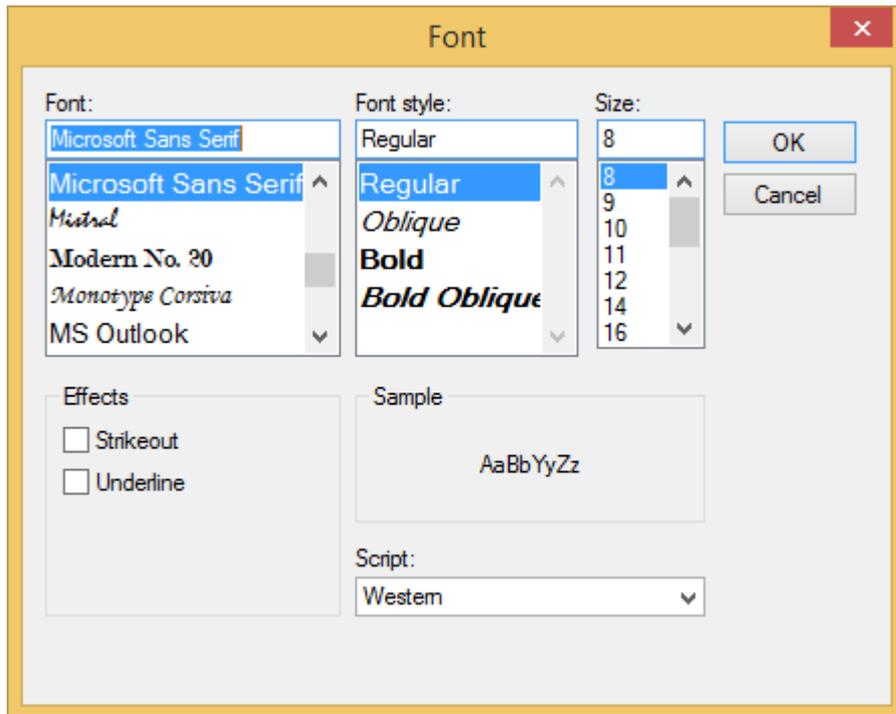
The properties command button exposes the **Chart Properties** designer for the **Header** and **Footer** elements when the user clicks on the properties command button.



Clicking on the  button opens the navigational tree view of the chart elements in the left pane of the **Chart Properties** designer.

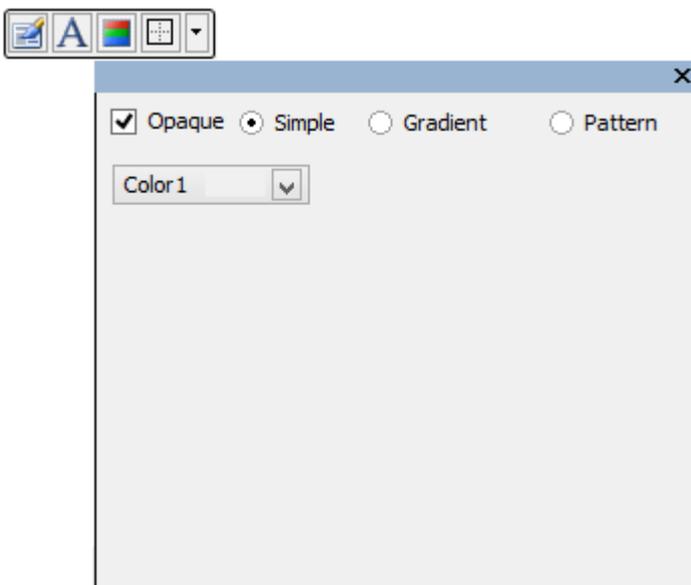
### Font button

The **Font** command button exposes the **Font** dialog box for the **Header** and **Footer** element. Here the Font style can be modified for the Header and Footer's text.



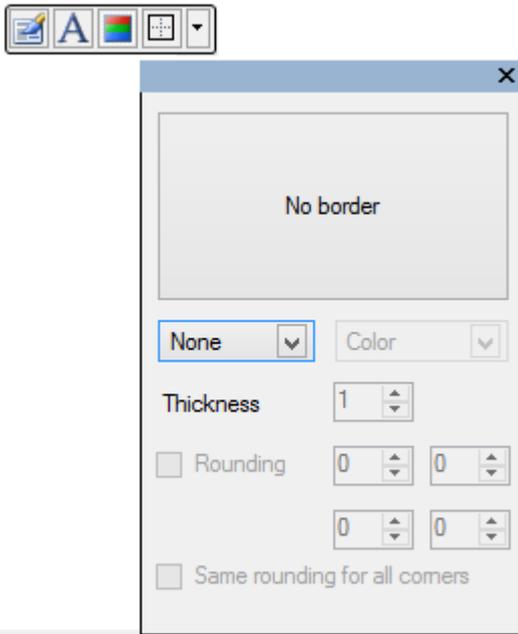
## Background button

The **Background** command button has a drop-down box that contains three different types of styles for the background and a color drop-down list box for the user to specify a color for the **Header** and **Footer's** background.



## Border button

The **Border** command button includes a drop-down box that contains editable Border styles and colors for the **Header** and **Footer's** border.



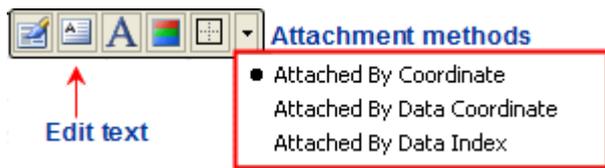
## Position button

The **Position** command button has a drop-down list box which includes a list of different compass directions (North, South, West, and East) for the user to choose from. The directions will position the **Header** or **Footer** in the North position which is above the Chart, the South position which is below the Chart, the West position which is to the left of the Chart, and the East position which is to the right of the Chart. The default compass position for the Header is north and the default position for the Footer is south.



## Label Toolbar

The floating toolbar for the Label element is almost the same as the Header/Footer toolbars except that it has an **Edit text** command for its text and different command items in its drop-down menu. Its drop-down menu has an **Attached by Coordinate**, **Attached By Data Coordinate**, and an **Attached By Data Index** command item. The **Attached By Coordinate** attaches the label anywhere on the chart. The number of pixels can be specified between the top-left corners of the chart to the **ChartLabel**. The **Attached By Data Coordinate** item attaches the label anywhere inside the PlotArea. The data coordinates can be specified. The **Attached By Data Index** item attaches the label to a specific data point on the chart. The series and point indices and the ChartGroup can be specified. For more information on how to use these attachment methods at design time please see [Attaching and Positioning Chart Labels](#). If you would like to see how to use these attachment methods programmatically through code, please see [Attaching the Chart Label by Pixel Coordinate](#), [Attaching the Chart Label by Data Coordinate](#), [Attaching the Chart Label by Data Point](#), or [Attaching the Chart Label by Data Point and Y Value](#). The figure below illustrates the unique command buttons in the Label toolbar.



## Exposing the Label Floating Toolbar

In order for the Label floating toolbar to appear you have to select the **Add/Edit labels** from the **PlotArea** toolbar and then add a label from the **Edit Label editor**.



## Label Floating Toolbar's Command Buttons

The section below lists all of the command buttons available in the **Label** toolbar and describes the functionality of each one.

### Properties Button

The **Properties** button for the Label toolbar exposes the Edit labels editor once it is clicked by the user. In the Edit label editor you can add or edit existing labels.

### Edit text Button

The **Edit text** button for the Label floating toolbar makes the textbox for the Label editable.

### Background Button

The **Background** button functions exactly like the rest of the Border command buttons for the **C1Chart** control toolbars.

### Border button

The **Border** command button functions exactly like the rest of the Border command buttons for the **C1Chart** control toolbars.

### Drop-Down menu

The drop-down menu for the **Label** toolbar contains three menu items for the user to choose from: **Attached By Coordinate**, **Attached By Data Coordinate**, and **Attached By Data Index**. The **Attached By Coordinate** attaches the label by its specified (X, Y) coordinate. The **Attached By Data** Coordinate attaches the label by its specified data coordinate, and the **Attached By Data Index** item attaches the label by its specified data index.

## Legend Toolbar

The floating toolbar for the Legend element is similar to the Header/Footer toolbars except that it includes an additional command button called Edit text. The Label toolbar also contains this command button.

The following figure illustrates the Legend toolbar:



## Exposing the Legend Floating Toolbar

In order for the **Legend** floating toolbar to appear you have to select the Legend item from the Chart toolbar's drop-down menu.



The section below lists all of the command buttons available in the **Legend** toolbar and describes the functionality of each one.

## Properties Button

The **Properties** button for the **Legend** floating toolbar exposes the **Chart Properties** designer for the Legend once it is clicked by the user.

## Edit text Button

The **Edit text** button for the **Legend** floating toolbar makes the textbox for the Legend editable.

## Background Button

The **Background** button functions exactly like the rest of the Border command buttons for the [C1Chart](#) control toolbars.

## Border button

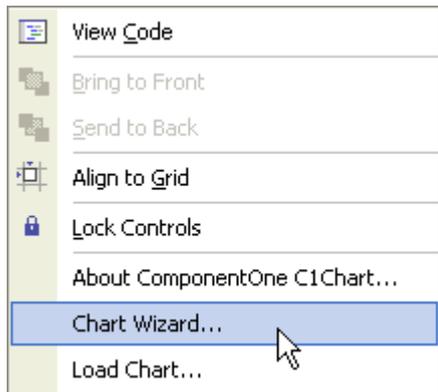
The **Border** command button functions exactly like the rest of the Border command buttons for the [C1Chart](#) control toolbars.

The next section describes the Chart Wizard interface and how it can be used to create 2D Charts at design time.

## Working with the Chart Wizard

The **Chart Wizard** provides an easy three step process to guide you through the basic steps for creating a chart. You can choose from various chart types, set up the chart's content for its header, footer, legend, and x and y axes, and create/edit the chart's data. The **Chart Wizard** includes a **Preview** tab so you can see how your chart looks after you have made changes to it.

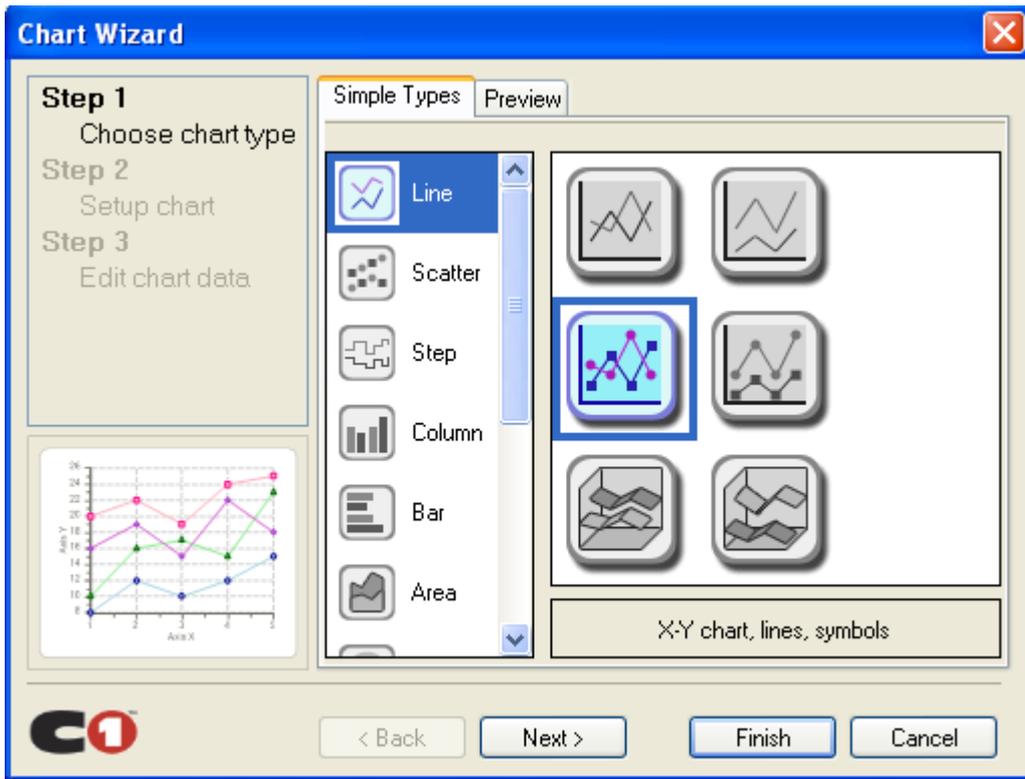
To access **Chart Wizard** at design time, right-click on the [C1Chart](#) control. Select **Chart Wizard** from the context menu:



## Step 1. Choose Chart Type

There are various chart types to choose from the chart wizard. This offers the ability to quickly create many different types of charts at design time. The chart types include line, scatter, column, bar, area, pie, stock, polar/radar, Gantt, Cylinder, Cone, and Pyramid. In addition each chart type has individual subtypes, which allows further selection of chart types. See [Specific 2D Charts](#) for more detail about each chart type.

## Simple Types tab

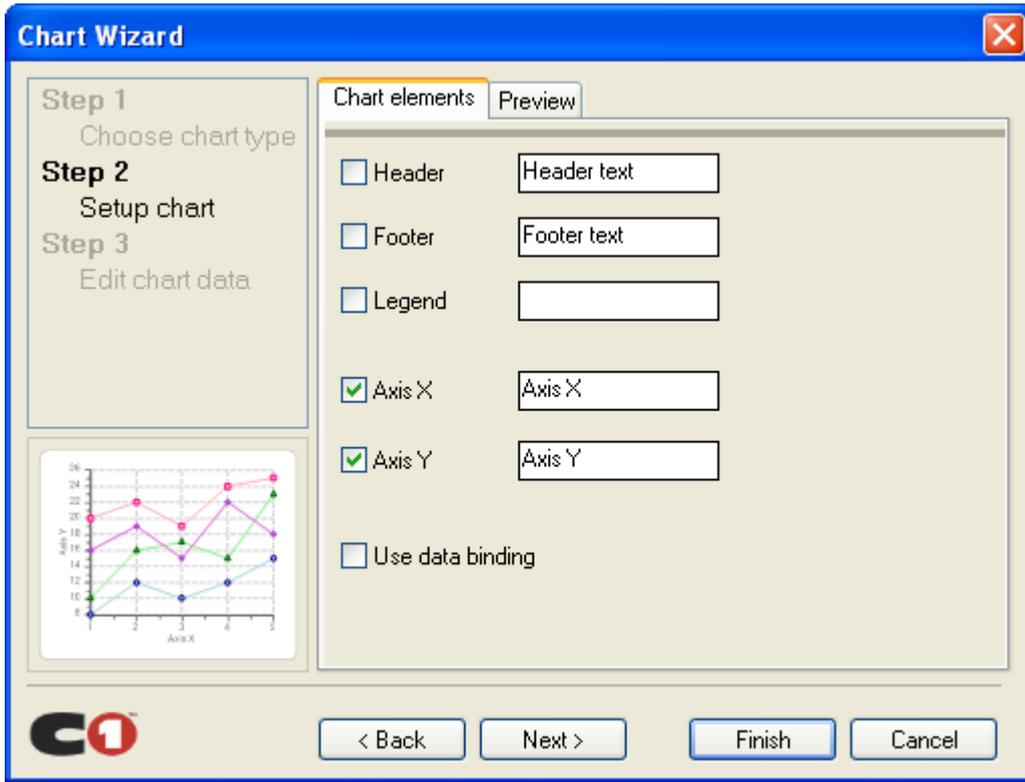


After a chart type is selected, choose Next to set up specific details for the new chart.

## Step 2. Setup Chart

The next step in the chart wizard provides you with the ability to create or modify the **Header**, **Footer**, and **Legend**, as well as the titles that you wish to specify for **Axis X** and **Axis Y**. You can specify a title for the X-axis or Y-axis by typing in the corresponding text box. Clicking on the check box enables the Chart element to appear in your chart. For more details about these elements, see [Customizing Chart Elements](#). You can check the **Use data binding** checkbox if you want to link your chart to an existing database.

### Chart elements tab



Once the chart element settings are specified, select **Next** to edit the chart data.

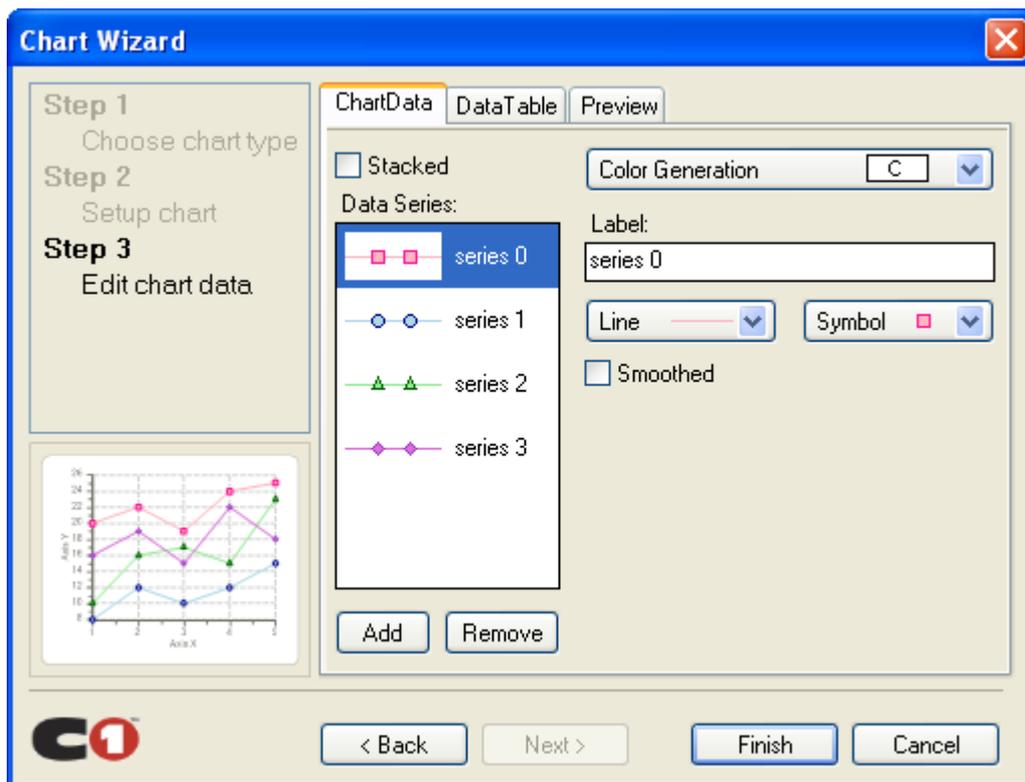
**Note:** At any time during the design of your chart, click the **Preview** tab to see what the final chart will look like.

## Step 3. Edit Chart Data

The next step in the Chart Wizard is to create or modify the appearance of the data chart series in the data area of the chart. This step includes a **ChartData**, **DataTable**, and **Preview** tab.

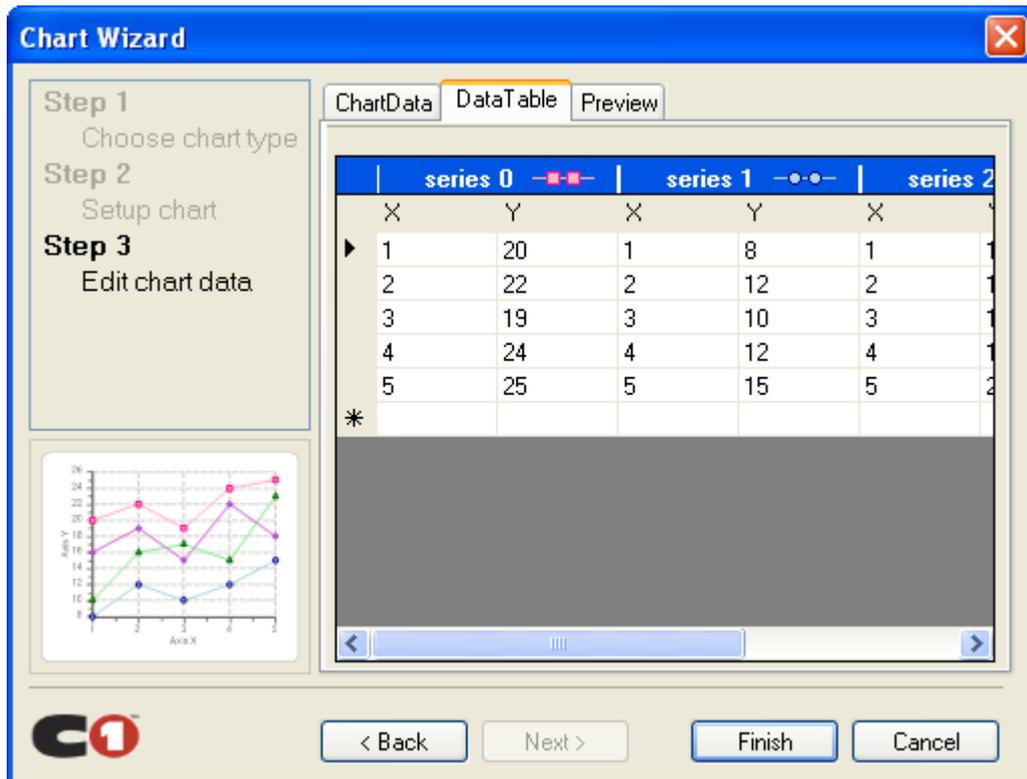
### ChartData tab

Under the **ChartData** tab, you can add or remove Data Series, specify the appearance of each data series such as the Labels, Linestyle, and Symbols. You can check the Smoothed checkbox if you want the line of a specific series in the data to appear smooth. This can be done by selecting a series in the Data Series list box and then checking the **Smoothed** check box.



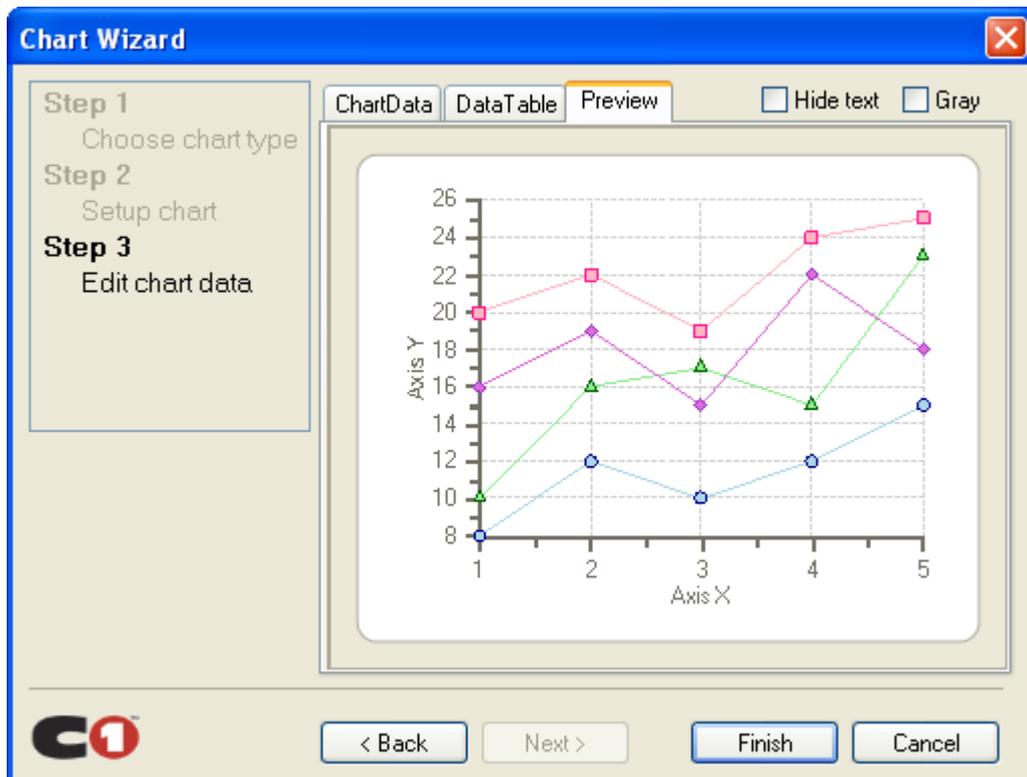
## DataTable tab

You can also select the **DataTable** tab to view all data associated with your chart. If you would like to change specific data, you can do so here.



## Preview tab

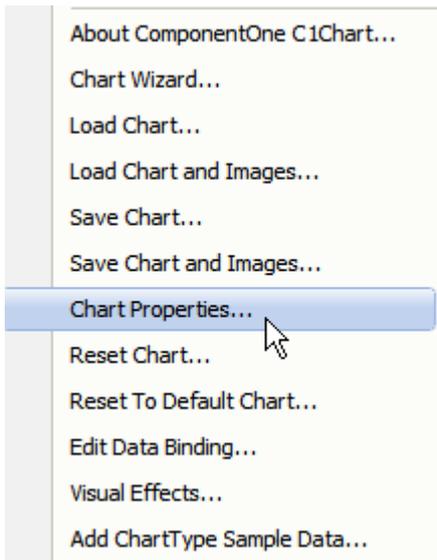
Finally, select the **Preview** tab to see your finished chart. Click **Finish** when the chart is completed.



## Working with the Chart Properties Designer

The **Chart Properties** designer provides an easy and interactive way to create and modify a new or existing chart. Like the **Chart Wizard**, it contains the same design and function as **Step 1: Choose Chart Type** and **Step 3: Edit chart data**. However, it also includes additional property settings for the x and y axis along with appearance settings for the header, footer, legend, chart area, and plot area of the chart.

The **Chart Properties** designer can be accessed at design time by right-clicking on the **C1Chart** control and selecting **Chart Properties** from its context menu:



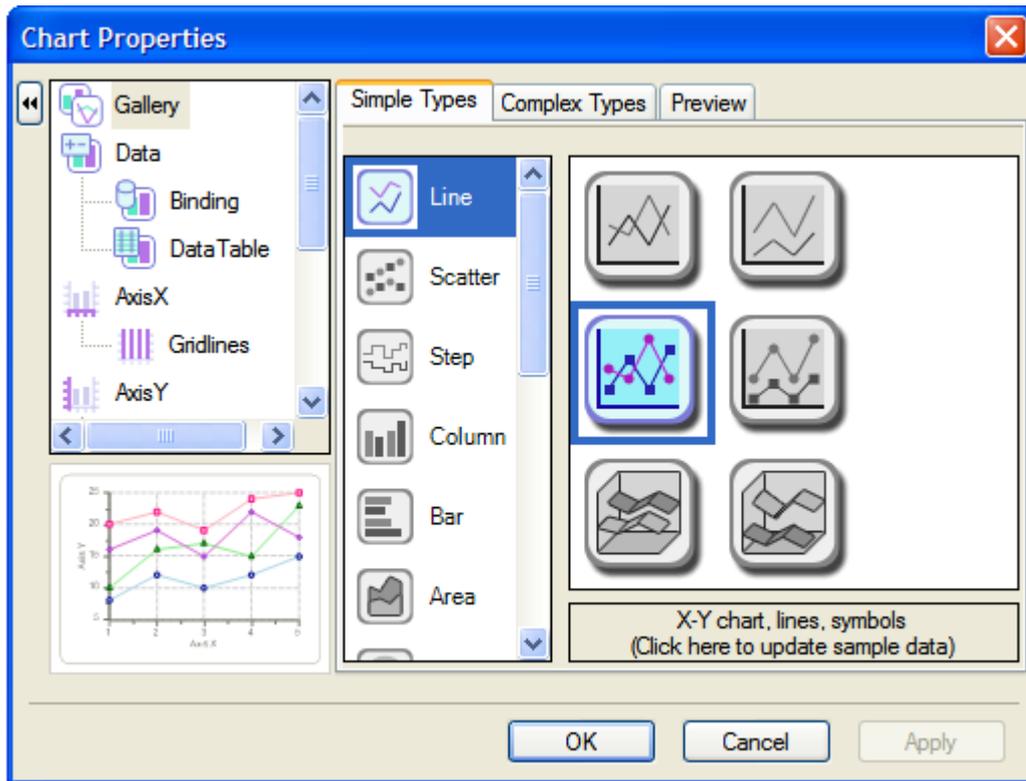
The **Chart Properties** designer provides more options to address specific details with the design of the chart you are developing. The following topics provide a visual detail of the **Chart Properties** designer interface and explain the functionality of each element in the **Chart Properties** designer.

## Gallery Element

The **Gallery** item in the left pane of the **Chart Properties** dialog box provides options for choosing a chart type and/or a sub-type of a chart. To see a description of all chart type selections, see [Specific 2D Charts](#). You can choose from a variety of simple chart types or you could click on complex types to add more functionality to your chart.

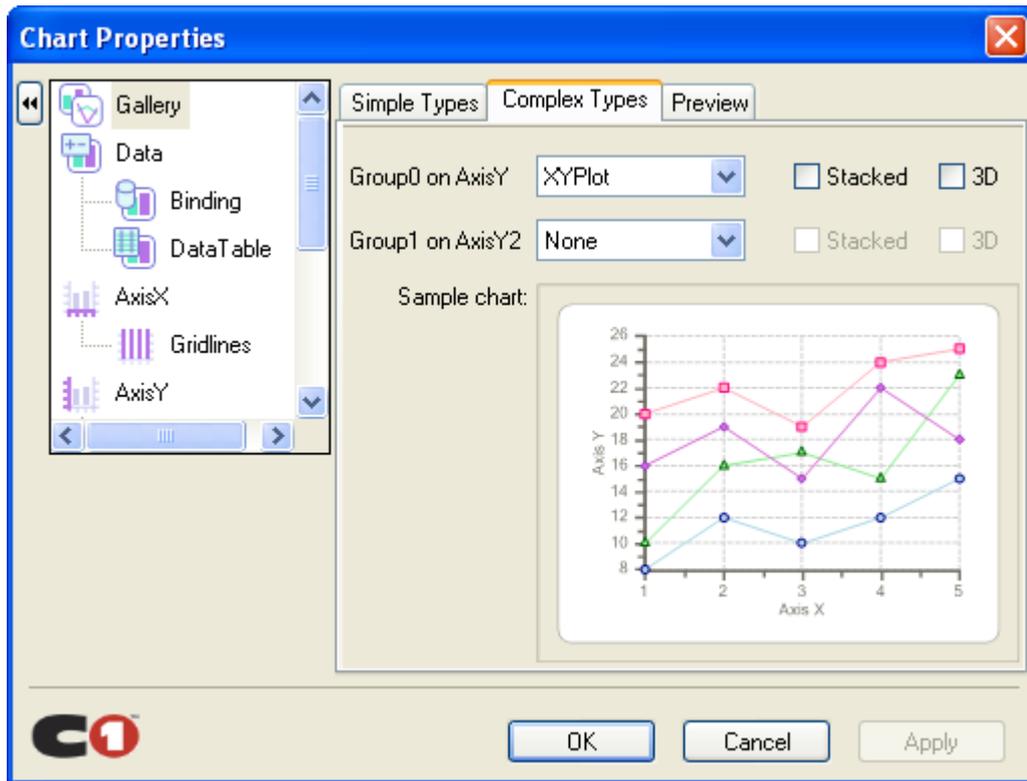
## Simple Types Tab

In the **Simple Types** tab you can choose from one of many simple chart types and then you can select a specialized chart located in list box next to the simple chart types.



## Complex Types Tab

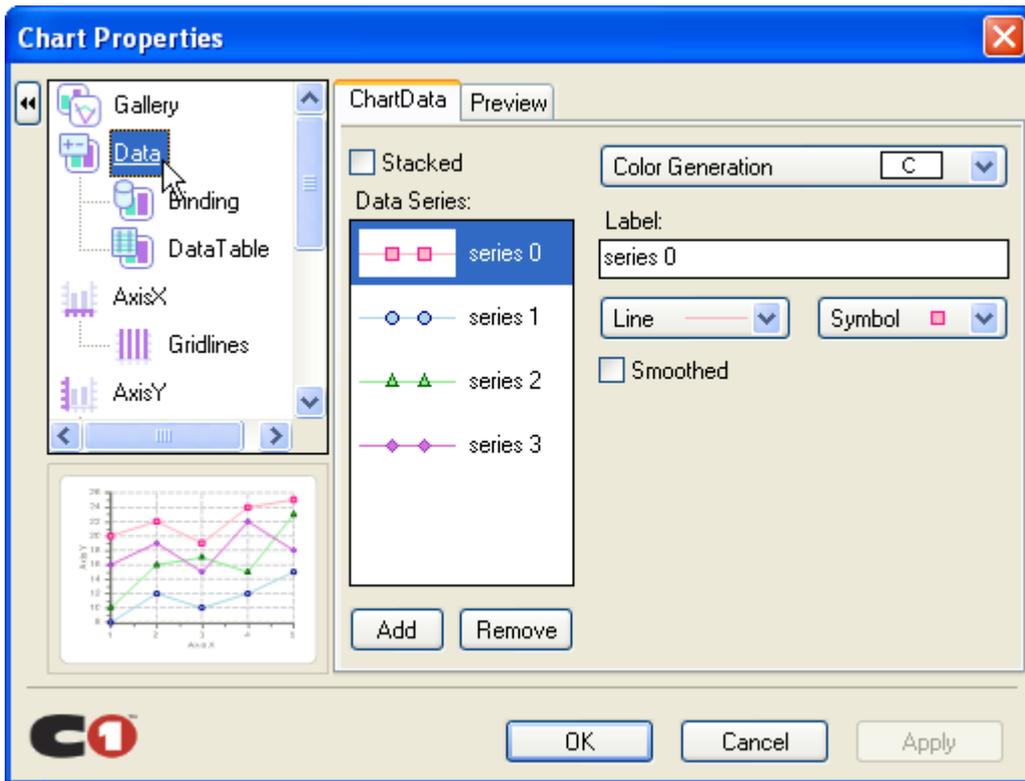
In the **Complex Types** tab you can specify whether you want to chart one or two chart groups. Also, you can select the type of chart you would like to create in the drop-down box for each group. For each group, you have the option to make the groups stacked and/or 3D. The following image illustrates the elements included in the **Complex Types** tab of the Gallery section in the Chart Properties dialog box.



**Note:** If you don't select a chart type for data [Group1] then the elements for data [Group1] will not appear in the list box in the left pane of the **Chart Properties** dialog box.

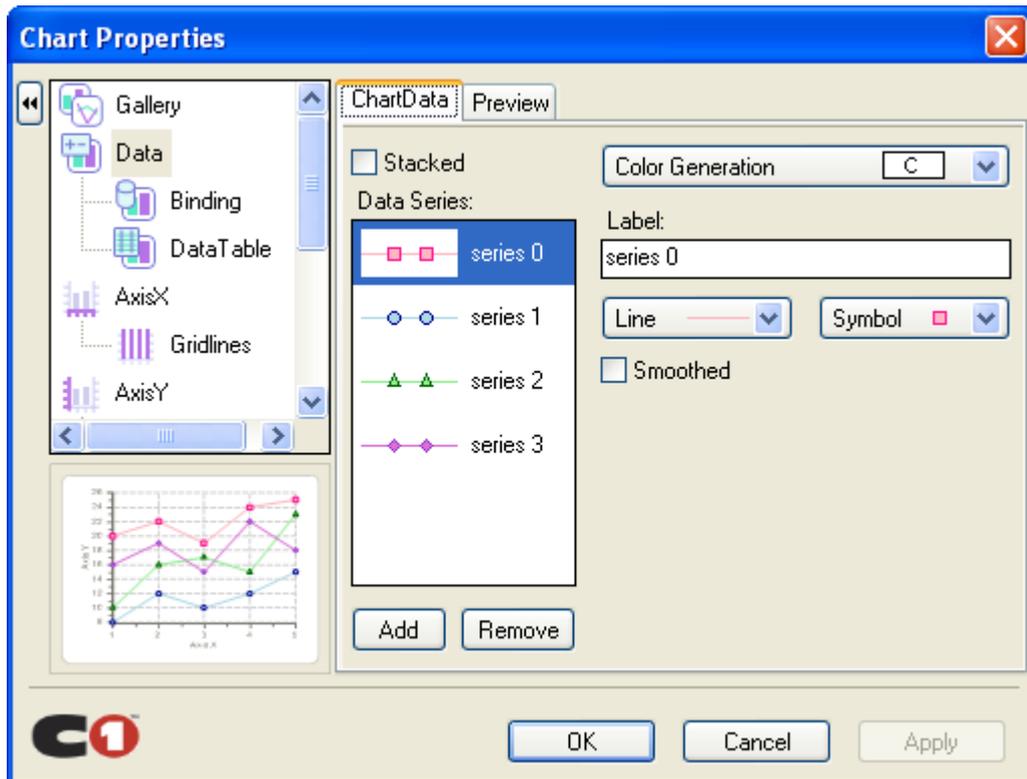
## Data Element

The **Data** element in the left pane of the **Chart Properties** dialog box provides a **ChartData** and **Preview** tab to create or modify the series in your chart data and to preview your current chart.



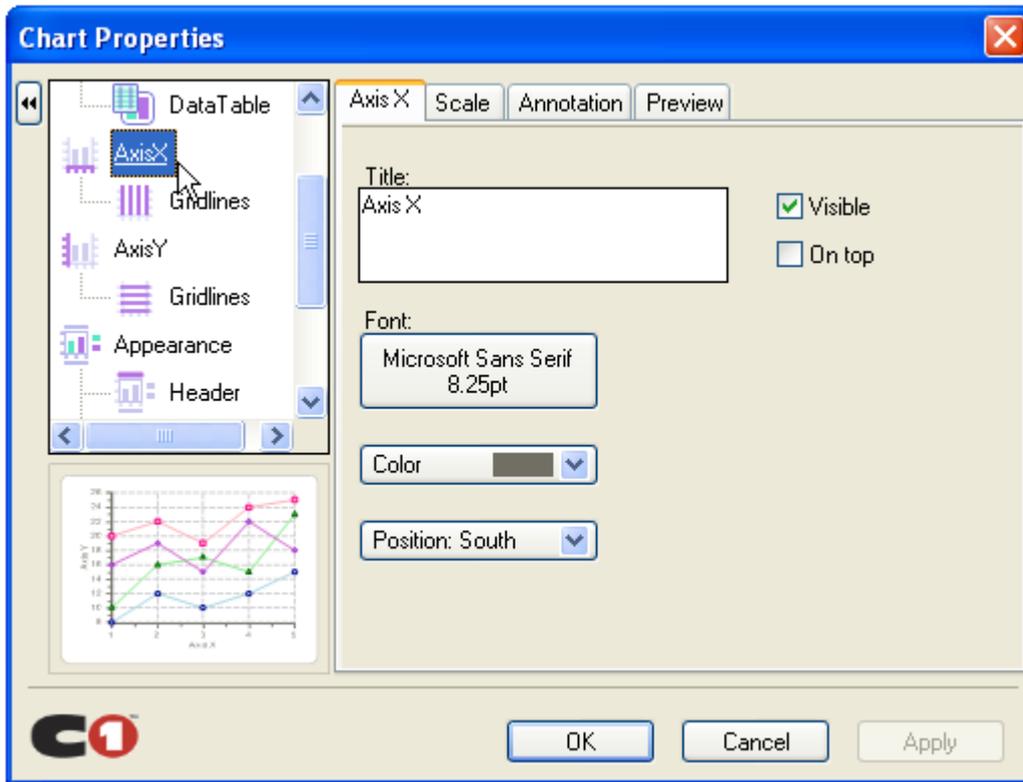
## Chart Data Tab

By selecting the **ChartData** tab you can add or remove data series in your chart, specify the label and color of each data series, and you can stack your data by clicking on the **Stacked** check box. To select a data source, click on Binding element and choose a data source. To modify the content in the table, click on DataTable and make any changes to the data. See [Defining the ChartDataSeries Object](#) or [Entering and Modifying Chart Data](#) or the [SeriesList](#) property to find out more about the ChartDataSeries object.

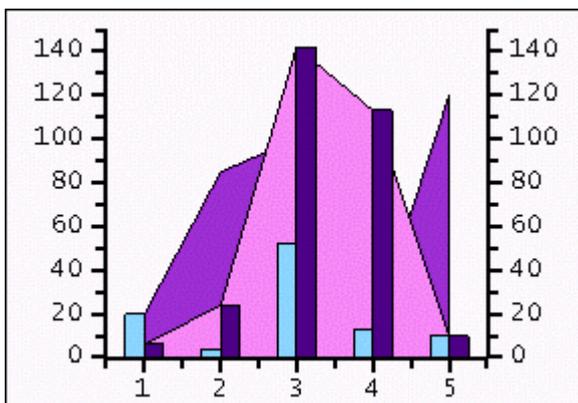


## Axis X , Axis Y, and Axis Y2 Elements

You can select **Axis X**, **Axis Y**, or **Axis Y2** to modify or create the format style, scale, and annotation of the axes. Each axes element (**AxisX**, **AxisY**, and **AxisY2**) contain the following tabs in the upper right pane of the Chart dialog box: the name of the selected element (**AxisX**, **AxisY**, or **AxisY2**), the **Scale**, the **Annotation**, and the **Preview** tab.



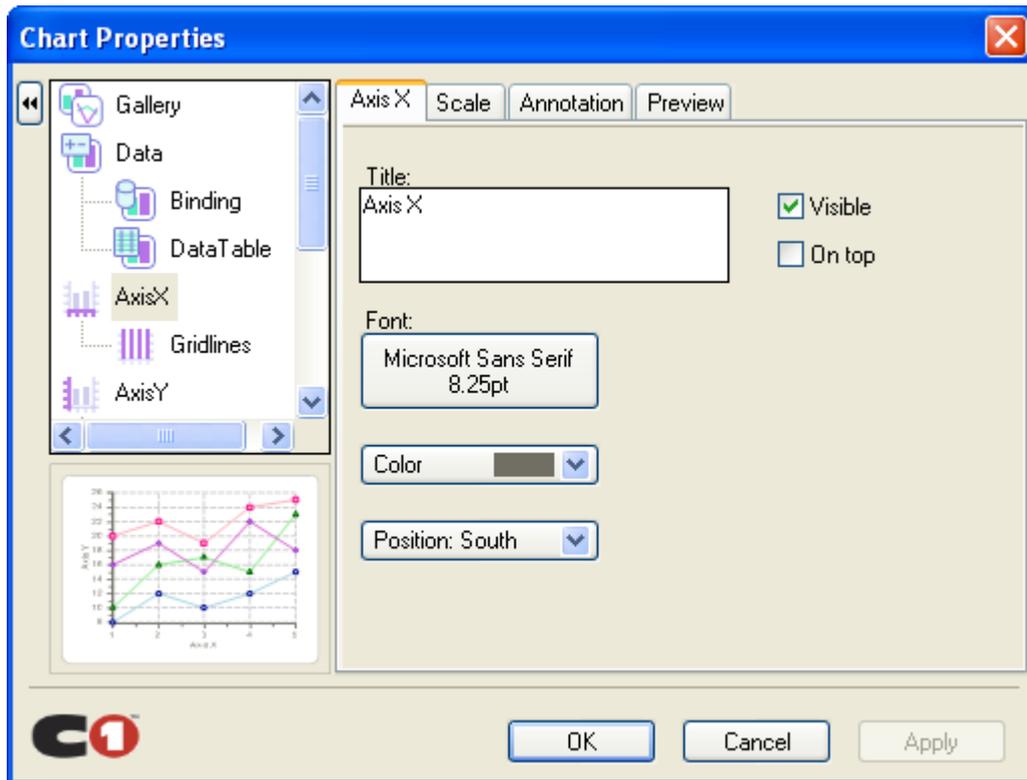
The Y2-axis element appears in the left pane of the **Chart Properties** designer once data is added to the second chart group (Group1). The data in the Y2-axis determines the maximum and minimum values of the axis in the same way that ChartGroup(0) data determines the maximum and minimum values of the Y-axis. Manual settings can be used to force alignment between axes. The picture below shows a chart that contains a Y2-axis.



In order to make the AxisY2 element appear in the left pane of the **Chart Properties** dialog box you must select a chart type for Data [Group1] in the **Complex Types** tab from the Gallery element. Also, after selecting a chart type for Data [Group1] you need to add a series to Data [Group1]. For more information see the [Complex Types Tab](#).

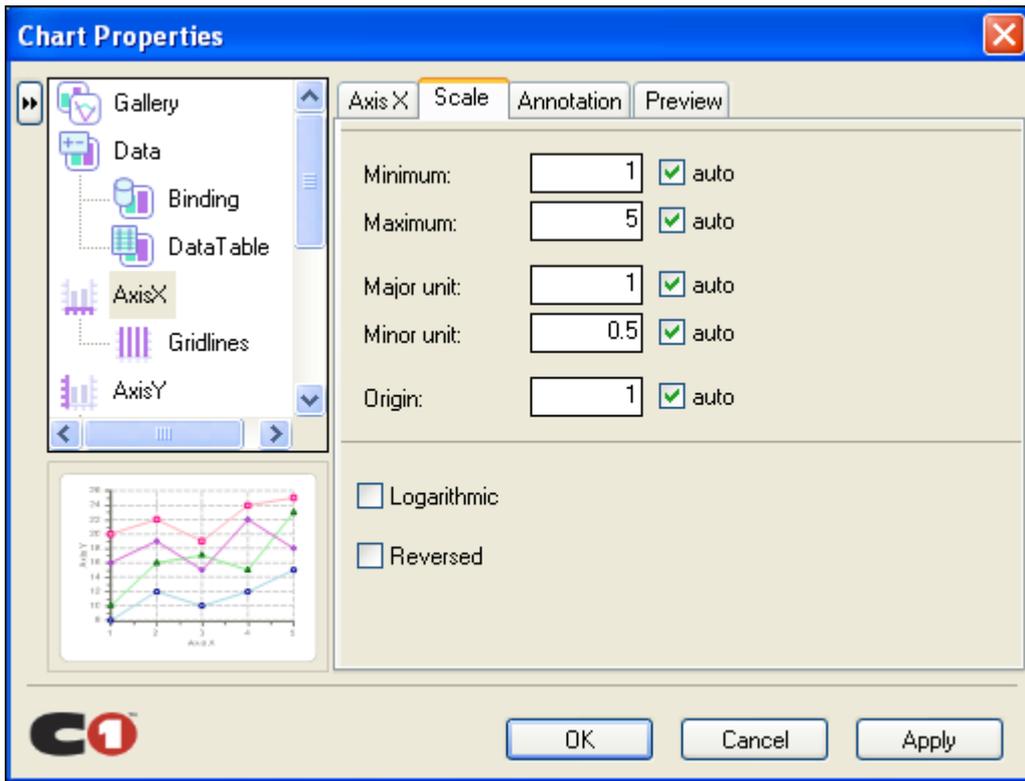
## Axis X, AxisY, or Axis Y2 Tab

The screen shot below represents the tab name for the selected Axis, in this example it is **Axis X**. In the **Axis** tab you can specify the **title**, **font**, **color**, and **position** of the axis. In addition you have the option to make the title for the axis appear on top by selecting the **On top** check box.



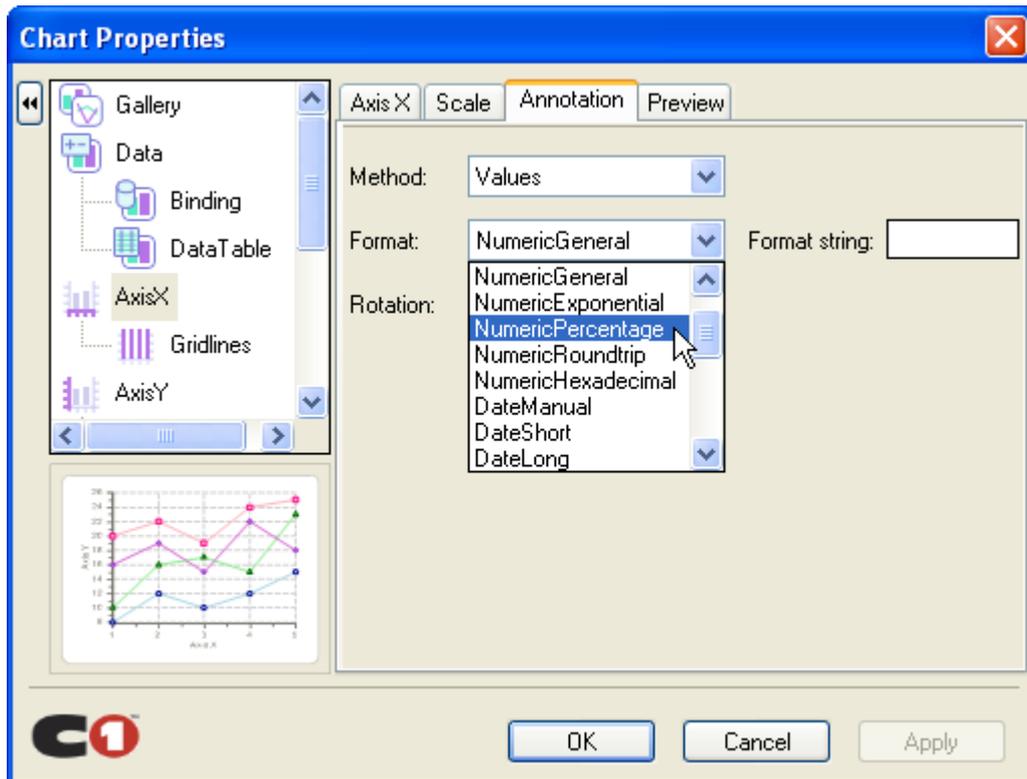
## Scale Tab

The **Scale** tab allows you to set the **Minimum**, **Maximum**, **Major Unit**, **Minor Unit**, and **Origin** property. In the example below the **auto** button was enabled so all the values for each property would be set to their default values. Your axis can also be **Reversed** and/or **Logarithmic** by clicking on the box.

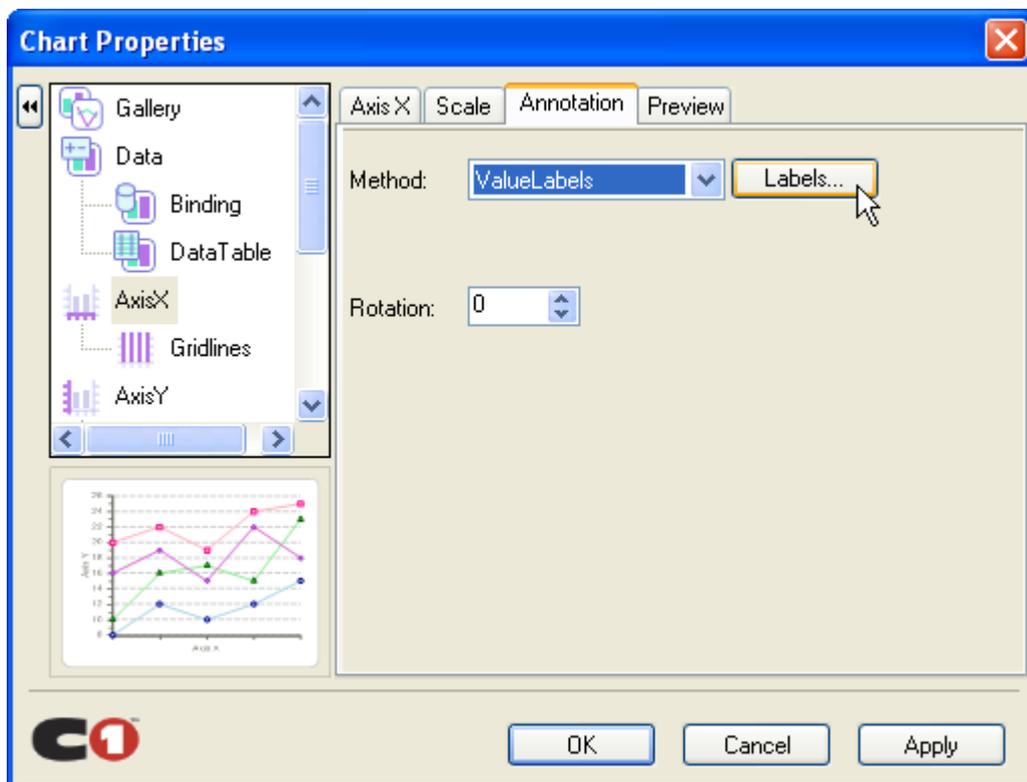


## Annotation Tab

The **Annotation** tab allows you to specify whether you would like the AxisX, AxisY, or AxisY2 to be labeled as Values or ValueLabels. If you select Values from the **Method** drop-down box a **Format** drop-down box will appear. Here you can select from various types of formats. Also, you can specify the format string in the **Format string** textbox. For more information see the [Values Annotation](#). For more information about the Format types, see FormatEnum Enumeration.

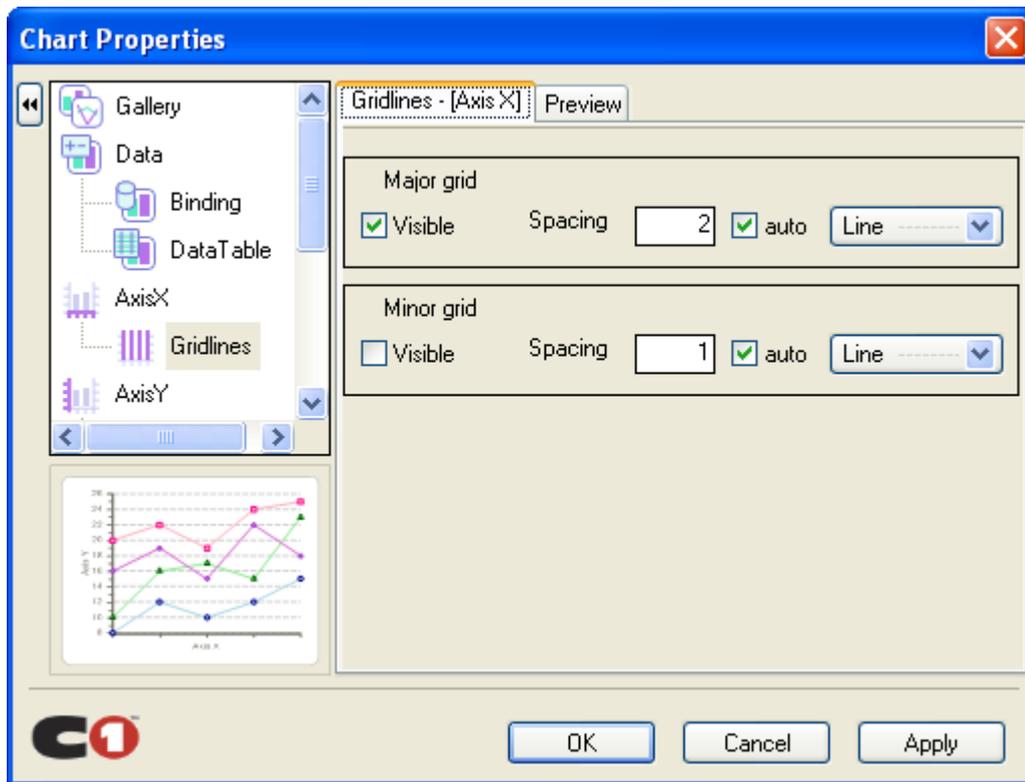


If you select **ValueLabels** from the **Method** drop-down box, a **Labels** button appears. Clicking on the **Labels** button exposes the **ValueLabel Collection Editor** dialog box. Here you can add members to your labels. For more information about the **ValueLabel Collection Editor** see the [ValueLabel Collection Editor](#).



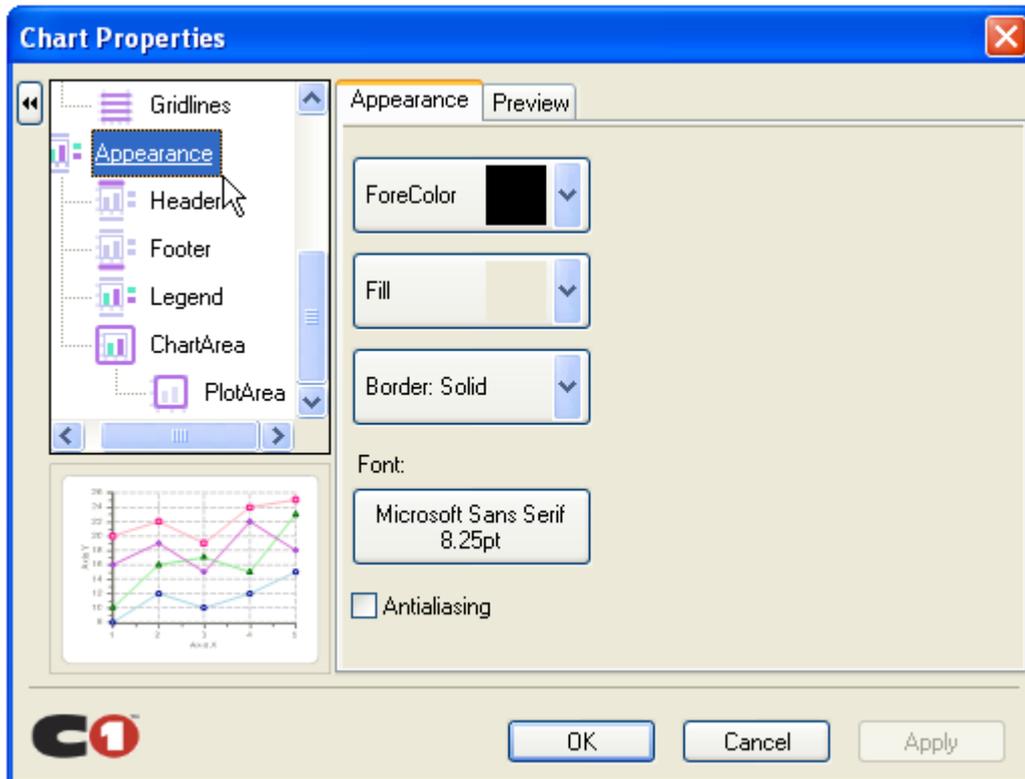
## Gridlines Tab

In the **Gridlines** tab you can add a major or minor gridline or both gridlines on either axes to improve the readability of the chart. If you would like to see the gridlines, simply check visible in the **Major Grid** and **Minor Grid** section. You can specify the spacing between each major or minor grid line by typing the amount in the **Spacing** text box. Also, you can specify the line style type for each grid line.



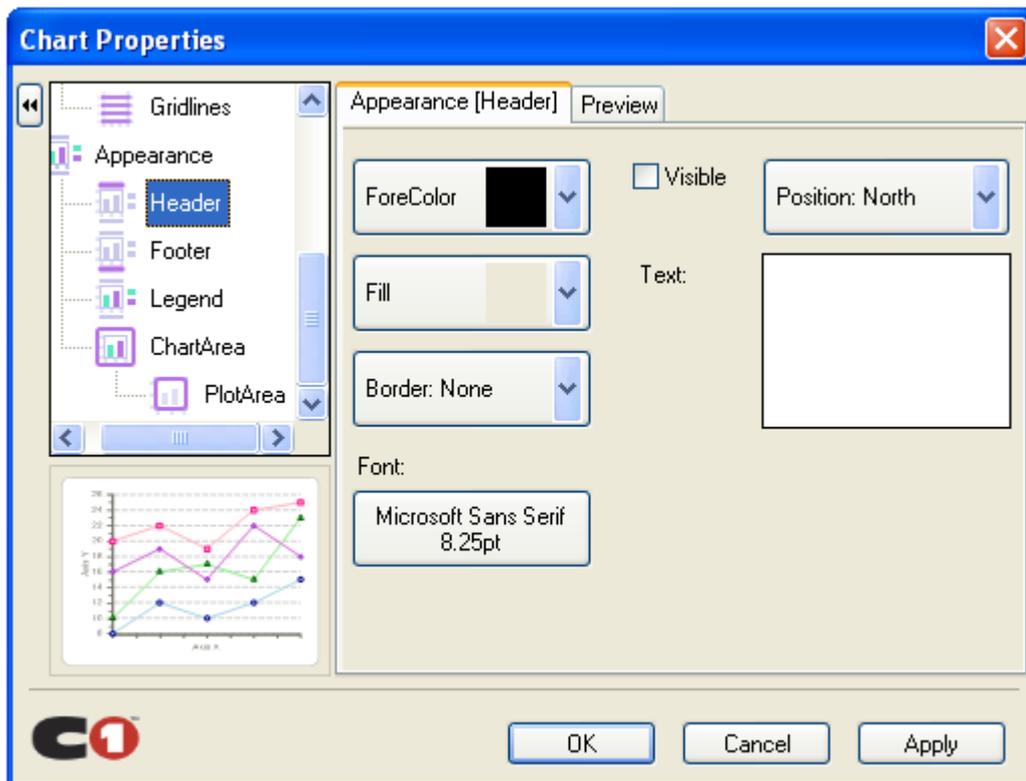
## Appearance Element

The **Appearance** element in the left pane of the **Chart Properties** designer provides properties in the **Appearance** tab for modifying the chart's forecolor, fill color, border style, font style, and whether or not to enable antialiasing. The sub elements of the Appearance element includes properties for modifying the appearance of the chart's Header, Footer, Legend, ChartArea, or PlotArea.



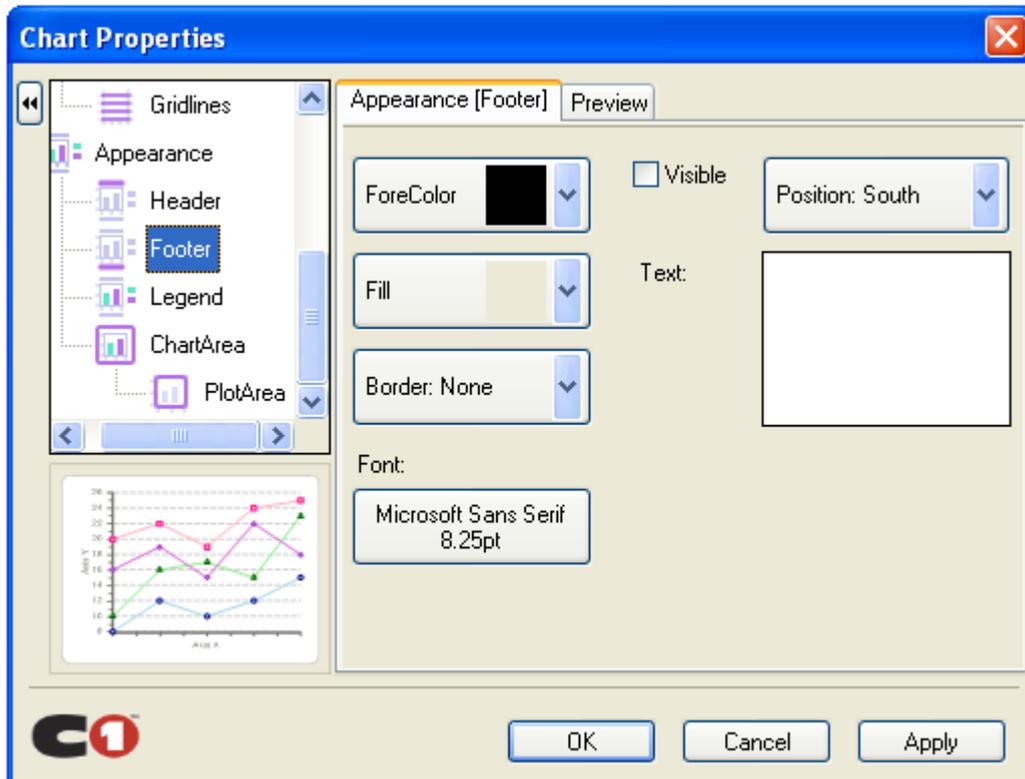
## Header Appearance Tab

The **Header** tab contains the same properties as the **Footer** tab: ForeColor, Fill, Border, Font, Position, Visible, and Text.



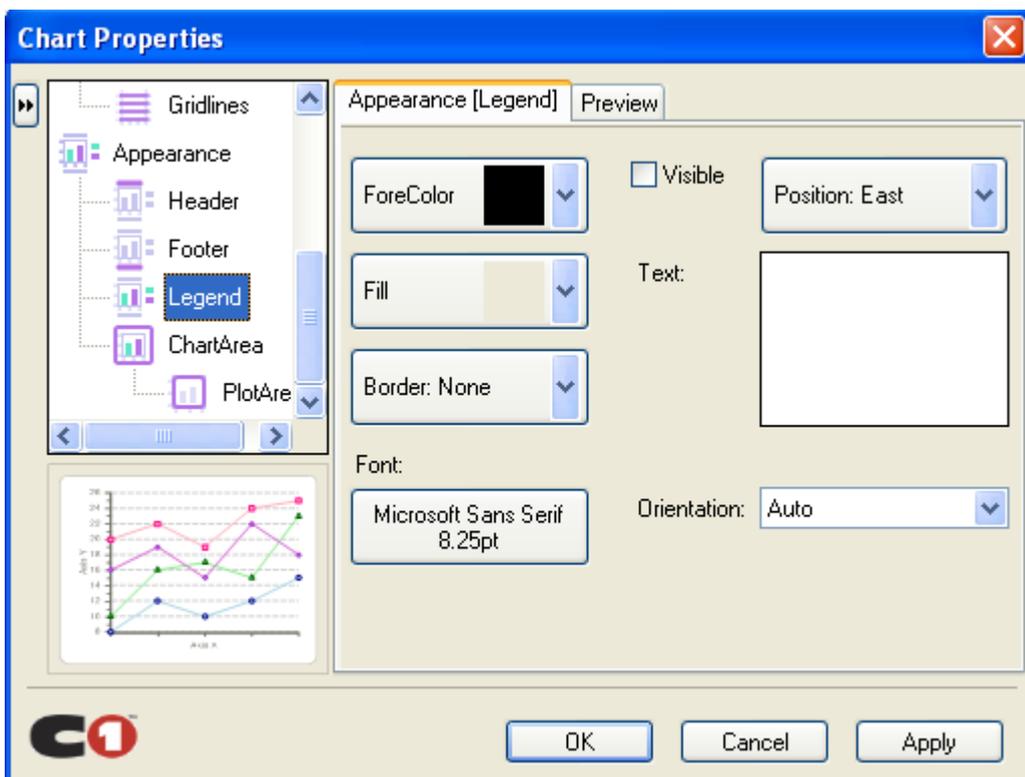
## Footer Appearance Tab

The **Footer** tab contains the same properties as the **Header** tab: ForeColor, Fill, Border, Font, Position, Visible, and Text.



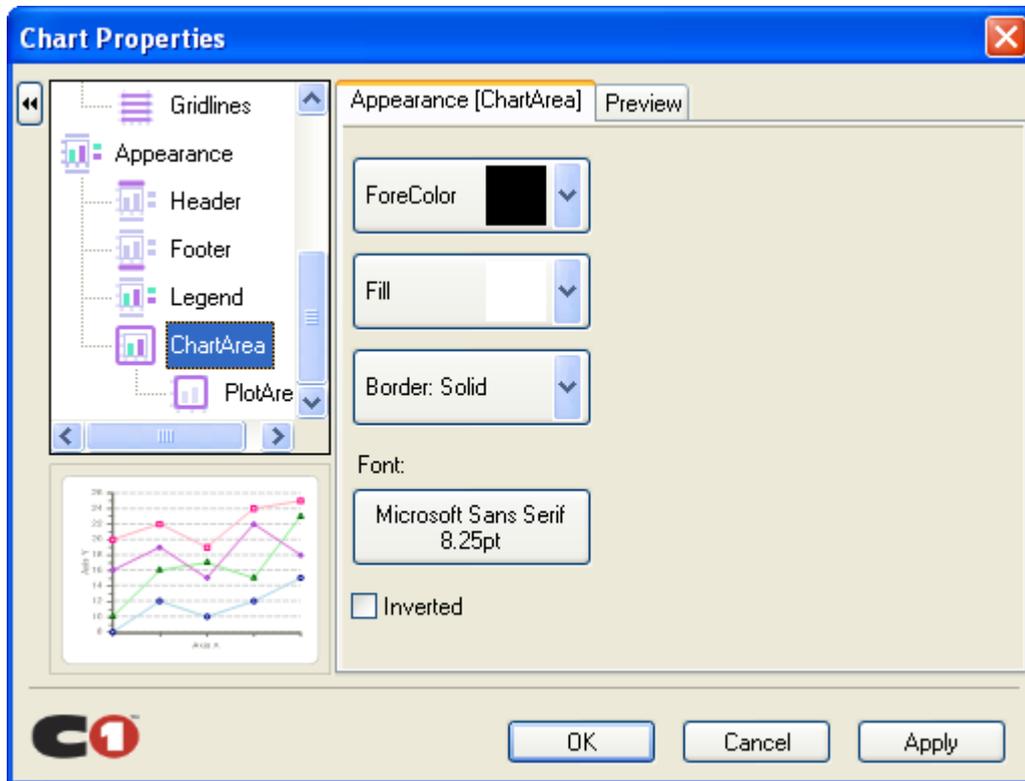
## Legend Appearance Tab

The **Legend** tab contains the same properties as the Header and Footer plus an additional **Orientation** property.



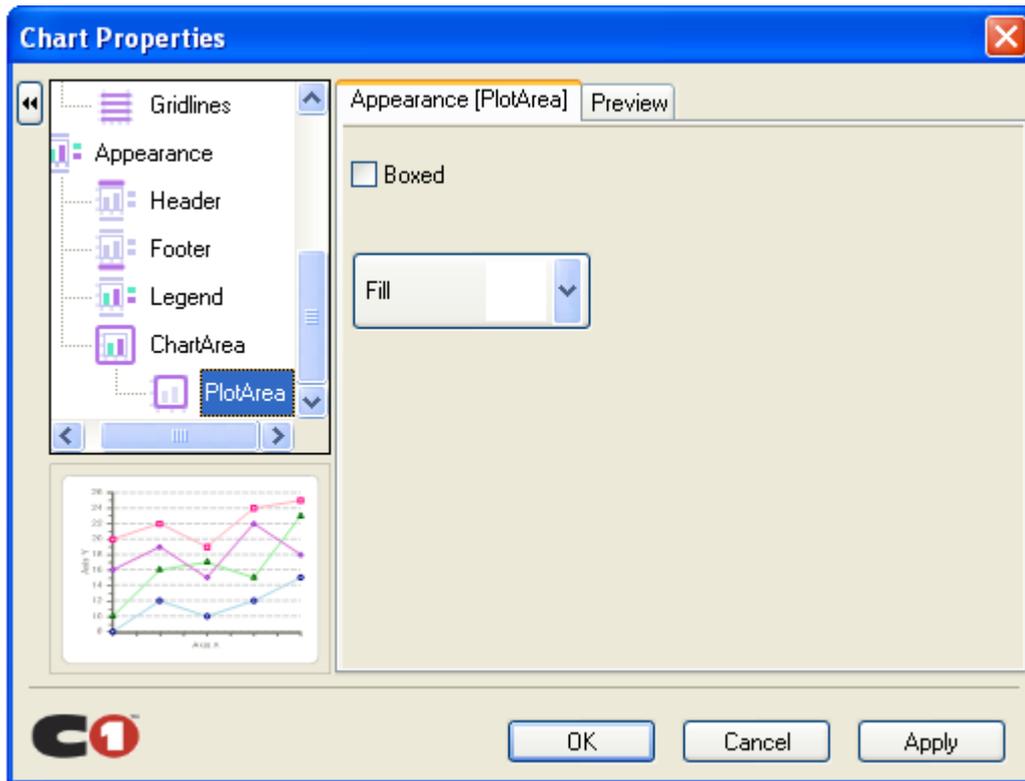
## ChartArea Appearance Tab

The **ChartArea** tab contains the **ForeColor**, **Fill**, **Border**, and **Font** properties, plus an **Inverted** property to invert the chart axes.



## PlotArea Appearance Tab

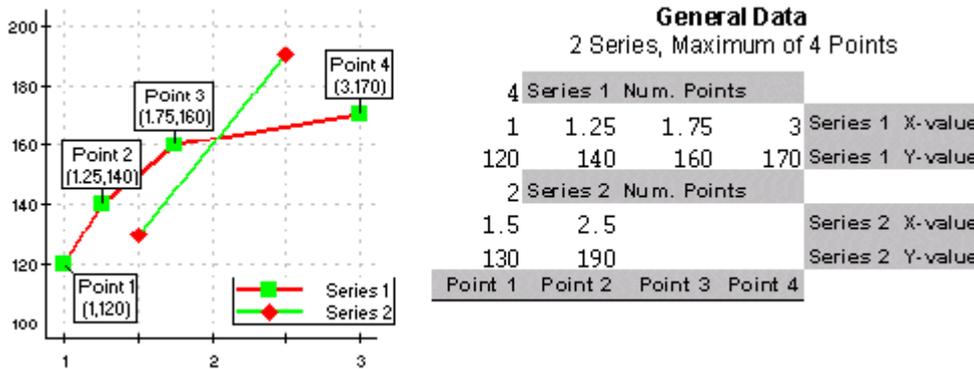
The **PlotArea** element contains a **Fill** and **Boxed** property. Clicking on the **Boxed** check box will create a box around the plot area in the chart.



## Charting Data

The 2D Chart displays data supplied in a general layout. Data gets loaded into the six ChartDataArray objects, which take a type Object. Data can be loaded from pre-filled arrays or can be entered in at design time.

A General layout is used with every chart type. It contains an X and Y array, along with Y1, Y2, Y3, and Y4 arrays. All of these arrays can take data or can be null. For instance, to plot an XY chart with only one series, only the X and Y arrays would contain data. The other Y arrays would remain empty. In addition, unlike an array layout, each series can have a different number of points, and does not have to have matching X values which allows for charting as in the following image:



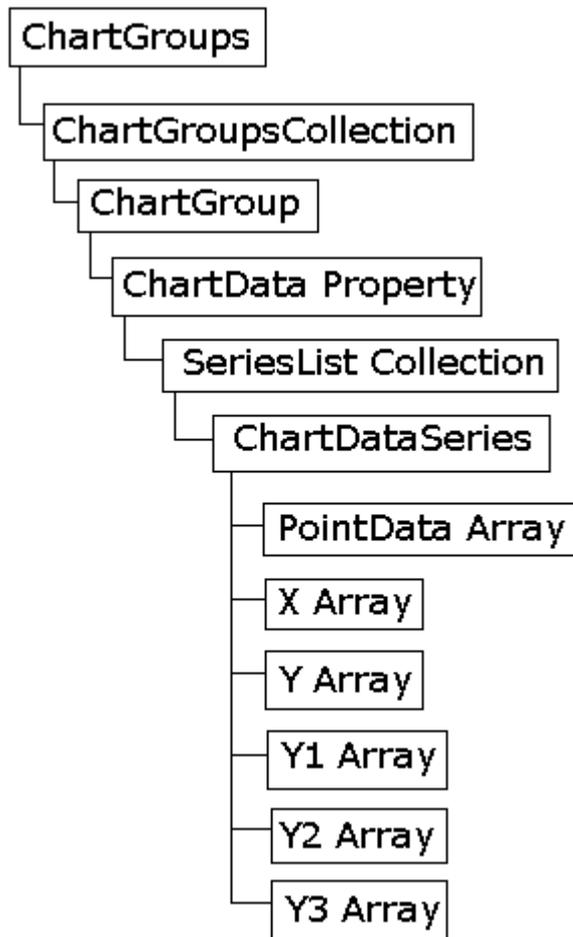
For convenience, a **PointData** property (an array of PointF structures) can also be used to supply X and Y data. The PointF values are not independent of the X and Y arrays, merely another form of data input and output. Changing values in the **PointData** property changes the X and Y arrays and vice versa.

The important General layout characteristics illustrated in the preceding image are the following:

- The points in each series have their own X- and Y-values.
- Each series can contain a different number of points.

## Defining the Chart Data Objects

**C1Chart** has a specific hierarchy of data related classes, collections, and properties. This section details each data object individually and provides information on how to access, modify, and create the data for the chart.



## Defining the ChartGroup Object

Data in a chart is organized into [ChartGroups](#). Each chart contains two ChartGroups (although most charts will only use the first ChartGroup), each of which is treated as a separate entity. ChartGroups allow more than one chart to be displayed in the ChartArea and provide access to many of the chart-specific properties.

In [C1Chart](#), a chart group is represented by a [ChartGroup](#) object. The ChartGroup objects are organized into the [ChartGroupsCollection](#), which is accessed through the ChartGroups object. This collection provides access to the ChartGroups through two methods.

First the individual chart groups can be accessed through the collection. For example, the following code shows how to access individual chart groups through the ChartGroupsCollection object:

### To write code in Visual Basic

Visual Basic

```
C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartType = Chart2DTypeEnum.XYPlot
```

### To write code in C#

C#

```
c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartType = Chart2DTypeEnum.XYPlot;
```

Secondly, the [Group0](#) and [Group1](#) properties of the ChartGroup's object return the associated ChartGroup and allow

circumvention of the long collection name by using the following code:

#### To write code in Visual Basic

Visual Basic

```
C1Chart1.ChartGroups.Group1.ChartType = Chart2DTypeEnum.XYPlot
```

#### To write code in C#

C#

```
c1Chart1.ChartGroups.Group1.ChartType = Chart2DTypeEnum.XYPlot;
```

The ChartGroupsCollection accepts usual iteration methods as illustrated below:

#### To write code in Visual Basic

Visual Basic

```
Dim cg As ChartGroup
For Each cg In C1Chart1.ChartGroups.ChartGroupsCollection
    cg.ChartType = Chart2DTypeEnum.XYPlot
Next
```

#### To write code in C#

C#

```
ChartGroup cg;
foreach ( cg In c1Chart1.ChartGroups.ChartGroupsCollection )
    cg.ChartType = Chart2DTypeEnum.XYPlot;
```

## Defining the ChartData Object

The [ChartGroup](#) object also contains the [ChartData](#) object. This object contains the [Hole](#) property, the [FunctionsList](#) property, the [SeriesList](#) property which returns the [ChartDataSeriesCollection](#) object, the [PointStylesList](#) property, and the [TrendsList](#) collection property. The [ChartGroup](#) object essentially provides access to all of the data-related objects and properties of the chart:

#### To write code in Visual Basic

Visual Basic

```
C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData
```

#### To write code in C#

C#

```
c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData;
```

The [FunctionsList](#) collection stores functions to be plotted in the Chart data or plot area. The [FunctionsList](#) property gets the [FunctionsCollection](#) object associated with the current [ChartData](#) object.

[PointStyles](#) provide a mechanism to mark specific data points with different visual attributes than other points of the same data series. [PointStyles](#) are contained by the [PointStylesCollection](#).

The [SeriesList](#) property returns a [ChartDataSeriesCollection](#) which is a collection of [ChartDataSeries](#) objects. [ChartDataSeries](#) objects contain all of the series and data for the chart. Series objects contain the [ChartDataArray](#) objects, which hold the chart's data.

The trend lines supported by chart with [TrendLine](#) objects, can be divided into two groups, including **regression** and **non-regression**. Regression trend lines are **polynomial**, **exponent**, **logarithmic**, **power** and **Fourier** functions that approximate the data which the functions trend.

The [ChartData](#) object also provides access to the [Hole](#) property. Data Holes are breaks in the continuity of the data set.

## Defining the ChartDataSeries Object

One of the more important objects in [C1Chart](#) is the [ChartDataSeries](#). The data series contains all of the data to be included in the chart and many important data-related properties.

The [ChartDataSeries](#) is contained in the [SeriesList](#) collection of the [ChartData](#) object, which is an object of the [ChartGroup](#):

### To write code in Visual Basic

Visual Basic

```
C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData.SeriesList(0)
```

### To write code in C#

C#

```
c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData.SeriesList[0];
```

The series object is crucial for data and data access for two reasons:

1. First, the number of series in the [SeriesList](#) collection of the [ChartData](#) object determines the number of data series to be displayed in the chart. If five sets of plots in an X-Y chart are desired, then there should be five series in the [SeriesList](#).
2. Secondly, the number of values specified in the [X](#), [Y](#), and [Y1](#), [Y2](#), [Y3](#) [ChartDataArray](#) objects within the series determine how many points that series will have and how it will be charted. If the chart has ten values in both the [X](#) and [Y](#) array objects for Series 0, and 5 values in both the [X](#) and [Y](#) arrays for Series 1, then the first series will have ten points and the second series will have five.

Note that the number of points in a series is the number of elements of the longest data array object applicable to the series. For example, an XY-Plot uses [X](#) and [Y](#) arrays. If [X](#) has five elements and [Y](#) has three elements, then the number of points is five, and the last two elements of [Y](#) are assumed to be data holes. Even if the [Y1](#) array has ten elements, the number of points is five because an XY-Plot does not use the [Y1](#) array. However, if the chart type is changed to **HiLo**, then [X](#), [Y](#) and [Y1](#) are all used and so the number of points would be ten, and both the [X](#) and [Y](#) arrays would be assumed to contain data holes in the missing elements.

## Defining the ChartDataArray Object

Each series contains six [ChartDataArray](#) objects. The [X](#), [Y](#), [Y1](#), [Y2](#), [Y3](#) data array objects hold arrays of data corresponding to the [X](#), [Y](#), [Y1](#), [Y2](#), and [Y3](#)-axes. The sixth data array object is the [PointData](#) object. This object accepts an array of [PointF](#) or [Point](#) values and is intended to be used as an alternative to the [X](#), [Y](#), [Y1](#), [Y2](#), [Y3](#) data array objects.

Each [ChartDataArray](#) is a read-only object that can only be modified through two methods. The [CopyDataIn](#) and [CopyDataOut](#) methods of the [ChartDataArray](#) copy arrays of data into these objects and return arrays of data from

these objects.

The `X`, `Y`, `Y1`, `Y2`, `Y3` data array objects accept `Object` data types. This allows a variety of .NET array types to be entered into the `ChartDataArray` objects (except for `PointData`).

## Entering and Modifying Chart Data

Under the `ChartDataSeries` object, the six `ChartDataArray` objects hold the arrays of data that the chart is drawn from. The `X`, `Y`, `Y1`, `Y2`, and `Y3` data array objects hold and return the data arrays for the X and Y-axes, while the `PointData` data array object holds and returns `PointF` and `Point` objects for the X and Y-axes.

## Loading and Extracting Chart Data

### CopyDataIn Method

In `C1Chart`, the `X`, `Y`, `Y1`, `Y2`, and `Y3` data array objects accept `Object` data types. Therefore, a variety of .NET type arrays may be entered into the `ChartDataArray` objects (except for `PointData`). The `CopyDataIn` method of the `ChartDataArray` takes an `Object` data type (which can be an array of various types) and loads it into the data array.

A good implementation would be to maintain a set of arrays for the data series and before the chart is to be drawn, update the `C1Chart`'s data array objects with the arrays of values. This allows full control over the data and provides control over batching the data values. The following code shows a brief example of how the arrays can be built and set to the chart's data array objects:

#### To write code in Visual Basic

##### Visual Basic

```
Dim xp(9) As Single
Dim yp(9) As Single
Dim i As Integer
For i = 0 To 9
    xp(i) = i
    yp(i) = i * i
Next i
With C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData.SeriesList(0)
    .X.CopyDataIn(xp)
    .Y.CopyDataIn(yp)
End With
```

#### To write code in C#

##### C#

```
float xp(10);
float yp(10);
int i;
for( i=0; i < 10; i++)
{
    xp[i] = i;
    yp[i] = i * i;
}
ChartDataSeries cds = c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData.SeriesList[0];
cds.X.CopyDataIn(xp);
cds.Y.CopyDataIn(yp);
```

### CopyDataOut Method

While the CopyDataIn method loads the array data into the ChartDataArray objects, the CopyDataOut method extracts data from the array objects. This method returns an Object data type, which contains the data array.

#### To write code in Visual Basic

##### Visual Basic

```
Dim xpo As Object
Dim cds As ChartDataSeries
cds = C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData.SeriesList(0)
xpo = cds.X.CopyDataOut()
```

#### To write code in C#

##### C#

```
object xpo;
ChartDataSeries cds;
cds = c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData.SeriesList[0];
xpo = cds.X.CopyDataOut();
```

To prevent having to then convert this Object to an array of the correct data type, the CopyDataOut method can also accept a parameter of System.Type, which causes it to then produce an array of the specified type. Because the actual data type is required for the parameter, the code must also include the GetType() function (typeof() in C#) to deliver the correct type. In the following example, the arrays are returned from the chart's ChartDataArray objects with the CopyDataOut method, but as data arrays of type Single:

#### To write code in Visual Basic

##### Visual Basic

```
Dim xpo As Single()
Dim ypo As Single()
With C1Chart1.ChartGroups.ChartGroupsCollection(0).ChartData.SeriesList(0)
xpo = .X.CopyDataOut(GetType(Single))
ypo = .Y.CopyDataOut(GetType(Single))
End With
```

#### To write code in C#

##### C#

```
ChartDataSeries cds;
cds = c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData.SeriesList[0];
float[] xpo = cds.X.CopyDataOut(typeof(float[]));
float[] ypo = cds.Y.CopyDataOut(typeof(float[]));
```

 **Note:** For C# users, the CopyDataOut implementation is slightly different. In C# array data types cannot be set to variables of type Object. The array can be set to the Object if the Object is explicitly cast to the correct data type.

#### To write code in Visual Basic

##### Visual Basic

```
Dim xpo() As Single
xpo = CType(C1Chart1.ChartGroups.ChartGroupsCollection(0), Single())_
.ChartData.SeriesList(0).X.CopyDataOut(GetType(Single))
```

#### To write code in C#

C#

```
float[] xpo;
xpo = (float[])c1Chart1.ChartGroups.ChartGroupsCollection[0].ChartData
.SeriesList[0].X.CopyDataOut(typeof(float[]));
```

## Changing Data Elements in the Data Arrays

The method of altering individual data elements in the data arrays involves the `ChartDataSeries` and the `ChartDataArray` objects.

Setting individual values through the .NET Properties window is made quite easy by the **ChartDataArray Collection Editor**. Data array editors exist for the `X`, `Y`, `Y1`, `Y2`, `Y3`, and `PointData` arrays. These editors can be accessed by clicking the **ellipsis** next to the appropriate data array in the **SeriesList Collection Editor** (which is available under the **ChartData** node of the **ChartGroupsCollection Editor**). Similar to the .NET collection editors, the **ChartDataArray Editor** has the array indices in a text box on the left and the array values on the right.

series 1	
X	Y
1	8
2	12
3	10
4	12
5	15



**Note:** The **ChartDataArray Collection Editor** also contains functionality for toggling the display between Date, DateTime, and Time display. If the values for the `ChartDataArray` are of `DateTime` type, then by clicking on the header above the value column, the values displayed will toggle between `DateTime`, `Date` and `Time` formats.

At run time the `ChartDataArray` elements are available like any normal array elements would be. Changing the values in the array objects is as easy as changing the values in an array:

### To write code in Visual Basic

Visual Basic

```
C1Chart1.ChartGroups.Group0.ChartData.SeriesList(0).X(4) = 4
```

### To write code in C#

C#

```
c1Chart1.ChartGroups.Group0.ChartData.SeriesList[0].X[4] = 4;
```

Retrieving the values from the `ChartDataArray` requires a similar process:

### To write code in Visual Basic

Visual Basic

```
Dim xval As Single
xval = C1Chart1.ChartGroups.Group0.ChartData.SeriesList(0).X(4)
```

### To write code in C#

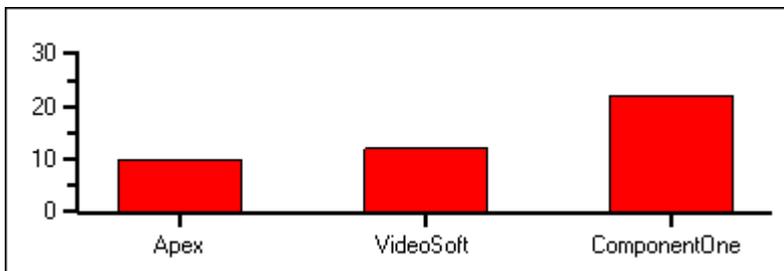
C#

```
float xval;  
xval = c1Chart1.ChartGroups.Group0.ChartData.SeriesList[0].X[4];
```

## Displaying String Values as Annotations

At times, it is very convenient to use string values as annotations along the X or Y axes, and numeric values along the other axis.

For example, suppose a data source contains 2 columns, one with a company name (string), and one with the number of ComponentOne products purchased by that company. To create a bar chart summarizing this data, one might create a series that uses the number of products on the Y axis, and then create an array with counting numbers and use this array for the X values. Then, create ValueLabels for the X axis, pairing each company name with a count number, thus producing a chart similar to the following:



Since this situation occurs often, [C1Chart](#) accepts [ChartDataArray](#) with strings, which automatically enumerates the array values and creates [ValueLabels](#) for the appropriate axis. Using this behavior, it is possible (and very convenient) to set the [DataField](#) properties for a [ChartDataSeries](#) to the string values directly and get the same results:

### To write code in Visual Basic

Visual Basic

```
C1Chart1.ChartGroups(0).ChartData.SeriesList(0).X.DataField = "CompanyNames"  
C1Chart1.ChartGroups(0).ChartData.SeriesList(0).Y.DataField = "UnitsPurchased"
```

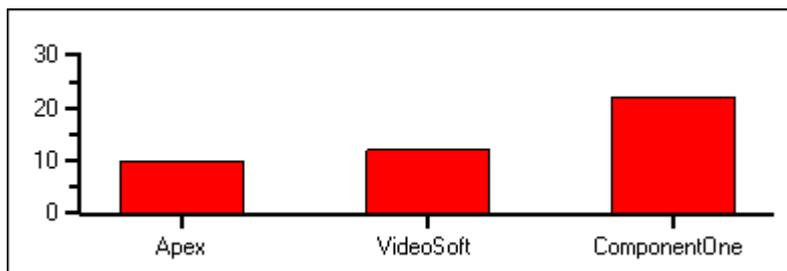
### To write code in C#

C#

```
c1Chart1.ChartGroups[0].ChartData.SeriesList[0].X.DataField = "CompanyNames";  
c1Chart1.ChartGroups[0].ChartData.SeriesList[0].Y.DataField = "UnitsPurchased";
```

## Mixing ChartDataArray Inputs

[C1Chart](#) allows some [ChartDataArrays](#) to be bound, and others to be set using the [AutoEnumerate](#) property of the [ChartDataSeries](#), or by using the [CopyDataIn\(\)](#) method of the [ChartDataArray](#) object. When data bound, the chart uses all of the bound data as its source of data for specified series data, and presents it in graphic form on the surface of the chart as directed by the series and other chart properties. When the [AutoEnumerate](#) property is set to **True** the x values of the [ChartDataArrays/series](#) will automatically set to their index value. While being able to set the data series to these different behaviors provides maximum flexibility, it also requires that excess default series are removed.



For example, the bar chart shown above contains only a single series (containing the number of ComponentOne products purchased by a company). So if data binding were used to provide input for C1Chart, then 3 of the 4 default chart series would need to be removed. Removing these series can be done at design time or runtime.

## Setting the X and Y Data Arrays Using Point Values

For charts that only use the X and Y-axes (XY-Plot, Bar, and so on), the [PointData](#) property can be a convenient shortcut. The [PointData](#) property of the [ChartDataSeries](#) object allows the entering of an array of [PointF](#) point values. This property is convenient for applications that use point data, but this array will only set the X and Y data arrays. This property cannot change the values in the data arrays for Y1, Y2, or Y3.

The following example builds a [PointF](#) array and loads it into a [ChartDataSeries](#):

### To write code in Visual Basic

#### Visual Basic

```
Dim ps() As PointF = _  
  
{  
    New PointF(30.0F, 20.0F), _  
    New PointF(60.0F, 24.0F), _  
    New PointF(90.0F, 42.0F), _  
    New PointF(120.0F, 13.0F), _  
    New PointF(150.0F, 10.0F) _  
}  
Dim s As New ChartDataSeries()  
C1Chart1.ChartGroups.Group1.ChartData.SeriesList.Add(s)  
s.PointData.CopyDataIn(ps)
```

### To write code in C#

#### C#

```
PointF [] ps =  
{  
    new PointF(30.0F, 20.0F),  
    new PointF(60.0F, 24.0F),  
    new PointF(90.0F, 42.0F),  
    new PointF(120.0F, 13.0F),  
    new PointF(150.0F, 10.0F)  
};  
ChartDataSeries s = new ChartDataSeries();  
c1Chart1.ChartGroups.Group1.ChartData.SeriesList.Add(s);  
s.PointData.CopyDataIn(ps);
```

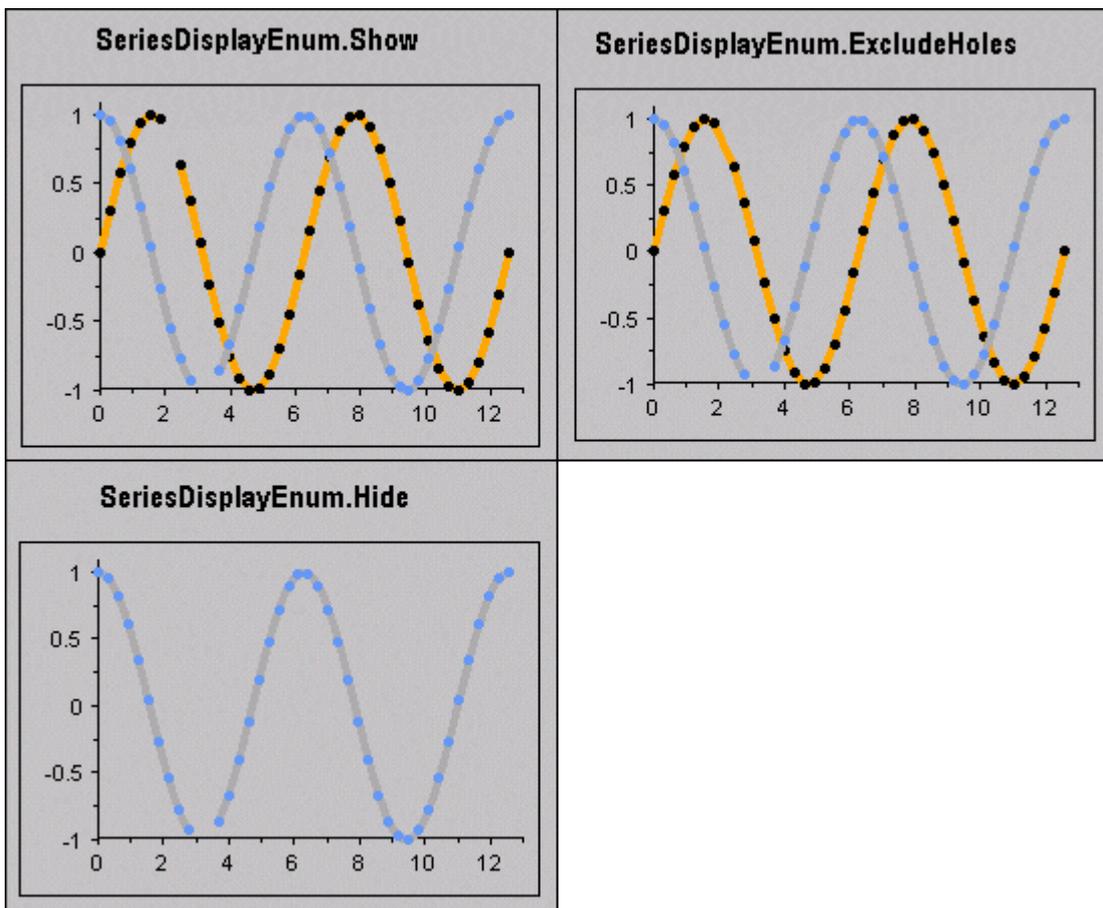
## Customizing the ChartDataSeries

This section describes how to show, hide, and exclude holes from the ChartDataSeries in the ChartArea as well as how to exclude series from a particular legend in C1Chart.

### Showing, Excluding, or Hiding a Series

Suppose there were hundreds of series that needed to be displayed in the chart. Since the chart can only be so large, the displayed series must be managed. The `Display` property of the `ChartDataSeries` provides this capability. The `Display` property accepts a `SeriesDisplayEnum` enumerated type. Setting this property to its different values allows a series to show, be hidden, or be excluded:

Value	Description
<code>SeriesDisplayEnum.Show</code>	The series is displayed in the ChartArea. This is the default setting.
<code>SeriesDisplayEnum.Hide</code>	The series is not displayed in the ChartArea, but the ChartArea (Max and Min) is not altered to account for the lost series.
<code>SeriesDisplayEnum.Exclude</code>	The series is not displayed in the ChartArea, but the ChartArea is altered to account for the lost series.
<code>SeriesDisplayEnum.ExcludeHoles</code>	The series is displayed, but the Data Hole values are ignored.



### Excluding a Series from the Legend

At times it is convenient to plot shapes, lines and curves that do not represent data, but rather data regions of interest, or error bands, and so on. In these instances, additional series may be added to the chart, but should be excluded from the legend. Each `ChartDataSeries` contains a Boolean property called **LegendEntry**. If set to **True**, as it is by default, the series will be listed in the legend. If **False**, however, the series will be charted, but not listed in the legend.

## Charting Irregular Data

Datasets, especially when they are dynamically created, can sometimes have irregular attributes that make it difficult for a charting control to handle. Thus, much time can be spent accounting for these inconsistencies before the data reaches the chart. Although `C1Chart` cannot modify these values automatically, it does accept irregular data and deals with it in a logical and consistent manner.

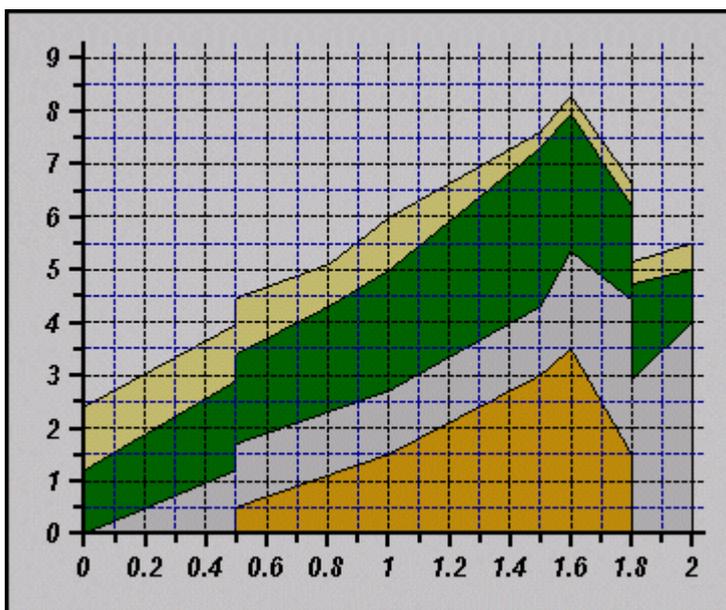
## Irregular Stacked Chart Data

Stacked charts are charts that have one or more series stacked upon a base series. All of the Y values for the chart are taken as distances from the previous series' Y value. Stacked charts are a simple issue when data is loaded in an array format where the X values all match and all the series are stacked on top quite neatly.

`C1Chart` provides more freedom by allowing data to be in a General Layout. General Layout means that not all series must have identical X values. While this freedom allows for charting of more non-standard sets of data, it poses a problem when using stacked charts.

## Irregular X-Axis Data

Problems arise with the stacked chart if the first series in the data set does not contain points that match up to the Min and the Max points of the other series. If the first series of the chart does not contain the amount of X values as successive series, then these series no longer have a definite starting point as in the image below. `C1Chart` deals with this irregularity in a logical manner. The data where the Y values for the first series do not exist are taken to be 0. Then each successive series is added onto this creating a discontinuity, best demonstrated by the following Stacked Area chart:



## Interpolating Y Value Data

Problems can also arise in a stacked chart if the X values of a successive series do not coincide with the first series. For instance, the first series of the chart could have x values at 2, 4, 6, and 8, while the second series has values at 3, 5, 7, and 9. A problem arises when the chart tries to discern where to place the first point of the second series. The point value that must be added to the Y value of the second series is somewhere between the first two points of the first series and is not explicitly part of the dataset.

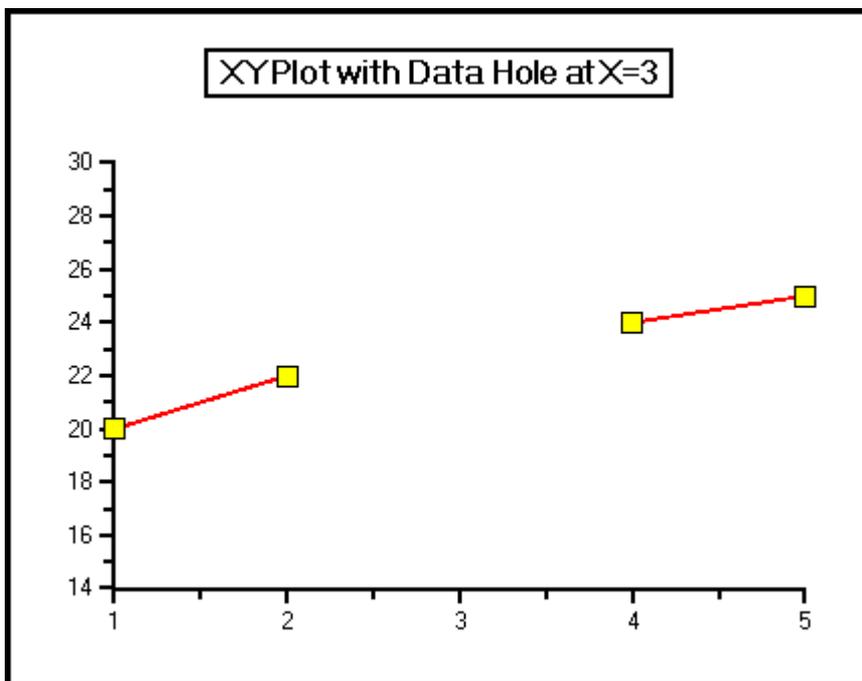
Instead of not displaying the data set, [C1Chart](#) deals with this irregularity in a logical manner. It interpolates a virtual point in between the first and second point of the first series based on the slope of the line. This interpolated value is added to the second series' Y value to find the new point for the stacked chart.

## Specifying Data Holes

When gathered data is charted, it is sometimes necessary to visually provide an indication in the chart that data for a particular value is not available. For example, you can indicate that information is missing. A Data Hole is a break in data that are being charted, and is used in the chart to visually indicate that data is missing.

In [C1Chart](#), holes are specified by setting the data point to be excluded to a specific, otherwise unused value in the data arrays. The [Hole](#) property of the ChartData object specifies the value to be used. For instance, assuming that the Hole value is set to  $-1000$ , and that data is unavailable for the third point of a series, a break or a hole can be introduced into the series plot by setting the Y value of the series point to  $-1000$ .

The default hole value is `Single.MaxValue` or `float.MaxValue` for Visual Basic and C#, respectively. Here is an example of a data hole in a **X-Y Plot** chart:

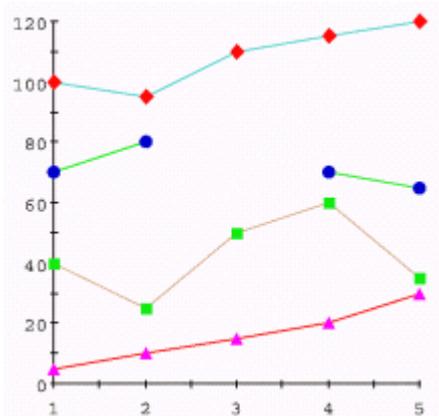


## Ignoring Data Holes

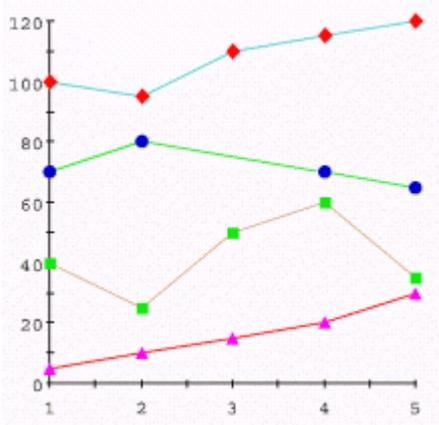
When data is obtained in arrays of specific length, programming is often simplified when all the arrays can be handled uniformly, even if some of the data points are missing and it is not desired to indicate the missing data. In this case, Data Holes can still be used as values for the missing data and [C1Chart](#) can simply ignore them entirely, as if those data array elements containing the Data Hole did not exist at all.

The **Display** property of the ChartDataSeries sets the display properties for a series. This property accepts a SeriesDisplayEnum enumeration. If the **Display** property is set to **SeriesDisplayEnum.Show**, then the Data Hole will be excluded from the plot as in the image above. If this property is set to SeriesDisplayEnum.ExcludeHoles the series will be displayed, but the holes will be ignored. This means that if an XY-Plot is being displayed, the line will continue from the point previous to the Data Hole directly to the point after the Data Hole as in the image below. Notice that although the plot line continues past the Data Hole, the Data Hole point is not charted.

## SeriesDisplayEnum.Show



## SeriesDisplayEnum.ExcludeHoles



## Plotting Functions

**C1Chart** has a built-in engine for plotting functions. As there are various types of functions used for different applications. **C1Chart** provides the various types of functions needed to create many applications.

There are two types of supported functions:

1. One-variable explicit functions
  - One-variable explicit functions are written as  $y=f(x)$  (see the [YFunction](#) class).
  - A few examples include: rational, linear, polynomial, quadratic, logarithmic, and exponential functions.
  - Commonly used by scientists and engineers, these functions can be used in many different types of finance, forecasts, performance measurement applications, and so on.
2. Parametric functions
  - The function is defined by a pair of equations, such as  $y=f_1(t)$  and  $x=f_2(t)$  where  $t$  is the variable/coordinate for functions  $f_1$  and  $f_2$ .
  - Parametric functions are special types of function because the  $x$  and  $y$  coordinates are defined by individual functions of a separate variable.
  - They are used to represent various situations in mathematics and engineering, from heat transfer, fluid

mechanics, electromagnetic theory, planetary motion and aspects of the theory of relativity, to name a few.

- For more information about the parametric function (see the [ParametricFunction](#) class).

In **C1Chart**, a code string can be used to calculate Y Functions and Parametric Functions.

To use a code string to calculate Y Function and Parametric Function you must implement one of the following:

- An interpretive string containing C# or VB source code
- Events
- A class which supports the `C1.Win.C1Chart.ISimpleFunction` interface

**C1Chart** includes a **FunctionBase Collection Editor** that is accessible through C1Chart's properties at design time. The **FunctionBase Collection Editor** consists of a windows form which conveniently allows the users to edit/create their functions. Using the editor, the user can add/remove one or many multiple functions, select what type of function (either a **Y function** or **Parametric function**), specify the code type and code language, just to name a few. For more information on the **FunctionBase Collection Editor**, see [FunctionBase Collection Editor](#).



**Note:** For a complete sample that demonstrates how to create Y functions and Parametric functions, see the sample, **FExplorer** located on <http://our.componentone.com/samples/?action=viewcontrol&control=7&platform=1>.

## Using a Code String to Define a Function

When an interpretive code string is used to define a function of a function class (`YFunction` or `ParametricFunction`), the string is compiled and the resulting code is dynamically included into the application. Execution speed will be the same as any other compiled code.

This provides a number of advantages which among others include:

- Built-in functions of C# or VB can be used when defining the function.
- Functions can be built by simple manipulation of strings.
- Functions are saved as part of the chart layout using the `SaveChartToFile()` or `SaveChartToString()` methods, and can be loaded directly into other projects as part of the data.

There are however, disadvantages as well, which among others include:

- Time delay when first assigning or changing code text due to pre-compilation.
- Types of source code limited to C# or VB.

For simple, one-variable explicit functions, the `YFunction` class object is used. This object has one code property, `CodeText`. For `YFunction` objects, the independent variable is always assumed to be "x".

For parametric functions, a pair of equations must be defined using the `ParametricFunction` class object. This object has two properties, one for each coordinate. The properties, `CodeTextX` and `CodeTextY` accept code in which the independent variable is always assumed to be "t".

For both class objects, the values of the independent variable are assumed to be uniformly distributed from the minimum to the maximum value of the independent variable, using the specified number of data points specified by the `PlotNumPoints` property.

Depending on complexity of function, three types of code are supported for `CodeText`, `CodeTextX` and `CodeTextY`. The code types include **Formula**, **Method** and **Unit**, each of which is described in the following section.

## Formula Code Types

For Formula code types, the [CodeText](#) property of a function class must contain code that calculates the value of a function using a parameter. The code should fit into a single line and represents the right hand side of an equation.

#### To write code in Visual Basic

##### Visual Basic

```
Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction()
yf.CodeType = C1.Win.C1Chart.FunctionCodeTypeEnum.Formula
yf.CodeLanguage = C1.Win.C1Chart.FunctionCodeLanguageEnum.VB
yf.CodeText = "x*x"
yf.MinX = -5
yf.MaxX = 5
yf.LineStyle.Color = Color.Red
yf.LineStyle.Thickness = 3
C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
```

#### To write code in C#

##### C#

```
C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction();
yf.CodeType = C1.Win.C1Chart.FunctionCodeTypeEnum.Formula;
yf.CodeText = "x*x";
yf.MinX = -5;
yf.MaxX = 5;
yf.LineStyle.Color = Color.Red;
yf.LineStyle.Thickness = 3;
c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add(yf);
```

## Method Code Types

For Method code types, the [CodeText](#) property of a function class must contain the body of a method that calculates the value of function and explicitly returns the value. The expected return value is of type Double. Note that VB syntax requires the inclusion of `vbNewLines` at the end of each statement in the code.

#### To write code in Visual Basic

##### Visual Basic

```
Dim code As String = _
    "Dim x2 As Double = x*x" & vbNewLine & _
    "if x<0 then " & vbNewLine & _
    "    return x" & vbNewLine & _
    "else" & vbNewLine & _
    "    return 0.5*x2" & vbNewLine & _
    " End If"
Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction()
yf.CodeType = C1.Win.C1Chart.FunctionCodeTypeEnum.Method
yf.CodeLanguage = C1.Win.C1Chart.FunctionCodeLanguageEnum.VB
yf.CodeText = code
yf.MinX = -5
yf.MaxX = 5
yf.LineStyle.Color = Color.Blue
```

```
yf.LineStyle.Thickness = 3
C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
```

### To write code in C#

```
C#
string code = "double x2 = x*x;" + "if( x<0)" + " return x;" + "else" + " return
0.5*x2;";
C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction();
yf.CodeType = C1.Win.C1Chart.FunctionCodeTypeEnum.Method;
yf.CodeText = code;
yf.MinX = -5;
yf.MaxX = 5;
yf.LineStyle.Color = Color.Blue;
yf.LineStyle.Thickness = 2;
c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add(yf);
```

## Unit Code Types

For Unit code types, the [CodeText](#) property of a function must contain the full compile unit text. The unit must include the class "Calculator" in the UserFunction namespace, and must implement the [ISimpleFunction](#) interface.

### To write code in Visual Basic

```
Visual Basic
Dim code As String = _
"Namespace UserFunction" & vbNewLine & _
" Class Calculator" & vbNewLine & _
" Implements ISimpleFunction" & vbNewLine & _
" Public Function Calculate(x As Double) As Double _" & vbNewLine & _
" Implements ISimpleFunction.Calculate" & vbNewLine & _
" Dim x2 As Double = x*x" & vbNewLine & _
" if( x<0)" & vbNewLine & _
" return -x" & vbNewLine & _
" else" & vbNewLine & _
" return -0.5*x2" & vbNewLine & _
" End If" & vbNewLine & _
" End Function" & vbNewLine & _
" End Class" & vbNewLine & _
"End Namespace"

Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction()

yf.CodeType = C1.Win.C1Chart.FunctionCodeTypeEnum.Unit
yf.CodeLanguage = C1.Win.C1Chart.FunctionCodeLanguageEnum.VB
yf.CodeText = code

yf.MinX = -5
yf.MaxX = 5
yf.LineStyle.Color = Color.Green
```

```
yf.LineStyle.Thickness = 3

C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
```

### To write code in C#

```
C#

string code =
"namespace UserFunction" +
"{ " +
"  class Calculator : ISimpleFunction" +
"  { " +
"    public double Calculate(double x)" +
"    { " +
"      double x2 = x*x;" +
"      if( x<0)" +
"        return -x;" +
"      else" +
"        return -0.5*x2;" +
"    }" +
"  }" +
"}";

C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction();

yf.CodeType = C1.Win.C1Chart.FunctionCodeTypeEnum.Unit;
yf.CodeText = code;

yf.MinX = -5;
yf.MaxX = 5;
yf.LineStyle.Color = Color.Green;
yf.LineStyle.Thickness = 2;

c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add( yf);
```

## Calculating the Value for Functions

The following subtopics below explain how to calculate an appropriate function value using either the CalculateX and CalculateY events or the Calculate method of the ISimpleFunction interface.

## Calculating the Function Value Using Events

Instead of adding source code to the [CodeText](#), [CodeTextX](#) or [CodeTextY](#) properties, an event delegate can be used to specify an event method to calculate the appropriate function value.

When using event delegates, the function value is calculated in event CalculateY for the YFunction class objects. For ParametricFunction class objects, two events, [CalculateX](#) and [CalculateY](#) must be set, one for each calculated coordinate. Non-null event delegates indicate that the event should be used to calculate the corresponding function value, even if the corresponding **CodeText** property is also specified in either the YFunction or ParametricFunction class objects. In order to use events, the programmer must provide the appropriate event handlers.

**To write code in Visual Basic****Visual Basic**

```
Private Sub Button_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button.Click
    Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction()
    AddHandler yf.CalculateY, AddressOf Function_Calculate
    yf.MinX = -5
    yf.MaxX = 5
    yf.LineStyle.Color = Color.DarkBlue
    yf.LineStyle.Thickness = 3
    C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
End Sub

Private Sub Function_Calculate(ByVal sender As Object, _
    ByVal e As C1.Win.C1Chart.CalculateFunctionEventArgs)
    e.Result = e.Parameter * e.Parameter * e.Parameter ' y = x*x*x
End Sub
```

**To write code in C#****C#**

```
private void button_Click(object sender, System.EventArgs e)
{
    C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction();
    yf.MinX = -5;
    yf.MaxX = 5;
    yf.LineStyle.Color = Color.DarkBlue;
    yf.LineStyle.Thickness = 2;
    yf.CalculateY += new C1.Win.C1Chart.CalculateFunctionEventHandler(
        Function_Calculate);
    c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add( yf);
}

private void Function_Calculate(object sender,
    C1.Win.C1Chart.CalculateFunctionEventArgs e)
{
    e.Result = e.Parameter * e.Parameter * e.Parameter; // y = x*x*x
}
```

## Calculating the Function Value Using the Calculate Method

As an alternative to the use of either a code string or events, the programmer may specify an instance of an object that implements `C1.Win.C1Chart.ISimpleFunction` interface. Such an object must inherit from the interface and implement a public function name, `Calculate`. The method, `Calculate` takes the independent variable (`Double`) as a parameter and returns the dependent variable (`Double`).

For the `YFunction` class objects, the `CustomFunction` property must be set to the `ISimpleFunction` implementation object. For the `ParametricFunction` class object, the `CustomFunctionX` and `CustomFunctionY` properties must be set to

appropriate objects implementing the `ISimpleFunction` interface.

#### To write code in Visual Basic

##### Visual Basic

```
Private Sub Button_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button.Click
    Dim yf As C1.Win.C1Chart.YFunction = New C1.Win.C1Chart.YFunction()
    yf.CustomFunction = New CustomFunction()
    yf.MinX = -5
    yf.MaxX = 5
    yf.LineStyle.Color = Color.DarkGreen
    yf.LineStyle.Thickness = 3
    C1Chart1.ChartGroups(0).ChartData.FunctionsList.Add(yf)
End Sub
Public Class CustomFunction
    Implements C1.Win.C1Chart.ISimpleFunction
    Public Function Calculate(ByVal x As Double) As Double _
        Implements C1.Win.C1Chart.ISimpleFunction.Calculate
        Return -x * x * x      ' y = - x*x*x
    End Function
End Class
```

#### To write code in C#

##### C#

```
private void button_Click(object sender, System.EventArgs e)
{
    C1.Win.C1Chart.YFunction yf = new C1.Win.C1Chart.YFunction();
    yf.MinX = -5;
    yf.MaxX = 5;
    yf.LineStyle.Color = Color.DarkGreen;
    yf.LineStyle.Thickness = 2;
    yf.CustomFunction = new CustomFunction();
    c1Chart1.ChartGroups[0].ChartData.FunctionsList.Add( yf);
}
class CustomFunction : C1.Win.C1Chart.ISimpleFunction
{
    public double Calculate( double x)
    {
        return -x*x*x; // y = -x*x*x
    }
}
```

## Working with TrendLines

The trend lines supported by chart with `TrendLine` objects can be divided into two groups, including regression and non-regression. In 2D charts, trend lines are typically used in X-Y line, bar, or scatter charts.

Non-regression trendlines are `MovingAverage`, `Average`, `Minimum`, and `Maximum`. `Moving Average` trendline is the average over the specified time.

Regression trend lines are **polynomial, exponent, logarithmic, power** and **Fourier functions** that approximate the data which the functions trend.

## Creating TrendLines

### Creating TrendLines at Design Time

TrendLines can be created at design time using the **TrendLine Collection Editor**. Using the collection editor, you may add, modify and remove trend lines. For more information on the **TrendLine Collection Editor**, see [TrendLine Collection Editor](#).

### Creating TrendLines Programmatically

To create a TrendLine programmatically, create instance of TrendLine object and set its properties. The TrendLine can be created using its constructor or AddNewTrendLine.TrendLinesCollection() method of TrendLinesCollection.

The following code adds a red TrendLine to ChartGroup(0) of the Chart:

#### To write code in Visual Basic

##### Visual Basic

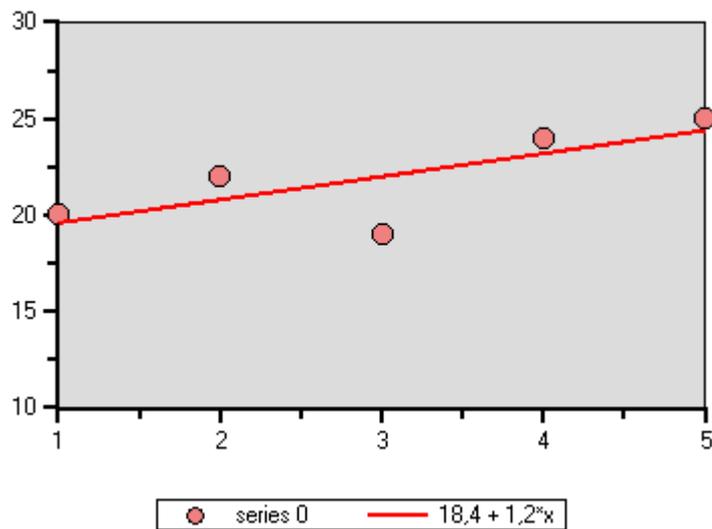
```
' create trend line
Dim tl As Cl.Win.C1Chart.TrendLine = New Cl.Win.C1Chart.TrendLine()
' first data series
tl.SeriesIndex = 0
' set line color and thickness
tl.LineStyle.Color = Color.Red
tl.LineStyle.Thickness = 2
' show formula in legend
tl.Text = "{#FORMULA}"
' add trend to the chart
c1Chart1.ChartGroups(0).ChartData.TrendsList.Add(tl)
```

#### To write code in C#

##### C#

```
// create trend line
Cl.Win.C1Chart.TrendLine tl = new Cl.Win.C1Chart.TrendLine();
// first data series
tl.SeriesIndex = 0;
// set line color and thickness
tl.LineStyle.Color = Color.Red;
tl.LineStyle.Thickness = 2;
// show formula in legend
tl.Text = "{#FORMULA}";
// add trend to the chart
c1Chart1.ChartGroups[0].ChartData.TrendsList.Add( tl);
```

This topic illustrates the following:



The red trend line connects the first and last data points.

## Regression TrendLines

For regression TrendLines there are two relevant class objects including `RegressionOptions` and `RegressionStatistics`. These objects describe the desired form of the trendline, and sum up the resulting trendline through statistics.

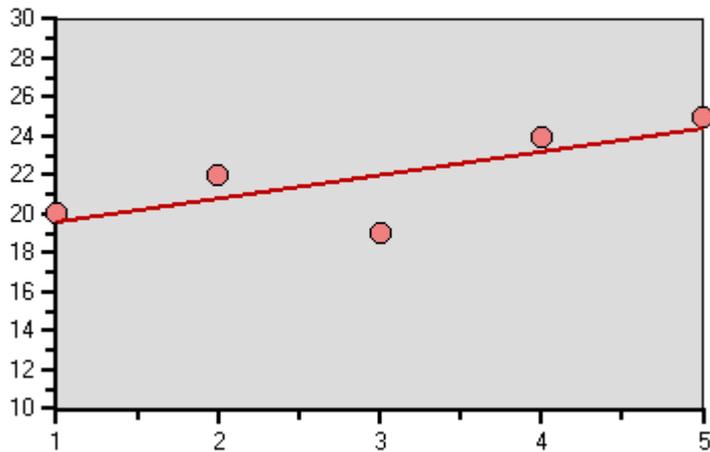
## Regression Options

The `RegressionOption` object allows specification of the regression model. The `NumTerms` property defines the number of coefficients to be used by a regression, and is relevant only for regressions with variable number of terms (polynomial and Fourier).

For a polynomial regression, the number of terms is one more than the order of the resulting polynomial. The maximum number of terms for a polynomial is the number of data points, and the minimum number of terms is two (linear).

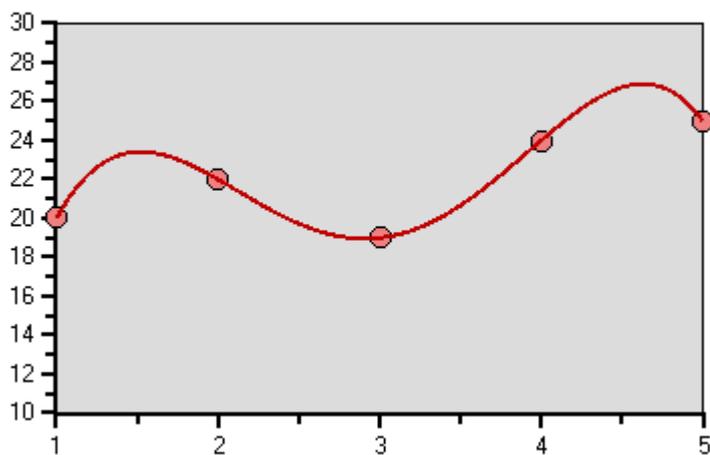
The following image illustrates a Linear Regression since its number of terms is two:

RegressionOptions.NumTerms = 2



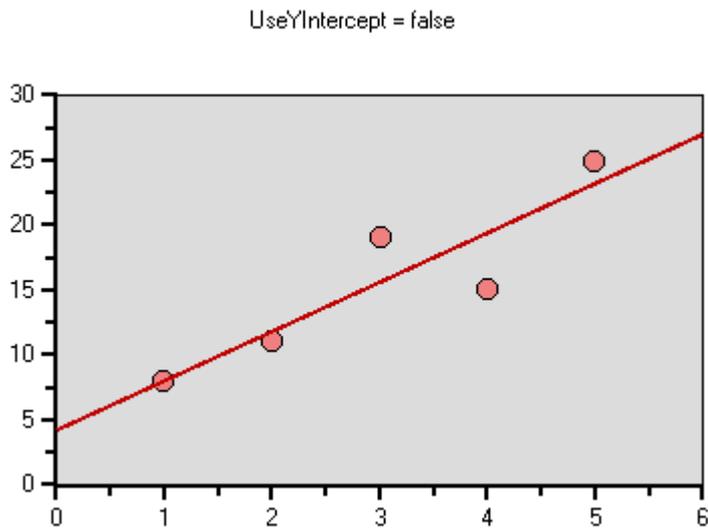
The following image illustrates a Polynomial regression since its number of terms is more than two.

RegressionOptions.NumTerms = 5

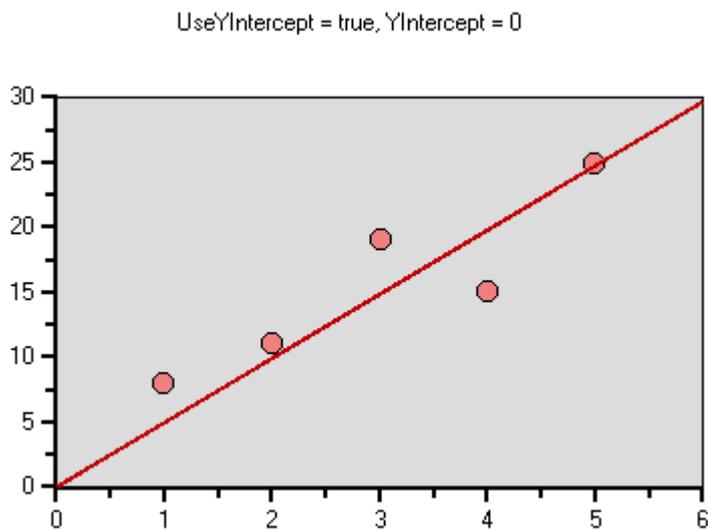


[UseYIntercept](#) property controls whether the first term of polynomial regression is fixed, when [UseYIntercept](#) is **True** the trend line intercepts the line  $x=0$  at  $y$ -coordinate that is defined by [YIntercept](#) property.

The following image illustrates a non-fixed Linear Regression since the [UseYIntercept](#) property is False.



The following image illustrates a fixed Linear Regression since the trendline is set to intercept the line  $x=0$  at  $y$ -coordinate that is defined by the **RegressionOptions.YIntercept** property:



## Regression Statistics

The **RegressionStatistics** property returns a **RegressionStatistics** object that contains the statistical results of a regression model. When a trend line is not regressive or the solution of a regression does not exist for current data, the **RegressionStatistics** property returns Nothing (VB) or null (C#).

The following table defines the values used in statistical equations:

Value	Definition
n	Number of points.
nt	Number of regression coefficients.
x values	$X_1, X_2, \dots, X_n$

y values	$y_1, y_2, \dots, y_n$
mean y value	$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$
Fitted y values	$\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n$

The following table displays the name, description, and formula for each member in the RegressionStatistics object:

Name	Description	Formula	Formula(UseYIntercept=true)
Ssr	The sum of squares due to regression	$SSR = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$	$SSR = \sum_{i=1}^n (\hat{y}_i * y_i)^2$
Sse	The sum of squares due to error	$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$	$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$
Rsq	The coefficient of determination (R-squared)	$r^2 = \frac{SSR}{SSR + SSE}$	$r^2 = \frac{SSR}{SSR + SSE}$
DF	The degrees of freedom	$df = n - nt$	$df = n - nt - 1$
F	The F-observed(F-statistic) value	$F = \frac{SSR * df}{SSE * (np - df - 1)}$	$F = \frac{SSR * df}{SSE * (np - df)}$

## Custom TrendLine

To implement a custom trendline a class implementing [ICustomTrendLine](#) interface must be created. The instance of this class is assigned to [CustomTrendLine](#) property of a TrendLine object. In this case all points of trendline must be fully defined by the class and [TrendLineType](#) property setting is not relevant. The following sample code implements a custom TrendLine that corresponds to data limits.

### To write code in Visual Basic

#### Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    _
    Handles Button1.Click
    ' create trend line
    Dim tl As C1.Win.C1Chart.TrendLine = _
        c1Chart1.ChartGroups(0).ChartData.TrendsList.AddNewTrendLine()
    ' setup line properties
    tl.LineStyle.Thickness = 3
    tl.LineStyle.Pattern = C1.Win.C1Chart.LinePatternEnum.Dash
    ' set custom trend line
    tl.CustomTrendLine = New CustomTrendLine()
End Sub

Public Class CustomTrendLine_ Implements C1.Win.C1Chart.ICustomTrendLine
    Private _x() As Double
    Private _y() As Double

```

```

    Public Sub Calculate(ByVal tl As Cl.Win.C1Chart.TrendLine, ByVal x() As
Double, _ ByVal y() As Double) Implements Cl.Win.C1Chart.ICustomTrendLine.Calculate
        If x Is Nothing Or x.Length = 0 Or y Is Nothing Or y.Length = 0 Then
            _x = Nothing
            _y = Nothing
            Return
        End If

        ' find min and max
        Dim xmin As Double = x(0), xmax = x(0)
        Dim ymin As Double = y(0), ymax = y(0)
        Dim i As Integer
        For i = 1 To x.Length - 1
            If x(i) < xmin Then
                xmin = x(i)
            ElseIf x(i) > xmax Then
                xmax = x(i)
            End If
            If y(i) < ymin Then
                ymin = y(i)
            ElseIf y(i) > ymax Then
                ymax = y(i)
            End If
        Next

        ' rectangle around data points
        _x = New Double(4) {}
        _y = New Double(4) {}
        _x(0) = xmin
        _y(0) = ymin
        _x(4) = _x(0)
        _y(4) = _y(0)
        _x(2) = xmax
        _y(2) = ymax
        _x(1) = _x(0)
        _y(1) = _y(2)
        _x(3) = _x(2)
        _y(3) = _y(0)

    End Sub

    Public Function GetXValues() As Double() _ Implements
Cl.Win.C1Chart.ICustomTrendLine.GetXValues
        Return _x
    End Function

    Public Function GetYValues() As Double() _ Implements
Cl.Win.C1Chart.ICustomTrendLine.GetYValues
        Return _y
    End Function

```

```

' don't use it just return something
Public Function GetY(ByVal x As Double) As Double _ Implements
C1.Win.C1Chart.ICustomTrendLine.GetY
    Return 0
End Function

Public ReadOnly Property Text() As String _ Implements
C1.Win.C1Chart.ICustomTrendLine.Text
    Get
        Return "Custom trend"
    End Get
End Property
End Class

```

**To write code in C#**

C#

```

private void button1_Click(object sender, System.EventArgs e)
{
    // create trend line
    C1.Win.C1Chart.TrendLine tl =
c1Chart1.ChartGroups[0].ChartData.TrendsList.AddNewTrendLine();
    // setup line properties
    tl.LineStyle.Color = Color.DarkRed;
    tl.LineStyle.Thickness = 3;
    tl.LineStyle.Pattern = C1.Win.C1Chart.LinePatternEnum.Dash;
    // set custom trend line
    tl.CustomTrendLine = new CustomTrendLine();
}

public class CustomTrendLine : C1.Win.C1Chart.ICustomTrendLine
{
    private double[] _x;
    private double[] _y;
    public void Calculate( C1.Win.C1Chart.TrendLine tl, double[] x, double [] y)

    {
        if( x==null || x.Length==0 || y==null || y.Length==0)
        {
            _x = null; _y = null;
            return;
        }
        // find min and max
        double xmin = x[0], xmax = x[0];
        double ymin = y[0], ymax = y[0];
        for( int i=1; i<x.Length; i++)
        {
            if( x[i] < xmin)

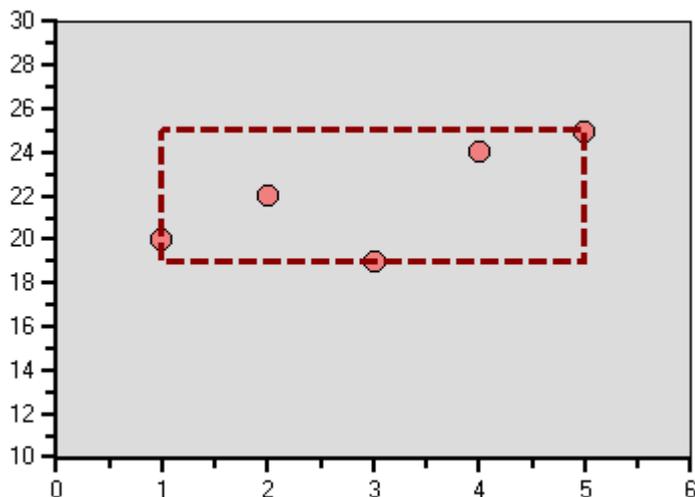
```

```
        xmin = x[i];
    else if( x[i] > xmax)
        xmax = x[i];
    if( y[i] < ymin)
        ymin = y[i];
    else if( y[i] > ymax)
        ymax = y[i];
}

// rectangle around data points
_x = new double[5];
_y = new double[5];
_x[0] = xmin; _y[0] = ymin;
_x[4] = _x[0]; _y[4] = _y[0];
_x[2] = xmax; _y[2] = ymax;
_x[1] = _x[0]; _y[1] = _y[2];
_x[3] = _x[2]; _y[3] = _y[0];
}

public double[] GetXValues() { return _x;}
public double[] GetYValues() { return _y;}
// don't use it just return something
public double GetY( double x) { return 0;}
public string Text { get{ return "Custom trend";}}
}
```

This topic illustrates the following:



A custom red dashed trend line is created around the data points.

## Working with PointStyles

PointStyles provide a mechanism to mark specific data points with different visual attributes than other points of the same data series. PointStyles are contained by the [PointStylesCollection](#). A PointStyle can be applied to an explicit

data point that is selected by series and point indexes or to a data point that meets to specific conditions -such as: x maximum of series, y maximum of all data, and so on. The [C1Chart](#) programmer can also specify custom conditions using the `Select` event. A [PointStyle](#) contains the same set of visual properties as `ChartDataSeries` `LineStyle`, `SymbolStyle` and `Offset`. These properties describe the appearance of a data point that meets the `PointStyle` condition. A `PointStyle` can be shown as legend item.

## Creating PointStyles

`PointStyles` can easily be created at design time through the **PointStyle Collection Editor** or programmatically through the `PointStyle` object.

### Creating PointStyles at Design Time

`PointStyles` can be created at design time using the **PointStyle Collection Editor**. Using the collection editor, you may add, modify and remove point styles. For more information on the properties in the **PointStyle Collection Editor**, see [PointStyle Collection Editor](#).

### Creating PointStyles Programmatically

The following code creates an instance of `PointStyle` object and sets its **LineStyle** and **SymbolStyle** properties:

#### To write code in Visual Basic

##### Visual Basic

```
Dim styles As C1.Win.C1Chart.PointStylesCollection = _
    c1Chart1.ChartGroups(0).ChartData.PointStylesList

' min value
Dim psmin As C1.Win.C1Chart.PointStyle = styles.AddNewPointStyle()
psmin.LineStyle.Pattern = C1.Win.C1Chart.LinePatternEnum.None
psmin.SymbolStyle.Color = Color.MistyRose
psmin.SymbolStyle.OutlineColor = Color.Blue
psmin.SymbolStyle.OutlineWidth = 2
psmin.SymbolStyle.Size = 10
psmin.Selection = C1.Win.C1Chart.PointStyleSelectionEnum.SeriesMinY

' show in legend
psmin.Label = "Y Min"
psmin.LegendEntry = True

' max value
Dim psmax As C1.Win.C1Chart.PointStyle = styles.AddNewPointStyle()
psmax.LineStyle.Pattern = C1.Win.C1Chart.LinePatternEnum.None
psmax.SymbolStyle.Color = Color.MistyRose
psmax.SymbolStyle.OutlineColor = Color.Red
psmax.SymbolStyle.OutlineWidth = 2
psmax.SymbolStyle.Size = 10
psmax.Selection = C1.Win.C1Chart.PointStyleSelectionEnum.SeriesMaxY

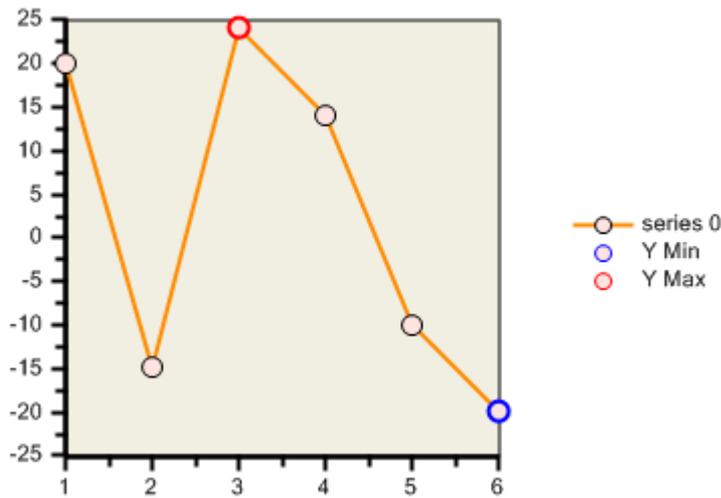
' show in legend
psmax.Label = "Y Max"
```

```
psmax.LegendEntry = True  
c1Chart1.Legend.Visible = True
```

**To write code in C#**

```
C#  
  
C1.Win.C1Chart.PointStylesCollection styles =  
c1Chart1.ChartGroups[0].ChartData.PointStylesList;  
  
// min value  
C1.Win.C1Chart.PointStyle psmin = styles.AddNewPointStyle();  
psmin.LineStyle.Pattern = C1.Win.C1Chart.LinePatternEnum.None;  
psmin.SymbolStyle.Color = Color.MistyRose;  
psmin.SymbolStyle.OutlineColor = Color.Blue;  
psmin.SymbolStyle.OutlineWidth = 2;  
psmin.SymbolStyle.Size = 10;  
psmin.Selection = C1.Win.C1Chart.PointStyleSelectionEnum.SeriesMinY;  
  
// show in legend  
psmin.Label = "Y Min";  
psmin.LegendEntry = true;  
  
// max value  
C1.Win.C1Chart.PointStyle psmax = styles.AddNewPointStyle();  
psmax.LineStyle.Pattern = C1.Win.C1Chart.LinePatternEnum.None;  
psmax.SymbolStyle.Color = Color.MistyRose;  
psmax.SymbolStyle.OutlineColor = Color.Red;  
psmax.SymbolStyle.OutlineWidth = 2;  
psmax.SymbolStyle.Size = 10;  
psmax.Selection = C1.Win.C1Chart.PointStyleSelectionEnum.SeriesMaxY;  
  
// show in legend  
psmax.Label = "Y Max";  
psmax.LegendEntry = true;  
c1Chart1.Legend.Visible = true;
```

PointStyles are added to the data points for the first series of C1Chart. Two special point styles are also added to represent the minimum and maximum value points for the Y-axis.



## Creating Custom PointStyles

To define custom point style condition the user must set [Selection](#) property of [PointStyle](#) to **PointStyleSelectionEnum.Custom** and provide event handler for the [Select](#) event. The following code creates a custom point style:

### To write code in Visual Basic

#### Visual Basic

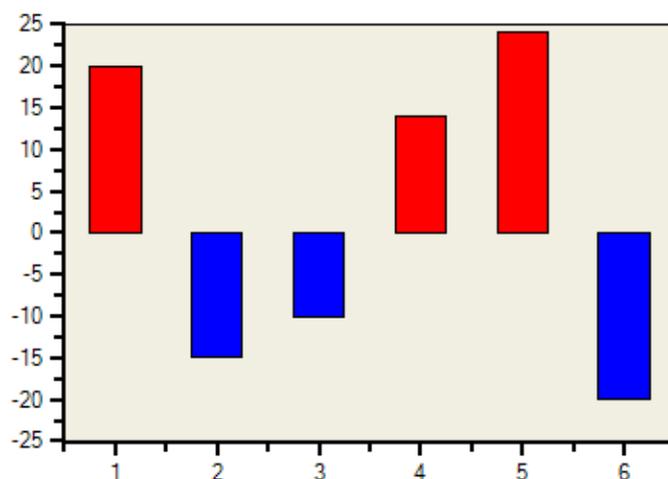
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles Button1.Click
    ' create point style
    Dim ps As Cl.Win.ClChart.PointStyle = New Cl.Win.ClChart.PointStyle()
    ' custom point style
    ps.Selection = Cl.Win.ClChart.PointStyleSelectionEnum.Custom
    AddHandler ps.Select, AddressOf PS_Select
    ' add point style
    clChart1.ChartGroups(0).ChartData.PointStylesList.Add(ps)
End Sub

Private Sub PS_Select(ByVal sender As Object, ByVal e As _
Cl.Win.ClChart.PointStyleSelectEventArgs)
    Dim ps As Cl.Win.ClChart.PointStyle = CType(sender, Cl.Win.ClChart.PointStyle)
    ' set visual appearance depending on y value
    Dim ds As Cl.Win.ClChart.ChartDataSeries = _
clChart1.ChartGroups(0).ChartData(e.SeriesIndex)
    Dim y As Double = Convert.ToDouble(ds.Y(e.PointIndex))
    If (y < 0) Then
        ps.LineStyle.Color = Color.Blue
    Else
        ps.LineStyle.Color = Color.Red
    End If
    ' apply to all points
    e.Selected = True
End Sub
```

## To write code in C#

```
C#  
  
private void button1_Click(object sender, System.EventArgs e)  
{  
    // create point style  
    C1.Win.C1Chart.PointStyle ps = new C1.Win.C1Chart.PointStyle();  
    // custom point style  
    ps.Selection = C1.Win.C1Chart.PointStyleSelectionEnum.Custom;  
    ps.Select += new C1.Win.C1Chart.PointStyleSelectEventHandler(PS_Select);  
    // add point style  
    c1Chart1.ChartGroups[0].ChartData.PointStylesList.Add( ps);  
}  
  
private void PS_Select( object sender, C1.Win.C1Chart.PointStyleSelectEventArgs e)  
{  
    C1.Win.C1Chart.PointStyle ps = sender as C1.Win.C1Chart.PointStyle;  
    // set visual appearance depending on y value  
    C1.Win.C1Chart.ChartDataSeries ds =  
    c1Chart1.ChartGroups[0].ChartData[e.SeriesIndex];  
    double y = Convert.ToDouble( ds.Y[e.PointIndex]);  
    if( y<0)  
        ps.LineStyle.Color = Color.Blue;  
    else  
        ps.LineStyle.Color = Color.Red;  
    // apply to all points  
    e.Selected = true;  
}
```

A custom blue point style is created for Y values less than zero and a custom red point style is created for y values greater than zero.



## Data Binding

Data binding is a process that allows one or more data consumers to be connected to a data provider in a synchronized manner. If you edit a value that is part of a bound dataset, the [C1Chart](#) controls connected to the same data source will change to reflect the new value.

Unlike many bound controls, **C1Chart** does not make use of currency. When data bound, the chart uses all of the bound data as its source of data for specified series data, and presents it in graphic form on the surface of the chart as directed by the series and other chart properties.

This process requires several simple steps, but requires some knowledge of the chart object model.

First, it is necessary to create a data source. Many data sources are available, including ADO.NET data source objects such as `DataTable`, `DataView`, `DataSet` and `DataViewManager`. Also, third-party data sources, such as ComponentOne `DataObjects` components, including `C1ExpressTable`, `C1ExpressView`, `C1ExpressConnection`, `C1DataView`, `C1DataTableSource` and `C1DataSet` may be used.

For details about creating ADO.NET data source objects, please refer to the .NET Framework documentation.

For details about using **DataObjects for .NET**, see the **DataObjects for .NET** documentation included in the ComponentOne Studio Enterprise.

The following sections explain how to bind the data to the chart by first setting the data source and then setting the data fields.

## Binding Data to C1Chart

To bind data to [C1Chart](#), first set the data source and then set the data field.

### Step 1: Setting the DataSource

Once a valid data source has been created, set the `C1Chart.DataSource` property. This can be done at design time or run time.

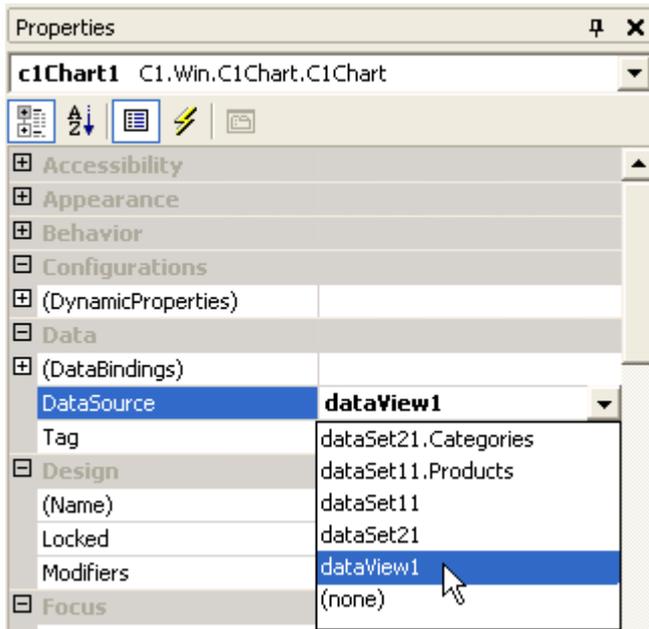
#### To set the DataSource property at design time:

You can set the `C1Chart.DataSource` property at design time using any of the following methods:

- **C1Chart's Properties window**

To set the `C1Chart.DataSource` property using `C1Chart's Properties window`:

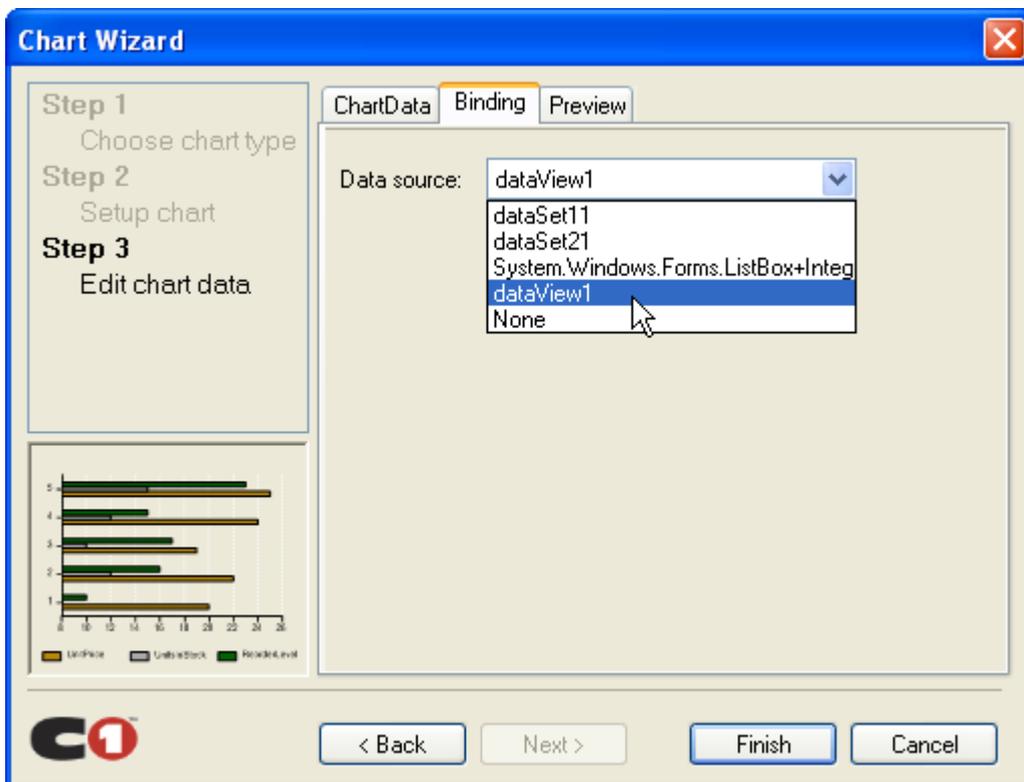
1. Select **C1Chart** from the Properties drop-down list box
2. Then select a value from the `C1Chart.DataSource` property drop-down list box.



- **Chart Wizard**

The `C1Chart.DataSource` property can also be set using the **Chart Wizard** or **Chart Properties** designer. To set the `C1Chart.DataSource` property using **Chart Wizard**, perform the following tasks:

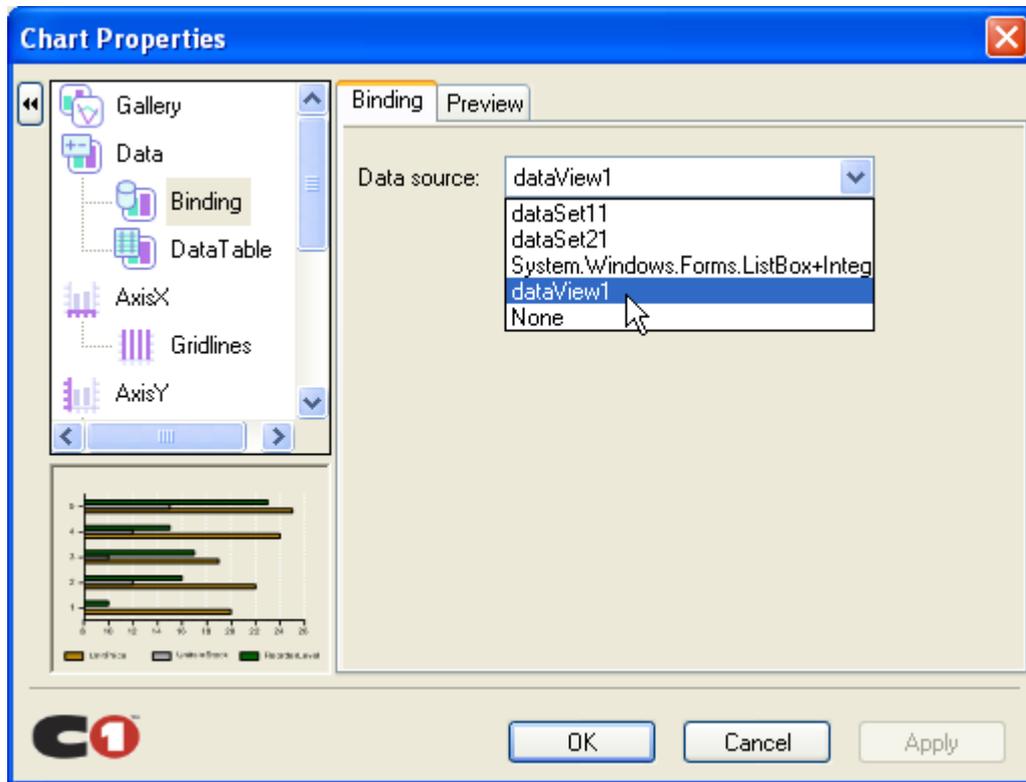
1. In **Step 3** of the **Chart Wizard**, select the **Binding** tab.
2. Select a value from the **Data source** drop-down list box.



- **Chart Properties designer**

To set the `DataSource` property using **Chart Properties** designer, perform the following:

1. Right-click on the **C1Chart** control and select **Chart Properties** to access the **Chart Properties** editor.
2. Expand the **Data** element, then select **Binding**.
3. In the **Binding** tab, select a value from the **Data source** drop-down list box.



### To set the chart DataSource property programmatically:

Use the following code to set the Char [DataSource](#) property to the data source object programmatically:

#### To write code in Visual Basic

Visual Basic

```
C1Chart1.DataSource = DataSet11
```

#### To write code in C#

C#

```
c1Chart1.DataSource = dataSet11;
```

### Step 2: Setting the data field

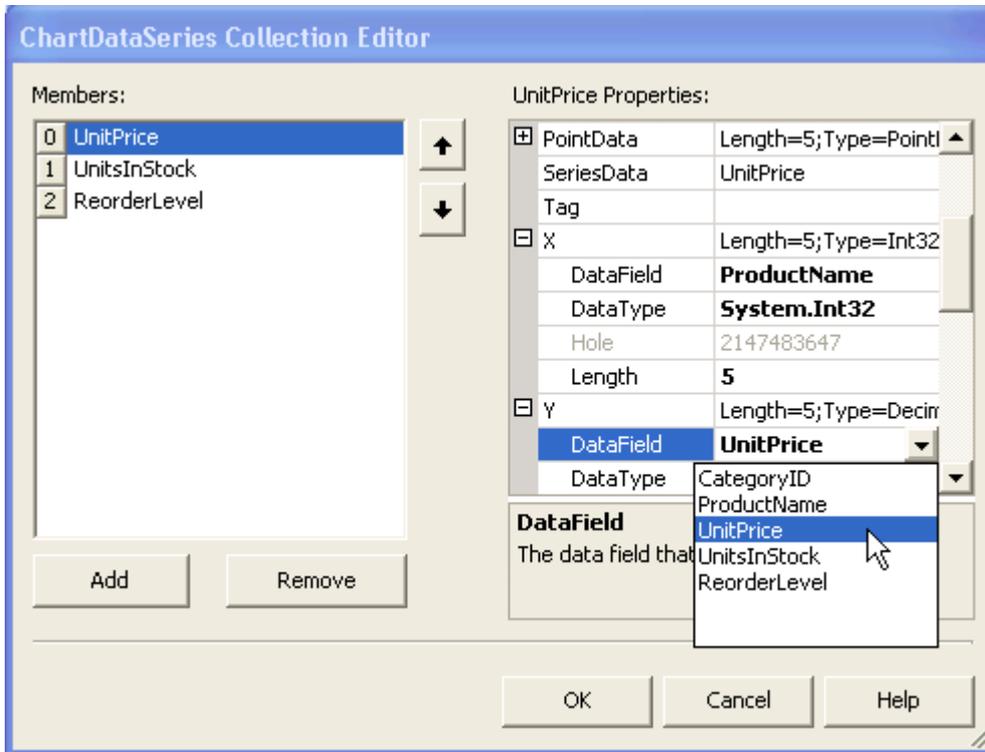
Once the **C1Chart** [DataSource](#) has been set, it is necessary to tell **C1Chart** which columns in the dataset should be applied to each of the [ChartDataSeries](#). In other words, it is necessary to specify which column of the data source should be used to specify the X values, which column specifies the Y values, and depending upon the [ChartType](#), the columns used for **Y1**, **Y2** and **Y3** of each [ChartDataSeries](#).

The set of **X** values, **Y** values, and so on, are held by the [ChartDataArray](#) objects of each [ChartDataSeries](#). These objects all have a [DataField](#) property which is used to specify the column of data in the data source that applies to the specific [ChartDataArray](#). The [DataField](#) property may be set at either design time or run time.

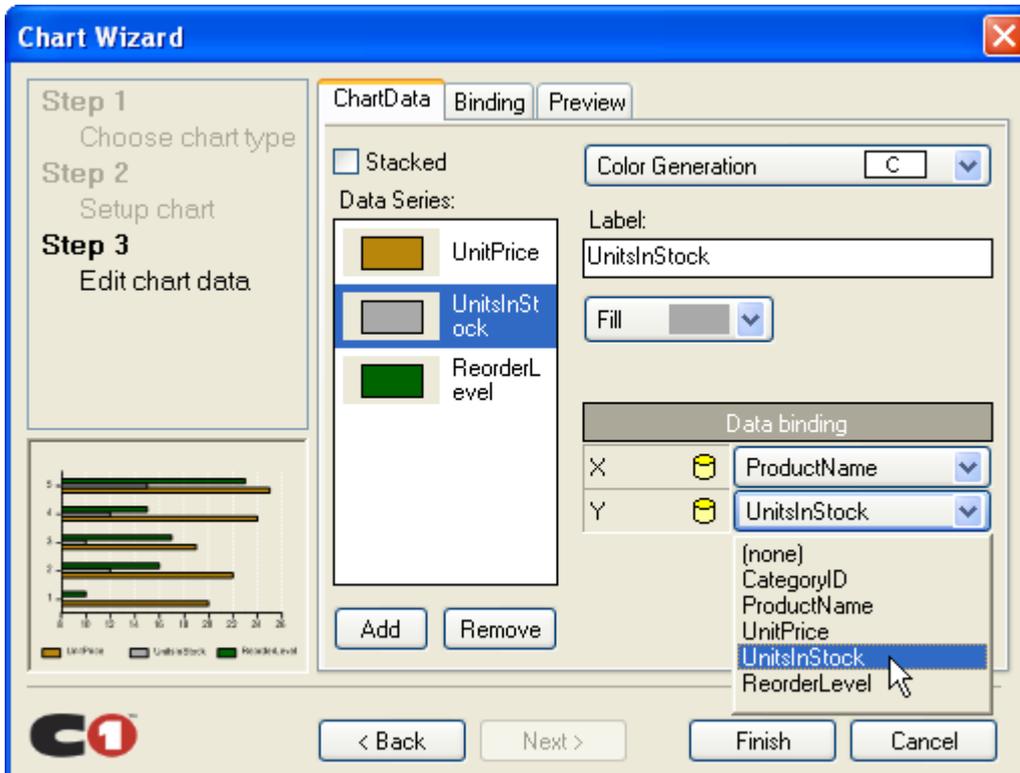
To set the data field using the designer

At design time, and using the Visual Studio .NET properties window, navigate through the [ChartGroups](#), [Group0](#),

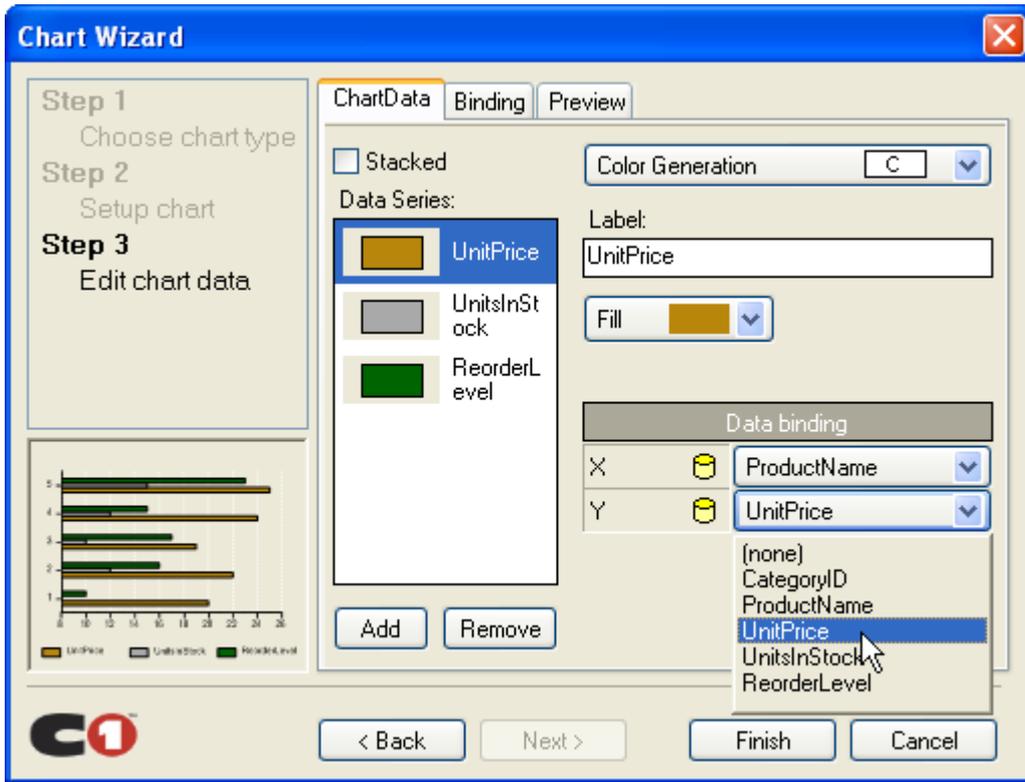
ChartData, and SeriesList, and open the SeriesList by double-clicking the **ellipsis** button. This opens the **ChartDataSeries Collection Editor**. Select the first series, and navigate the **X** property. The **X** property is a ChartDataArray object. Expand the **X** node. Select the DataField property, then select an available value from its drop-down list box.



The DataField can be set from both the **Chart Wizard** and **Chart Properties** designers as well. The following image depicts the DataField in the X and Y drop-down list box of the **Chart Wizard** dialog box:



The following image depicts the DataField in the X and Y drop-down list box of the **Chart Properties** designer:



To set the data field using code

At run time, the DataField property is available on each ChartDataArray object of each ChartDataSeries. Note that the DataField property is set to a string value which indicates the name of the data source column.

### To write code in Visual Basic

Visual Basic

```
C1Chart1.ChartGroups(0).ChartData.SeriesList(0).X.DataField = "QuickStop.ShippedDate"
C1Chart1.ChartGroups(0).ChartData.SeriesList(0).Y.DataField = "QuickStop.SaleAmount"
```

### To write code in C#

C#

```
c1Chart1.ChartGroups[0].ChartData.SeriesList[0].X.DataField = "QuickStop.ShippedDate";
c1Chart1.ChartGroups[0].ChartData.SeriesList[0].Y.DataField = "QuickStop.SaleAmount";
```

Please note the syntax of the string. In code sample above, *QuickStop* is the TableName containing the column, and *ShippedDate* and *SaleAmount* are the column names. Whether or not the TableName appears in the string depends upon the data source itself. In the above case, a DataSet object with multiple tables is used as a data source, so the TableName is required. If a specific table in the DataSet object were used as the data source, then only the column names would be used.

## Binding the Chart Directly to the Data Source

In most cases, data needs to be summarized before it can be charted, so there is a layer between the data source and the actual chart. In some cases, however, the data you want to plot may already be available in a DataView or similar

data source object. In these cases, you can bind the chart directly to the data source object.

To bind a `C1Chart` control to a data source, you should start by setting the control's `DataSource` property to a data source object such as a `DataView` or `DataTable`. Then bind individual series to columns on the data source object using the `DataSeries.X.DataField` and `DataSeries.Y.DataField` properties.

You can do all this using the **Chart Wizard**, but if you prefer to do it in code here is a sample that demonstrates the entire process:

#### To write code in Visual Basic

##### Visual Basic

```
' DataBinding is only available with C1Chart
' version 1.0.20034.13244 and later.
' get chart data
Dim sql As String = "select * from products"
Dim conn As String = "provider=... c:\nwind.mdb"
Dim da As New OleDbDataAdapter(sql, conn)
da.Fill(dt)

' bind chart to table (each series will bind to a field on the table).
clchart.DataSource = dt

' clear data series collection.
Dim dsc As ChartDataSeriesCollection = clchart.ChartGroups(0).ChartData.SeriesList
dsc.Clear()

' add unit price series.
Dim ds As ChartDataSeries = dsc.AddNewSeries()

'ds.AutoEnumerate = true ' (in case you don't want to set the X values).
ds.X.DataField = "ProductName"
ds.Y.DataField = "UnitPrice"

' add units in stock series.
ds = dsc.AddNewSeries()
ds.X.DataField = "ProductName"
ds.Y.DataField = "UnitsInStock"

' apply filters, sorting, etc.
dt.DefaultView.RowFilter = "CategoryID = 4"
```

#### To write code in C#

##### C#

```
// DataBinding is only available with C1Chart.
// version 1.0.20034.13244 and later.
// get chart data
string sql = "select * from products";
string conn = @"provider=... c:\nwind.mdb";
OleDbDataAdapter da = new OleDbDataAdapter(sql, conn);
da.Fill(dt);
```

```
// bind chart to table (each series will bind to a field on the table).
clchart.DataSource = dt;
// clear data series collection.
ChartDataSeriesCollection dsc = clchart.ChartGroups[0].ChartData.SeriesList;
dsc.Clear();

// add unit price series.
ChartDataSeries ds = dsc.AddNewSeries();

//ds.AutoEnumerate = true; // (in case you don't want to set the X values).
ds.X.DataField = "ProductName";
ds.Y.DataField = "UnitPrice";

// add units in stock series.
ds = dsc.AddNewSeries();
ds.X.DataField = "ProductName";
ds.Y.DataField = "UnitsInStock";

// apply filters, sorting, etc.
dt.DefaultView.RowFilter = "CategoryID = 4";
```

The code retrieves the *NWind Products* table, then creates two data series, each bound to a column on the data source.

The most interesting part of the bound table is that it maintains a dynamic connection to the data source. Any changes applied to the data source, including value changes, filters, sorting, new or deleted records, and so on, are automatically reflected on the chart.

## Functional Program Using Data Binding

The following code, when added to a new project with the [C1Chart](#) control added to the form, demonstrates a fully functional program using data binding. Please note that it is also necessary to create and handle the **Form\_Load** event. The database used in this sample is the NorthWind database supplied and used by Microsoft, as well as by ComponentOne Studio Enterprise. If the file path does not match your installation, it will be necessary to change the database before execution.

The following sample manipulates the database and **C1Chart** entirely through code. However, using the Microsoft .NET IDE and its property sheet, the entire setup can be managed at design time. Please note that the use of two *DataAdapters* with *TableMapping* allows a single *DataSet* to be used as the *DataSource*, with two series charted from the same table.

### To write code in Visual Basic

#### Visual Basic

```
Private Sub BindMultipleSeriesViewsAndChartSetup(ByVal chart As
C1.Win.C1Chart.C1Chart)

    ' following objects are in namespace System.Data.OleDb
    Dim connect As OleDbConnection = New OleDbConnection()
    Dim adapt1 As OleDbDataAdapter = New OleDbDataAdapter()
    Dim adapt2 As OleDbDataAdapter = New OleDbDataAdapter()
    Dim select1 As OleDbCommand = New OleDbCommand()
    Dim select2 As OleDbCommand = New OleDbCommand()
```

```

' set up the connect to the ComponentOne sample database.
connect.ConnectionString = _
    "Provider=Microsoft.Jet.OLEDB.4.0;" + _
    "User ID=Admin;" + _
    "Data Source=C:\Program Files\ComponentOne Studio.Net\common\C1NWIND.MDB;"
+ _
    "Jet OLEDB:Engine Type=5;"

' select Save-A-Lot entries in [Sales Totals by Amount]
select1.CommandText = _
    "SELECT SaleAmount, ShippedDate, CompanyName " + _
    "FROM [Sales Totals by Amount] " + _
    "WHERE (CompanyName = 'Save-a-lot Markets') " + _
    "ORDER BY ShippedDate"
select1.Connection = connect

' select Quick-Stop entries [Sales Totals by Amount]
select2.CommandText = _
    "SELECT SaleAmount, ShippedDate, CompanyName " + _
    "FROM [Sales Totals by Amount] " + _
    "WHERE (CompanyName = 'QUICK-Stop') " + _
    "ORDER BY ShippedDate"
select2.Connection = connect

' using System.Data.Common namespace for the Mapping objects.
' TableMapping is used to allow multiple views of the same table
' to appear in the same DataSet.

' set up the adapter, adapt1.
adapt1.SelectCommand = select1
Dim ColumnMaps_SaveALot As DataColumnMapping() = _
{
    New DataColumnMapping("SaleAmount", "SaleAmount"), _
    New DataColumnMapping("CompanyName", "CompanyName"), _
    New DataColumnMapping("ShippedDate", "ShippedDate") _
}
adapt1.TableMappings.Add(New DataTableMapping("Table", "SaveALot", _
    ColumnMaps_SaveALot))

' set up the adapter, adapt2.
adapt2.SelectCommand = select2

Dim ColumnMaps_QuickStop As DataColumnMapping() = _
{
    New DataColumnMapping("SaleAmount", "SaleAmount"), _
    New DataColumnMapping("CompanyName", "CompanyName"), _
    New DataColumnMapping("ShippedDate", "ShippedDate") _
}

adapt2.TableMappings.Add(New DataTableMapping("Table", "QuickStop", _

```

```
ColumnMaps_QuickStop))

' Create the dataset and fill it from all adapters.
Dim ds As DataSet = New DataSet()
adapt1.Fill(ds)
adapt2.Fill(ds)

' set up the chart, assigning the DataSource, DataFields and properties.
chart.Dock = DockStyle.Fill
chart.DataSource = ds
Dim sc As ChartDataSeriesCollection =
chart.ChartGroups(0).ChartData.SeriesList
sc.RemoveAll()

' Add the Save-A-Lot series.
Dim s As ChartDataSeries = sc.AddNewSeries()
s.Label = "Save-A-Lot"
s.X.DataField = "SaveALot.ShippedDate"
s.Y.DataField = "SaveALot.SaleAmount"

' Add the Quick-Stop series.
s = sc.AddNewSeries()
s.Label = "Quick-Stop"
s.X.DataField = "QuickStop.ShippedDate"
s.Y.DataField = "QuickStop.SaleAmount"

' Set up the Axes and Legend.
chart.ChartArea.AxisX.AnnoFormat = FormatEnum.DateShort
chart.ChartArea.AxisY.AnnoFormat = FormatEnum.NumericCurrency
chart.ChartArea.AxisY.Min = 0

' Change to a bar chart.
chart.ChartGroups[0].ChartType = Chart2DTypeEnum.Bar
chart.ChartGroups[0].ShowOutline = false

' Position, Orient and Show the Legend
chart.Legend.Compass = CompassEnum.North
chart.Legend.Orientation = LegendOrientationEnum.Horizontal
chart.Legend.Visible = true
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    C1Chart1.Location = New Point(0)
    C1Chart1.Size = Me.ClientSize
    BindMultipleSeriesViewsAndChartSetup(C1Chart1)
End Sub
```

### To write code in C#

C#

```
private void BindMultipleSeriesViewsAndChartSetup(C1.Win.C1Chart.C1Chart chart)
{
    // following objects are in namespace System.Data.OleDb
    OleDbConnection connect = new OleDbConnection();
    OleDbDataAdapter adapt1 = new OleDbDataAdapter();
    OleDbDataAdapter adapt2 = new OleDbDataAdapter();
    OleDbCommand select1 = new OleDbCommand();
    OleDbCommand select2 = new OleDbCommand();

    // set up the connect to the ComponentOne sample database
    connect.ConnectionString =
        "Provider=Microsoft.Jet.OLEDB.4.0;" +
        "User ID=Admin;" +
        "Data Source=C:\\Program Files\\ComponentOne
Studio.Net\\common\\C1NWIND.MDB;" + "Jet OLEDB:Engine Type=5;";

    // select Save-A-Lot entries in [Sales Totals by Amount]
    select1.CommandText =
        "SELECT SaleAmount, ShippedDate, CompanyName " +
        "FROM [Sales Totals by Amount] " +
        "WHERE (CompanyName = \'Save-a-lot Markets\') " +
        "ORDER BY ShippedDate";
    select1.Connection = connect;

    // select Quick-Stop entries [Sales Totals by Amount]
    select2.CommandText =
        "SELECT SaleAmount, ShippedDate, CompanyName " +
        "FROM [Sales Totals by Amount] " +
        "WHERE (CompanyName = \'QUICK-Stop\') " +
        "ORDER BY ShippedDate";
    select2.Connection = connect;

    // using System.Data.Common namespace for the Mapping objects.
    // TableMapping is used to allow multiple views of the same table
    // to appear in the same DataSet.

    // set up the adapter, adapt1.
    adapt1.SelectCommand = select1;

    DataColumnMapping [] ColumnMaps_SaveALot =
    {
        new DataColumnMapping("SaleAmount", "SaleAmount"),
        new DataColumnMapping("CompanyName", "CompanyName"),
        new DataColumnMapping("ShippedDate", "ShippedDate")
    };

    adapt1.TableMappings.Add(new DataTableMapping("Table", "SaveALot",
ColumnMaps_SaveALot));

    // set up the adapter, adapt2.
    adapt2.SelectCommand = select2;
```

```
DataColumnMapping [] ColumnMaps_QuickStop =
{
    new DataColumnMapping("SaleAmount", "SaleAmount"),
    new DataColumnMapping("CompanyName", "CompanyName"),
    new DataColumnMapping("ShippedDate", "ShippedDate")
};

adapt2.TableMappings.Add(new DataTableMapping("Table", "QuickStop",
ColumnMaps_QuickStop));

// Create the dataset and fill it from all adapters.
DataSet ds = new DataSet();
adapt1.Fill(ds);
adapt2.Fill(ds);

// set up the chart, assigning the DataSource, DataFields and properties.
chart.Dock = DockStyle.Fill;
chart.DataSource = ds;
ChartDataSeriesCollection sc = chart.ChartGroups[0].ChartData.SeriesList;
sc.RemoveAll();

// Add the Save-A-Lot series.
ChartDataSeries s = sc.AddNewSeries();
s.Label = "Save-A-Lot";
s.X.DataField = "SaveALot.ShippedDate";
s.Y.DataField = "SaveALot.SaleAmount";

// Add the Quick-Stop series.
s = sc.AddNewSeries();
s.Label = "Quick-Stop";
s.X.DataField = "QuickStop.ShippedDate";
s.Y.DataField = "QuickStop.SaleAmount";

// Set up the Axes and Legend.
chart.ChartArea.AxisX.AnnoFormat = FormatEnum.DateShort;
chart.ChartArea.AxisY.AnnoFormat = FormatEnum.NumericCurrency;
chart.ChartArea.AxisY.Min = 0;

// Change to a bar chart.
chart.ChartGroups[0].ChartType = Chart2DTypeEnum.Bar;
chart.ChartGroups[0].ShowOutline = false;

// Position, Orient and Show the Legend
chart.Legend.Compass = CompassEnum.North;
chart.Legend.Orientation = LegendOrientationEnum.Horizontal;
chart.Legend.Visible = true; chart.Legend.Visible = true;
}

private void Form1_Load(object sender, System.EventArgs e)
{
    clChart1.Location = new Point(0);
}
```

```
c1Chart1.Size = this.ClientSize;  
BindMultipleSeriesViewsAndChartSetup(c1Chart1);  
}
```

## Charting Labels

[ChartLabels](#) is a label that displays in front of the chart data. The chart does not adjust to fit [ChartLabels](#), since they are completely independent. [ChartLabels](#) are useful when highlighting an important data point, but can also be used generally to provide information on data or on the chart.

A [Label](#) object defines an independent rectangular region that can be attached to a chart. The [LabelsCollection](#) contains all of the chart labels defined for a particular chart. A chart label can be added to the chart at design time or programmatically through the [ChartLabels](#) object. [C1Chart](#) provides an **Edit Labels** designer so you can conveniently add, remove, or modify existing labels through the designer.

The following code shows how to access a [ChartLabels](#) individually by specifying the index number in the [LabelsCollection](#):

### To write code in Visual Basic

Visual Basic

```
C1Chart1.ChartLabels.LabelsCollection(0).Text = "First label in collection"
```

### To write code in C#

C#

```
c1Chart1.ChartLabels.LabelsCollection[0].Text = "First label in collection";
```

The following code shows how to create a [Label](#) object, by calling the [AddNewLabel](#) method:

### To write code in Visual Basic

Visual Basic

```
Dim Clabel As C1.Win.C1Chart.Label  
Clabel = C1Chart1.ChartLabels.LabelsCollection.AddNewLabel()
```

### To write code in C#

C#

```
C1.Win.C1Chart.Label Clabel;  
Clabel = c1Chart1.ChartLabels.LabelsCollection.AddNewLabel();
```

There is no limit to the number of [ChartLabels](#) a chart can contain. Each [ChartLabel](#) has [label](#), [interior color](#), [border](#) and [attachment](#) attributes that can be customized.

The [Label](#) object provides a number of properties that help define and position the [ChartLabel](#). The most important of these properties are:

- The [Compass](#) property specifies the positioning of the label around its anchor. For instance, setting the [Compass](#) property to *East* will place the label to the right of the anchor.
- The [AttachMethod](#) property specifies by which method the label will be attached to the chart. It can be attached by *Coordinate*, *DataCoordinate*, *DataIndex*, or by the *DataIndexY*.
- The [Text](#) property specifies the text to appear in the [ChartLabel](#).
- The [Connected](#) property is a Boolean, which specifies whether a line is to be drawn from the chart label to its attached location. If **True**, the line is drawn.
- The [Offset](#) property specifies the distance, in the anchor direction, from the [ChartLabel](#) to its attached location, and is of type [Long](#).

- The [DefaultLabelStyle](#) property sets a default style that all the labels will inherit from. For instance, if the `BackColor` of the default style is set to black, all future labels will have a black back color.

A `Label` object defines an independent rectangular region that can be attached to a chart. The `LabelsCollection` contains all of the `ChartLabels` defined for a particular chart.

## Attaching and Positioning Chart Labels

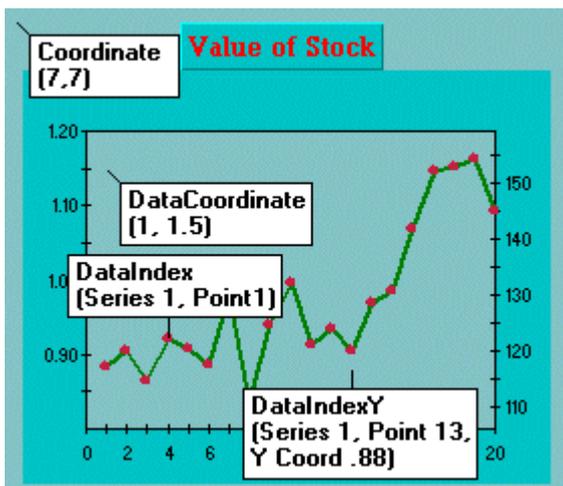
When defining a `ChartLabel`, it is necessary to specify how to attach it to the chart and where to position it relative to its attachment point.

The attachment method chosen depends on what the `ChartLabel` is going to be used for. The attachment method takes an [AttachMethodEnum](#) value with four possible values: `Coordinate`, `DataCoordinate`, `DataIndex`, and `DataIndexY`.

`ChartLabels` can be attached to a chart in one of four ways:

- To a pixel (x,y) coordinate
- To a data (x,y) coordinate in the `ChartArea`
- To a (series, point) in the `ChartArea`
- To a (series, point, Y value) in the `ChartArea`

The following image illustrates the `ChartLabel` attachment methods:



The following list explains each attachment method for the `ChartLabels`:

- `AttachMethodEnum.Coordinate` attaches the label anywhere on the chart. The number of pixels can be specified between the top-left corners of the chart to the `ChartLabel`.
- `AttachMethodEnum.DataCoordinate` attaches the label anywhere inside the `PlotArea`. The data coordinates can be specified. This method is not allowed on Pie charts using `PointLabels` for the X-axis annotation. If any part of the `ChartLabel` falls outside of the `ChartArea`, it is clipped.
- `AttachMethodEnum.DataIndex` attaches the label to a specific data point on the chart. The series and point indices, and the `ChartGroup` can be specified.
- `AttachMethodEnum.DataIndexY` attaches the label to a distance above or below a specific data point. The series and point indices, the `ChartGroup`, and the Y-coordinate can be specified. This is most useful for Bar and Stacking Bar charts.

Use the `AttachMethod` property of the `Label` class to set the attachment method, and the properties of the `AttachMethodData` class to set the attachment point. These properties can be accessed programmatically through the `Label` class or at design time through the `Labels Collection` editor or the `Edit Labels` designer. For more information on setting the `ChartLabel` methods at design time using Chart's Smart Designer, see [Attaching and](#)

Positioning Chart Labels.

## Attaching the Chart Label by Pixel Coordinate

In a coordinate attachment the label can be attached anywhere on the chart. The number of pixels between the top-left corner of the chart to the ChartLabel can be specified.

To attach a ChartLabel to a pixel coordinate, set the `AttachMethod` property to `AttachMethodEnum.Coordinate`, and then under the `AttachMethodData` object set the `X` and the `Y` properties to the coordinates at which the ChartLabel is to be attached. For example, the following attaches a ChartLabel to the coordinates specified:

### To write code in Visual Basic

#### Visual Basic

```
With C1Chart1.ChartLabels.LabelsCollection(0)
    .AttachMethod = AttachMethodEnum.Coordinate
    .AttachMethodData.X = 250
    .AttachMethodData.Y = 250
End With
```

### To write code in C#

#### C#

```
C1.Win.C1Chart.Label lab = c1Chart1.ChartLabels.LabelsCollection[0];
lab.AttachMethod = AttachMethodEnum.Coordinate;
lab.AttachMethodData.X = 250;
lab.AttachMethodData.Y = 250;
```

For more information on how to attach the chart label by pixel coordinate at design time see, [Attach By Coordinate](#).

## Attaching the Chart Label by Data Coordinate

To attach a ChartLabel to a data coordinate on the chart, set the `AttachMethod` property to `AttachMethodEnum.DataCoordinate`, and then under the `AttachMethodData` object set the `X` and `Y` properties to the data coordinates at which the ChartLabel is to be attached. For example, the following attaches a ChartLabel to the data coordinate (1.1, 15.8):

### To write code in Visual Basic

#### Visual Basic

```
With C1Chart1.ChartLabels.LabelsCollection(0)
    .AttachMethod = AttachMethodEnum.DataCoordinate
    .AttachMethodData.X = 1.1
    .AttachMethodData.Y = 15.8
End With
```

### To write code in C#

#### C#

```
C1.Win.C1Chart.Label lab = c1Chart1.ChartLabels.LabelsCollection[0];
lab.AttachMethod = AttachMethodEnum.DataCoordinate;
```

```
lab.AttachMethodData.X = 1.1;  
lab.AttachMethodData.Y = 15.8;
```

ChartLabels can only be attached to a data coordinate if the chart is an Area or XY-Plot. Any part of the ChartLabel that falls outside of the ChartArea is clipped.

For more information on how to attach the chart label by data coordinate at design time see, [Attach by Data Coordinate](#).

## Attaching the Chart Label by Data Point

To attach a ChartLabel to a data point, set the **AttachMethod** property to `AttachMethodEnum.DataIndex`, and then under the `AttachMethodData` object set the **GroupIndex**, **SeriesIndex**, and **PointIndex** properties. These properties specify the ChartGroup, Series, and Point indices. `PointIndex` refers to the element of the data array. `SeriesIndex` refers to the element of the series array. Each series has arrays of data.

The following code example attaches a ChartLabel to a data point:

### To write code in Visual Basic

```
Visual Basic  
  
With C1Chart1.ChartLabels.LabelsCollection(0)  
    .AttachMethod = AttachMethodEnum.DataIndex  
    .AttachMethodData.GroupIndex = 0  
    .AttachMethodData.SeriesIndex = 1  
    .AttachMethodData.PointIndex = 2  
End With
```

### To write code in C#

```
C#  
  
C1.Win.C1Chart.Label lab = c1Chart1.ChartLabels.LabelsCollection[0];  
lab.AttachMethod = AttachMethodEnum.DataIndex;  
lab.AttachMethodData.GroupIndex = 0;  
lab.AttachMethodData.SeriesIndex = 1;  
lab.AttachMethodData.PointIndex = 2;
```

For more information on how to attach the chart label by data point at design time see, [Attach by Data Index](#).

## Attaching the Chart Label by Data Point and Y Value

The two methods, attaching the ChartLabel by data coordinate and attaching the ChartLabel by data point, can be combined so that a ChartLabel is attached to a location whose X-coordinate is specified by a data point and whose Y-coordinate is specified by a chart value. Note that this method is not supported in Pie charts.

To attach the ChartLabel by data point and Y value, set the **AttachMethod** property to `AttachMethodEnum.DataIndexY`, and then under the `AttachMethodData` object set the **GroupIndex**, **SeriesIndex**, **PointIndex**, and **Y** properties.

### To write code in Visual Basic

```
Visual Basic  
  
With C1Chart1.ChartLabels.LabelsCollection(0)
```

```
.AttachMethod = AttachMethodEnum.DataIndex
.AttachMethodData.GroupIndex = 0
.AttachMethodData.SeriesIndex = 1
.AttachMethodData.PointIndex = 2
.AttachMethodData.Y = 15.8
```

End With

## To write code in C#

C#

```
C1.Win.C1Chart.Label lab = c1Chart1.ChartLabels.LabelsCollection[0];
lab.AttachMethod = AttachMethodEnum.DataIndex;
lab.AttachMethodData.GroupIndex = 0;
lab.AttachMethodData.SeriesIndex = 1;
lab.AttachMethodData.PointIndex = 2;
lab.AttachMethodData.Y = 15.8;
```

In this example, the ChartLabel location is specified as follows:

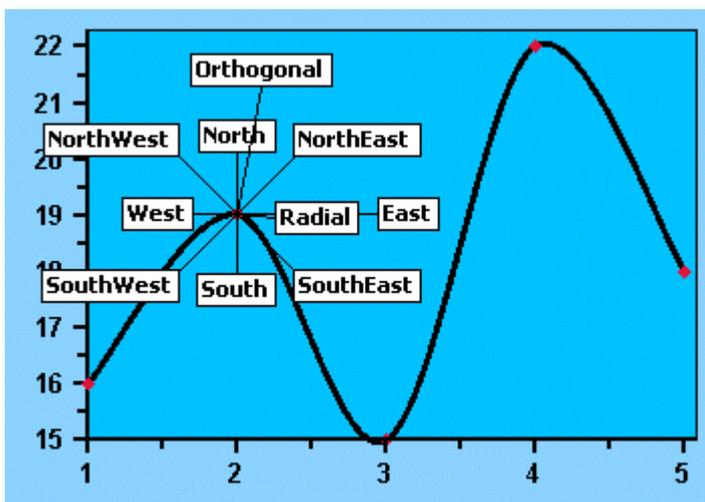
- The X-coordinate of the second point in the first data series becomes the X-coordinate of the chart label.
- The Y-coordinate specified by the Y property becomes the Y-coordinate of the chart label.

 **Note:** This method is not supported in Pie charts.

For more information on how to attach the chart label by data point at design time see, [Attach by Data Index](#).

## Anchoring Chart Labels

Once attached to a point on the chart, setting the [Compass](#) property sets the anchoring position for the ChartLabel. The property takes an enumeration of type [LabelCompassEnum](#), and the following shows the valid anchoring positions:



The [Compass](#) property acts similarly on ChartLabels as it does other objects in .NET. The Label is placed away from the point according to the specified compass direction. The [LabelCompassEnum.Radial](#) setting generally applies only to Polar, Radar, and Pie charts. It places the Label out radially from the center of the chart. This can be a convenient feature if a chart, such as a Pie chart, has a multitude of labels to be shown. The [LabelCompassEnum.Orthogonal](#)

setting places the Label perpendicularly from the path of the series line. For instance, if there was a straight horizontal line as the series plot, then the Orthogonal position would be exactly like the North [Compass](#) setting, likewise if there was a straight vertical line as the series plot, then the Orthogonal position would be exactly like the West [Compass](#) setting.

## Customizing Chart Labels

### Chart Labels Rotation

Use the [Rotation](#) property to set or change the rotation of a ChartLabel. Rotation is available at design time under the **DefaultLabelStyle** node of the **ChartLabels** object or through the **Label Collection Editor**. For more information on the **Label Collection Editor**, see [Label Collection Editor](#).

The [Rotation](#) property can be set to Rotate0, Rotate90, Rotate180, or Rotate270. If you would like to set a specific angle other than 90, 180, or 270 degrees then you can override these values and specify the clockwise rotation angle around the connection point of the label.

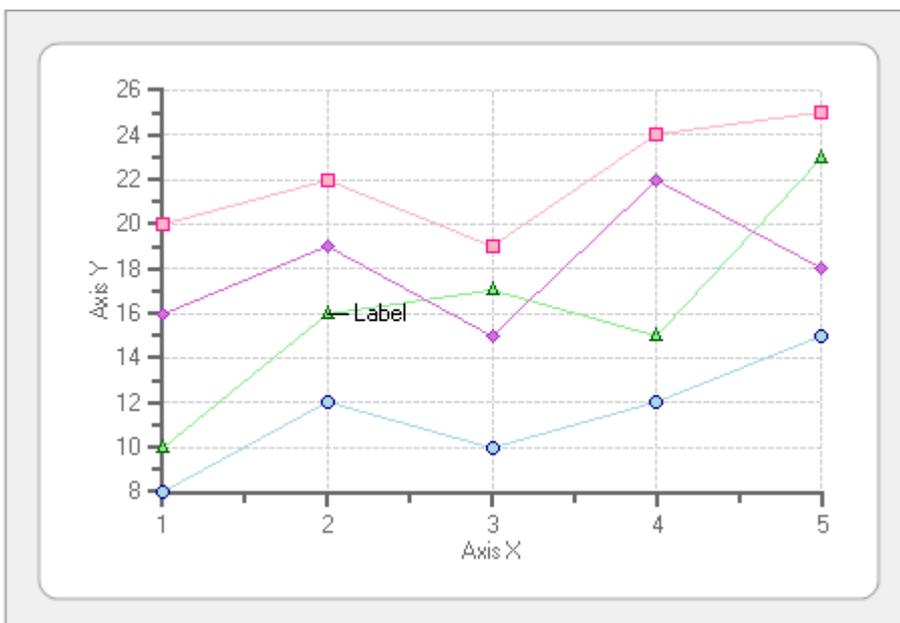
The [RotationOverride](#) property is useful for charts that require special labeling like psychometric charts where there is multiple axes with constant lines of various parameters such as enthalpy, humidity, temperature, etc.

 **Note:** The [RotationOverride](#) does not apply to **Radial** or **RadialText** compass values.

### Chart Labels Connecting Line

Set the [Connected](#) property to **True** to show a connecting line from the data coordinate to the associated label. A numeric value should be specified for the [Offset](#) property to make room to draw the connecting line from the data coordinate to the label.

The following image shows a connecting line with an offset value of 10 pixels:



### Chart Labels Border

Use the **BorderStyle** and **Thickness** border properties to customize the border's ChartLabel. These properties are

available at design time under the Style node of the ChartLabels Collection Editor. For more information on the ChartLabels Collection Editor, see [Label Collection Editor](#). This property can also be set in the DefaultLabelStyle node.

For more information on how to customize the label's appearance see, [Modify the Appearance of the Chart Labels using the Properties Window](#).

### Chart Labels Colors

Use the .NET Color properties to set the font for the ChartLabels. The color properties are available at design time under the Style node of the **ChartLabels Collection Editor**. For more information on the **ChartLabels Collection Editor**, see [Label Collection Editor](#). See [Chart Colors](#) for more information. This property can also be set in the **DefaultLabelStyle** node.

## Chart Labels Font

Use the font properties to customize the font used for a ChartLabel. These are available at design time under the Style node of the **ChartLabels Collection Editor**. For more information on the ChartLabels Collection Editor, see [Label Collection Editor](#). This property can also be set in the **DefaultLabelStyle** node.

## Setting a Default Label Style

The **DefaultLabelStyle** property of the ChartLabels object sets a default style for all current and future ChartLabels. This property can be useful in many ways, but two are outlined here.

First, this property sets default style properties that all future Labels will inherit. Secondly, since all labels inherit this default style, if a default style property is changed, all of the ChartLabels will then take on this attribute. For instance, suppose 100 ChartLabels created at design time should have had green as the BackColor, the following code could be inserted into the application:

### To write code in Visual Basic

Visual Basic

```
c1Chart1.ChartLabels.DefaultLabelStyle.BackColor = Color.Green
```

### To write code in C#

C#

```
c1Chart1.ChartLabels.DefaultLabelStyle.BackColor = Color.Green;
```

If style attributes for individual ChartLabels have to be changed, then these attributes can be set on the specific ChartLabel object manually.

## Chart Area and Plot Area Objects

This section details the important **ChartArea** objects individually and provides information on how to access, modify, and create the appearance for the properties in the ChartArea object.

The following table defines the main objects included in the ChartArea object.

Property	Description
<a href="#">Alignment</a>	Gets or sets the text alignment within the axis display. Inherits from the ChartArea.
<a href="#">Compass</a>	Allows you to set the position of the axis. For example, you may want to display the X-axis above the data instead of below.
<b>C1.Win.C1Chart.Style.Font</b>	Sets the font used to display the values along the axis.
<a href="#">ForeColor</a>	Sets the color used to display the axis, tickmarks, and values.
<a href="#">OnTop</a>	Gets or sets whether axis and gridlines appear on top of the chart image.
<a href="#">Reversed</a>	Gets or sets whether the axis is normal or reversed (ascending or descending).
<a href="#">Text</a>	Sets a string to display next to the axis (this is typically used to describe the variable and units being depicted by the axis).
<a href="#">Rotation</a>	Sets the orientation of the Text string.

## Axes

The **X**, **Y**, and **Y2** properties of the ChartArea object returns Axis objects that allow you to customize the appearance of the chart axes. The axes are represented by sub-properties of the **ChartArea** property: ChartArea.AxisX, ChartArea.AxisY, and ChartArea.AxisY2. Each of these properties returns an Axis object with the following main properties:

Property	Description
<a href="#">Alignment</a>	Gets or sets the text alignment within the axis display. Inherits from the ChartArea.
<a href="#">Compass</a>	Allows you to set the position of the axis. For example, you may want to display the X-axis above the data instead of below.
<b>C1.Win.C1Chart.Style.Font</b>	Sets the font used to display the values along the axis.
<a href="#">ForeColor</a>	Sets the color used to display the axis, tickmarks, and values.
<a href="#">OnTop</a>	Gets or sets whether axis and gridlines appear on top of the chart image.
<a href="#">Reversed</a>	Gets or sets whether the axis is normal or reversed (ascending or descending).
<a href="#">Text</a>	Sets a string to display next to the axis (this is typically used to describe the variable and units being depicted by the axis).
<a href="#">Rotation</a>	Sets the orientation of the Text string.

## Axis Position

Axis annotation typically appears beside its axis. This may be a problem on charts with an origin that is not at the axis

minimum or maximum. The chart can automatically determine where to place annotation in different situations, depending on the chart type. The [Compass](#) property can also specify annotation placement for an axis. The Compass value for the X-axis can be set to either **North** or **South**, while the value for a Y-axis can be set to either **East** or **West**. By default, the X-axis is set to **South** and the Y-axis is set to **West**.

## Axis Appearance

### Alignment

The [Alignment](#) property can be set to three different settings: **Center**, **Near**, or **Far**. Setting the alignment to center centers the axis title in comparison to the ChartArea. Setting the alignment to Near places the axis title to the left side of the ChartArea. Setting the alignment to Far places the axis title to the right side of the Chart Area.

### Font

The [Font](#) size and style for the axis title can be changed by manipulating the Font object of the axis. To access the font properties at design time click the **ellipsis** next to the Font node or expand the Font node under the axis object in the Visual Studio Properties window. To programmatically modify the Axis font properties, enter the following:

#### To write code in Visual Basic

```
Visual Basic
Dim f As Font = New Font("Arial", 8, FontStyle.Bold)
ClChart1.ChartArea.AxisX.Font = f
```

#### To write code in C#

```
C#
Font f = new Font("Arial", 8, FontStyle.Bold);
clChart1.ChartArea.AxisX.Font = f;
```

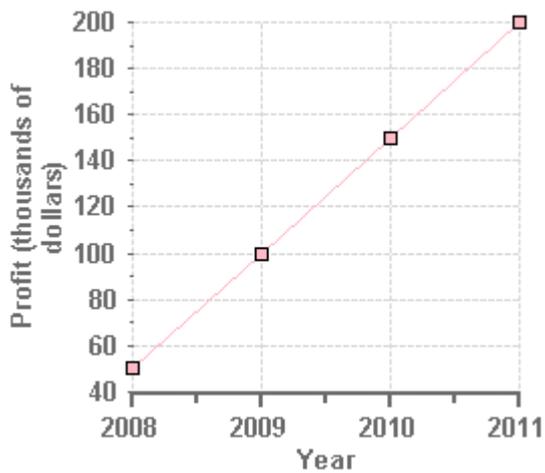
### ForeColor

The [ForeColor](#) property changes the color of the axis line, tick marks, and title. To change the [ForeColor](#) set the [ForeColor](#) property to a valid color in code, or the property can also be accessed at design time under the Axis object in the Visual Studio Properties window.

### Thickness

The thickness of the axis is determined by the [Thickness](#) property. The value is specified in pixels. Axes with a zero thickness value are fully drawn except for the axis line itself.

The following chart displays the Y-Axis with its [Thickness](#) property set to zero:



## Axis Title and Rotation

Adding a title to an axis clarifies what is charted along that axis. Axis titles can be added to **Area**, **XY-Plot**, **Bar**, **HiLo**, **HiLoOpenClose** or **Candle** charts. The title or the annotation along the axis can also be rotated. To see how to rotate the axis annotation see, [Axis Annotation Rotation](#).

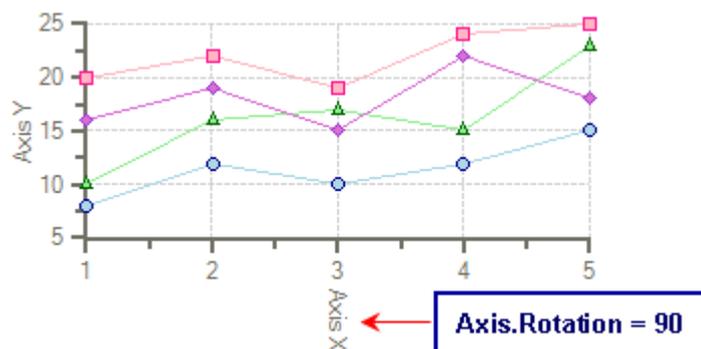
## Adding an Axis Title

Use the axis `Text` property to add a title to an axis. To remove the title, delete the default text for the `AxisX` or `AxisY` object.

 **Note:** An axis title for the Y axis cannot be added to Polar, Radar or Filled Radar charts.

## Rotating Axis Title

Use the `Rotation` property to rotate the axis title to 90, 180, or 270 degrees. The 90 and 270-degree rotations are most efficient for vertical axes.



## Axis Tick Marks

The chart automatically sets up the axis with both major and minor ticks. Customizing the tick spacing or attributes is as easy as manipulating a set of properties.

The [TickMajor](#) and [TickMinor](#) properties set the state of the Axis' tick marks. This property can be set to any of the [TickMarksEnum](#) values.

## Tick Mark Position

These values set where and if the tick marks will be displayed:

Value	Description
<a href="#">TickMarksEnum.None</a>	No tick marks along axis.
<a href="#">TickMarksEnum.Cross</a>	Tick marks cross over axis.
<a href="#">TickMarksEnum.Outside</a>	Tick marks located outside chart area on axis.
<a href="#">TickMarksEnum.Inside</a>	Tick marks located inside chart area on axis.

## Tick Mark Spacing

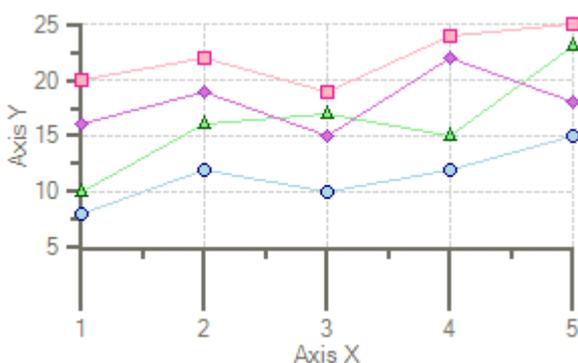
The [AutoMajor](#) and [AutoMinor](#) properties set whether the tick marks are to be automatically configured by the chart. When both these properties are set to **True**, the chart uses the current data to logically place the major and minor tick marks. When the [AutoMajor](#) property is true, its not necessary to enable overlap for axis annotations.

The [UnitMajor](#) and [UnitMinor](#) properties set the units by which the ticks will be spaced. When the [UnitMajor](#) property is set, the [UnitMinor](#) property is automatically set by the chart to half the UnitMajor value. Although the chart automatically sets the [UnitMinor](#) property, it also can be manually changed to a different value.

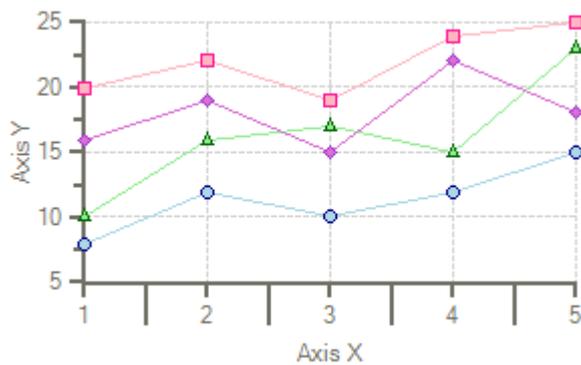
## Tick Mark Length

You can increase the length of the Major and Minor tick marks using the [TickFactorMajor](#) and [TickFactorMinor](#) properties. Tick marks are sized based on the thickness of the axis line as well as the tick factor. Doubling the tick factor doubles the length of the axis tick mark. Values are limited to integers from 1 to 20. Values outside of this range are ignored.

The following chart image has the [TickFactorMajor](#) property set to 5. This makes the tick length for the major tick marks five times longer.



The following chart image has the [TickFactorMinor](#) property set to 5. This makes the tick length for the minor tick marks five times longer.



## Axis Grid Lines

Grid lines are lines that appear perpendicular with major/minor tick marks at unit major/minor intervals. The lines that appear perpendicular to an axis at Major intervals are controlled by the [GridMajor](#) property and the lines that appear perpendicular to an axis at Minor intervals are controlled by the [GridMinor](#) property. Grid lines can help improve the readability of the Chart when you are looking for exact values. The major and minor grid lines appearances are controlled by the [ChartGridStyle](#) properties.

### To set the major grid lines appearance properties

Use the [GridMajor](#) property to set the major grid lines appearance properties. The [GridMajor](#) property gets the `C1.Win.C1Chart.ChartGridStyle` object that controls the appearances for the major grid lines.

### To set the minor grid lines appearance properties

Use the [GridMinor](#) property to set the minor grid lines appearance properties. The [GridMinor](#) property gets the `C1.Win.C1Chart.ChartGridStyle` object that controls the appearances for the minor grid lines.

## OnTop

The [OnTop](#) property specifies whether the axis grid lines and tick marks are drawn over the chart image or behind it. Setting this property to **True** will bring the grid lines and the tick marks to the foreground over the chart image, while setting it to **False** will bring the chart image to the foreground.

## Axis Bounds

Normally a graph displays all of the data it contains. However, a specific part of the chart can be displayed by fixing the axis bounds.

The chart determines the extent of each axis by considering the lowest and highest data value and the numbering increment. Setting the [Min](#), [Max](#), [AutoMin](#), and [AutoMax](#) properties allows the customization of this process.

## Axis Min and Max

Use the [Min](#) and [Max](#) properties to frame a chart at specific axis values. If the chart has X-axis values ranging from 0 to 100, then setting [Min](#) to 0 and [Max](#) to 10 will only display the values up to 10.

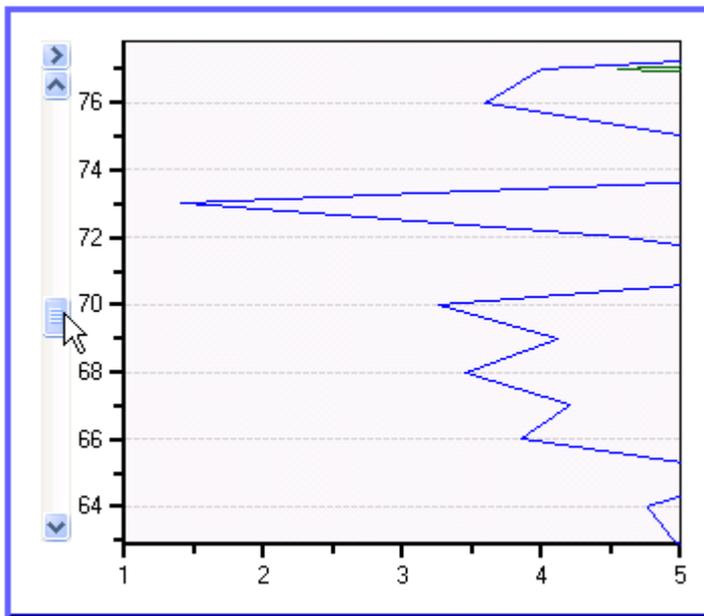
The chart can also calculate the [Min](#) and [Max](#) values automatically. If the [AutoMax](#) and [AutoMin](#) properties are set to

**True** then the chart automatically formats the axis numbering to fit the current data set.

 **Note:** The X-axis cannot be set for Polar charts.

## Axis Scrolling and Scaling

In circumstances when you have a substantial amount of X-values or Y-values in your chart data, you can add a scrollbar to the axes. Adding a scrollbar can make the data on the chart easier to read by scrolling through it so you can closely view pieces of data one at a time. The following image has the `AxisScrollbar` set to the `AxisY` object:



A scrollbar can appear on the X-Axis or Y-Axis simply by assigning the `Scrollbar` property to either the `AxisX` or `AxisY` object and then specifying the `Min` and `Max` values of the chart's data series. Setting the `Min` and `Max` values will prevent the scrollbar from changing the Axis values when you are scrolling.

## Scrollbar Appearance

`C1Chart`'s `AxisScrollbar` class provides a number of useful properties to control the scrollbar's appearance such as its style, size, and alignment.

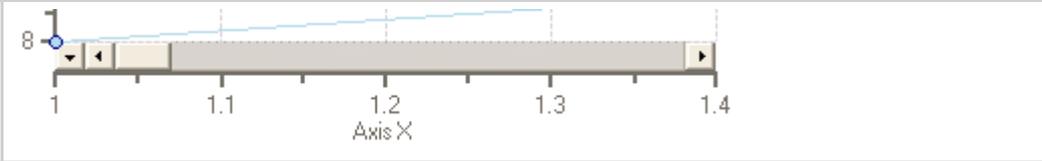
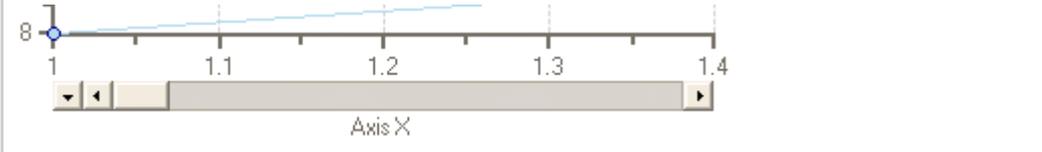
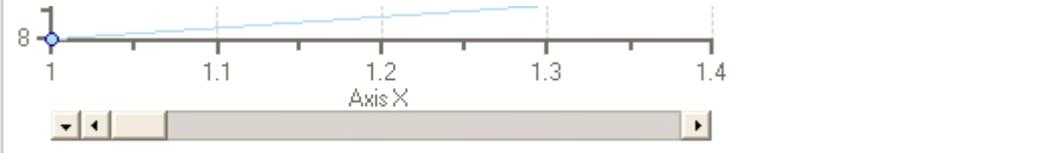
`C1Chart`'s scrollbar appearance can be determined by the `Appearance` property.

The following table represents the three different axis scroll bar styles to choose from when you set its `Appearance` property:

Value	Appearance
Normal	
Flat	
XP	

The scrollbar's width can be specified through the `Size` property and its alignment can be set through its `Alignment` property.

The following table represents the three different scrollbar alignments to choose from when you set its [Alignment](#) property:

Value	Appearance
Near	
Center	
Far	

## Scrollbar Scaling

You can use scaling while you are scrolling through large amounts of data to zoom in on important data.

Scaling can be implemented at run time by clicking on the scroll bar buttons, or in code, using the [ScaleMenuItems](#) property.



A built-in scale menu appears when the [Buttons](#) property is set to `ScaleAndScrollButtons` (default).

The following table includes the values' descriptions for the [Buttons](#) property:

Value	Description
<b>NoButtons</b>	No buttons appear on the scrollbar.
<b>ScrollButtons</b>	Scroll buttons appear on the scrollbar.
<b>ScaleButton</b>	A scale dropdown button appears. When the dropdown button is clicked a built-in scale menu appears with default scale items.
<b>ScaleAndScrollButtons</b>	The default setting. Both the scroll buttons and scale button appears on the scrollbar.

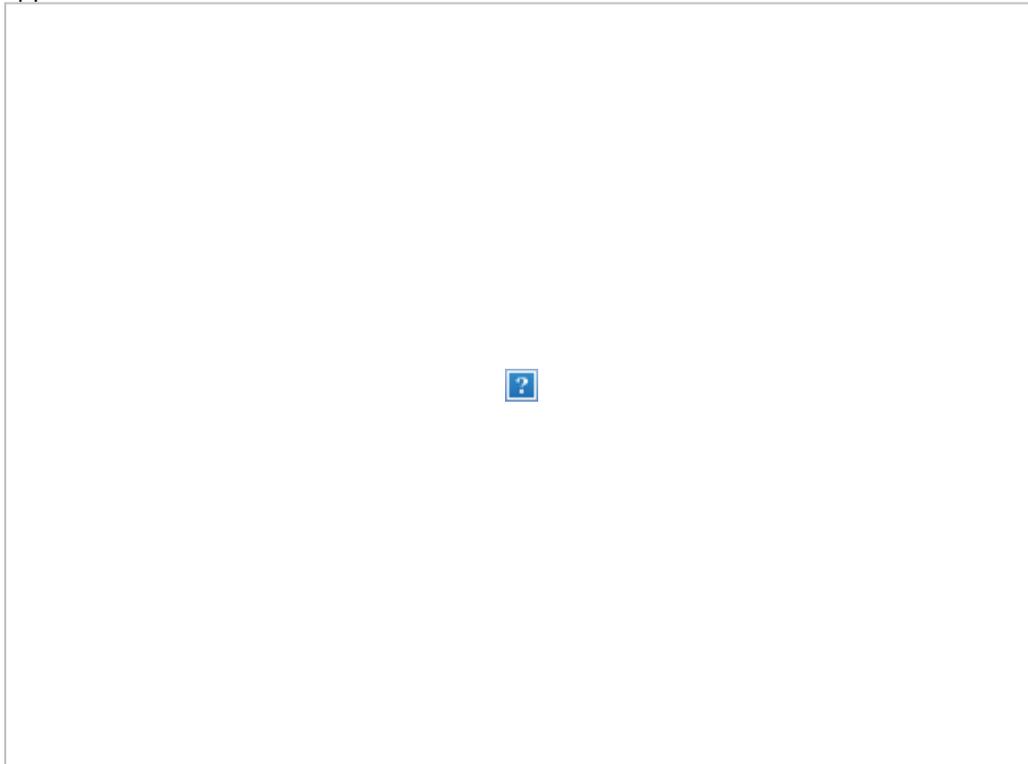
If you prefer your own menu for the scale menu, you can create it and assign it to the [ScaleMenu](#) property.

Individual members of the [ScaleMenuItem](#) collection can be accessed through design-time using the **ScaleMenuItem Collection Editor** or through code by using the [ScaleMenuItems](#) property to tie the new collection for the scale menu items to the axis scrollbar.

## Using the Designer

To add scale items to the scale menu at design time, complete the following:

1. Right-click on the **C1Chart** control and select **Properties** from its context menu.
2. In the Properties window, expand the **ChartArea** node, expand the **AxisX**, **AxisY**, or **AxisY2** node, and then expand the **ScrollBar**.
3. Click on the **ellipsis** button next to the **ScaleMenuItems** property. The **ScaleMenuItem Collection Editor** appears.



4. Click **Add** to add a new item to the collection and then set its **Scale** and **Text** properties to the desired value.
5. Click **OK** when you are finished.

## Using Code

To add custom menu items to the scale menu, you can clear the existing collection and add the scale items to the collection like the following:

### To write code in Visual Basic

#### Visual Basic

```
With Me.C1Chart1.ChartArea.AxisX.ScrollBar
    .ScaleMenuItems.Clear()
    .ScaleMenuItems.Add(0.1, "1:10")
    .ScaleMenuItems.Add(0.2, "2:10")
    .ScaleMenuItems.Add(0.3, "3:10")
    .ScaleMenuItems.Add(0.4, "4:10")
    .ScaleMenuItems.Add(0.5, "5:10")
    .ScaleMenuItems.Add(0.6, "6:10")
    .ScaleMenuItems.Add(0.7, "7:10")
    .ScaleMenuItems.Add(0.8, "8:10")
    .ScaleMenuItems.Add(0.9, "9:10")
    .ScaleMenuItems.Add(1.0, "10:10")
End With
```

```
.Scale = 0.1'initial value of scale
.Visible = True
End With
```

## To write code in C#

```
C#
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Clear();
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(0.1, "1:10");
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(0.2, "2:10");
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(0.3, "3:10");
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(0.4, "4:10");
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(0.5, "5:10");
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(0.6, "6:10");
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(0.7, "7:10");
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(0.8, "8:10");
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(0.9, "9:10");
c1Chart1.ChartArea.AxisX.ScrollBar.ScaleMenuItems.Add(1.0, "10:10");
c1Chart1.ChartArea.AxisX.ScrollBar.Scale = 0.1 ;//initial value of scale
c1Chart1.ChartArea.AxisX.ScrollBar.Visible = true;
```

The example given in the code above will clear the existing collection and add new scale menu items to the scrollbar on the x-axis. The [ScaleMenuItems](#) property is used to tie the new collection to the axis scrollbar on the chart. The first parameter to the Add method is the scale, which must be a double value between 0 and 1 inclusive. The second parameter, text, is the text that appears in the menu.

## Axis Scroll Events

AxisScroll events are provided for the C1Chart and AxisScrollBar objects to use when you need to determine more information on axis scrolling. For example, using the AxisScroll events you can provide detailed information about which axis is being scrolled, what value the min and max values are before and after scrolling through the data, what type of scrolling is being performed, and the orientation of the scrollbar.

The sender in the AxisScroll event can be either the AxisScrollBar object or the c1Chart1 object. If both the Chart and the AxisScrollBar events are set, the AxisScrollBar event fires first. However, the [AxisScroll](#) event fires for all axes, and sets the AxisId property of the AxisScrollEventArgs object to indicate the axis that changed.

The AxisScrollEventArgs class provides data for the AxisScrollEvent which fires whenever the built-in axis scrollbar changes value.

The following table provides information on the AxisScrollEventArgs properties:

Property	Description
<a href="#">AxisID</a>	An enumeration value that identifies the axis being scrolled.
<a href="#">NewValue</a>	Gets or sets the value of the axis scrollbar as it will be after event completion. The axis scrollbar value indicates the fraction between the minimum and maximum.
<a href="#">OldValue</a>	Gets the value of the axis scrollbar before the event. The axis scrollbar value indicates the fraction between the minimum and maximum.
<a href="#">ScrollEventType</a>	Gets a value indicating the type of scroll event such as ThumbPosition, ThumbTrack, EndScroll, LargeIncrement, SmallIncrement, etc.

**ScrollOrientation** Gets a value indicating the orientation of the axis scrollbar.

The following example demonstrates how to reference all of the properties from the [AxisScrollEventArgs](#) class using the AxisScroll event:

#### To write code in Visual Basic

##### Visual Basic

```
'type the Imports directive for the namespace
Imports Cl.Win.ClChart
Public Sub New()
    InitializeComponent()

    ' set the chart dock to fill.

    clChart1.Dock = DockStyle.Fill

    ' setup easy access to the scroll bar object

    Dim scrollbar As Cl.Win.ClChart.AxisScrollBar =
clChart1.ChartArea.AxisX.ScrollBar
    scrollbar.Scale = 0.1

    ' set the scale to 1/10th of the full axis width.

    scrollbar.Visible = True

    ' make the scrollbar visible.
    ' add the new AxisScroll event to the AxisScrollBar object.

    AddHandler scrollbar.AxisScroll, AddressOf XAxis_ScrollEvent

    ' add the new AxisScroll event to the Chart object.

    AddHandler clChart1.AxisScroll, AddressOf XAxis_ScrollEvent
End Sub

Public Sub XAxis_ScrollEvent(ByVal sender As Object, ByVal e As
Cl.Win.ClChart.AxisScrollEventArgs) Handles ClChart1.AxisScroll

    ' AxisScrollEventArgs are similar to that of regular scrollbars
    ' except that they also include an Axis ID.

    Dim sb As New StringBuilder()
    sb.AppendLine("" & Chr(10) & "AxisScroll Event Data")

    ' sender can be either the AxisScrollBar object or the clChart1 object.
    ' If both the Chart and the AxisScrollBar events are set, the AxisScrollBar.
    ' event fires first. However, the Chart event fires for ALL axes, and sets
    ' the AxisId property of the AxisScrollEventArgs object to indicate the axis
    that changed.
```

```

sb.AppendLine(" Sender is: " + sender.ToString())

' AxisId is an enum value that specifies which axis is involved.
' C1.Win.C1Chart.AxisIdEnum

sb.AppendLine(" ID: " + e.AxisId.ToString())

' OldValue and NewValue represent the fraction of the value change.
' from the Axis.Min to Axis.Max. OldValue is before the event.
' NewValue is the fraction that will be after the event.
' Note that NewValue is the only value that can be set. If
' is set NewValue = OldValue, then the event is effectively cancelled.

sb.AppendLine(" OldVal: " + e.OldValue.ToString())
sb.AppendLine(" NewVal: " + e.NewValue.ToString())

' EventType indicates how the scrollbar has been changed.
' SmallIncrement/SmallDecrement indicates the button was pressed.
' LargeIncrement/LargeDecrement indicates the scrollbar was pressed
' between the thumb and the button.
' TrackBar indicates that the scrollbar has been dragged.

sb.AppendLine(" EventType: " + e.ScrollEventType.ToString())
sb.AppendLine(" Orientation: " + e.ScrollOrientation.ToString())
System.Diagnostics.Debug.WriteLine(sb.ToString())

End Sub

```

**To write code in C#**

C#

```

//type the using directive for the namespace

using C1.Win.C1Chart;
public Form1()
{
    InitializeComponent();

    // set the chart dock to fill.
    c1Chart1.Dock = DockStyle.Fill;

    // setup easy access to the scroll bar object
    C1.Win.C1Chart.AxisScrollBar scrollbar = c1Chart1.ChartArea.AxisX.ScrollBar;
    scrollbar.Scale = 0.1; // set the scale to 1/10th of the full axis width.
    scrollbar.Visible = true; // make the scrollbar visible

    // add the new AxisScroll event to the AxisScrollBar object.
    scrollbar.AxisScroll += new
C1.Win.C1Chart.AxisScrollEventHandler(XAxis_ScrollEvent);

    // add the new AxisScroll event to the Chart object
    c1Chart1.AxisScroll += new

```

```
C1.Win.C1Chart.AxisScrollEventHandler(XAxis_ScrollEvent);
}

public void XAxis_ScrollEvent(object sender, C1.Win.C1Chart.AxisScrollEventArgs e)
{
    // AxisScrollEventArgs are similar to that of regular scrollbars
    // except that they also include an Axis ID.

    StringBuilder sb = new StringBuilder();
    sb.AppendLine("\nAxisScroll Event Data");

    // sender can be either the AxisScrollBar object or the c1Chart1 object.
    // If both the Chart and the AxisScrollBar events are set, the AxisScrollBar
    // event fires first. However, the Chart event fires for ALL axes, and sets
    // the AxisId property of the AxisScrollEventArgs object to indicate the
    // axis that changed.

    sb.AppendLine(" Sender is: " + sender.ToString());

    // AxisId is an enum value that specifies which axis is involved.
    // C1.Win.C1Chart.AxisIdEnum

    sb.AppendLine(" ID: " + e.AxisId.ToString());

    // OldValue and NewValue represent the fraction of the value change from the
    Axis.Min to Axis.Max. OldValue is before the event,
    // NewValue is the fraction that will be after the event.

    sb.AppendLine(" OldVal: " + e.OldValue.ToString());
    sb.AppendLine(" NewVal: " + e.NewValue.ToString());

    // EventType indicates how the scrollbar has been changed.
    // SmallIncrement/SmallDecrement indicates the button was pressed.
    // LargeIncrement/LargeDecrement indicates the scrollbar was pressed
    // between the thumb and the button.
    // TrackBar indicates that the scrollbar has been dragged.

    sb.AppendLine(" EventType: " + e.ScrollEventType.ToString());
    sb.AppendLine(" Orientation: " + e.ScrollOrientation.ToString());
    System.Diagnostics.Debug.WriteLine(sb.ToString());
}
```

The code outputs the AxisScroll event data similar to the following:

```
AxisScroll Event Data
Sender is: Cl.Win.ClChart.ClChart
ID: X
OldVal: 0.251141552511416
NewVal: 0.253678335870117
EventType: ThumbTrack
Orientation: HorizontalScroll

AxisScroll Event Data
Sender is: Cl.Win.ClChart.AxisScrollBar
ID: X
OldVal: 0.253678335870117
NewVal: 0.256215119228818
EventType: ThumbTrack
Orientation: HorizontalScroll

AxisScroll Event Data
Sender is: Cl.Win.ClChart.ClChart
ID: X
OldVal: 0.253678335870117
NewVal: 0.256215119228818
EventType: ThumbTrack
Orientation: HorizontalScroll
```

## Axis Logarithmic Scaling

When data is shown with large differences in scale or when data is expected to vary exponentially on the same chart, it is often convenient to use logarithmic scaling on one or more axes. On a logarithmic axis, equal distance along it depicts an equal percentage change. Each axis can be set to scale logarithmically by setting the `IsLogarithmic` property to **True**. If logarithmic scaling is used on one or both axes, the chart is called a Log Scale chart.

With logarithmic scaling, values are physically spaced based upon the logarithm of the value instead of the value itself. This is useful when quantities are charted over a very wide range, and when depiction of geometric and/or exponential relationships is desired.

Unlike **arithmetic charts** where changes are measured in terms of direct units, **log scale charts** show changes in terms of percentage change. For example, in a **log scale chart** measuring dollars, a change from \$1 to \$2 is a 100 percent change so the distance on the chart axis from \$1 to \$2 would be the same from \$50 to \$100. Whereas in an **arithmetic chart**, a change from \$50 to \$100 would make the distance on the axis from \$50 to \$100 appear much larger on the chart because it is a change of \$50 as opposed to just \$1.

## Commonly Used Logarithms

Logarithms can be expressed using any base value, including integers and floating point values. The two most commonly used types of logarithms include:

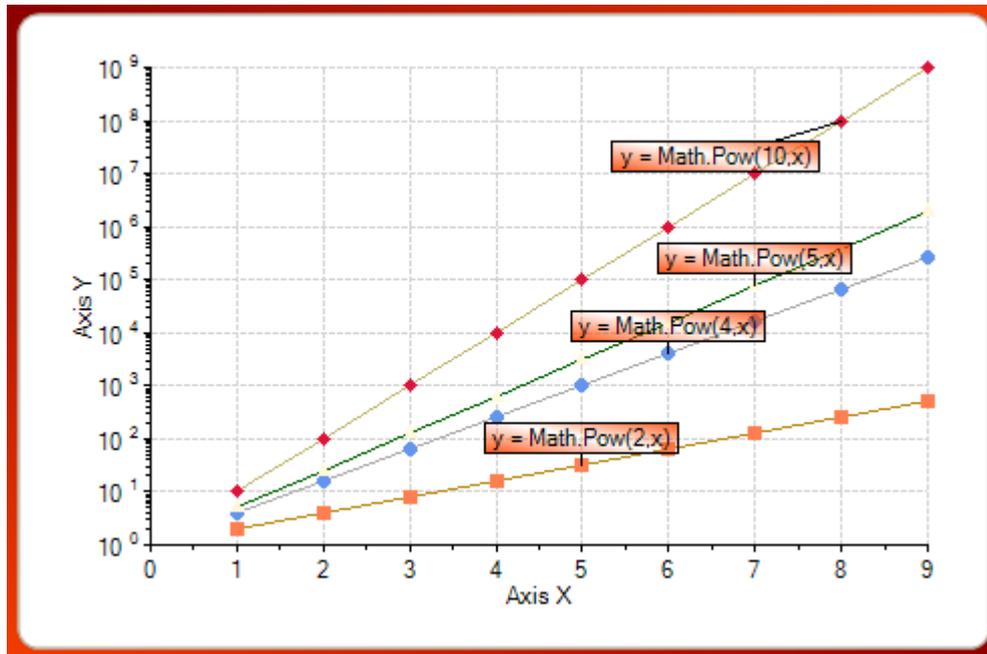
- **Common logarithms** – Use 10 as the base so its written as  $\log 100 = 2$ .
- **Natural logarithms** – Use the mathematical constant  $e$  as the base.

## Logarithmic Base

When the `IsLogarithmic` property is set to true you can specify the value for the logarithmic base using the `LogarithmicBase` property. The default value is 10. If a natural log scale is desired, set the `LogarithmicBase` to a

value less than or equal to 1. If the value is 1 then the natural logarithm of 1 would be 0, since  $e^0 = 1$ . A natural logarithm is the logarithm to the base e. Note that Logarithmic scaling does not make mathematical sense when values are less than or equal to zero. Therefore, negative and zero values are not plotted against axes that have the `IsLogarithmic` property set to true.

The following image shows how the **C1Chart** appears when the `LogarithmicBase` is set to its default value 10, which is the common logarithm:



## Sample Available

For a complete sample that demonstrates how to use logarithmic scaling on C1Chart, see the sample, **LogPlots** located on <http://our.componentone.com/samples/?action=viewcontrol&control=7&platform=1>.

## Logarithmic scaling output for the annotations on the axis

The following format works for any log base used in C1Chart, if:

- The `AnnoFormat` property = `NumericManual`
- The `AnnoFormatString` property contains `"**"`.

If both of these are true, then the string is output and the log in the logarithmic base is output to `{0}`.

So if the following values exist:

- `LogarithmicBase` = 10,
- `AnnoFormat` = `NumericManual`,
- `AnnoFormatString` = `"10**{0}"`

Then the annotations on the axis are displayed as:  $power_{10}$  where "power" is a superscript.

## Criteria used for Logarithmic Scaling

The following additional criteria must be following for logarithmic axes:

- Any data that is less than or equal to zero is not graphed (it is treated as a data hole), since a logarithmic axis only handles data values that are greater than zero. For the same reason, axis and data minimum/maximum bounds and origin properties cannot be set to zero or less.
- Axis numbering increment, ticking increment, and precision properties have no effect when the axis is logarithmic.
- For a logarithmic X-axis, the chart type must be either plot, bubble, area, HiLo, HiLoOpenClose or candle. For the Y-axis, the chart type must be either plot, bubble, area, polar, HiLo, HiLoOpenClose, candle, radar or filled radar.
- The annotation method for the X-axis cannot be TimeLabels.

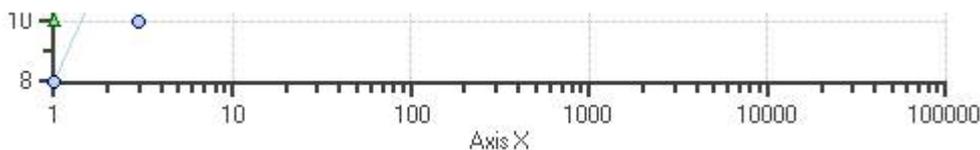
## UnitMajor and Logarithmic Axes

For logarithmic axis scaling, [UnitMajor](#) is taken as a multiplier of the base value of each cycle that provides a hint as to the annotation spacing within each cycle of the logarithmic base. That is  $(\text{UnitMajor} * \text{base cycle value})$  is approximately the annotation value increment within each cycle. For integer logarithmic base values, the result is usually exact. For floating point values, annotations are rounded to nice numbers as for linear scaling.

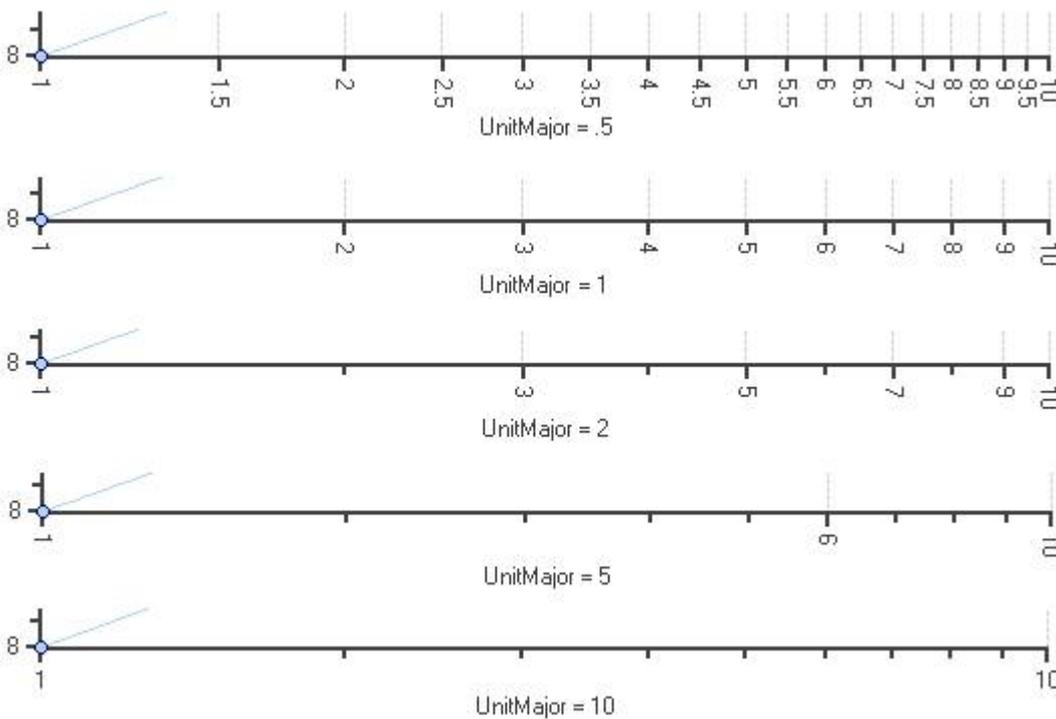
## Detailed Explanation of UnitMajor and Logarithmic Axes

Often, when logarithmic scales are used, the bounds of a chart axis will span multiple cycles of the logarithmic base. In these cases, the usual linear specification of [UnitMajor](#) no longer makes sense, as a value appropriate for a given cycle makes little sense for the previous or next cycles. For the [UnitMajor](#) setting to be of value, it must pertain to values relative to each cycle of the logarithmic base.

If this doesn't make sense to you, think about what single, fixed, incremental value you might use for the following axis:



Following the above reasoning, for logarithmic axes, the chart assumes that [UnitMajor](#) specifies the fraction of the base value for each cycle. Consider the following examples:



In each case, the base cycle value is 1. For each cycle the next annotation value = previous number + (base value of cycle \* UnitMajor). The maximum value of the UnitMajor is the LogarithmicBase. The automatic value of UnitMajor is always the LogarithmicBase.

When all of the annotation values are calculated, a nice rounding algorithm is applied so the numbers are relatively easy to read. The behavior may seem a bit odd, but it is the result of accommodating any logarithmic base while at the same time obtaining numbers for the annotations that are reasonable to read.

For example, the plots above are log-base 10 values, but there are also natural-logs to consider such as log-base 2, log-base-x, etc.

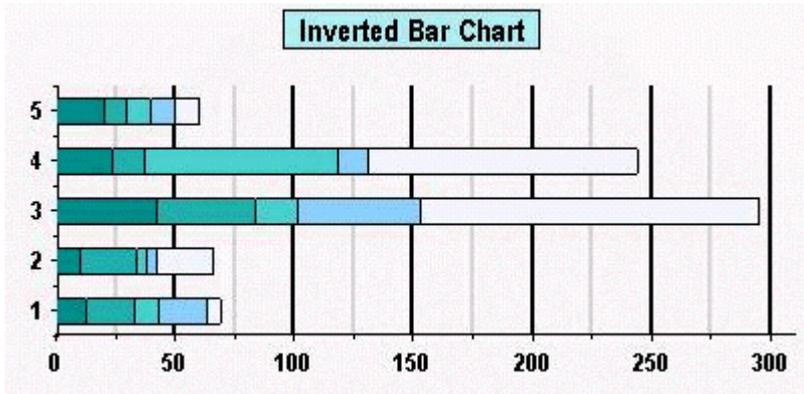
## Inverted and Reversed Chart Axes

When a data set contains X or Y values which span a large range, sometimes the normal chart setup does not display the information most effectively. Formatting a chart with a vertical Y-axis and axis annotation that begins at the minimum value can sometimes be more visually appealing if the chart could be inverted or the axes reversed.

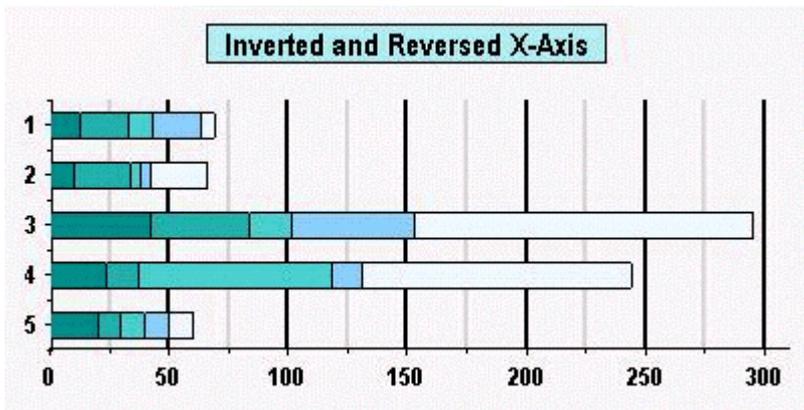
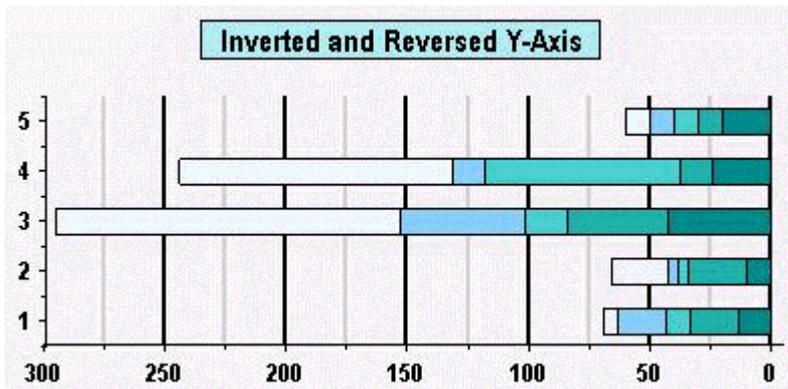
Therefore, C1Chart provides the Inverted property of the ChartArea and the Reversed property of the axis.

Setting the Inverted property of the ChartArea to **True** will invert the axes. This means that the X-axis will take the place of the Y-axis, and the Y-axis will take the place of the X-axis. Initially the chart is setup with the X-axis displayed horizontally and the Y-axis displayed vertically. This feature is commonly used in Bar Charts.

**Note:** An Inverted Bar Chart is a standard Bar Chart with the Inverted property set to **True**. Thus, Inverted Bar Charts do not have a special ChartType.



Setting the [Reversed](#) property of the ChartArea to **True** will reverse the axes. This means that the Max side of the axis will take the place of the Min side of the axis, and the Min side of the axis will take the place of the Max side of the axis. Initially, the chart displays the Minimum value on the left side of the X-axis, and on the bottom side of the Y-axis. Setting the [Reversed](#) property of the Axis, however will juxtapose these Maximum and Minimum values.



## Axes Annotation

The annotation along each axis is an important part of any chart. The 2D chart annotates the axes with numbers based on the data entered into the ChartDataArray objects.

The chart automatically produces the most natural annotation possible, even as chart data changes. Annotation properties can be modified to perfect this process.

The following properties represent the format and the layout for the annotation of the axes in [C1Chart](#):

Property	Description
----------	-------------

<a href="#">AnnoFormat</a>	A set of predefined formats used to format the values displayed next to the axis.
<a href="#">AnnoFormatString</a>	The .NET formatting string used to format the values displayed next to the axis when AnnoFormat is set to "NumericManual" or "DateManual". If the AnnoFormat is set to either NumericManual or DateManual and the AnnoFormatString is empty, then the chart uses an algorithm to find the "best" formatting available.
<a href="#">AnnoMethod</a>	Determines what values are displayed next to the axis. Options are "Values", which displays the actual series values, or "ValueLabels", which displays the elements in the ValueLabels collection.
<a href="#">ValueLabels</a>	A collection of text/value pairs to display next to the axis when AnnoMethod is set to "ValueLabels". This property is useful when you want to display strings along an axis instead of numeric values (for example, you may be charting product prices and want to display the product names along the X-axis).
<a href="#">AnnotationRotation</a>	Allows you to rotate the values so they take up less space along the axis.
<a href="#">NoAnnoOverlap</a>	Enables you to determine whether axis annotations are permitted to overlap.

**C1Chart** includes the following types of axis annotation:

- Values Annotation
- ValueLabels Annotation
- Mixed Annotation

The next three topics explain the different types of Axis annotations as well as location and rotation for the axes annotations.

## Values Annotation

Values Annotation is an implementation where the chart automatically generates numeric annotation based on the data itself. Values Annotation can be used for any axis, with any chart type, and with any data layout. It is controlled by the following properties of the axis:

Property	Description
<a href="#">AnnoFormat</a>	Gets or sets the axis annotation format.
<a href="#">AnnoFormatString</a>	Gets or set the annotation format string used with manual formats.

Setting the **AnnoFormat** property to one of the [FormatEnum](#) values will format the data entered into the array data into one of .NET's twenty-two formats. Setting AnnoFormat to either **FormatEnum.NumericManual** or **FormatEnum.DateManual** allows a format string to be entered into the [AnnoFormatString](#) property. Manual format strings are designated in the .NET framework and below are links to more information:

## Date Time Format Strings

The Date Time format strings are divided into two categories:

1. Standard Date Time Format Strings
2. Custom Date Time Format Strings

## Custom Numeric Format Strings

You can also customize your format strings by using the custom numeric format strings.

If `AnnoFormat` is set to `FormatEnum.NumericManual` and `AnnoFormatString` is empty, then the numeric values are formatted the same as `FormatEnum.NumericGeneral`.

If `AnnoFormat` is set to `FormatEnum.DateManual` and `AnnoFormatString` is empty, then the date and time values are formatted as appropriate for the time span of the axis as defined by the axis maximum and minimum values.

## Value Labels Annotation

A very flexible type of annotation, `ValueLabels` displays text defined at a specific axis coordinate. This annotation is useful in labeling only specific coordinates or when producing annotation in a form that the chart does not provide. `ValueLabels` annotation can be used for any axis, with any chart type, and with any data layout, with the exception of Pie charts.

Non-stacked Pie charts can display labels for each pie/point, by using the X axis `ValueLabels`. The `ValueLabels` are applied to pies or points in the order they appear in the `ValueLabel` collection. The labels are only displayed if the X axis `AnnoMethod` property is set to `ValueLabels`. The display of the labels is also controlled by the X axis properties, including `Alignment`, `AnnoMethod`, `Compass`, `Font`, `ForeColor`, `Rotation`, `ValueLabels`, `VerticalText` and `Visible` properties. No other X axis properties affect the labels. The `Inverted` property also affects the label display, in that it determines whether the X axis `Compass` property values will be North, South or East, West.

If the `AnnoMethod` property is set to `AnnotationMethod.ValueLabels` the chart places labels at explicit locations along an axis. The `ValueLabels` property, which is a `ValueLabels` collection, supplies this list of strings and their locations. For example, the following code sets chart labels at the locations 1, 2 and 3:

### To write code in Visual Basic

#### Visual Basic

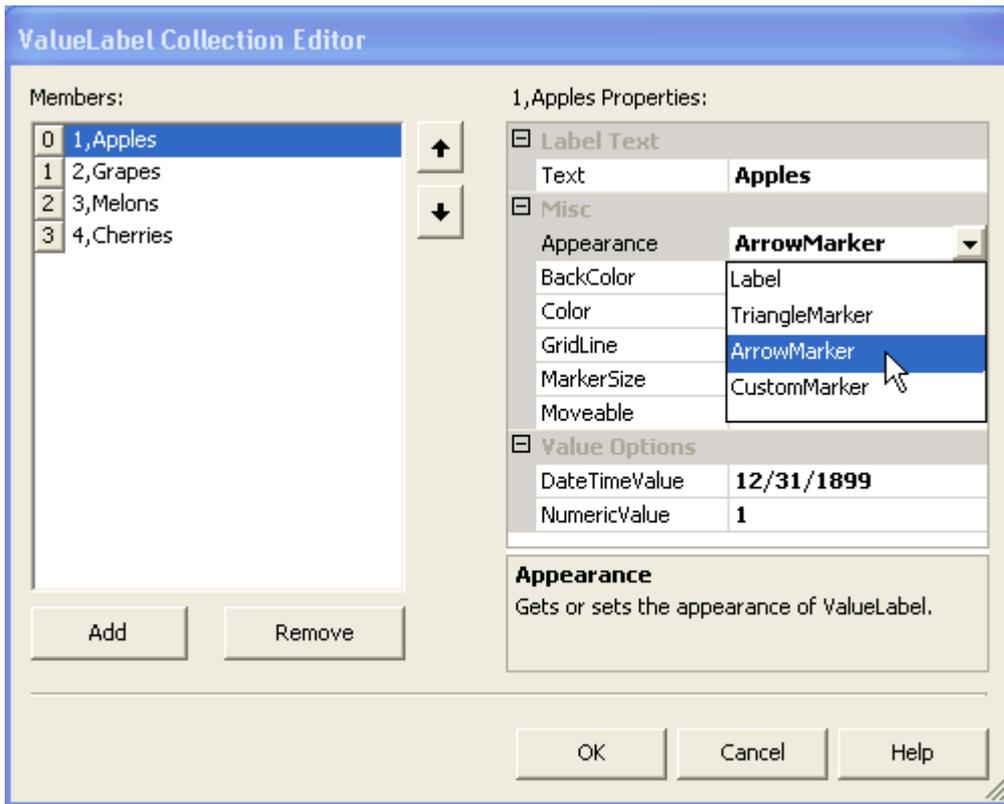
```
With c1Chart1.ChartArea.AxisX.ValueLabels
    .Add(1, "one")
    .Add(2, "two")
    .Add(3, "three")
End With
c1Chart1.ChartArea.AxisX.AnnoMethod = AnnotationMethodEnum.ValueLabels
```

### To write code in C#

#### C#

```
ValueLabelsCollecton vlColl = c1Chart1.ChartArea.AxisX.ValueLabels;
vlColl.Add(1, "one");
vlColl.Add(2, "two");
vlColl.Add(3, "three");
c1Chart1.ChartArea.AxisX.AnnoMethod = AnnotationMethodEnum.ValueLabels;
```

`ValueLabels` can also be modified at design time through the **ValueLabel Collection Editor**. For more information on the **ValueLabel Collection Editor**, see [ValueLabel Collection Editor](#).



The following chart illustrates the [Text](#), [Appearance](#), [BackColor](#), [Color](#), [MarkerSize](#), and [Moveable](#) properties from the ValueLabel class.

Notice that label's appearance for the numeric value's 1 and 2 are displayed as an arrow. You can set the appearance property for your label at design time or run time.

At design time you can set the label's appearance property by selecting one of the following label types: **Label**, **Triangle Marker**, **ArrowMarker**, or **CustomMarker** in the **ValueLabel Collection Editor**. The appearance property of the label class gets the value of the **ValueLabelAppearanceEnum**. At run time you can set the label's appearance property by using the following code:

### To write code in Visual Basic

Visual Basic

```
vl.Appearance = ValueLabelAppearanceEnum.ArrowMarker
```

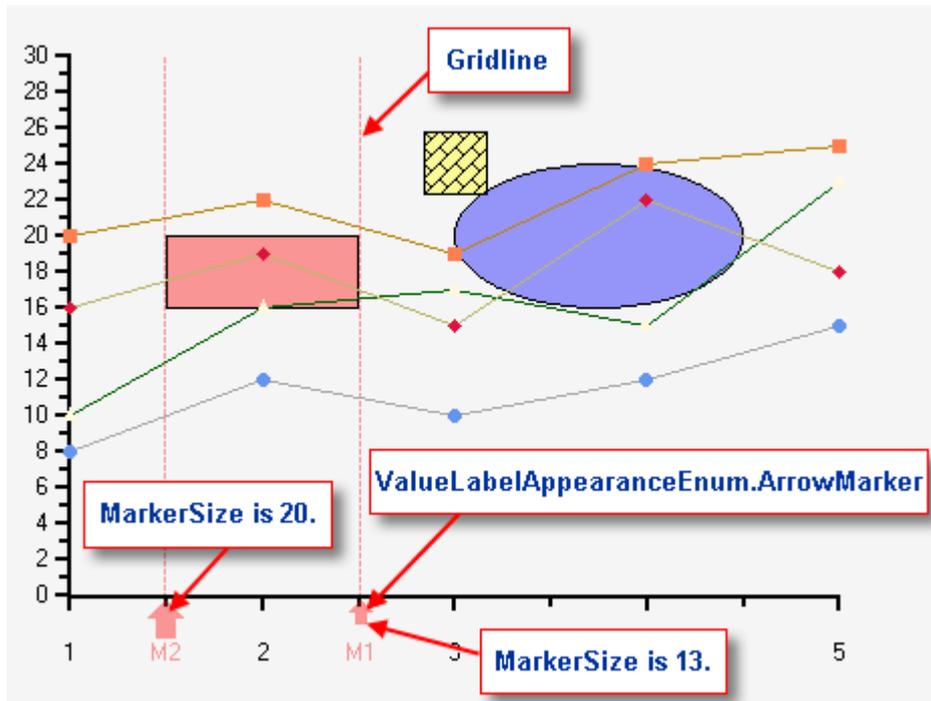
### To write code in C#

C#

```
vl.Appearance = ValueLabelAppearanceEnum.ArrowMarker;
```

**Note:** In the example above the text, vl, represents the variable name of the value label.

The [MarkerSize](#) property illustrated below shows how the markers appear when they are set to different sizes. By setting the [Moveable](#) property to **True** for each marker label, you can slide **M1** and **M2** along the x-axis. This is often useful as an input device for specifying axis values that affect charts with Cartesian coordinates. The two grid lines associated with the ValueLabels below help serve as a visual aid for the two markers.



Individual members of the [ValueLabels](#) collection can be accessed in one of three ways: by typing in the ValueLabel index (0-based), by referencing the ValueLabel value to be replaced, or by referencing the ValueLabel text value. For instance, the following three lines of code all set the first member of the ValueLabels collection that has a Value of 1 to the text "one":

### To write code in Visual Basic

#### Visual Basic

```
C1Chart1.ChartArea.AxisX.ValueLabels(0).Text = "one"
C1Chart1.ChartArea.AxisX.ValueLabels(1.0).Text = "one"
C1Chart1.ChartArea.AxisX.ValueLabels("one").Text = "one"
```

### To write code in C#

#### C#

```
c1Chart1.ChartArea.AxisX.ValueLabels[0].Text = "one";
c1Chart1.ChartArea.AxisX.ValueLabels[1.0].Text = "one";
c1Chart1.ChartArea.AxisX.ValueLabels["one"].Text = "one";
```

## Mixed Annotation

A very flexible type of annotation, Mixed includes automatic annotations and value labels. Each type of annotation will result in visible text defined at a specific axis coordinate. This annotation allows superposition of value and value labels.

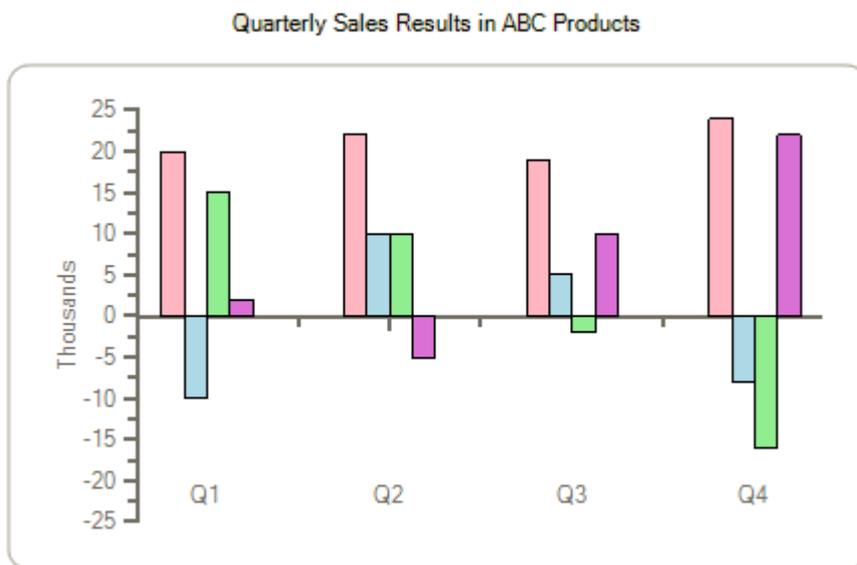
## Axis Annotation Location

The location can be set for the axis annotations by choosing one of the following values in the [TickLabelsEnum](#) enumeration.

Value	Description
<b>TickLabelsEnum.None</b>	No annotations along the axis.
<b>TickLabelsEnum.High</b>	Annotations are drawn near the maximum value of the perpendicular axis, and inside the plot area. For X-Axis annotation, if data is not available for Group0 and is available for Group1, the Y2 axis determines the location, otherwise the Y axis is used. Annotations which overlap the crossing axis are eliminated.
<b>TickLabelsEnum.Low</b>	Annotations are drawn near the minimum value of the perpendicular axis, and inside the plot area. For X axis annotations, if the data is not available for Group0 and is available for Group1, the Y2 axis determines the location, otherwise the Y axis is used. Annotations which overlap the crossing axis are eliminated.
<b>TickLabelsEnum.NextToAxis</b>	Annotations are drawn next to the axis. This is the default value.

High and Low specify that maximum and minimum positions of the cross axis. For example, if High is specified for the X axis TickLabelEnum, then the annotations are placed near the Maximum value of the Y axis, which is not necessarily the top of the chart (*Reversed* = true).

The High or Low values are typically used if you have values below the Axis-Y origin. In the following example, specifying `AxisX.TickLabels = TickLabelsEnum.Low` has placed them near the -25 value of the Y axis.



To programmatically set the value for the TickLabelsEnum:

### To write code in Visual Basic

```

Visual Basic
Dim ax As Axis = c1Chart1.ChartArea.AxisX
ax.TickLabels = TickLabelsEnum.Low
    
```

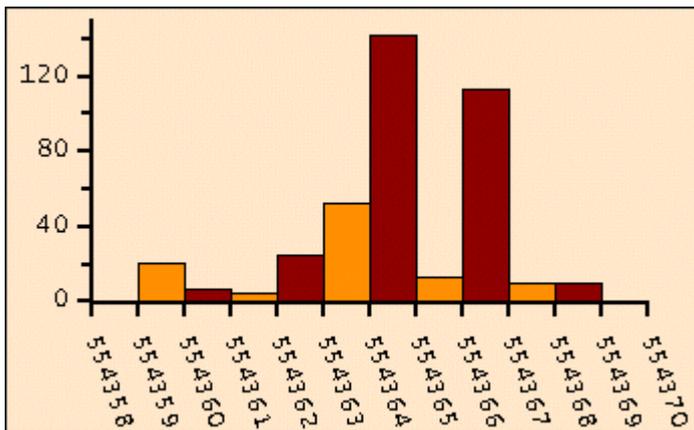
### To write code in C#

```

C#
Axis ax = c1Chart1.ChartArea.AxisX;
Ax.TickLabels = TickLabelsEnum.Low;
    
```

## Axis Annotation Rotation

Use the axis [AnnotationRotation](#) property to rotate the axis annotation counterclockwise the specified number of degrees. This property is especially useful if the X-axis is crowded with annotation. Rotating the annotations +/- 30 – 60 degrees allows a much larger number of annotations in a confined space on horizontal axes. By utilizing the [AnnotationRotation](#) property, the X-axis annotation does not overlap, as shown below:

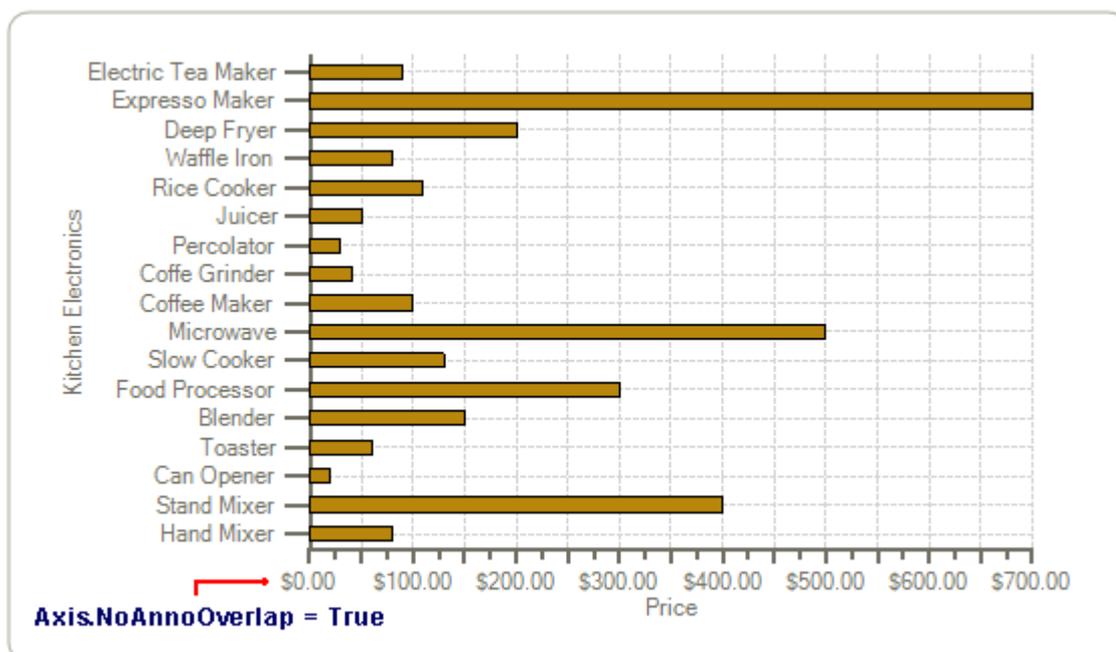


## Axis Annotation Overlap

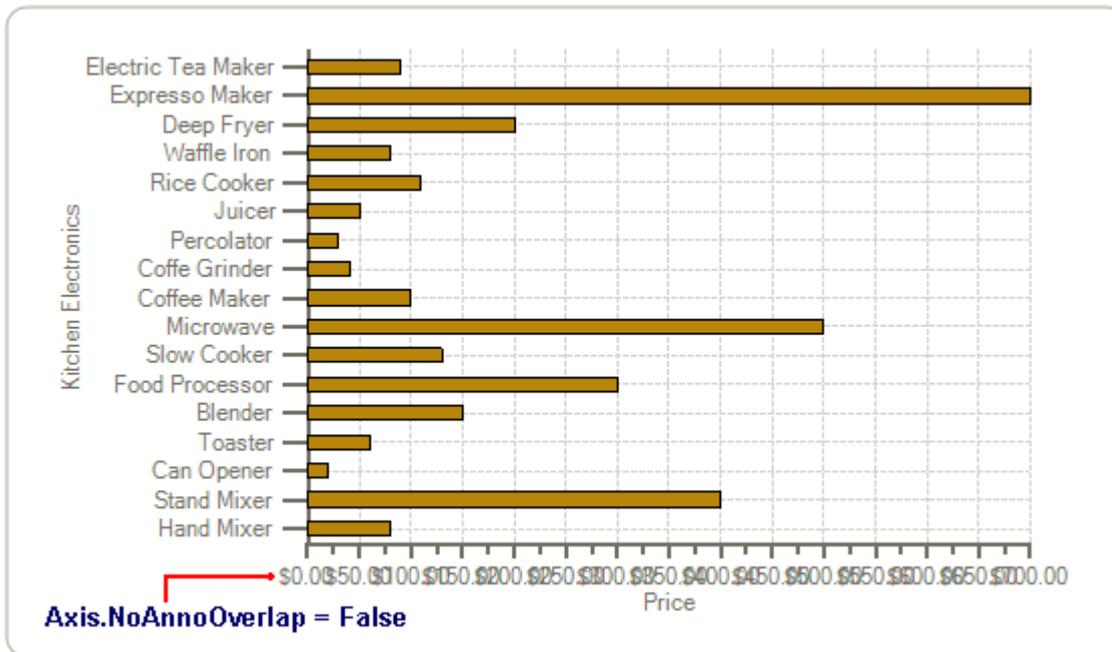
In cases when you don't have enough room on the axes to display your annotation you can set the [NoAnnoOverlap](#) property to True so it will suppress the drawing of any axis annotation that would overlap a previously drawn annotation regardless of the reason for the overlap.

If you have [AutoMajor](#) property set to true then you don't need to use the [NoAnnoOverlap](#) property because the AutoMajor property will automatically calculate the best value for the major tick marks.

The following chart displays the [NoAnnoOverlap](#) property set to True:



The following chart display the `NoAnnoOverlap` property set to False:



## Plot Area

The data is plotted in the plot area of the chart. In the PlotArea object you can customize or create styles for the PlotArea. For instance you can use the `BackColor` and `BackColor2` to specify the back colors of the plot area. You can also set the `ForeColor` of the PlotArea by setting the `ForeColor` property.

The table below shows the Plot area's appearance properties and their functions:

Property	Description
<code>AlarmZones</code>	The AlarmZones property gets the AlarmZones Collection object associated with the current PlotArea object.
<code>BackColor</code>	Gets or sets the back color of the PlotArea. Inherits from the ChartArea.
<code>BackColor2</code>	Gets or sets the second back color.
<code>Boxed</code>	Gets or sets whether the PlotArea is enclosed in a box.
<code>ForeColor</code>	Gets or sets the ForeColor of the plot area. Inherits from the ChartArea.
<code>GradientStyle</code>	Gets or sets the type of gradient background filling.
<code>HatchStyle</code>	Gets or sets the type of hatch background filling.
<code>Opaque</code>	Gets or sets whether the PlotArea background is opaque.
<code>Size</code>	Gets the size of the PlotArea in chart control client coordinates.
<code>View3D</code>	Gets the View3D object.
<code>Visible</code>	Gets or sets the PlotArea visibility.

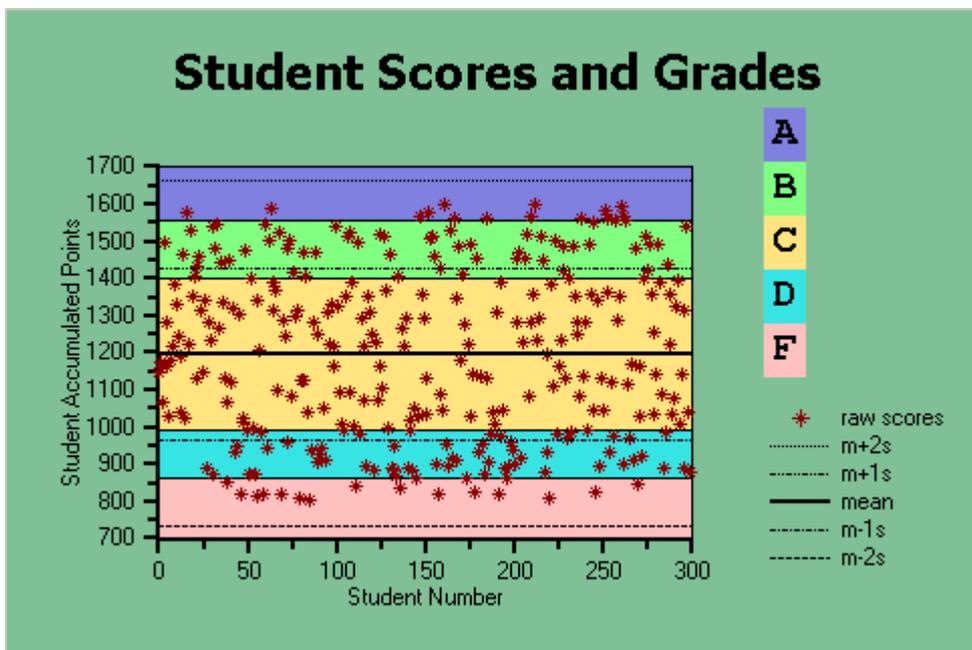
`C1Chart` includes an **AlarmZone Collection Editor** that can be accessed through `C1Chart`'s properties at design-time. The **AlarmZone Collection Editor** consists of a windows form which conveniently allows the users to edit/create

AlarmZones. Using the editor, the user can add/remove one or many multiple alarm zones, and modify or set the properties for each alarm zone. For more information on the **AlarmZone Collection Editor**, see [AlarmZone Collection Editor](#).

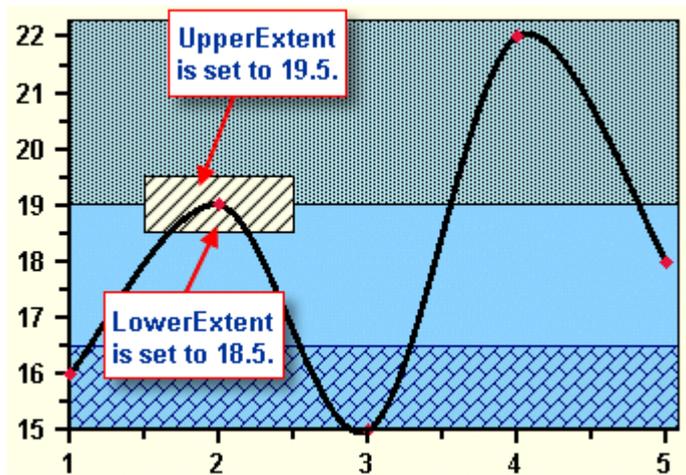
The next section explains what an alarm zone is and how it is used in a chart.

## Alarm Zones

Alarm Zones are a series of bands or shapes that can be placed behind the plotted data, but in front of the chart background. Generally, Alarm Zones are used in much the same manner as grid lines, but the ability to modify the Alarm Zones allows them to be more useful and visually appealing. Also, Alarm Zones can be used to highlight important **y-values** in charts. For example, the chart below uses five different alarm zones (A, B, C, D, F) that represent the student's grades. Each Alarm Zone is shown in a different color to avoid confusion with the other Alarm Zones. Notice how the Alarm Zones help in showing the important **y-values** (in this case, the students' accumulated points/grades).



By using the [UpperExtent](#) and [LowerExtent](#) properties, each Alarm Zone band can be adjusted to specific values. The graphic below displays an Alarm Zone band adjusted to a low **y - value** of **18.5** and a high **y - value** of **19.5** by using the [LowerExtent](#) and [UpperExtent](#) properties.



 **Note:** Alarm Zones are most effectively used with **XY-Plot**, **Bar**, **Stacking Bar** and **Candle** charts.

You can specify the shape of your alarm zone by using the [Shape](#) property. You can set an Alarm Zone to an **elliptical**, **rectangle**, or **polygon** shape. The [Shape](#) property can be set at design time in the **AlarmZone Collection Editor**, or at run time. In code, you'll use the **AlarmZoneShapeEnum** property to get the **elliptical**, **rectangle**, or **polygon** shape. The code below creates a rectangular shape for the alarm zone.

#### To write code in Visual Basic

Visual Basic

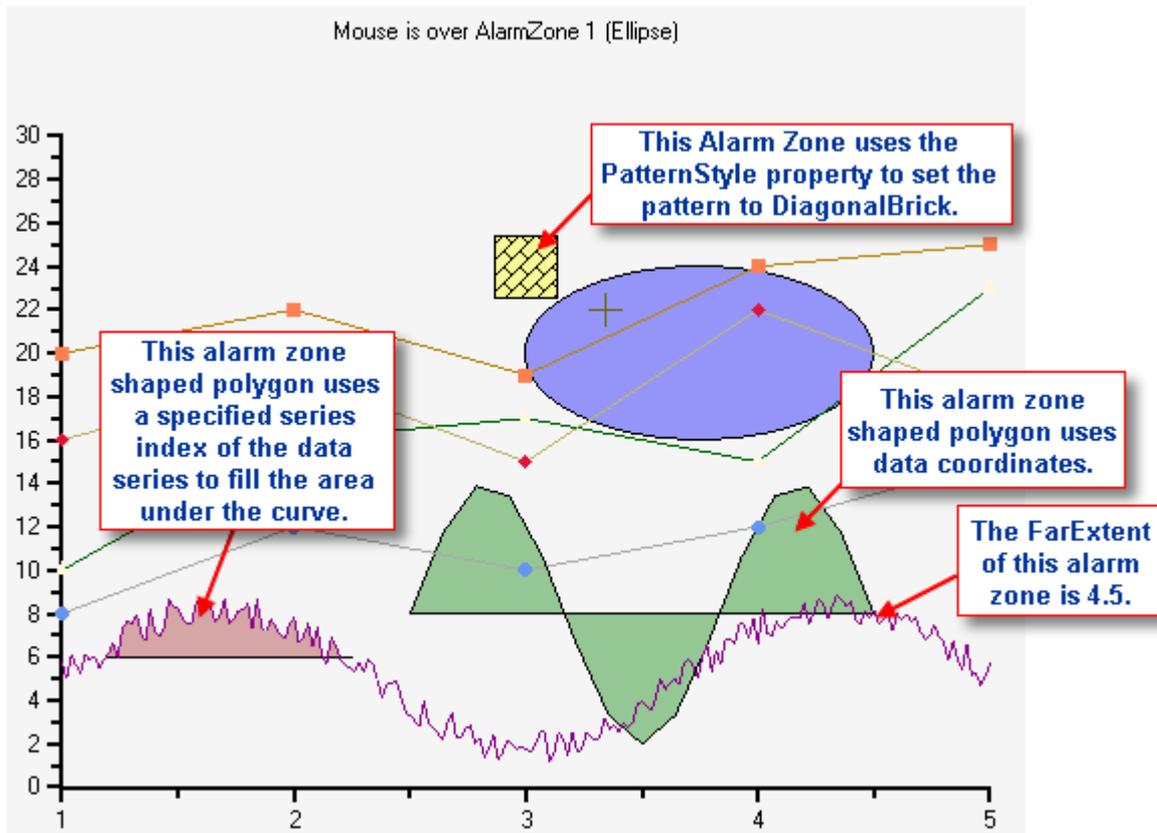
```
alarmzone.Shape = AlarmZoneShapeEnum.Rectangle
```

#### To write code in C#

C#

```
alarmzone.Shape = AlarmZoneShapeEnum.Rectangle;
```

The chart below illustrates several alarm zone properties. Notice how the text below states what alarm zone the cursor is pointing to. This technique can be achieved using the **AlarmZoneAtCoord** method. The [AlarmZoneAtCoord](#) method retrieves a reference to the foremost AlarmZone that lies under the specified coordinates. The array and DataSeries are two different source types used for the polygon data. In the chart below the green polygon uses the array source to get the polygon coordinates filled in by the user. The other polygon uses the DataSeries to get the specified series index from the user.



Alarm Zones support the following shapes: rectangle, elliptical, and polygon. When a polygon is used the data for the polygon can be specified explicitly or by specifying a data series (XY Plots only). This allows for easy filling of specified areas under a curve, just like the polygon in the chart above.

## Adding Alarm Zones

AlarmZones can be added at design time through the **AlarmZonesCollection Editor** in the Visual Studio Properties window or programmatically through the AlarmZones object.

### To programmatically add an AlarmZone member to the AlarmZonesCollection:

#### To write code in Visual Basic

```
Visual Basic
c1Chart1.ChartArea.PlotArea.AlarmZones.AddNewZone()
```

#### To write code in C#

```
C#
c1Chart1.ChartArea.PlotArea.AlarmZones.AddNewZone();
```

### To access the collection editor at design time:

1. Right-click on the **C1Chart2D** control and select Properties from its context menu.
2. In the Properties Window, expand the ChartArea node in the Properties window, then expand the PlotArea node, and click the **ellipsis** next to the **AlarmZones** property.

For more information on the **AlarmZone Collection Editor**, see [AlarmZone Collection Editor](#).

## Defining Alarm Zones Properties

Alarm Zones have two basic sets of properties. The first set of properties determines the boundaries of the Alarm Zone in reference to the chart's data values. Setting the [NearExtent](#) and [FarExtent](#) properties to values within the bounds of the X-axis sets the **Left** and **Right** boundaries of the AlarmZone. Conversely, setting the [UpperExtent](#) and [LowerExtent](#) properties to values within the bounds of the **Y-axis** sets the **Upper** and **Lower** boundaries of the Alarm Zone. For instance, to set the green AlarmZone in the above image the [NearExtent](#) would be 2.5, the [FarExtent](#) would be 4.5, the [LowerExtent](#) would be 8.0, and the [UpperExtent](#) would be 14.0. Notice that the names of these properties are [UpperExtent](#), [LowerExtent](#), [NearExtent](#), and [FarExtent](#) rather than **X**, **Y**, **Height**, and **Width**. One important benefit to Alarm Zones is that they are not static, but actually tied to the chart's data. This means that as the data changes, the Alarm Zones will change in relation to the data and not like one of the chart's visual attributes. To prevent the AlarmZones from vanishing when the data changes you can set the [MinHeight](#) and [MinWidth](#) properties. Also, the [MinHeight](#) and [MinWidth](#) properties allows the placement of an AlarmZone with fixed dimensions.

The second set of properties determines the style characteristics of the Alarm Zone. The [Shape](#), [BackColor](#) and [PatternStyle](#) properties can be set for the Alarm Zone. You can choose from three different types of shapes for Alarm Zones: **rectangle**, **ellipse**, and **polygon**. If you create a polygon shaped alarm zone you need to specify the [PolygonSource](#) property from the **AlarmZone Collection Editor** at design time or at run time. Setting the [ForeColor](#) property sets the color of the pattern specified by the [PatternStyle](#) property.

## Customizing Chart Elements

When the chart's data and axes are formatted properly, its elements can be customized to make it look clearer and more professional. The following topics cover tasks that customize the appearance of a chart.

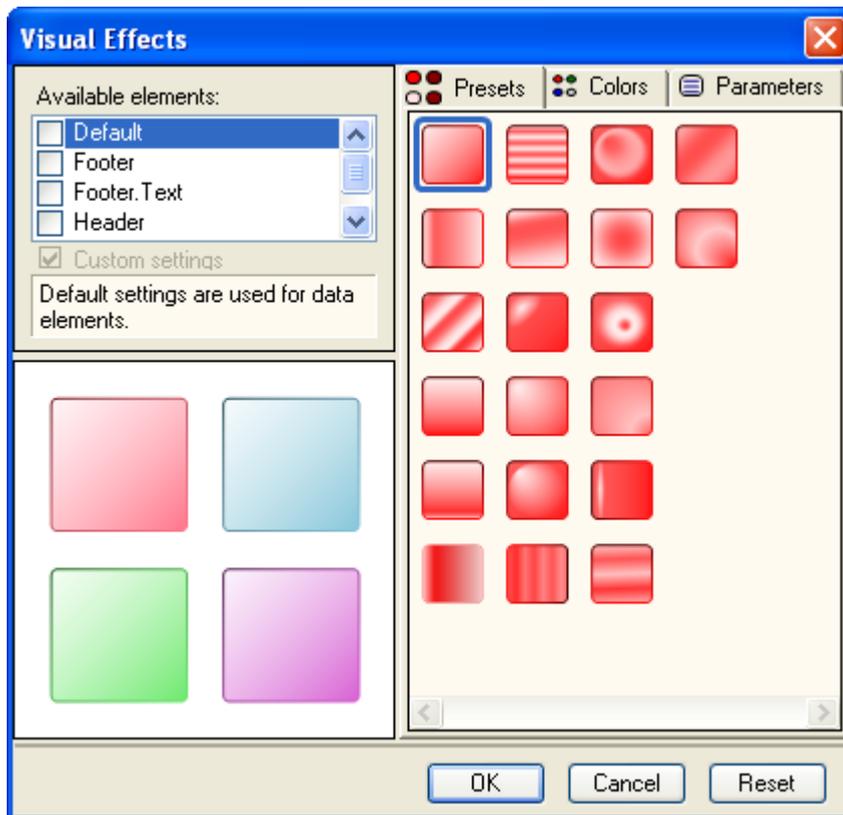
## Visual Effects Designer

**Visual Effects** is a tool used for visually enhancing the Chart2D control's elements such as the data series, header, and footer. Any existing project can use the new features provided by this tool. The chart's appearance can dramatically improve in a few simple steps using the **Visual Effects** designer.

### Limitation

VisualEffects rendering may not work effectively in case of very complex and large data arrays or when highest performance is required.

When you run the interactive **Visual Effects** designer it appears like the following.



For more information on accessing the **Visual Effects** designer, see [Access the Visual Effects Designer](#).

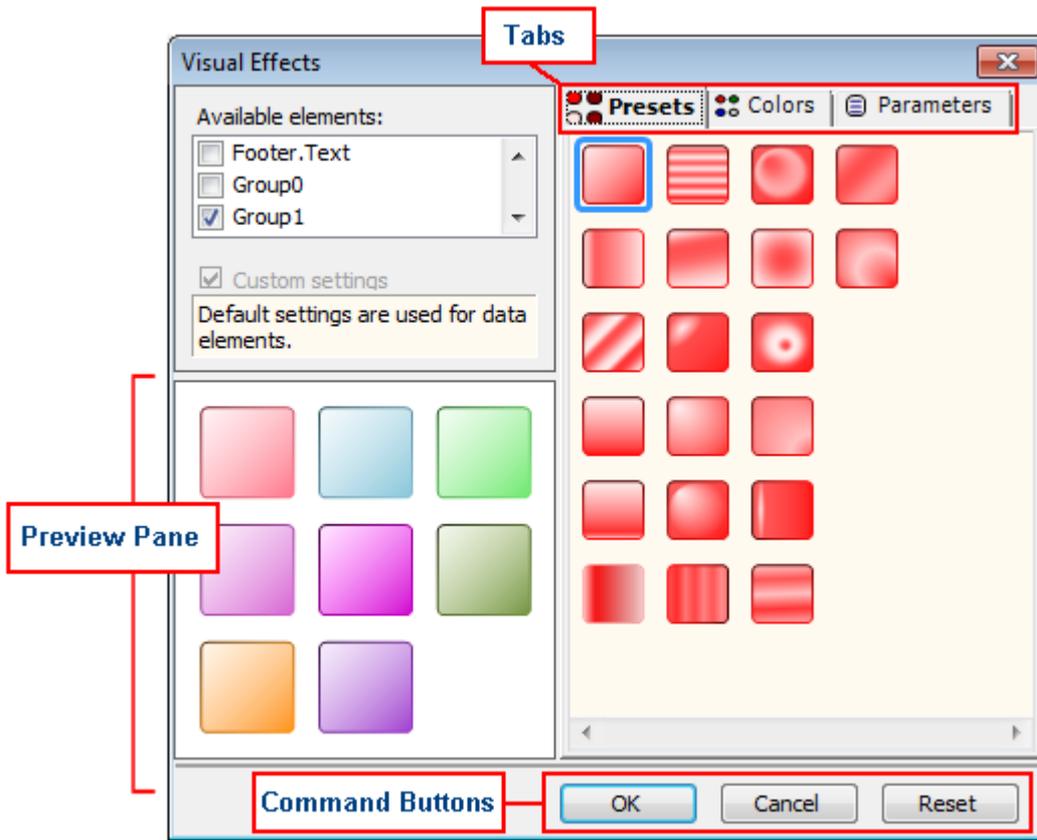
## Navigating the Visual Effects Designer

The **Visual Effects** designer has a very intuitive interface.

The designer consists of two parts. The first part appears on the left side of the **Visual Effects** designer. It displays a list box of chart elements that support VisualEffects rendering and a preview pane that shows the current element's appearance.

The second part appears on the right side of the **Visual Effects** designer. The second part contains three tabs: **Presets**, **Colors**, and **Parameters**.

In addition to the Chart elements, Preview Pane, and tab pages there is also command buttons located at the bottom of the designer. The command buttons consist of an **OK**, **Cancel**, and **Reset** button. The **OK** button closes the **Visual Effects** designer and saves all of the changes. The **Cancel** button cancels the **Visual Effects** designer and cancels the changes made in the editing mode. The **Reset** button resets the value of the selected property.



## Visual Effects Elements

The Visual Effects elements may be applied independently to different chart elements. Each element that supports Visual Effects can have its own custom settings or inherit them from the parent element.

The following table provides a description for each available element in the **Visual Effects** designer:

Element name	Parent element	Description
<b>Default</b>		Used for painting all data elements.
<b>Header</b>	Default	Header background.
<b>Header.Text</b>	Header	Header text
<b>Footer</b>	Default	Footer background.
<b>Footer.Text</b>	Footer	Footer text.

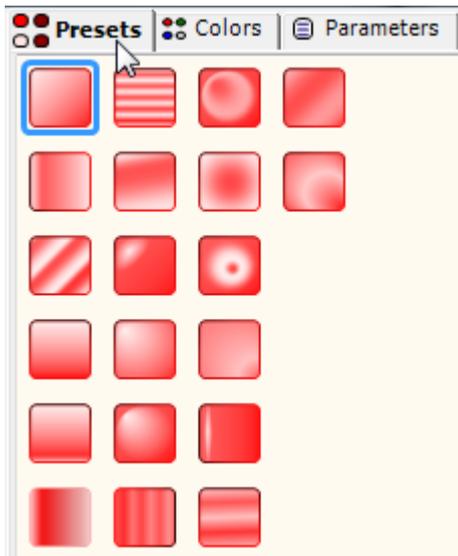
For more information on how to apply the visual effects to the chart elements, see [Adding Visual Effects to Chart Elements](#).

## Visual Effects Designer's Tabs

The **Visual Effects** designer consists of a **Presets**, **Colors**, and **Parameters** tab.

### Presets

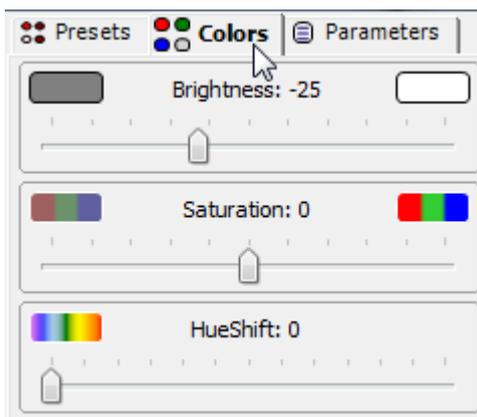
The **Presets** tab appears like the following:



The **Presets** tab contains a list of built-in settings for the available **Chart2D** elements. This makes it simple to visually enhance one or all of the available **Chart2D** elements with one simple click.

### Colors

The **Colors** tab appears like the following:



The **Colors** tab page includes a slider for the color's brightness, saturation, and hue shift. Brightness, Saturation, and Hue represent the red, blue, and green color scheme. This makes it possible to achieve any type of color tone you want for the chart element by changing its brightness and saturation rather than limiting you from specific colors. You can even change all of the colors using the HueShift slider. For more information on using the color slider for chart elements, see [Use the Color Sliders to Enhance an Existing Color for the Chart Data Series](#) or [Use the Color Sliders to Enhance an Existing Color for the Chart Header and Footer](#).

#### Brightness Slider

The brightness is the lightness or darkness of a tone. To increase the level of lightness, slide the **Brightness** slider to

the right. To decrease the level of lightness, slide the **Brightness** slider to the left.

## Saturation Slider

Saturation is the intensity of a hue from gray tone to pure vivid tone. The values in the Saturation slider range from -100 to 100 with the lowest value representing no saturation (a gray tone) to the highest value 100 representing high saturation (pure tone).

To make the color more bright and intense, move the Saturation slider to the right. The color intensity increases the further you slide the Saturation slider to the right.

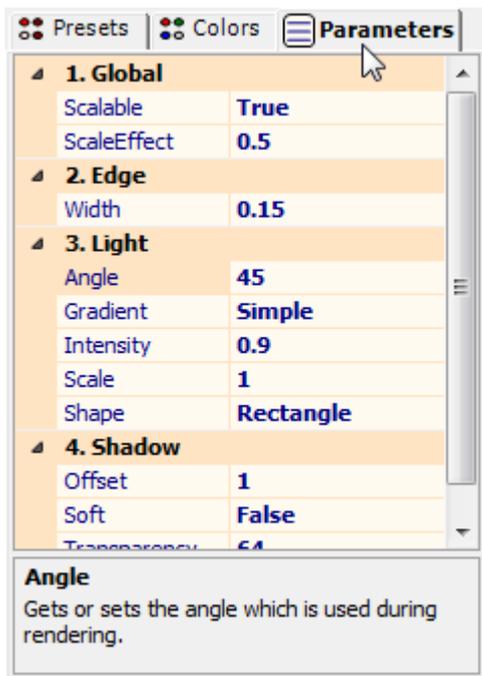
To make the color dull, move the Saturation slider to the left. The color intensity decreases the further you slide the Saturation slider to the left.

## HueShift Slider

Hue represents the specific tone of the color wheel. The value of the Hue depends on the values of the brightness and saturation. If there is high saturation, but no brightness (value of 0) the Hue tone appears grayish-black. If there is high brightness, but no saturation, the Hue tone appears opaque/white.

## Parameters

The **Parameters** tab appears like the following:



The **Parameters** tab includes properties for the Chart2D elements scaling, edge, light, and shadow effects. With these properties you have endless possibilities in customizing your chart's elements. For more information on these properties, see [Visual Effects Parameters](#).

## Visual Effects Parameters

This section describes each Visual Effects parameter and how they relate to one and another.

## Scaling and Edge Effects

The **Scalable** property is set to **True** by default. When the **Scalable** property is enabled you can use the **ScaleEffects**

property to set the scale factor for the image. You can set the scale effect factor from **0** to **1**. The default value is **0.5**. Given a scaling factor greater than **0.5**, scaling will brighten the image and less than **0.5**, scaling will darken the image.

You can achieve a more brightening or darkening effect using the **ScaleEffects** property more than using the **Offset** property since it preserves the relative contrast of the image better. You will not notice the effects of the **ScaleEffects** property if the **Offset** property is zero.

You can specify the width of the element's edge by setting its **Width** property. When the **Scalable** property is true it is measured in relative units, otherwise it's measured in pixels.

For more information on applying scaling and edge effects to a chart element, see [Increase the Size of the Symbols in the Data Series](#).

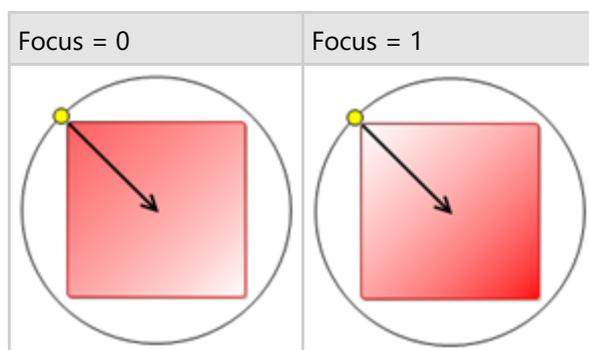
## Light Effects

You can customize the chart element's appearance even more by using applying angle, gradient, intensity, scaling, and shape to change the effects of the light source.

You can change the angle of the visual rendering by setting its **Angle** property to a different degree. The default value for the **Angle** is 45 degrees. The value of the **Angle** property ranges from -180 to 180 degrees. As you change the value of the **Angle** the arrow points to the color and it move around the square in counter-clockwise direction.

You can also apply a light gradient to the chart element using the **Gradient** property. The default setting is Simple. The **Gradient** has three different settings: **Simple**, **SigmeBell**, and **Triangle**. When you select SigmeBell or Triangle for the light Gradient a **Focus** property appears. The default value for the Focus property is 0.1. As you increase the **Focus**, the light gradually moves in opposite direction from the previous position. For example, when the **Focus** property changes from 0 to 1 the light position moves in an opposite direction for the SigmeBell or Triangle gradient.

The following table illustrates the transition from the light position when **Focus** is zero and when **Focus** is 1.



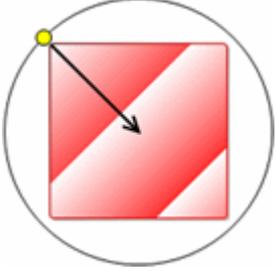
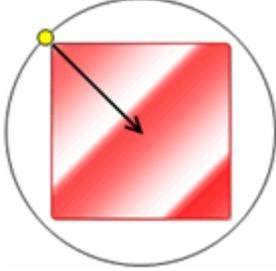
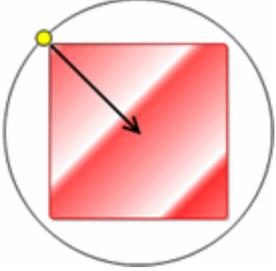
When the Light intensity is set to 0, the Gradient effects are not noticeable. The light gradient effects become more distinguishable when you increase the light intensity. The default value for the Intensity is 0.9.

You can show repetitive light patterns in a chart element by setting the **Scale** property to a value less than 1. As you decrease the Scale, the light pattern repeats more. The values of the **Scale** property range from 0 to 1.

 **Note:** The **Scale** property is only applicable for the Rectangle light shape.

The following table shows the effect of the different light gradients when the Intensity property is 1 and the **Scale** property is 0.4.

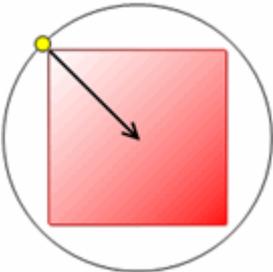
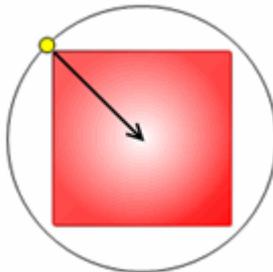
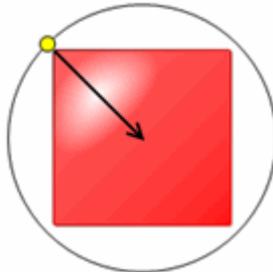
Light Gradient	Image

<b>Simple</b>	
<b>SigmeBell</b>	
<b>Triangle</b>	

You can change the shape of the light to rectangle, ellipse, or edge by using the **Shape** property in the **Visual Effects** designer. The default shape is a rectangle.

When you set the light shape to Edge, the Gradient and Scale properties are not available since these properties do not visually affect the edge light shape. When you select the Ellipse shape, two more properties, Shift and Size are added for further customization. You can shift the ellipse light shape across the element by sliding the Shift slider. You can also increase the size of the ellipse light shape by moving the Size slider from left to right.

The following table illustrates how the rectangle, ellipse, and edge shape appear for the chart element:

Rectangle	Ellipse	Edge
		

For more information on applying light effects to a specific chart element, see [Add a Light Pattern to the Chart Header and Footer](#), [Add a Light Shape to the Chart Header and Footer](#), or [Adjust the Focus of the Light to the Chart Header and Footer](#).

## Shadow Effects

You can add shadow effects by increasing the value of the **Offset** property. The shadow is more noticeable at values

from 3 to 5 (the highest). You can darken your shadow by increasing the transparency. The default value for the transparency is 64. The shadow effects on the element can appear softer if you enable the soft property.

For more information on applying the shadow effects to a specific chart element, see [Add a Shadow to the Chart Header and Footer](#).

## Chart Themes

You can easily apply many predefined themes to the **C1Chart** control or modify them using the **Themes** designer. For more information, see the [Themes for WinForms documentation](#).

## Chart Titles

A chart can have two titles, one called the **Header** and one called the **Footer**. A title consists of one or more lines of text with an optional border both can be customized. Because each title can be positioned above, below, and to the right or left of the chart, it is not necessary to adhere to the traditional concept of **Header** and **Footer** at the top and bottom of an object. In addition, the text alignment, positioning, colors, and font used for the Header or Footer can be modified.

## Title, Text, and Alignment

The Header or Footers' text can be customized using the Smart Designer, **Chart Properties** designer, or through the [Text](#) property.

To add, change, or remove text from the Header or Footer use thDesigner's Header or Footer toolbar or use the Text property. The Text property is located in the Properties window under the corresponding title node (Footer or Header).

To specify whether to center, left-justify, or right-justify the title, use the [HorizontalAlignment](#) or [VerticalAlignment](#) property.

These properties can be accessed at design time through the Style node in either the Header or Footer node in the Visual Studio Properties window.

## Title Position and Size

Use the Title's **Compass** property to specify where to position the Titles relative to the ChartArea. Select from four compass points around the ChartArea.

Use the X and Y properties of the [Location](#) object to customize the location of the title. To restore location auto-selection set left or top to -1. These properties can be accessed at design time through the Location node in either the Header or Footer node in the Visual Studio Properties window.

Use the Width and Height properties of the [Size](#) object to customize the size of the title. To restore auto size selection set width or height to -1. These properties can be accessed at design time through the Size node in either the Header or Footer node in the Visual Studio Properties window.

See [Chart Elements Position and Size](#) for more information.

## Title Border

Use the Type and Width border properties to customize the title's border. These properties can be accessed through the Style node in either the Header or Footer node in the Visual Studio Properties window. See [Chart Borders](#) for more

information.

## Title Colors and Gradient Effects

You can use the `ForeColor` and `BackColor` properties to customize the background and text colors of a title.

These properties are located under `Style` node in either the `Header` or `Footer` node in the Visual Studio Properties window. See [Chart Colors](#) for more information.

## Title Font

Use the `Font` properties to customize the font used for a title. These are located on the `Style` node in either the `Header` or `Footer` node in the Visual Studio Properties window. See [Chart Fonts](#) for more information.

## Chart Legend

The chart automatically generates a Legend whenever data exists in the chart. The chart assigns the name specified in the `ChartDataSeries` object for the series as the series identifier. The symbols that accompany the series name in the Legend are determined by the `LineStyle` and `SymbolStyle` for that series. The positioning, border, colors and font used for the Legend can be customized.

## Legend Position

Use the `Compass` property to specify where to position the Legend relative to the `ChartArea`. Select from four compass points around the `ChartArea`. `Compass` can be accessed at design time under the `Legend` node of the Visual Studio Properties window.

By default the chart automatically positions the legend. Use the `X` and `Y` Location properties to fine-tune the positioning. These properties can be accessed at design time through the `Location` node that is under the `Legend` node of the Visual Studio Properties window.

By default the chart automatically calculates size of the Legend. Use the `Width` and `Height` size properties to fine-tune the size of Legend. These properties can be accessed under the **Size** node that is under the `Legend` node of the Visual Studio Properties window.

See [Chart Elements Position and Size](#) for more information.

## Legend Title

Use the `Text` property to specify the legend title. The legend title appears centered at the top of the Legend. The `Text` property can be accessed at design time under the `Legend` node in the Visual Studio Properties window.

## Legend Border

Use the `Type` and `Width` border properties to customize the title's border. These properties can be accessed at design time under the `Style` node that is under the `Legend` node of the Visual Studio Properties window. See [Chart Borders](#) for more information.

## Legend Colors

Use the `BackColor` and `ForeColor` properties to customize background and text colors of the Legend. These can be accessed at design time through the Style node under the Legend node of the Visual Studio Properties window. See [Chart Colors](#) for more information.

## Legend Font

Use the Font properties to customize the font used for the Legend. These can be accessed at design time through the Style node under the Legend node of the Visual Studio Properties window. See [Chart Fonts](#) for more information.

## Line and Symbol Styles for the Series

The `LineStyle` and `SymbolStyle` properties of the `ChartDataSeries` allow display aspects of each series to be perfected. Not limited to only the lines or symbols of a series line, these properties can change the color of a **Pie** chart slice, change the color of a **Radar** chart series, or set the width of a **HiLo** chart series.

The `LineStyle` property contains the `Color`, `Pattern`, `Thickness`, `LineJoin`, and `MiterLimit` properties. Generally these properties set the color, pattern, joining style, miter limit of the joints of the drawn line, and thickness attributes for a lines of a series.

The `SymbolStyle` property contains the `Color`, `Pattern`, and `Thickness` properties. Generally these properties set the color, patter, and thickness attributes for the symbols of a series.

Below is a table that lists how these two properties affect each chart type. Notice that for some chart types these properties have no effect, and for others these properties set the attributes for something other than a line or a symbol:

Value	Effect for LineStyle Property	Effect for SymbolStyle Property
<code>Chart2DTypeEnum.XYPlot</code>	Changes Color, Thickness, and Pattern of the line that connects data points.	Changes Series Symbol Style.
<code>Chart2DTypeEnum.Pie</code>	Changes the BackColor of the series (piece).	No effect.
<code>Chart2DTypeEnum.Bar</code>	Changes the BackColor of the series.	No effect.
<code>Chart2DTypeEnum.Area</code>	Changes the BackColor of the series.	No effect.
<code>Chart2DTypeEnum.Polar</code>	Changes the BackColor, Pattern, and Thickness for Series Line.	Changes the Series Symbol Style, BackColor, and Thickness.
<code>Chart2DTypeEnum.Radar</code>	Changes the BackColor, Pattern, and Thickness for the Series Line.	Changes the Series Symbol Style, BackColor, and Thickness.
<code>Chart2DTypeEnum.HiLo</code>	Changes the BackColor, Style, and Thickness for the Series Line.	No effect.
<code>Chart2DTypeEnum.HiLoOpenClose</code>	Changes the BackColor, Style, and Thickness for the Series Line.	No effect.
<code>Chart2DTypeEnum.Candle</code>	Changes the BackColor for falling stock prices, and the thickness of the candle.	Changes the BackColor for rising stock prices and the thickness of the candle.

A more elaborate method of styling the plot involves the use of a brush instead of only color as specified by the `LineStyle` and `SymbolStyle` properties. A brush can provide a richer, more unique appearance than just color -- including hatching, gradients and textures. For more information about using brushes, see [Custom Brushes for](#)

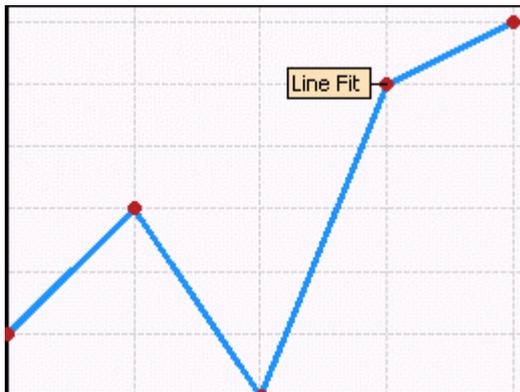
Plotting Data.

## Line FitType

For charts that contain plot lines (XY-Plot, Area, Polar, and Radar), the **FitType** property sets how the line fits to the points of the chart. The property takes a FitTypeEnum enumeration.

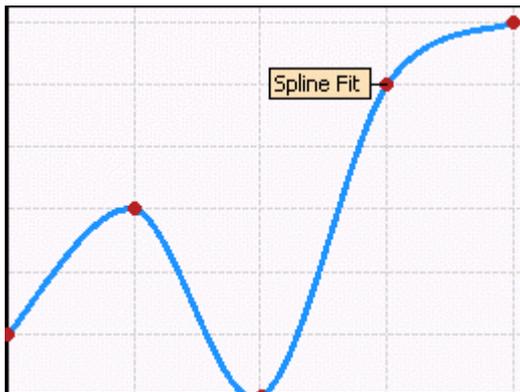
### FitTypeEnum.Line

The Line fit is the default fit for the plot lines that is where the line travels directly from point to point. The Line fit is illustrated below.



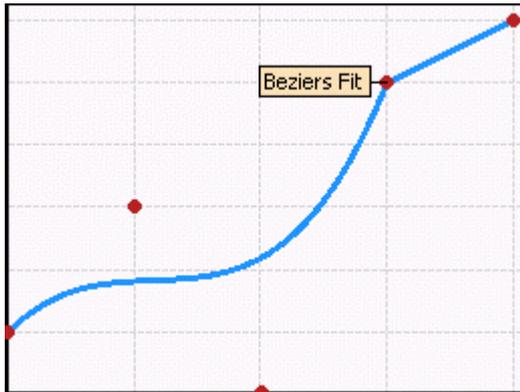
### FitTypeEnum.Spline

The Spline fit is where the line is curved to still pass through the plots, but provide a smooth curved plot line nonetheless. Those familiar with how curved lines are created in most drawing applications will likely be familiar with the Spline fit type.



### FitTypeEnum.Beziers

The Beziers fit is where the line does not necessarily pass through all data points, but the points act as magnets, pulling the curve in certain directions and influencing the way the curve bends.



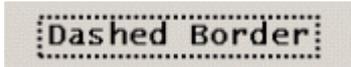
## Chart Borders

Adding a border to part of the chart can help highlight important information, or simply make the chart look more attractive. The border style, color, rounding, and width can be set using the respective properties: [BorderStyle](#), [Color](#), [Rounding](#), and [Thickness](#). Using these properties you can create customized borders for any of the following chart elements:

- Header and Footer titles
- Legend
- ChartArea
- The entire chart

The following table defines and illustrates the effect of each value in the [BorderStyleEnum](#):

Member name	Description	Effect
<b>NotSet</b>	Border style is not set and is inherited from C1Chart class.	
<b>None</b>	No border.	
<b>Empty</b>	Empty border.	
<b>Solid</b>	Solid line border.	
<b>Raised</b>	Raised 3D border, drawn using system colors.	
<b>Inset</b>	Inset 3D border with bevel.	
<b>RaisedBevel</b>	Raised 3D border with bevel.	
<b>InsetBevel</b>	Inset 3D border with bevel.	
<b>Groove</b>	Compound border (inset+raised).	
<b>Fillet</b>	Compound border (raised+inset).	
<b>Double</b>	Double solid line border.	

<b>Dashed</b>	Dashed line border.	
<b>Opaque</b>	The opaque border style ensures that anti-aliasing is turned off when drawing the border. Opaque borders ignore <b>Rounding</b> settings. This border style can be useful for generating chart images to be use with transparent backgrounds.	

The border properties can be modified at design time using the [Chart Properties](#) designer, Properties window, or [Chart Smart Designer](#). These properties are located under the **Style** nodes in the Visual Studio Properties window, which can be found on the Control, ChartArea, Titles, Legend, and ChartLabels objects.

## To change the border style of the ChartArea at design time

To change the border style of the ChartArea element at design time using the Visual Studio Properties window, complete the following:

1. Expand the **ChartArea** node in the c1Chart1 properties window.
2. Expand the **Style->Border** property.
3. Click on the dropdown arrow next to the **BorderStyle** property and select a border style, for example **Dashed**.

## To change the border style of the ChartArea programatically

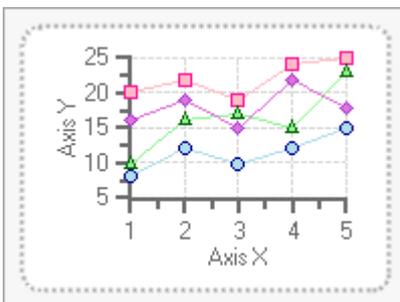
### To write code in Visual Basic

```
Visual Basic
c1Chart1.ChartArea.Style.Border.BorderStyle = BorderStyleEnum.Dashed
```

### To write code in C#

```
C#
c1Chart1.ChartArea.Style.Border.BorderStyle = BorderStyleEnum.Dashed;
```

The dashed border appears around the ChartArea element like the following:

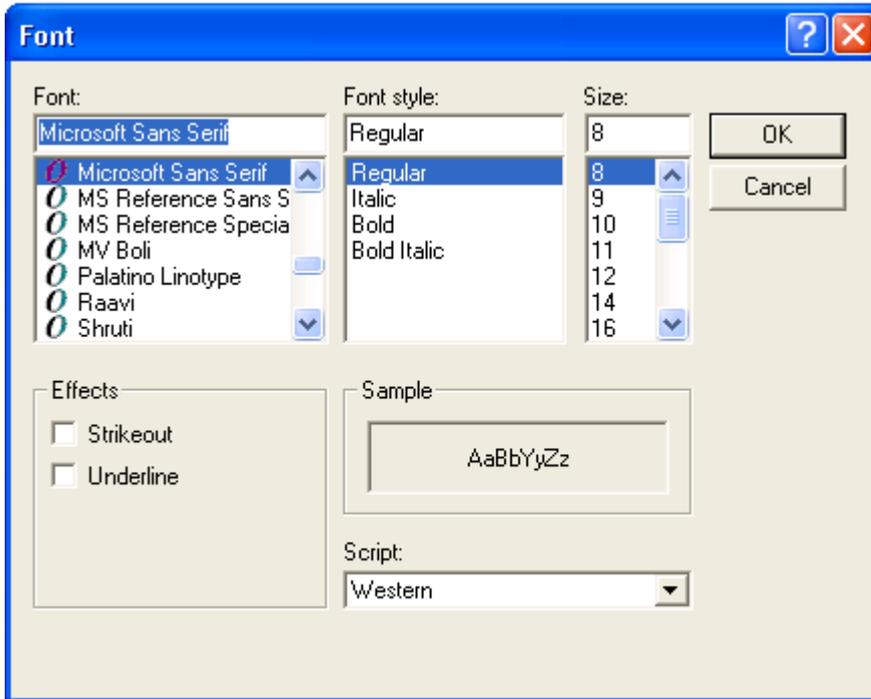


## Chart Fonts

Charts can have more impact when fonts are customized for different chart elements. Font size can be adjusted to make an element better suit the overall size of the chart.

## To Change Fonts

Use the standard .NET Font property editor to set the font, style, and size attributes. Font properties are located under the Style nodes of the Visual Studio Properties window.



## Chart Colors

Color can powerfully enhance a chart's visual impact. Colors can be customized using color names, selecting a color scheme, using RGB values, or interactively using a color chooser. Each of the following visual elements in the chart has a background and foreground color that can be customized:

- The entire chart
- Data series
- Header and Footer titles
- Legend
- ChartArea
- Each ChartLabel added to the chart

## Choosing Colors Interactively

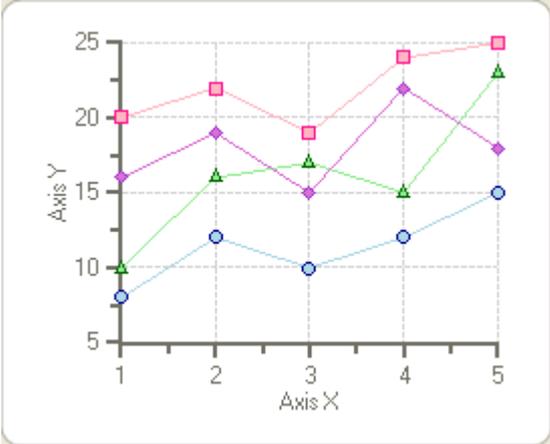
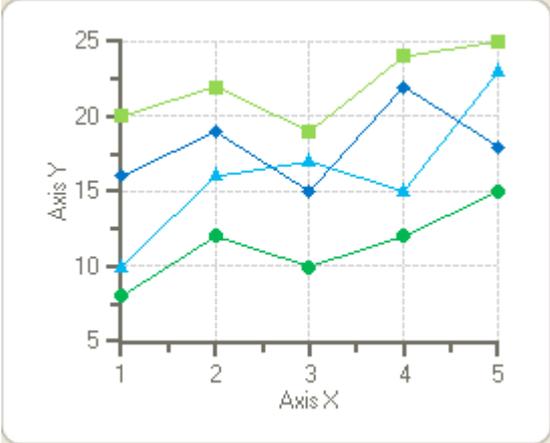
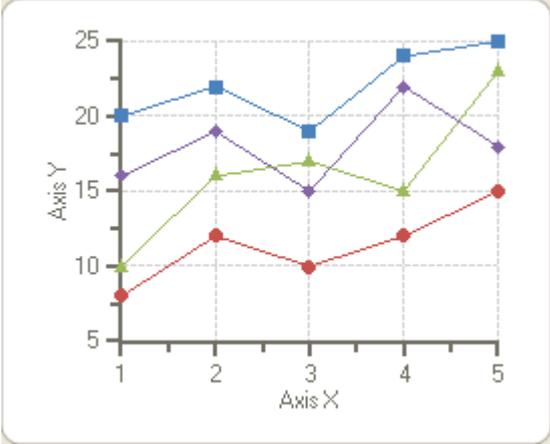
Colors can be selected interactively using .NET's color dialog that works like the standard windows color dialog. Choose from windows basic colors, custom colors, or interactively choose from a full color spectrum.

## Setting the Color Scheme for the Data Series

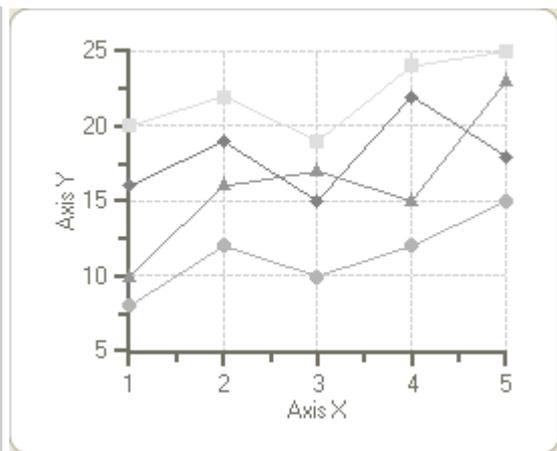
The data series color scheme can be selected by using the [ColorGeneration](#) property. By default, [C1Chart](#) uses the [ColorGeneration.Custom](#) setting which specifies the standard color generation. The remaining options mimic the color themes of Microsoft Office.

Available color schemes for the data series are listed below:

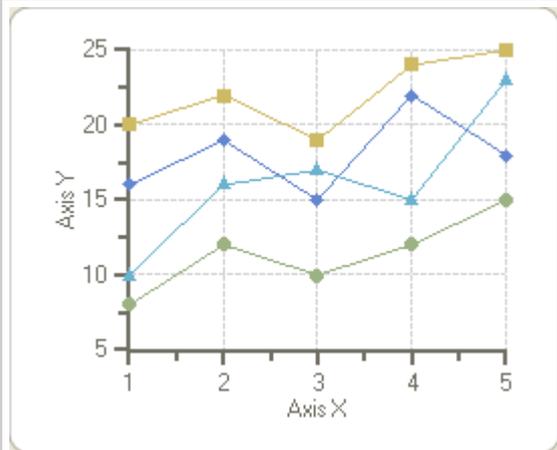


Color Generation Setting	Description or Preview
CopyCurrentToCustom	Copies the currently specified color group into the custom group.*
Custom (default)	
Standard	
Office	

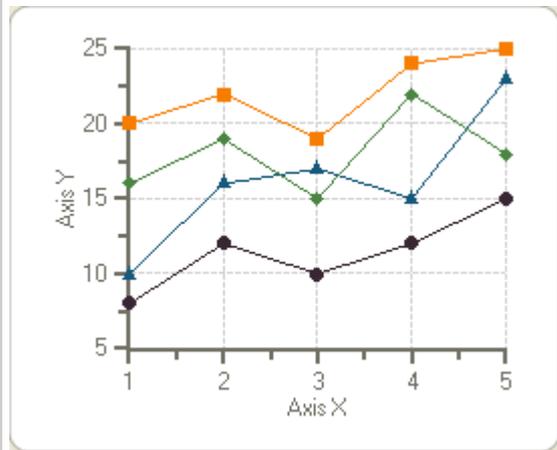
GrayScale



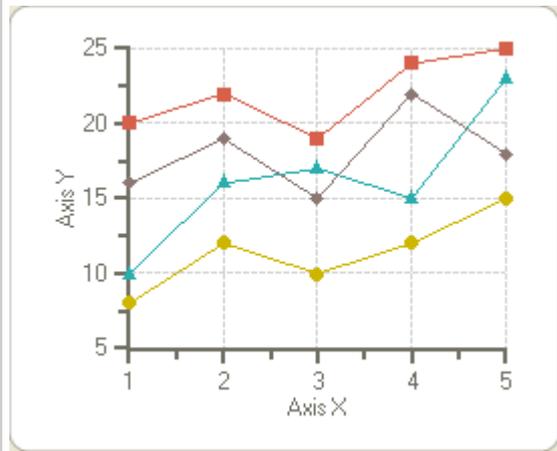
Apex



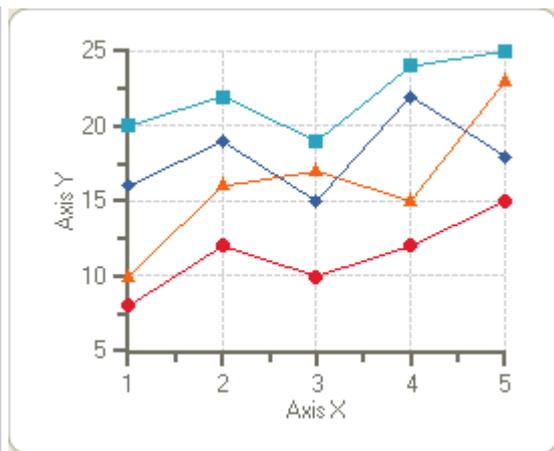
Aspect



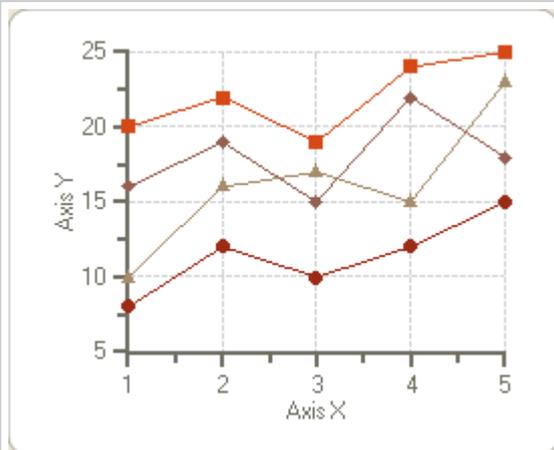
Civic



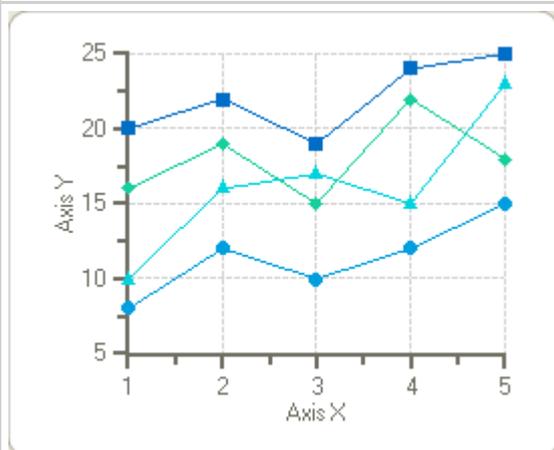
Concourse



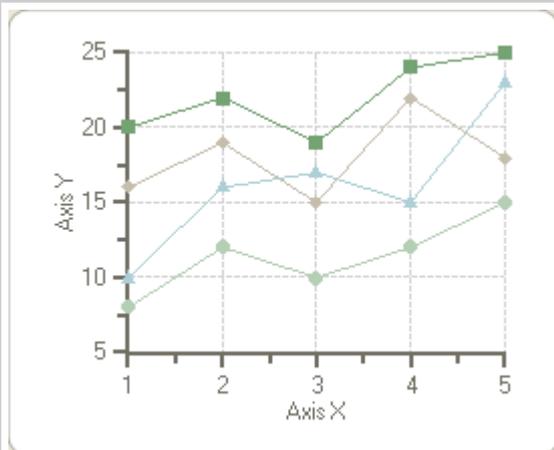
Equity



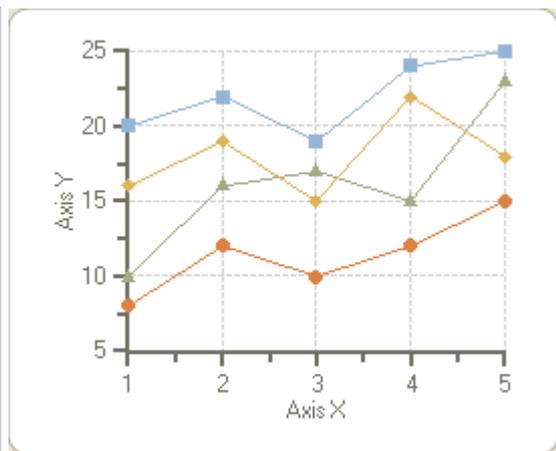
Flow



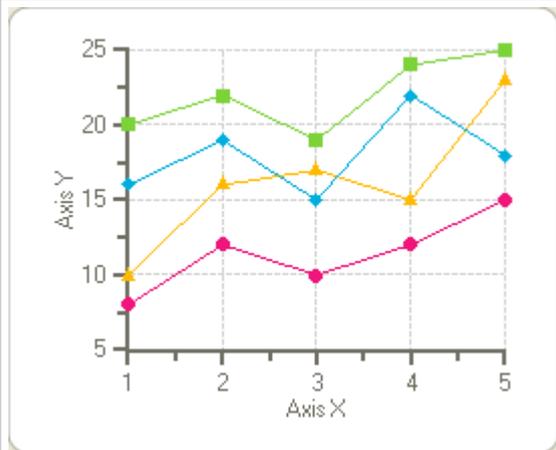
Foundry



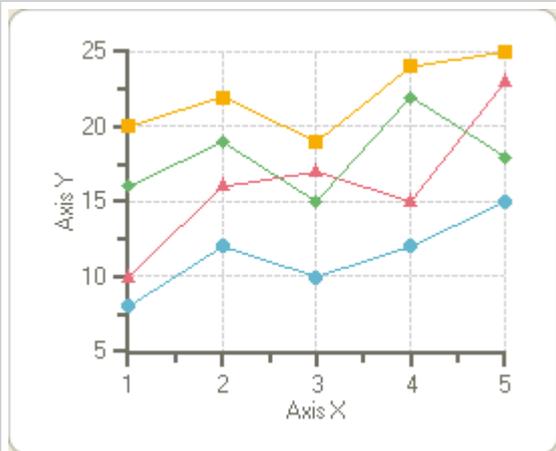
Median



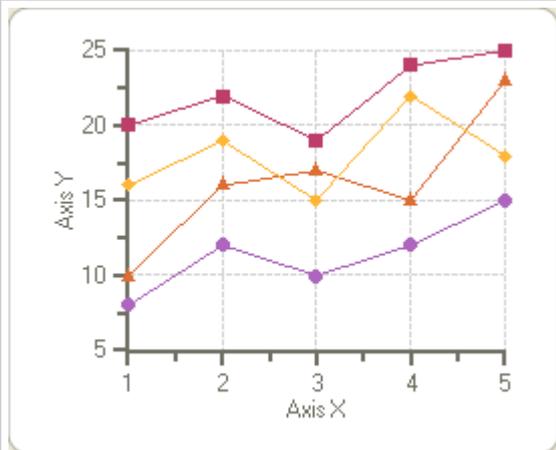
Metro



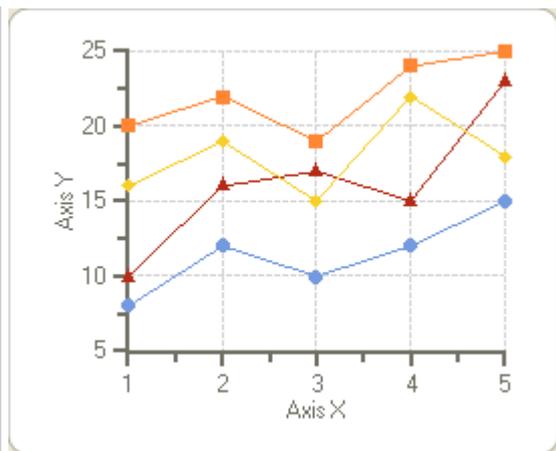
Module



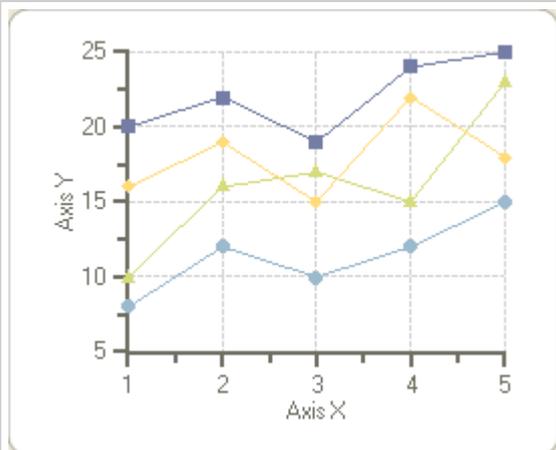
Opulent



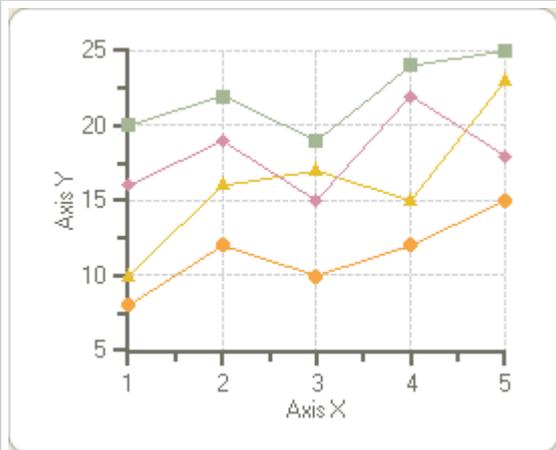
Oriel



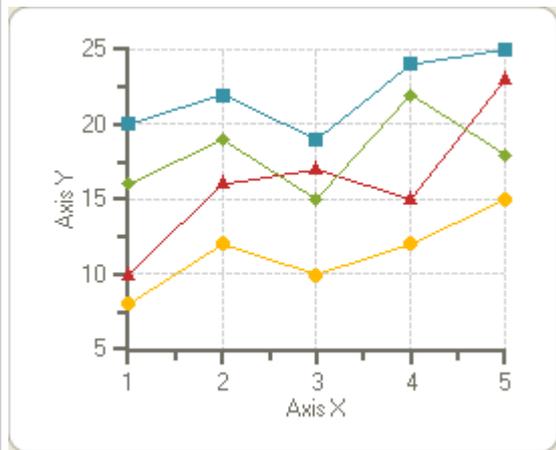
Origin



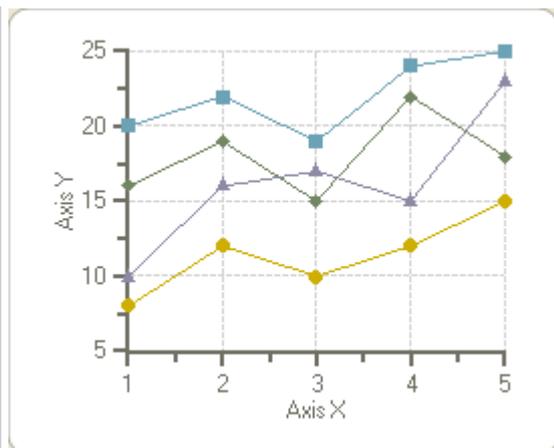
Paper



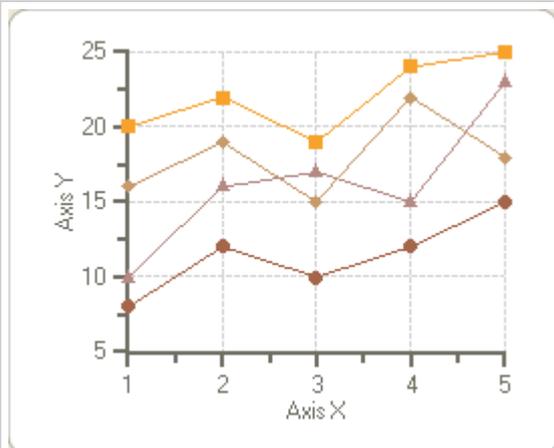
Solstice



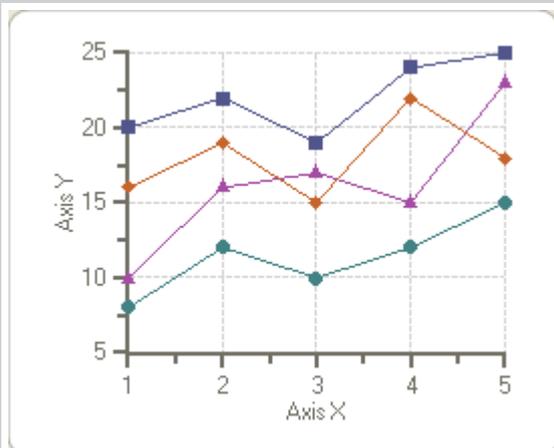
Technic



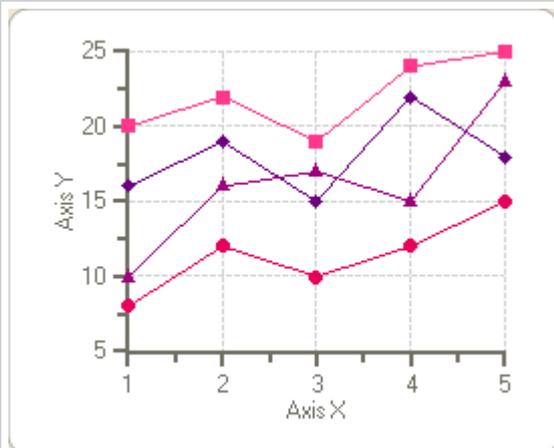
Trek



Urban



Verve



Note that when the [ColorGeneration](#) property is set to the **CopyToCustom** value, the current colors are copied to the Custom color generation, and the property is automatically set to **ColorGeneration.Custom** for further customization. Also note that this operation is automatically performed whenever the chart data is loaded by string, file or by the designer. As color customizations are made to the [LineStyle](#) and [SymbolStyle](#) of each [ChartDataSeries](#), these customizations affect both the current state and the Custom state values.

## Specifying RGB Colors

Alternately, 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.

[C1Chart](#) has a **Visual Effects** designer that enables you to set the hue, saturation, and brightness of a color for the data series, header and footer titles, and the legend elements for more information on using the **Visual Effects** designer to specify a new color through the hue, saturation, and brightness, see [Use the Color Sliders to Enhance an Existing Color for the Chart Data Series](#) or [Use the Color Sliders to Enhance an Existing Color for the Chart Header and Footer](#).

## 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.

## Chart Elements Position and Size

Each of the main chart elements (Header, Footer, Legend, ChartArea, and ChartLabels) has properties that control its position and size. While the chart can automatically control the positioning of any element except ChartLabels and the size of any element, these properties can also be customized.

When the chart controls positioning, it first allows space for the Header, Footer, and Legend, if they exist (size is determined by contents, border, and font). The ChartArea is sized and positioned to fit into the largest remaining rectangular area. Positioning adjusts when other chart properties change.

## Changing Location

Use the **X** property of the Location to specify the number of pixels from the edge of the chart to the left edge of the chart element. Use the **Y** property of the Location to specify the number of pixels from the edge of the chart to the top of the chart element. Set X and Y to -1 to allow the chart to automatically position the element.

These properties are located on the Location node, found on the ChartArea, Titles, Legend, and ChartLabels nodes of the Visual Studio Properties window.

## Changing Width and Height

Use the Width and Height location properties to specify the width and height of the chart elements. Set these properties to -1 to allow the chart to automatically size the chart elements.

These properties are located and enabled at design time under the Location nodes of the Visual Studio Properties window.

## Custom Brushes for Plotting Data

This section provides information on using the [DrawDataSeriesEventArgs](#) for creating a custom brush during plotting.

Whenever a data series is plotted, the **DrawDataSeries** event of [C1Chart](#) fires. This event allows defining the selection of a custom brush during plotting. The event sender object is the [C1Chart.ChartDataSeries](#) to be plotted. The event data ([DrawDataSeriesEventArgs](#)) have the following properties:

- The [Brush](#) property defines the brush that will be used for rendering the data series. You should create your own brush and assign it to this property if you want to use to a custom brush.
- The [DisposeBrush](#) is Boolean value that specifies whether the brush should be disposed after using. If **DisposeBrush** is set to False then you must dispose the brush yourself.
- The [GroupIndex](#) property defines index of chart group.
- The [SeriesIndex](#) property defines index of chart group.
- The [Bounds](#) property represents rectangle area in which data will be plotted. This rectangle will be useful when creating gradient brushes.

## Creating a Hatched Brush

The following sample code represents the handler that creates hatched brush.

Please note that the given code snippet shows the initial declarations of some objects using their full namespace. In subsequent use of those and related objects, the namespace is omitted for brevity.

### To write code in Visual Basic

#### Visual Basic

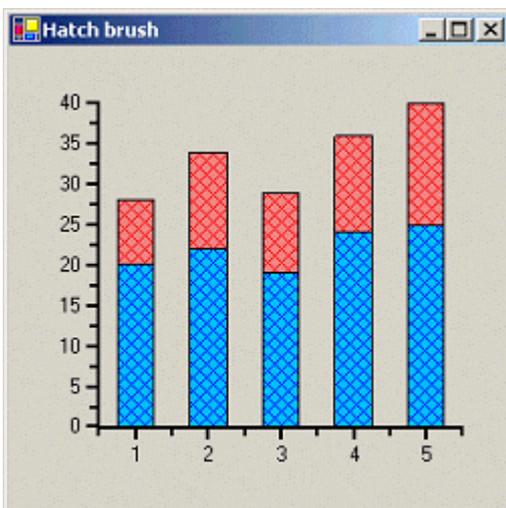
```
Private Sub C1Chart1_DrawDataSeries(ByVal sender As Object, _
    ByVal e As C1.Win.C1Chart.DrawDataSeriesEventArgs) _
    Handles C1Chart1.DrawDataSeries
    Dim ds As C1.Win.C1Chart.ChartDataSeries = sender
    Dim foreclr As Color = ds.SymbolStyle.Color
    Dim backclr As Color = ds.LineStyle.Color
    Dim hb As System.Drawing.Drawing2D.HatchBrush
    hb = New HatchBrush(HatchStyle.OutlinedDiamond, foreclr, backclr)
    e.Brush = hb
End Sub
```

### To write code in C#

C#

```
private void c1Chart1_DrawDataSeries(object sender,
C1.Win.C1Chart.DrawDataSeriesEventArgs e)
{
    C1.Win.C1Chart.ChartDataSeries ds = (ChartDataSeries)sender;
    Color forecolor = ds.SymbolStyle.Color;
    Color backcolor = ds.LineStyle.Color;
    System.Drawing.Drawing2D.HatchBrush hb;
    hb = new HatchBrush(HatchStyle.OutlinedDiamond, forecolor, backcolor);
    e.Brush = hb;
}
```

The following image displays the hatch brush bar chart:



## Creating a Gradient Brush

The following sample code represents the handler that creates linear gradient brush.

Please note that the given code snippet shows the initial declarations of some objects using their full namespace. In subsequent use of those and related objects, the namespace is omitted for brevity.

### To write code in Visual Basic

Visual Basic

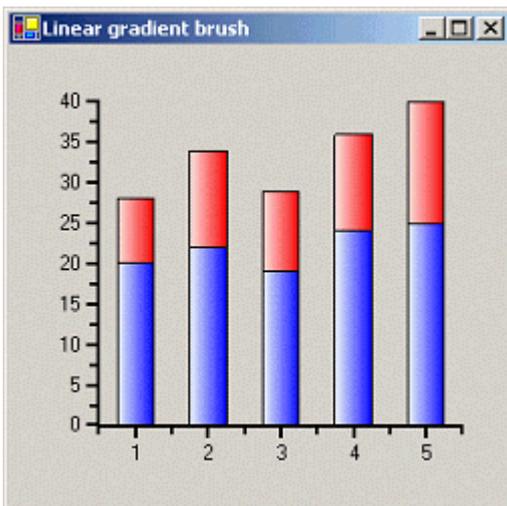
```
Private Sub C1Chart1_DrawDataSeries(ByVal sender As Object, _
    ByVal e As C1.Win.C1Chart.DrawDataSeriesEventArgs) _
    Handles C1Chart1.DrawDataSeries
    Dim ds As C1.Win.C1Chart.ChartDataSeries = sender
    Dim clr1 As Color = ds.LineStyle.Color
    Dim clr2 As Color = ds.SymbolStyle.Color
    If (e.Bounds.Height > 0 And e.Bounds.Width > 0) Then
        Dim lgb As System.Drawing.Drawing2D.LinearGradientBrush = _
            New LinearGradientBrush(e.Bounds, clr1, clr2, LinearGradientMode.Horizontal)
        e.Brush = lgb
    End If
End Sub
```

## To write code in C#

C#

```
private void c1Chart1_DrawDataSeries(object sender,
    C1.Win.C1Chart.DrawDataSeriesEventArgs e)
{
    C1.Win.C1Chart.ChartDataSeries ds = (ChartDataSeries)sender;
    Color clr1 = ds.LineStyle.Color;
    Color clr2 = ds.SymbolStyle.Color;
    if(e.Bounds.Size.Height > 0 && e.Bounds.Size.Width > 0)
    {
        System.Drawing.Drawing2D.LinearGradientBrush lgb = new
        LinearGradientBrush(e.Bounds, clr1, clr2, LinearGradientMode.Horizontal);
        e.Brush = lgb;
    }
}
```

The following image shows the linear gradient brush:



## Loading and Saving Charts, Data, and Images

In [C1Chart](#) each chart can be loaded from and saved to an external XML file or a string value. In addition, the chart can be saved to any number of image files, which allows more flexibility in archiving, accessing data, and creating images of the chart.

There are two different methods that can be used to save/load the chart to an XML file or string value by either saving/loading the entire chart (data and formatting) or by saving/loading only the chart's data.

### Saving/Loading chart's styles and data

When the entire chart is saved/loaded, all of the formatting of the axis, the titles, and the chart area is saved to an XML file.

The following methods are used when saving the chart's formatting and data:

- [LoadChartFromFile](#)
- [SaveChartToFile](#)
- [LoadChartFromString](#)
- [SaveChartToString](#)

In cases where you need to save or load the background images from the chart elements such as the chart background, header and footer images, etc. you can use the following save/load methods:

- [LoadChartAndImagesFromFile](#)
- [SaveChartAndImagesToFile](#)
- [LoadChartAndImagesFromString](#)
- [SaveChartAndImagesToString](#)

### Saving/Loading chart's data

When only the chart's data is saved/loaded the external file is smaller in size, but when this file is loaded back into the chart, only the data will be restored.

The following methods are used when saving/loading only the chart's data:

- [SaveDataToFile](#)
- [LoadDataFromFile](#)

## Loading and Saving to a String

In addition to loading and saving to an XML file, the chart has the ability to save to a string value. This string can then be saved in an application and retrieved at a future time.

C1Chart provides four methods for loading and saving chart to a string: [SaveChartToString](#), [LoadChartFromString](#), [SaveChartAndImagesToString](#), and [LoadChartAndImagesFromString](#).

The [SaveChartToString](#) and [LoadChartFromString](#) methods enable you to save a copy of the chart to a string value in the application. This string value contains all the set properties, methods, and events for the chart, in essence, its formatting. The chart can save to this file, and then if the structure remains intact, load a chart from this file.

The [SaveChartAndImagesToString](#) and [LoadChartAndImagesFromString](#) methods does the same as the [SaveChartToString](#) and [LoadChartFromString](#) methods except that they capture the various background images of the chart elements (chart, header, footer, labels). To save the chart to a string at run time, call the [SaveChartToString](#) method, this takes no parameters and returns the string value:

**To write code in Visual Basic**

## Visual Basic

```
Dim ChartString As String
ChartString = C1Chart1.SaveChartToString()
```

**To write code in C#**

## C#

```
string ChartString;
ChartString = c1Chart1.SaveChartToString();
```

To load the chart from a string value at run time, call the [LoadChartFromString](#) method, which takes the saved chart string as its only parameter:

**To write code in Visual Basic**

## Visual Basic

```
C1Chart1.LoadChartFromString(ChartString)
```

**To write code in C#**

## C#

```
c1Chart1.LoadChartFromString(ChartString);
```

## Loading and Saving Chart to a File

In C1Chart, each chart can be loaded from and saved to an external XML file using the Save Chart/Load Chart commands from C1Chart's context menu at design-time or programmatically using any of the following methods:

- LoadChartFromFile
- SaveChartToFile
- LoadChartAndImagesFromFile
- SaveChartAndImagesToFile

The SaveChartAndImagesToFile and LoadChartAndImagesFromFile methods does the same as the SaveChartToFile and LoadChartFromFile methods except that they capture the various background images of the chart elements (chart, header, footer, labels).

If you only want to save the chart's data you can do so using the SaveDataToFile and LoadDataFromFile methods of the ChartData object.

To save just the chart's data to XML, call the SaveDataToFile method, which takes the file's path as a parameter:

**To write code in Visual Basic**

## Visual Basic

```
C1Chart1.ChartGroups.Group0.ChartData_
    .SaveDataToFile("C:\ComponentOneDocs\chartdata1.xml")
```

**To write code in C#**

C#

```
c1Chart1.ChartGroups.Group0.ChartData
    .SaveDataToFile("C:\\ComponentOneDocs\\chartdata1.xml");
```

To load the chart's data back from XML, call the `LoadDataFromFile` method, which takes the file's path as a parameter:

**To write code in Visual Basic**

Visual Basic

```
C1Chart1.ChartGroups.Group0.ChartData_
    .LoadDataFromFile("C:\\ComponentOneDocs\\chartdata1.xml")
```

**To write code in C#**

C#

```
C1Chart1.ChartGroups.Group0.ChartData_
    .LoadDataFromFile("C:\\ComponentOneDocs\\chartdata1.xml")
```

An in-depth discussion of the structure of the XML file is not covered here, but the file can be opened as a text file and easily dissected.

## Saving Chart Images

The `C1Chart` provides the ability to create a single image of the entire chart as it is drawn on the screen or printer. By calling the `SaveImage` method, all of the items within the chart bounds can be saved to the clipboard, a byte array, a stream, or to an image file.

The `SaveImage` method can save to four different output types by using one of eight different overloaded parameter sets. For each output type, there is an option to save the chart image as it appears on the screen, or to save the chart image as a specified size.

To save the chart image to the clipboard, simply specify an image format, the size parameter is not specified here:

**To write code in Visual Basic**

Visual Basic

```
C1Chart1.SaveImage(System.Drawing.Imaging.ImageFormat.Bmp)
```

**To write code in C#**

C#

```
c1Chart1.SaveImage(System.Drawing.Imaging.ImageFormat.Bmp);
```

To save the chart image to an image file, simply specify the pathname for the new image and an image format, the size parameter is not specified here:

**To write code in Visual Basic**

Visual Basic

```
C1Chart1.SaveImage("C:\\temp\\ChartImages\\CandleChart.bmp", _
    System.Drawing.Imaging.ImageFormat.Bmp)
```

## To write code in C#

C#

```
c1Chart1.SaveImage("C:\\temp\\ChartImages\\CandleChart.bmp",  
System.Drawing.Imaging.ImageFormat.Bmp);
```

To save the chart image to a stream, simply specify the stream object and an image format, the size parameter is not specified here:

## To write code in Visual Basic

Visual Basic

```
Dim coutstream As New System.IO.MemoryStream()  
C1Chart1.SaveImage(coutstream, System.Drawing.Imaging.ImageFormat.Bmp)
```

## To write code in C#

C#

```
System.IO.MemoryStream coutstream = new System.IO.MemoryStream();  
c1Chart1.SaveImage(coutstream, System.Drawing.Imaging.ImageFormat.Bmp);
```

To save the chart image as a byte array, simply specify the byte array, and an image format, the size parameter is not specified here:

## To write code in Visual Basic

Visual Basic

```
Dim bytes() As Byte  
C1Chart1.SaveImage(bytes, System.Drawing.Imaging.ImageFormat.Bmp)
```

## To write code in C#

C#

```
Byte[] bytes;  
c1Chart1.SaveImage(bytes, System.Drawing.Imaging.ImageFormat.Bmp);
```

## End-User Interaction

There are many operations that can be performed on the chart which are not covered in this documentation. Each user of the chart component has different requirements for functionality and features. One way in which to accommodate the many requests for functionality is to provide a set of tools to create functionality not currently in the component.

In [C1Chart](#) there is a set of built-in tools such as conversion methods and interactive built-in tools such as rotation, scaling, and zooming chart that help customize and further develop applications. Chart conversion methods allow the handling of instances where the end user's cursor travels over the chart and performs an action.

The following sections explain how to use the different types of coordinate conversion methods and built-in tools for run-time interaction.

## Coordinate Conversion Methods

Events such as the **MouseMove** event of .NET reports coordinate values and provide pixel coordinates from the chart area. This is useful to a degree, but it does not report to the application anything about the underlying data coordinates, data points, or chart areas.

Chart conversion methods allow the handling of instances where the end user's cursor travels over the chart and performs an action.

The coordinate conversion methods of [C1Chart](#) help to provide this information from the pixel coordinates alone. By knowing just the pixel coordinates you can use any of the **C1Chart** coordinate conversion methods listed in the following table to determine data values for a particular series, data coordinates, and data indices:

Pixel to Data Coordinate Conversion Methods	Description
<a href="#">SeriesFromCoord</a>	Obtains the group and series indices of the legend entry nearest to the specified client coordinates. Mouse coordinates are specified in client coordinates.
<a href="#">CoordToDataCoord</a>	Calculates the data coordinates of a point in the PlotArea given chart client coordinates.
<a href="#">CoordToDataIndex</a>	Returns series and point indices and distance to the closest data point in the group given client coordinates.

If you know the data coordinates or data indices, but don't know the pixel coordinates you can use the following methods to convert the data coordinates or data indices into pixel coordinates:

Data to Pixel Coordinate Conversion Methods	Description
<a href="#">DataIndexToCoord</a>	Calculates the client coordinates of a point in the PlotArea given the data

	coordinates.
<a href="#">DataCoordToCoord</a>	Returns the client coordinates of a specified data point.

When used in conjunction with .NET's MouseMove event these tools allow for creation of interesting application-specific features like handling a double-click in the Legend and chart ToolTips. The MouseMove event of .NET reports the coordinates of the user's cursor as it is dragged over the chart to get constantly updating data on where the user's cursor is located.

The following sections in this chapter briefly explain how to use each type of coordinate conversion method and provides an example of each one.

## Converting Coordinates to Series

In the C1Chart Legend, series are represented by the series name and a symbol containing a sample of the [SymbolStyle](#) or [LineStyle](#). The [SeriesFromCoord](#) method of the Legend returns the nearest associated series from a pixel coordinate value located in the legend. For instance, suppose that the user clicks on the symbol for the first series in the Legend, the pixel coordinates of the user's mouse at this moment when input into the SeriesFromCoord method would return group and series values from the Group and Series parameters. The following is an example of this:

### To write code in Visual Basic

#### Visual Basic

```
Dim LegendGroup As Integer
Dim LegendSeries As Integer
C1Chart1.Legend.SeriesFromCoord(170, 275, LegendGroup, LegendSeries)
```

### To write code in C#

#### C#

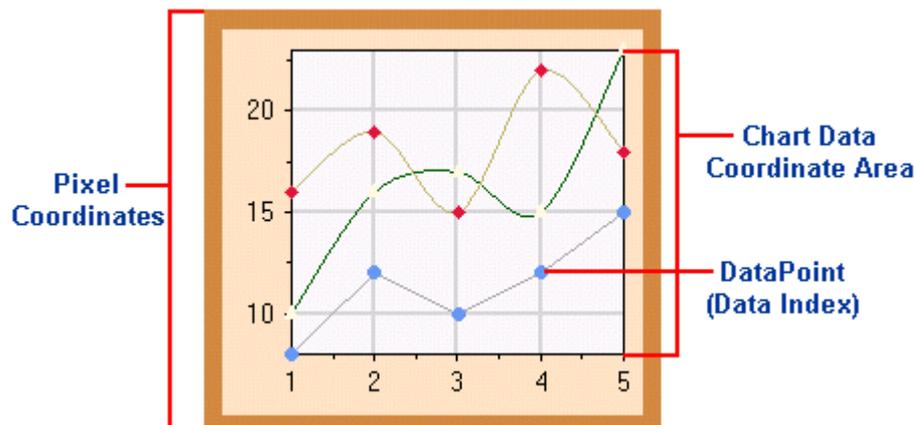
```
int LegendGroup = 0;
int LegendSeries = 0;
c1Chart1.Legend.SeriesFromCoord(170, 275, ref LegendGroup, ref LegendSeries);
```



**Note:** For a complete sample using this method, see [DataStyl](#) or [PieStuff](#) sample located on <http://helpcentral.componentone.com/Samples.aspx>.

## Converting Pixel Coordinates into Data Points and Vice Versa

[C1Chart](#) also allows conversion of pixel coordinate values into their associated Data Coordinate or Data Index and vice versa. The [CoordToDataCoord](#), [CoordToDataIndex](#), [DataCoordToCoord](#), and [DataIndexToCoord](#) are methods which convert pixel coordinates into these values and vice versa. The following graphic illustrates which areas of **C1Chart** are governed by the pixel coordinates, and which area is governed by the data coordinates (PlotArea):



## Converting Pixel Coordinates into Data Points

The two methods that convert pixel coordinate values into chart data coordinates and data indices are the [CoordToDataCoord](#) and [CoordToDataIndex](#) methods of the [ChartGroup](#) object. Both of these methods take a coordinate value, presumably from a [MouseMove](#) event, and return a [PlotArea](#) coordinate or the nearest data point.

### CoordToDataCoord Method

The [CoordToDataCoord](#) method takes four parameters. The first two parameters (e.X and e.Y in the example below) are the pixel coordinates from the [MouseMove](#) event. The second two parameters are integer values that the method will fill with the X and Y Data Coordinate data. The following code is an example of this method in the **MouseMove** event:

#### To write code in Visual Basic

##### Visual Basic

```
Dim CoordXOutput As Double Dim CoordYOutput As Double
    C1Chart1.ChartGroups.Group0.CoordToDataCoord(e.X, e.Y, CoordXOutput,
CoordYOutput)
    Debug.WriteLine("X
Data Coordinate: " & CoordXOutput.ToString())
    Debug.WriteLine("Y Data Coordinate: " & CoordYOutput.ToString())
```

#### To write code in C#

##### C#

```
double CoordXOutput = 0;
    double CoordYOutput = 0;
    c1Chart1.ChartGroups.Group0.CoordToDataCoord(e.X, e.Y, CoordXOutput,
CoordYOutput);
c1Chart1.ChartGroups.Group0.CoordToDataCoord(e.X, e.Y, ref CoordXOutput, ref
CoordYOutput);
    ConsoleDebug.WriteLine("X Data Coordinate: " +
CoordXOutput.ToString()); ConsoleDebug.WriteLine("Y Data Coordinate: " +
CoordYOutput.ToString());
```

 **Note:** For a complete sample using the [CoordToDataCoord](#) method, please see the [StepChart](#) or the [CoordToMapping3D](#) sample located <http://helpcentral.componentone.com/Samples.aspx>.

## CoordToDataIndex Method

The `CoordToDataIndex` method takes six parameters. The first two parameters (e.X and e.Y in the example below) are the pixel coordinates from the `MouseMove` Event. The third parameter is a `CoordinateFocusEnum` value. This is an enumeration that specifies which axis to focus on when determining which data point is closer and its distance from the pixel coordinate. For instance, if X is selected as the focus, then it will return the nearest point that has the closest X-coordinate, regardless of the value of the Y-coordinate. The fourth and fifth parameters are integer values that the method will fill with the appropriate Series and Point indices. The sixth parameter is an integer value that the method fills with the distance from the pixel coordinate specified to the data point in pixels. The following code is an example of this method in the `MouseMove` event:

### To write code in Visual Basic

#### Visual Basic

```
Dim SeriesOutput As Integer Dim PointOutput As Integer Dim DistanceOutput As Integer
    ClChart1.ChartGroups.Group0._ CoordToDataIndex(e.X, e.Y,
CoordinateFocusEnum.XandYCoord, _ ref SeriesOutput, ref PointOutput, ref
DistanceOutput)
    Debug.WriteLine("Series Index: " & SeriesOutput.ToString())
Debug.WriteLine("Point Index: " & PointOutput.ToString())
    Debug.WriteLine("Distance From Point: " & DistanceOutput.ToString())
```

### To write code in C#

#### C#

```
int SeriesOutput = 0;
    int PointOutput = 0;
    int DistanceOutput = 0;
    clChart1.ChartGroups.Group0.CoordToDataIndex(e.X, e.Y,
CoordinateFocusEnum.XandYCoord,
ref SeriesOutput, ref PointOutput, ref DistanceOutput);
    ConsoleDebug.WriteLine("Series Index: " + SeriesOutput.ToString());
    ConsoleDebug.WriteLine("Point
Index: " + PointOutput.ToString());
    ConsoleDebug.WriteLine("Distance From Point: " + DistanceOutput.ToString());
```



**Note:** For a complete sample using the `CoordToDataIndex` method, see **DataStyl**, **PieStuff**, **StepChart**, or **Scatter** samples located on <http://helpcentral.componentone.com/Samples.aspx>.

## Converting Data Points into Pixel Coordinates

The two methods that convert either data coordinates or data indices into pixel coordinates are the **DataIndexToCoord** and **DataCoordToCoord** methods of the `ChartGroup` object. The `DataIndexToCoord` method takes a series and point index and returns the client coordinates of the specified data point. The `DataCoordToCoord` method takes a set of data coordinates and returns a pixel coordinate. These methods are quite similar to the other conversion methods.

## DataIndexToCoord Method

The `DataIndexToCoord` method takes four parameters, and returns a pixel coordinate as seen in the following example:

#### To write code in Visual Basic

##### Visual Basic

```
Dim CoordX, CoordY As Integer
C1Chart1.ChartGroups.Group0.
    DataIndexToCoord(ChartSeries, ChartPoint, CoordX, CoordY)
Debug.WriteLine("X Chart Coordinate: " & CoordX.ToString())
Debug.WriteLine("Y Chart Coordinate: " & CoordY.ToString())
```

#### To write code in C#

##### C#

```
int CoordX=0, CoordY=0;
c1Chart1.ChartGroups.Group0.
    DataIndexToCoord(ChartSeries, ChartPoint, ref CoordX, ref CoordY);
Debug.WriteLine("X Chart Coordinate: " + CoordX.ToString());
Debug.WriteLine("Y Chart Coordinate: " + CoordY.ToString());
```

## DataCoordToCoord Method

The `DataCoordToCoord` method takes four parameters, and returns a pixel coordinate as seen in the following example:

#### To write code in Visual Basic

##### Visual Basic

```
Dim CoordX, CoordY As Integer
C1Chart1.ChartGroups.Group0.
    DataCoordToCoord(DataCordX, DataCoordY, CoordX, CoordY)
Debug.WriteLine("X Chart Coordinate: " & CoordX.ToString())
Debug.WriteLine("Y Chart Coordinate: " & CoordY.ToString())
```

#### To write code in C#

##### C#

```
int CoordX=0, CoordY=0;
c1Chart1.ChartGroups.Group0.
    DataCoordToCoord(DataCordX, DataCoordY, ref CoordX, ref CoordY);
ConsoleDebug.WriteLine("X Chart Coordinate: " + CoordX.ToString());
ConsoleDebug.WriteLine("Y Chart Coordinate: " + CoordY.ToString());
```



**Note:** For a complete sample using the `DataCoordToCoord` method, see the PropGrid or the FExplorer samples located on <http://helpcentral.componentone.com/Samples.aspx>.

## Rotating, Scaling, Translating, and Zooming

**C1Chart** contains built in tools that simplify the implementation of interactive behaviors for the end user. The end user can explore, rotate and zoom chart using combinations of mouse and shift keys. In addition to the built in tools provided for the end-user, **C1Chart** also makes it possible for the end-user to modify any of the chart elements, such as Axis labels. This can be done by using the Microsoft property grid and assigning the **C1Chart** object to it or using the [ShowProperties](#) method. The [ShowProperties](#) method has four overloads.

### End-User Interaction

**C1Chart** contains built in tools that simplify the implementation of interactive behaviors for the end user. The end user can explore, rotate and zoom chart using combinations of mouse and shift keys.

The control center for interactive features is the [Interaction](#) property of **C1Chart**. The Interaction object has a number of properties that allow customization of the interface. All of the properties can be set or changed at design time through the Properties window or the **Action Collection Editor** and programmatically through the [Interaction](#) class. For more information on the **Action Collection Editor** see, [Action Collection Editor](#).

The Interaction class includes the following properties:

- The [IsDefault](#) property is a Boolean value that indicates whether the action settings are default. To restore defaults, set the **IsDefault** property to **True**.
- The [Enabled](#) property switches on/off interaction. Note that by default interaction is disabled.
- The [Actions](#) property contains a collection of actions that are available for chart.

The following list reveals the supported actions:

- Rotate action allows changing viewing angle. This action is available only for chart with 3D effects.
- Scale action increases or decreases the scale of chart along the selected axis or axes.
- Translate action provides the opportunity to scroll through plot area.
- Zoom action allows the user to select rectangular area for viewing.

The scaling, translation and zooming are available only for chart with Cartesian axes.

Each Action object provides set of properties that help to customize behavior of action.

- The [Name](#) gets the name of the action.
- The [Axis](#) property specifies what axis will be involved. It is possible to use only one axis for the action. For example, scaling or translation can be applied in horizontal (x-axis) or vertical (y-axis) direction.
- The [MouseButton](#) and [Modifier](#) properties specify the mouse button and key (ALT, CONTROL or SHIFT) combination that will invoke the execution of the action.

## Controlling Transformation

The transformations applied to the chart can be tuned by using Transform event of **C1Chart**. When scaling, rotating or zooming occurs, the [Transform](#) event fires. Adding a handler to this event allows adjusting limits of axes or to cancel the action.

The following example limits the range of x-axis from -10 to 10 by canceling action when axis minimum or maximum is beyond the range.

### To write code in Visual Basic

Visual Basic

```
Private Sub C1Chart1_Transform(ByVal sender As Object, _ ByVal e As
```

```

C1.Win.C1Chart.TransformEventArgs) Handles C1Chart1.Transform
  If e.MinX < -10 Or e.MaxX > 10 Then
    e.Cancel = True
  End If
End Sub

```

### To write code in C#

```

C#
private void c1Chart1_Transform(object sender, C1.Win.C1Chart.TransformEventArgs e)
{
  if( e.MinX < -10 || e.MaxX > 10)
    e.Cancel = true;
}

```

## Creating a Zoom Effect

To create a zoom effect with the chart, simply adjust the axis. For example, you can add two button controls (**Zoom in** and **Zoom out** buttons) to your application which will adjust the axis at run time when each is pressed. Here is a code example showing how to handle the **Button\_Click** events to control zooming:

### To write code in Visual Basic

```

Visual Basic
' Controls zooming in
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
  Dim xMin As Double = Me.C1Chart1.ChartArea.AxisX.Min
  Dim xMax As Double = Me.C1Chart1.ChartArea.AxisX.Max
  Dim xChange As Double = (xMax - xMin) * 0.05
  Me.C1Chart1.ChartArea.AxisX.Min = xMin + xChange
  Me.C1Chart1.ChartArea.AxisX.Max = xMax - xChange
  Dim yMin As Double = Me.C1Chart1.ChartArea.AxisY.Min()
  Dim yMax As Double = Me.C1Chart1.ChartArea.AxisY.Max
  Dim yChange As Double = (yMax - yMin) * 0.05
  Me.C1Chart1.ChartArea.AxisY.Min = yMin + yChange
  Me.C1Chart1.ChartArea.AxisY.Max = yMax - yChange
End Sub
' Controls zooming out
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button2.Click
  Dim xMin As Double = Me.C1Chart1.ChartArea.AxisX.Min
  Dim xMax As Double = Me.C1Chart1.ChartArea.AxisX.Max
  Dim xChange As Double = (xMax - xMin) * 0.05
  Me.C1Chart1.ChartArea.AxisX.Min = xMin - xChange
  Me.C1Chart1.ChartArea.AxisX.Max = xMax + xChange
  Dim yMin As Double = Me.C1Chart1.ChartArea.AxisY.Min()
  Dim yMax As Double = Me.C1Chart1.ChartArea.AxisY.Max
  Dim yChange As Double = (yMax - yMin) * 0.05
  Me.C1Chart1.ChartArea.AxisY.Min = yMin - yChange

```

```
Me.ClChart1.ChartArea.AxisY.Max = yMax + yChange
End Sub
```

### To write code in C#

```
C#
'// Controls zooming in
private void button1_Click(object sender, System.EventArgs e)
{
    double xmin = this.clChart1.ChartArea.AxisX.Min;
    double xmax = this.clChart1.ChartArea.AxisX.Max;
    double xchange = (xmax - xmin) * 0.05;
    this.clChart1.ChartArea.AxisX.Min = xmin + xchange;
    this.clChart1.ChartArea.AxisX.Max = xmax - xchange;
    double ymin = this.ClChart1.ChartArea.AxisY.Min();
    double ymax = this.ClChart1.ChartArea.AxisY.Max;
    double ychange = (ymax - ymin) * 0.05;
    this.clChart1.ChartArea.AxisY.Min = ymin + ychange;
    this.clChart1.ChartArea.AxisY.Max = ymax - ychange;
}
'// Controls zooming out
private void button2_Click(object sender, System.EventArgs e)
{
    double xmin = this.clChart1.ChartArea.AxisX.Min;
    double xmax = this.clChart1.ChartArea.AxisX.Max;
    double xchange = (xmax - xmin) * 0.05;
    this.clChart1.ChartArea.AxisX.Min = xmin - xchange;
    this.clChart1.ChartArea.AxisX.Max = xmax + xchange;
    double ymin = this.clChart1.ChartArea.AxisY.Min();
    double ymax = this.clChart1.ChartArea.AxisY.Max;
    double ychange = (ymax - ymin) * 0.05;
    this.clChart1.ChartArea.AxisY.Min = ymin - ychange;
    this.clChart1.ChartArea.AxisY.Max = ymax + ychange;
}
```

## Zooming, Panning, and Scaling with Multiple Y-axes

To use interaction features, such as zooming, panning, and scaling, with a second y-axis, you should set the appropriate [Axis](#) property, for example:

### To write code in Visual Basic

```
Visual Basic
ClChart1.Interaction.Actions("Zoom").Axis = AxisFlagEnum.AxesAll
```

### To write code in C#

```
C#
clChart1.Interaction.Actions["Zoom"].Axis = AxisFlagEnum.AxesAll;
```

## Chart for WinForms Tutorials

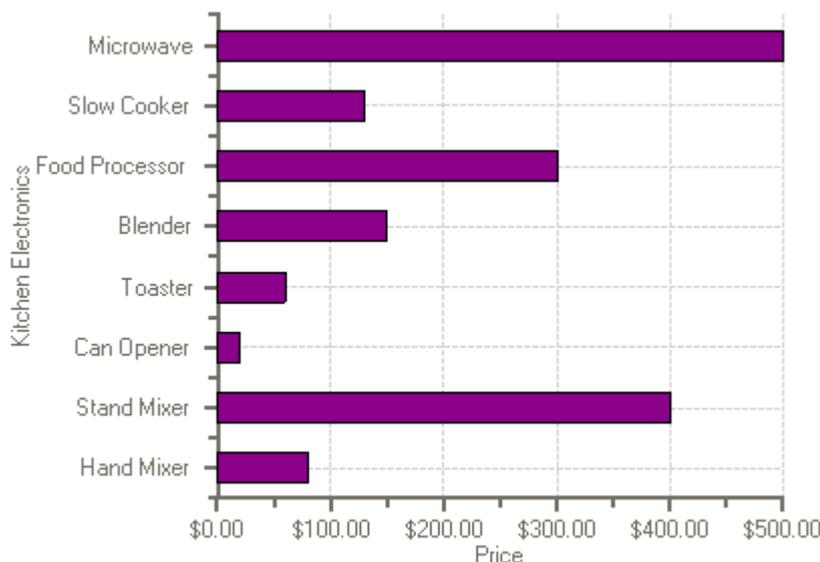
The following steps will walk you through creating basic charts such as Bar, Pie, X-Y Plot, and Candle charts. The tutorials in this section include:

Tutorial	Description
<a href="#">Bar Chart Tutorial</a>	This tutorial shows how to create a simple Bar chart at design time and through code.
<a href="#">Line Chart Tutorial</a>	This tutorial shows how to create a Line chart at design time and through code.
<a href="#">Pie Chart Tutorial</a>	This tutorial shows how to create a Pie chart at design time and through code.
<a href="#">Candle Chart Tutorial</a>	This tutorial shows how to create a Candle chart at design time and through code.

## Bar Chart Tutorial

This section provides step-by-step instructions for building a simple Bar Chart. The graph shows the information as a simple bar chart with one y-axis that represents the numeric values for each product price and the x-axis values provide the spacing between the points and display the label attached to each data point.

The following chart illustration is shown before you begin creating the chart:



### To create a Bar Chart at design time

This topic walks you through the steps for creating a simple bar chart using the **Chart Properties** designer. In this topic you will learn how to set the specific chart type, modify the data series, add data to the chart, and annotate the axes all at design-time in a few simple steps.

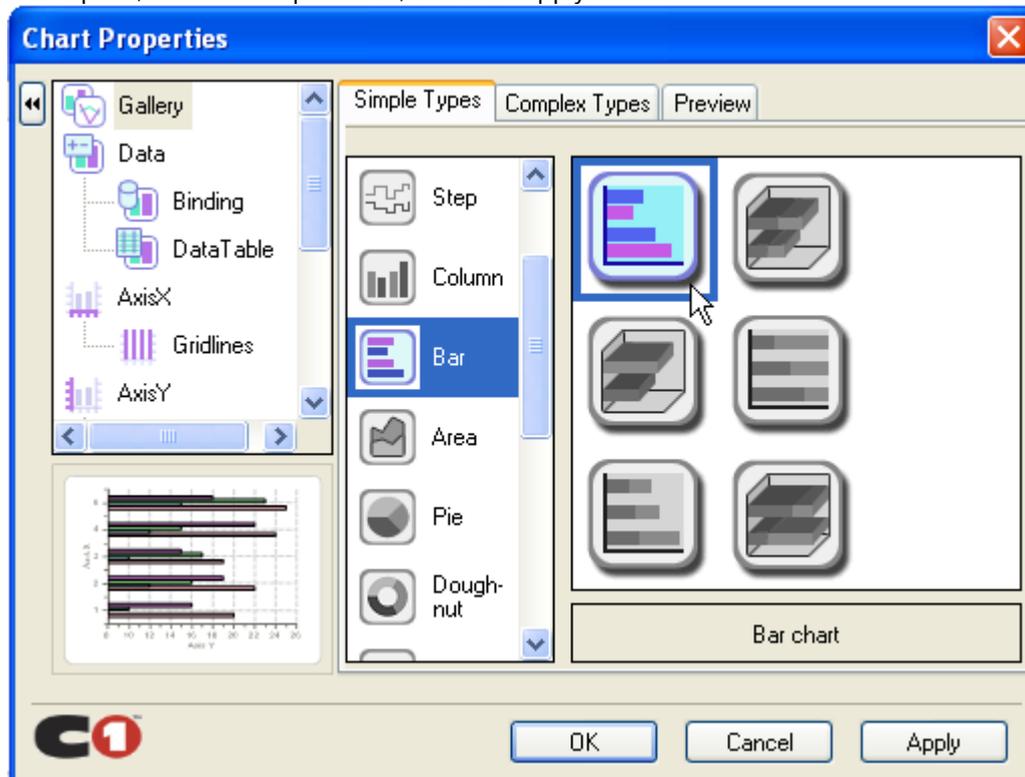
This task assumes you have already added the C1Chart control to the form.

## Set the Chart Type

The first step in configuring a chart through the smart tag wizard is to select a gallery type from the wealth of chart

types available. From basic chart types like Bar and Lines to more complex charts like Polar and Surface, C1Chart allows you to deal with any data visualization needed.

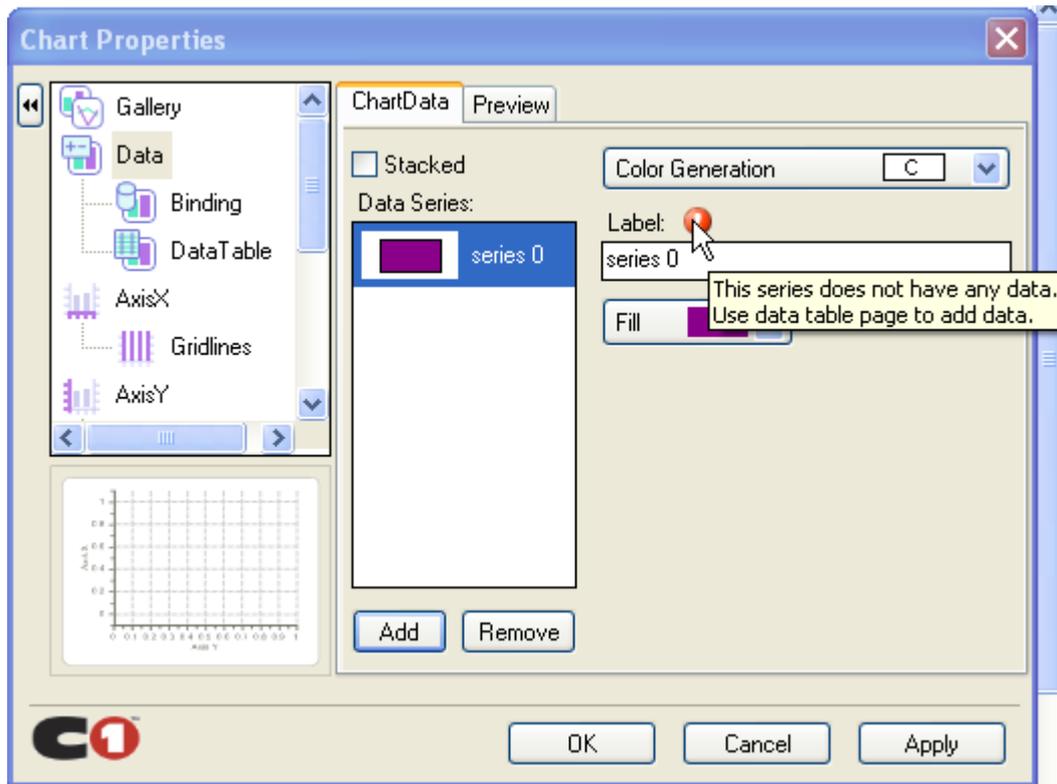
1. Right-click on the **C1Chart** control and click **Chart Properties** from its short-cut menu.
2. Select **Gallery** from the treeview pane, then select **Bar** from the **Simple Types** list.
3. In the pane, select the top-left Bar, and click Apply.



## Modify the Data Series

4. Select **Data** from the treeview pane.
5. Click **Remove** to remove Series0, Series 1, Series 2, and Series 3.
6. Click **Add** to add a new series for the Y-values that will display the product names.

A warning will appear to remind you that you must add data to the new series.



## Add Data to the Data Series

7. Select **Data->DataTable** from the treeview pane. Add the following data to the data table:
8. Click in the X table and enter the following numerical X-Y-values: (0, 80), (1, 400), (2, 20), (3, 60), (4, 150), (5, 300), (6, 130), and (7, 500).

 **Note:** Click Tab to move the cursor to the next X-Y value.

9. Click **Apply** to update the chart.

A preview image of the updated chart appears in the lower left pane of the Chart Properties designer.

## Configure Axes

Next we will annotate the x-axis and y-axis. We will use the value labels annotation for the x-axis. For more information on the value labels annotation, see the [Value Labels Annotation](#) topic.

10. Select **AxisX** from the treeview pane, then select the annotation tab.
11. Select **ValueLabels** from the Method dropdown list box.

A **Labels** button appears.

12. Click on the **Labels** button to open the **ValueLabel Collection Editor**.
13. Click the **Add** button eight times to add eight value labels.
14. Select the first value label in the **Members** list box and set its **NumericValue** property to 0 and its **Text** property to "Hand Mixer".
15. Set the remaining value labels' properties to the following:
  - o ValueLabel1's **NumericValue** to 1 and **Text** to "Stand Mixer".
  - o ValueLabel2's **NumericValue** to 2 and **Text** to "Can Opener".
  - o ValueLabel3's **NumericValue** to 3 and **Text** to "Toaster".
  - o ValueLabel4's **NumericValue** to 4 and **Text** to "Blender".

- ValueLabel5's **NumericValue** to 5 and **Text** to "Food Processor".
  - ValueLabel6's **NumericValue** to 6 and **Text** to "Slow Cooker".
  - ValueLabel7's **NumericValue** to 7 and **Text** to "Microwave".
16. 16. Click **OK** to apply the value labels to the x-axis annotation.
  17. 17. Select **AxisY** from the treeview pane, then select the annotation tab.
  18. 18. Select **NumericCurrency** from the Format dropdown listbox.

## Set Titles for Axes

19. 19. To set the AxisX Title, select **AxisX** from the treeview pane and set its **Title** to "Kitchen Electronics".
20. 20. To set the AxisY Title, select **AxisY** from the treeview pane and set its **Title** to "Price".
21. 21. Click **OK** to close the **Chart Properties** designer.

### To create a Bar chart programmatically

A simple Bar chart can be created programmatically using the following steps:

1. Add the **C1Chart** control to the form.
2. Right-click the form and select View Code to view the code file, then add the following directive to declare the **C1.Win.C1Chart** namespace:

#### To write code in Visual Basic

```
Visual Basic
Imports C1.Win.C1Chart;
```

#### To write code in C#

```
C#
using C1.Win.C1Chart;
```

3. Double click Form1 and add the following code in the **Form1\_Load** procedure to create the simple bar chart:

#### To write code in Visual Basic

```
Visual Basic
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Clear previous data
    C1Chart1.ChartGroups(0).ChartData.SeriesList.Clear()

    ' Add Data
    Dim ProductNames() As String = {"Hand Mixer", "Stand Mixer", "Can Opener",
    "Toaster" , "Blender" , "Food Processor" , "Slow Cooker" , "Microwave"}
    Dim PriceX() As Integer = {80, 400, 20, 60, 150, 300, 130, 500}

    ' Create first series
    Dim ds1 As C1.Win.C1Chart.ChartDataSeries = _
        C1Chart1.ChartGroups(0).ChartData.SeriesList.AddNewSeries()
    ds1.Label = "PriceX"
    ds1.X.CopyDataIn(ProductNames)
    ds1.Y.CopyDataIn(PriceX)
```

```

' Set chart type
c1Chart1.ChartArea.Inverted = True
c1Chart1.ChartGroups(0).ChartType = _
c1.Win.C1Chart.Chart2DTypeEnum.Bar

' Set axes titles
c1Chart1.ChartArea.AxisX.Text = "Kitchen Electronics"
c1Chart1.ChartArea.AxisY.Text = "Price"

' Set format for the Y-axis annotation
    c1Chart1.ChartArea.AxisY.AnnoFormat = FormatEnum.NumericCurrency
End Sub

```

**To write code in C#**

C#

```

private void Form1_Load(object sender, EventArgs e)
{
    // Clear previous data
    c1Chart1.ChartGroups[0].ChartData.SeriesList.Clear();

    // Add Data
    string[] ProductNames = { "Hand Mixer", "Stand Mixer", "Can Opener",
    "Toaster" ,"Blender" ,"Food Processor" ,"Slow Cooker" ,"Microwave"};
    int[] PriceX = { 80, 400, 20, 60, 150, 300, 130, 500 };

    // create single series for product price
    C1.Win.C1Chart.ChartDataSeries ds1 =
c1Chart1.ChartGroups[0].ChartData.SeriesList.AddNewSeries();
    ds1.Label = "Price X";

    //Use the CopyDataIn method of the ChartDataArray object to copy the X and
Y value data into the data series
    ds1.X.CopyDataIn(ProductNames);
    ds1.Y.CopyDataIn(PriceX);

    // Set chart type
    c1Chart1.ChartArea.Inverted = true;
    c1Chart1.ChartGroups[0].ChartType = C1.Win.C1Chart.Chart2DTypeEnum.Bar;

    // Set axes titles
    c1Chart1.ChartArea.AxisX.Text = "Kitchen Electronics";
    c1Chart1.ChartArea.AxisY.Text = "Price";

    //Set format for the Y-axis annotation
    c1Chart1.ChartArea.AxisY.AnnoFormat = FormatEnum.NumericCurrency;
}

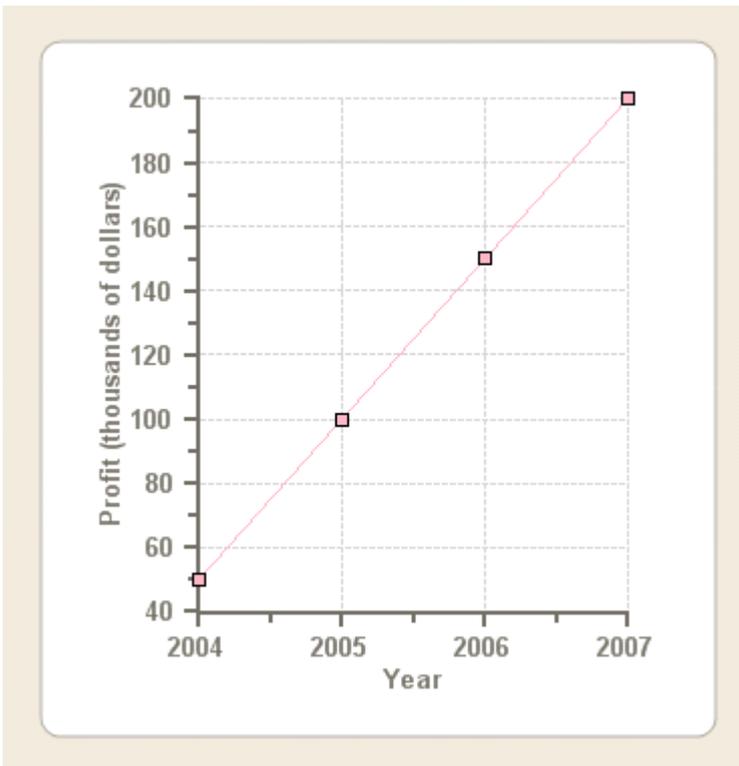
```

**Line Chart Tutorial**

This section provides instructions for building a Line Chart with symbols which is the default chart type. A Line chart is another simple way to show the relationship of the data.

In this example we will create a Line chart with symbols. There will be only one line so we will use one data series. The line will display the company's profit growth over time. The horizontal axis, in this example, Axis X, represents years and the vertical axis, in this example, Axis Y, represents profit in thousands of dollars.

Once you complete the following steps your chart will appear like the following line chart with symbols:



**To create a Line Chart with Symbols at design time**

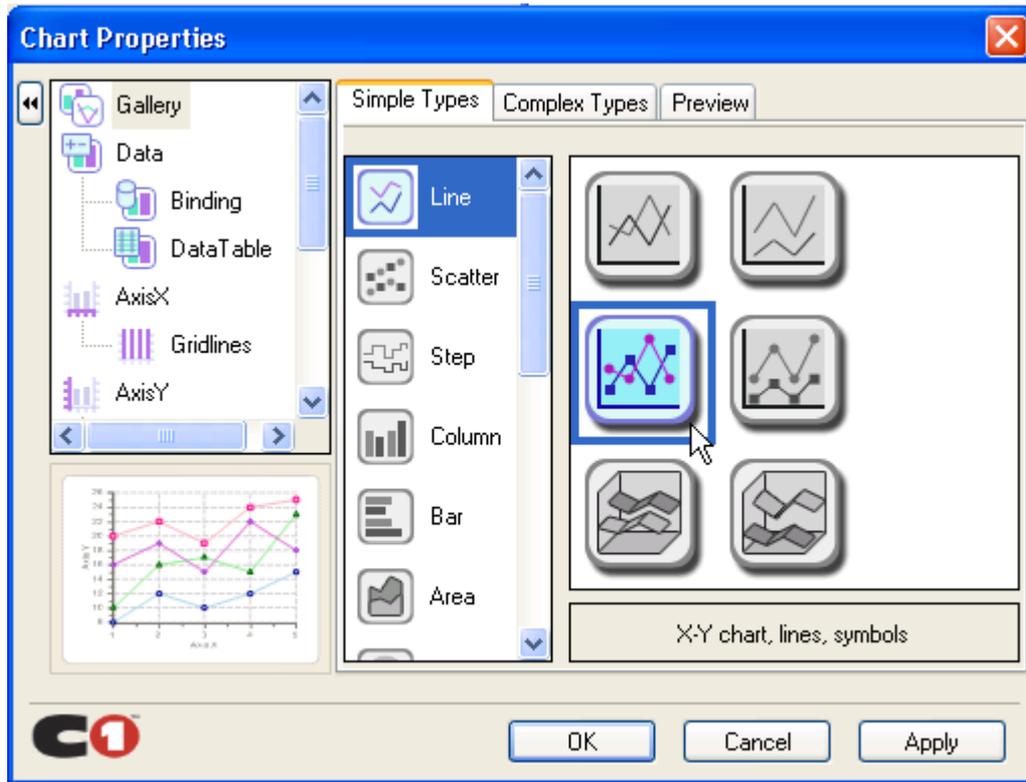
**This task assumes you have already added the C1Chart control to the form.**

## Set the Chart Type

The first step in configuring a chart through the Chart Properties designer is to select a gallery type from the available chart types.

1. Right-click on the **C1Chart** control and click **Chart Properties** from its short-cut menu.
2. Select **Gallery** from the treeview pane, then select **Line** from the **Simple Types** list.
3. In the pane next to the main chart types, select the X-Y chart, lines, symbols subtype, and click **Apply**.

The default Line chart will add two data series which will create two lines.



## Modify the Data Series

4. Select **Data** from the treeview pane.
5. Click **Remove** to remove Series 1, Series 2, and Series 3.

## Add Data to the Data Series

6. Select Data->DataTable from the treeview pane. Click in the XY data table and enter the following numerical X-Y-values: (2004,50), (2005,100), (2006, 150), (2007, 100). **Note:** Click Tab to move the cursor to the next X or Y value.
7. Delete the last XY value from the default line chart.
8. Click **Apply** to update the chart.

A preview image of the updated chart appears in the lower left pane of the Chart Properties designer.

## Modify Axes Appearance

Next we will modify the X and Y axis default title and we will also change the axes font style using the Chart Properties designer. Select **AxisX** from the treeview pane, then select the annotation tab.

9. Select **AxisX** from the tree.
10. In the AxisX tab, type the title, "Year", then click on the **Font** button. The Font dialog box appears.
11. Change the Font style to Bold and the Font size to 10, then click **OK**.
12. Click **Apply** in the **Chart Properties** designer. The modifications for the X-Axis appear on the Line chart.
13. Select AxisY from the tree.
14. In the AxisY tab, type the title, "Profit (thousands of dollars)", then click on the **Font** button. The Font dialog box appears.
15. Change the Font style to Bold and the Font size to 10, then click **OK**.
16. Click **Apply** in the **Chart Properties** designer. The modifications for the Y-Axis appear on the Line chart.

17. Click **OK** to close the **Chart Properties** designer.

Congratulations! You created a Line symbol chart using the Chart Properties designer.

### To create a Line Chart with Symbols programmatically

## A Line chart can be created programmatically using the following steps:

1. Add **C1Chart** to the form.
2. Right-click the form and select View Code to view the code file, then add the following directive to declare the **C1.Win.C1Chart** namespace:

#### To write code in Visual Basic

```
Visual Basic
Imports C1.Win.C1Chart;
```

#### To write code in C#

```
C#
using C1.Win.C1Chart;
```

3. Double click Form1 and add the following code in the **Form1\_Load** procedure to create the simple pie chart:

#### To write code in Visual Basic

```
Visual Basic
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    'create data for the chart
    Dim xdata() As Double = {2004, 2005, 2006, 2007}
    Dim ydata() As Double = {50, 100, 150, 200}

    'clear previous series
    C1Chart1.ChartGroups(0).ChartData.SeriesList.Clear()

    'add one series to the chart
    Dim ds As C1.Win.C1Chart.ChartDataSeries = _
C1Chart1.ChartGroups(0).ChartData.SeriesList.AddNewSeries()

    'copy the x and y data
    ds.X.CopyDataIn(xdata)
    ds.Y.CopyDataIn(ydata)

    'set the chart type
    C1Chart1.ChartGroups(0).ChartType =
C1.Win.C1Chart.Chart2DTypeEnum.XYPlot

    'create new font for the X and Y axes
```

```
Dim f As Font = New Font("Arial", 10, FontStyle.Bold)
C1Chart1.ChartArea.Style.ForeColor = Color.DarkGray
C1Chart1.ChartArea.AxisX.Font = f
C1Chart1.ChartArea.AxisX.Text = "Year"
C1Chart1.ChartArea.AxisX.GridMajor.Visible = True
C1Chart1.ChartArea.AxisX.GridMajor.Color = Color.LightGray
C1Chart1.ChartArea.AxisY.Font = f
C1Chart1.ChartArea.AxisY.Text = "Profit (thousands of dollars)"
C1Chart1.ChartArea.AxisY.GridMajor.Visible = True
C1Chart1.ChartArea.AxisY.GridMajor.Color = Color.LightGray

'change the default line style appearance
ds.LineStyle.Color = Color.LightPink
ds.LineStyle.Pattern = LinePatternEnum.Solid
ds.LineStyle.Thickness = 1

'change the default symbol style appearance
ds.SymbolStyle.Shape = SymbolShapeEnum.Box
ds.SymbolStyle.Color = Color.LightPink
ds.SymbolStyle.OutlineColor = Color.Black
ds.SymbolStyle.Size = 5
ds.SymbolStyle.OutlineWidth = 1

'set the bgcolor for the plot area
C1Chart1.ChartArea.PlotArea.BackColor = Color.White

End Sub
```

### To write code in C#

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    //create data for the chart
    double[] xdata = { 2004, 2005, 2006, 2007 };
    double[] ydata = { 50, 100, 150, 200 };

    //clear previous series
    c1Chart1.ChartGroups[0].ChartData.SeriesList.Clear();

    //add one series to the chart
    C1.Win.C1Chart.ChartDataSeries ds =
c1Chart1.ChartGroups[0].ChartData.SeriesList.AddNewSeries();

    //copy the x and y data
    ds.X.CopyDataIn(xdata);
    ds.Y.CopyDataIn(ydata);

    //set the chart type
    c1Chart1.ChartGroups[0].ChartType =
```

```
C1.Win.C1Chart.Chart2DTypeEnum.XYPlot;

    //create new font for the X and Y axes
    Font f = new Font("Arial", 10, FontStyle.Bold);
    c1Chart1.ChartArea.Style.ForeColor = Color.DarkGray;
    c1Chart1.ChartArea.AxisX.Font = f;
    c1Chart1.ChartArea.AxisX.Text = "Year";
    c1Chart1.ChartArea.AxisX.GridMajor.Visible = true;
    c1Chart1.ChartArea.AxisX.GridMajor.Color = Color.LightGray;
    c1Chart1.ChartArea.AxisY.Font = f;
    c1Chart1.ChartArea.AxisY.Text = "Profit (thousands of dollars)";
    c1Chart1.ChartArea.AxisY.GridMajor.Visible = true;
    c1Chart1.ChartArea.AxisY.GridMajor.Color = Color.LightGray;

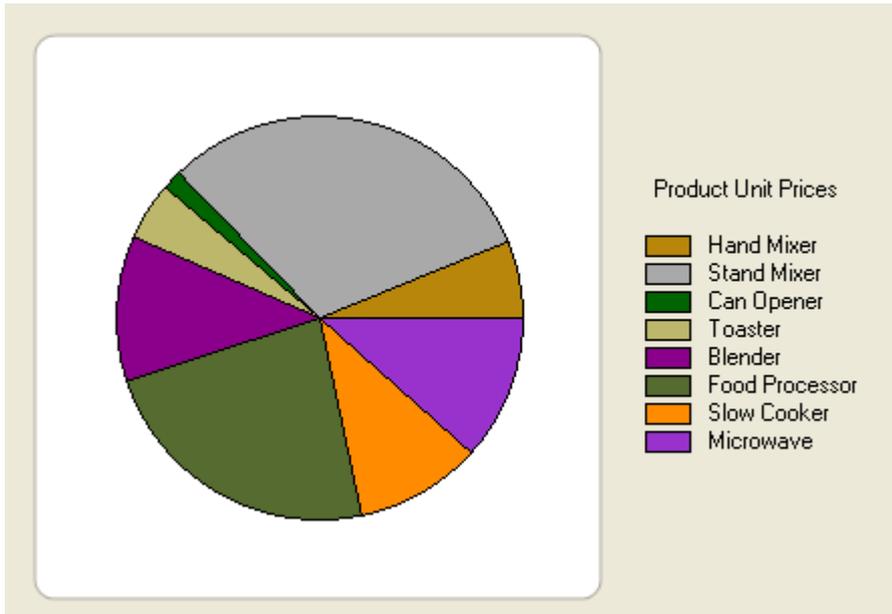
    //modify line style appearance
    ds.LineStyle.Color = Color.LightPink;
    ds.LineStyle.Pattern = LinePatternEnum.Solid;
    ds.LineStyle.Thickness = 1;

    //modify the symbol style appearance
    ds.SymbolStyle.Shape = SymbolShapeEnum.Box;
    ds.SymbolStyle.Color = Color.LightPink;
    ds.SymbolStyle.OutlineColor = Color.Black;
    ds.SymbolStyle.Size = 5;
    ds.SymbolStyle.OutlineWidth = 1;
    c1Chart1.ChartArea.PlotArea.BackColor = Color.White;
}
```

## Pie Chart Tutorial

This section provides step-by-step instructions for building a Pie Chart. A Pie chart is used to display simple values. Unlike other C1Chart types, the Pie chart must have more than one series in order to display multiple slices. If there was only one series the Pie would just be a circle without any slices. In this example we will create a pie with eight slices so we will have eight series with a single point each as opposed to a single series with eight points in it.

The graph shows the information as a Pie chart with each product representing one slice of the pie.



Both steps are shown below for creating the Pie chart at design time or programmatically through the Chart objects.

### To create a Pie chart at design time

This task assumes you have already added the C1Chart control to the form.

## Set the Chart Type

The first step in configuring a chart through the **Chart Properties** designer is to select a gallery type from the available chart types.

1. Right-click on the **C1Chart** control and click **Chart Properties** from its short-cut menu.
2. Select Gallery from the treeview pane, then select Pie from the Simple Types list.
3. In the pane, select the top-left Pie, and click **Apply**.

The default Pie will add five pies with each having four data series.

## Modify the Data Series

4. Select Data from the treeview pane.
5. Click **Remove** to remove all of the series.
6. Click **Add** to add a new series for the Y-values that will display the product's prices.

A warning will appear to remind you that you must add data to the new series.

7. Click **Add** seven times to add the remaining seven data series so the Pie will have eight slices.
8. Select Data->DataTable from the treeview pane.
9. Click on the first field in the data table and enter the following (X,Y) values: (1, 80), (1, 400), (1, 20), (1, 60), (1, 150), (1, 300), (1, 130), and (1, 500).
10. Select the Data item from the treeview pane and then select the Data Series in the Data Series listbox and enter the Label for each data series in the **Label** textbox to the following:
  - Series0 to "Hand Mixer (\$80.00)"
  - Series1 to "Stand Mixer (\$400.00)"
  - Series2 to "Can Opener (\$20.00)"
  - Series3 to "Toaster (\$60.00)"
  - Series4 to "Blender (\$150.00)"

- Series5 to "Food Processor (\$300.00)"
- Series6 to "Slow Cooker (\$130.00)"
- Series7 to "Microwave (\$500.00)"
- Add the Legend to the Pie Chart

To enable the legend, select Legend under **Appearance** in the **Chart Properties** designer treeview pane and select the **Legend** checkbox.

### To create a Pie chart programmatically

This task assumes you have already added the C1Chart control to the form.

## A Pie chart can be created programmatically using the following steps:

1. Add the following directive to declare the **C1.Win.C1Chart** namespace:

#### To write code in Visual Basic

```
Visual Basic
Imports C1.Win.C1Chart;
```

#### To write code in C#

```
C#
using C1.Win.C1Chart;
```

2. Double click Form1 and add the following code in the **Form1\_Load** procedure to create the simple pie chart:

#### To write code in Visual Basic

```
Visual Basic
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Set chart type
    C1Chart1.ChartArea.Inverted = True
    C1Chart1.ChartGroups(0).ChartType = C1.Win.C1Chart.Chart2DTypeEnum.Pie

    ' Clear previous data
    C1Chart1.ChartGroups(0).ChartData.SeriesList.Clear()

    ' Add Data
    Dim ProductNames As String() = {"Hand Mixer", "Stand Mixer", "Can
Opener", "Toaster", "Blender", "Food Processor", "Slow Cooker", "Microwave"}
    Dim PriceX As Integer() = {80, 400, 20, 60, 150, 300, 130, 500}

    'get series collection
    Dim dscoll As ChartDataSeriesCollection =
C1Chart1.ChartGroups(0).ChartData.SeriesList
    For i As Integer = 0 To PriceX.Length - 1
        'populate the series
```

```

        Dim series As ChartDataSeries = dscoll.AddNewSeries()
        'Add one point to show one pie
        series.PointData.Length = 1
        'Assign the prices to the Y Data series
        series.Y(0) = PriceX(i)
        'format the product name and product price on the legend
        series.Label = String.Format("{0} ({1:c})", ProductNames(i),
PriceX(i))
        Next

        ' show pie Legend
        C1Chart1.Legend.Visible = True
        'add a title to the chart legend
        C1Chart1.Legend.Text = "Product Unit Prices"
End Sub

```

### To write code in C#

```

C#
private void Form1_Load(object sender, EventArgs e)
{

    // Set chart type
    c1Chart1.ChartArea.Inverted = true;
    c1Chart1.ChartGroups[0].ChartType = C1.Win.C1Chart.Chart2DTypeEnum.Pie;

    // Clear previous data
    c1Chart1.ChartGroups[0].ChartData.SeriesList.Clear();

    // Add Data
    string[] ProductNames = { "Hand Mixer", "Stand Mixer", "Can Opener",
    "Toaster" ,"Blender" ,"Food Processor" ,"Slow Cooker" ,"Microwave"};
    int[] PriceX = { 80, 400, 20, 60, 150, 300, 130, 500 };

    //get series collection
    ChartDataSeriesCollection dscoll =
c1Chart1.ChartGroups[0].ChartData.SeriesList;

    //populate the series
    for (int i=0; i < PriceX.Length; i++)
    {

        ChartDataSeries series = dscoll.AddNewSeries();
        //Add one point to show one pie
        series.PointData.Length = 1;
        //Assign the prices to the Y Data series
        series.Y[0] = PriceX[i];
        //format the product name and product price on the legend
        series.Label = string.Format("{0} ({1:c})", ProductNames[i],
PriceX[i]);
    }
}

```

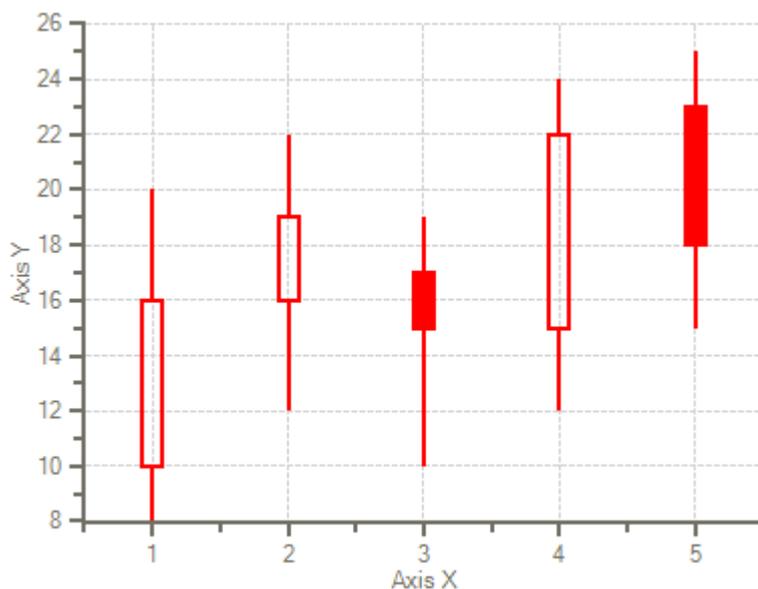
```
// show pie Legend
c1Chart1.Legend.Visible = true;

//add a title to the chart legend
c1Chart1.Legend.Text = "Product Unit Prices";
}
```

## Candle Chart Tutorial

This section provides step-by-step instructions for building a Candle Chart. The graph shows the information as a candle chart with four Y variables on the y-axis that represent the hi, low, open, and close values of the stock and the x-axis values provide the position for each candle.

The following chart illustration is shown before you begin creating the chart:



Both steps are shown below for creating the Candle chart at design time or programmatically through the Chart objects.

### To create a candle chart at design time

This topic walks you through the steps for creating a candle chart using the **Chart Properties** designer. In this topic you will learn how to set the specific chart type, modify the data series, add data to the chart, and format the appearance of the candle all at design-time in a few simple steps.

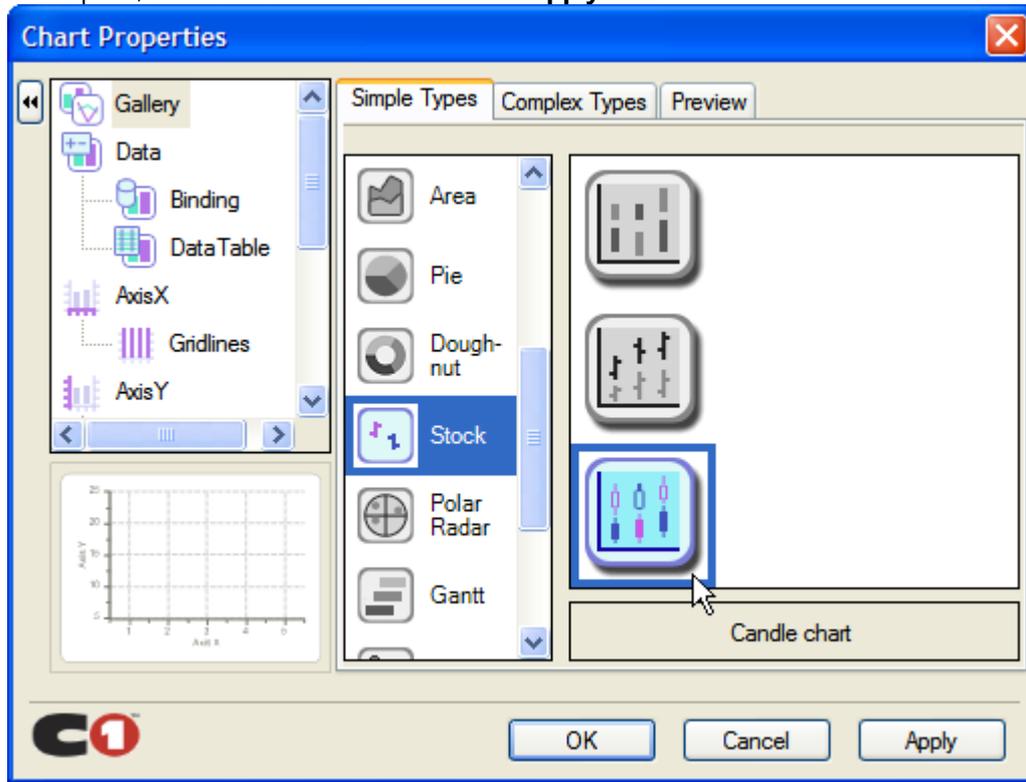
 **Note:** This task assumes you have already added the C1Chart control to the form.

## Set the Chart Type

The first step in configuring a chart through the smart tag **Chart Properties** is to select a gallery type from the wealth of chart types available. From basic chart types like Bar and Lines to more complex charts like Polar and Surface, C1Chart allows you to deal with any data visualization needed.

1. Right-click on the **C1Chart** control and click **Chart Properties** from its short-cut menu.

2. Select **Gallery** from the treeview pane, then select **Stock** from the **Simple Types** list.
3. In the pane, select the Candle chart and click **Apply**.



## Modify the Data Series

4. Select **Data** from the treeview pane.
5. Click **Remove** to remove Series0, Series 1, Series 2, and Series 3.
6. Click **Add** to add a new series for the Y-values that will display the values for each stock. A warning will appear to remind you that you must add data to the new series.

## Add Data to the Data Series

7. Select **Data->DataTable** from the treeview pane. Add the following data to the data table:
8. Add data values as in the following table.

series 0				
X	Y (High)	Y1 (Low)	Y2 (Open)	Y3 (Close)
1	8	20	10	16
2	12	22	16	19
3	10	19	17	15
4	12	24	15	22
5	15	25	23	18
*				

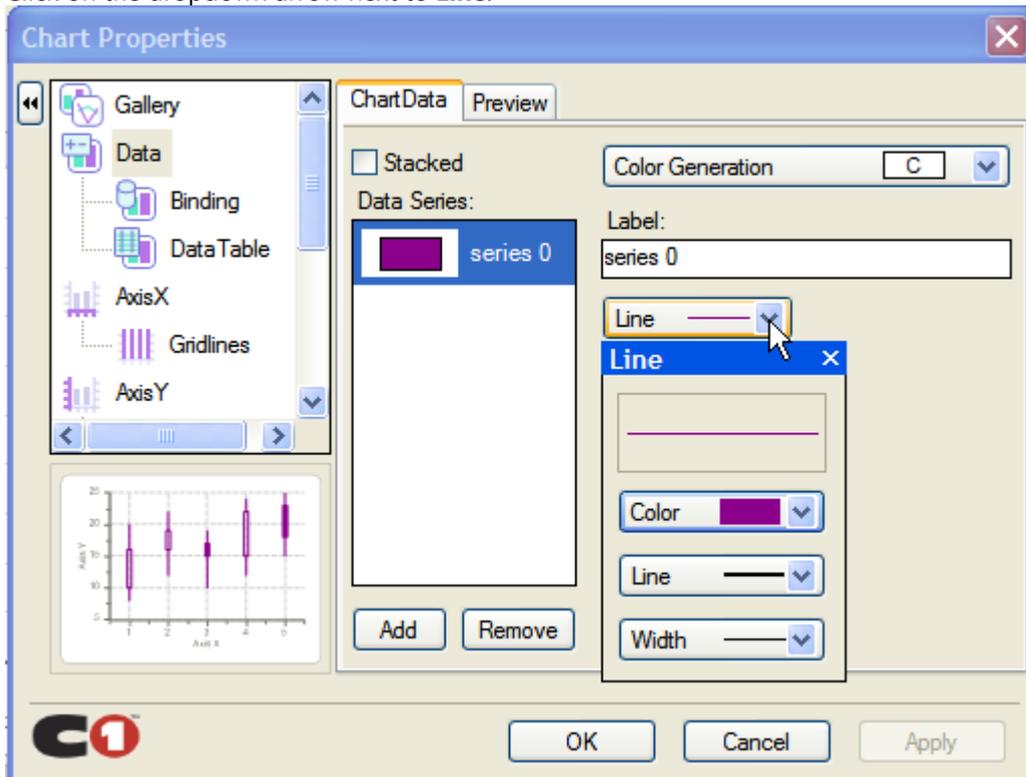
**Note:** X represents the position on the x-axis, Y represents the Hi value, Y1 represents the low value, Y2 represents the open value, and Y3 represents the closing value.

- Click **Apply** to update the chart  
A preview image of the updated chart appears in the lower left pane of the Chart Properties designer.

## Format Candle

Next we will change the default line color to red, increase the body width of the candle, and increase the line thickness. We will use the `LineStyle` property to change the color of the lines to red, the `SymbolStyle` property to change the body width of the candle, and `LineStyle` property to change the thickness of the lines.

- Select **Data** from the treeview pane, and then select the ChartData tab.
- Click on the dropdown arrow next to **Line**.



- Click on the dropdown arrow next to **Color** and select Red.
- Click on the dropdown arrow next to **Width** and select the second line.
- Click **OK** to save and close the **Chart Properties** dialog box.
- Open the C1Chart properties window and click on the ellipsis button next to **SeriesList**. The **ChartDataSeries Collection Editor** appears.
- Expand the **SymbolSize** and set the **SymbolStyle.Size** property to 10.

The body of the candle increase from its default size, 5, to 10.

### To create a candle chart programmatically

A Candle chart can be created programmatically using the following steps:

- Add the following directive to declare the **C1.Win.C1Chart** namespace:

#### To write code in Visual Basic

```
Visual Basic
Imports C1.Win.C1Chart
```

**To write code in C#**

C#

```
using C1.Win.C1Chart;
```

2. Double click Form1 and add the following code in the **Form1\_Load** procedure to create the simple Candle chart:

**To write code in Visual Basic**

Visual Basic

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    c1Chart1.ChartGroups(0).ChartData.SeriesList.AddNewSeries()

    'Declare the x and y variables as double to represent the x, y, y1, y2, and
    y3 values and assign the values
    Dim x As Double() = New Double() {1, 2, 3, 4, 5}
    Dim y_hi As Double() = New Double() {8, 12, 10, 12, 15}
    Dim y_low As Double() = New Double() {20, 22, 19, 24, 25}
    Dim y_open As Double() = New Double() {10, 16, 17, 15, 23}
    Dim y_close As Double() = New Double() {16, 19, 15, 22, 18}

    'copy the x, y, y1, y2, and y3 data into the chart group
    c1Chart1.ChartGroups(0).ChartData(0).X.CopyDataIn(x)
    c1Chart1.ChartGroups(0).ChartData(0).Y.CopyDataIn(y_hi)
    c1Chart1.ChartGroups(0).ChartData(0).Y1.CopyDataIn(y_low)
    c1Chart1.ChartGroups(0).ChartData(0).Y2.CopyDataIn(y_open)
    c1Chart1.ChartGroups(0).ChartData(0).Y3.CopyDataIn(y_close)

    'assign the candle chart to the chart type
    c1Chart1.ChartGroups(0).ChartType = C1.Win.C1Chart.Chart2DTypeEnum.Candle
    'Make the rising candles transparent to show rising prices
    c1Chart1.ChartGroups(0).HiLoData.FillTransparent = True
    'Declare the color for the lines
    c1Chart1.ChartGroups(0).ChartData.SeriesList(0).LineStyle.Color =
System.Drawing.Color.Red
    'Increase the line thickness
    c1Chart1.ChartGroups(0).ChartData.SeriesList(0).LineStyle.Thickness = 2
    'Increase the body width of the candle
    c1Chart1.ChartGroups(0).ChartData.SeriesList(0).SymbolStyle.Size = 10

End Sub
```

**To write code in C#**

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    c1Chart1.ChartGroups[0].ChartData.SeriesList.AddNewSeries();

    //Declare the x and y variables as double to represent the x, y, y1, y2,
```

```
and y3 values and assign the values
    double[] x = new double[] { 1, 2, 3, 4, 5 };
    double[] y_hi = new double[] { 8, 12, 10, 12, 15 };
    double[] y_low = new double[] { 20, 22, 19, 24, 25 };
    double[] y_open = new double[] { 10, 16, 17, 15, 23 };
    double[] y_close = new double[] { 16, 19, 15, 22, 18 };

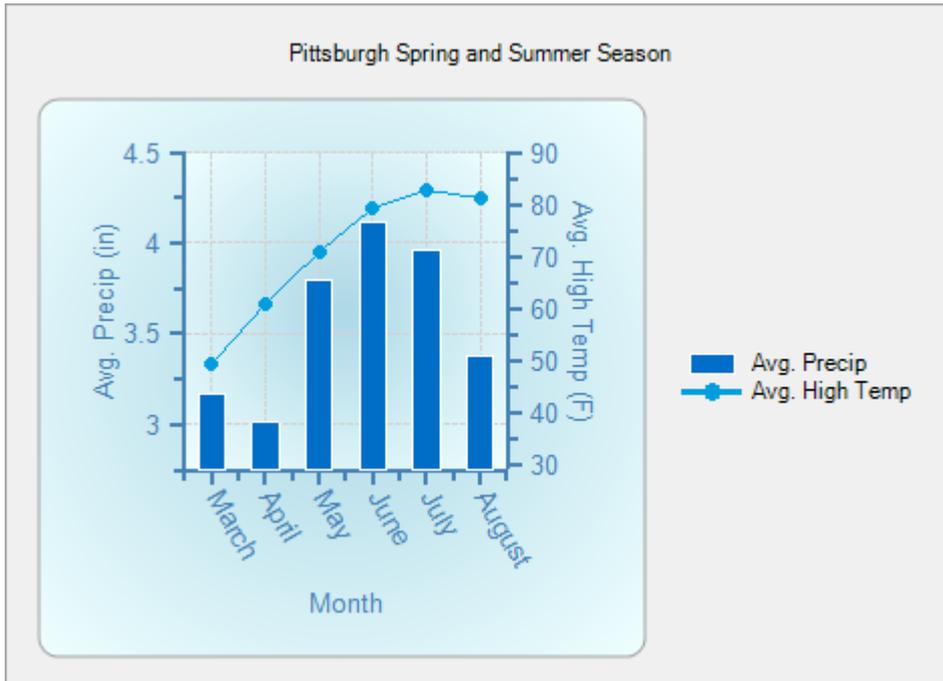
    //copy the x, y, y1, y2, and y3 data into the chart group
    c1Chart1.ChartGroups[0].ChartData[0].X.CopyDataIn(x);
    c1Chart1.ChartGroups[0].ChartData[0].Y.CopyDataIn(y_hi);
    c1Chart1.ChartGroups[0].ChartData[0].Y1.CopyDataIn(y_low);
    c1Chart1.ChartGroups[0].ChartData[0].Y2.CopyDataIn(y_open);
    c1Chart1.ChartGroups[0].ChartData[0].Y3.CopyDataIn(y_close);

    //assign the candle chart to the chart type
    c1Chart1.ChartGroups[0].ChartType =
C1.Win.C1Chart.Chart2DTypeEnum.Candle;
    //Make the rising candles transparent to show rising prices
    c1Chart1.ChartGroups[0].HiLoData.FillTransparent = true;
    //Declare the color for the lines
    c1Chart1.ChartGroups[0].ChartData.SeriesList[0].LineStyle.Color =
System.Drawing.Color.Red;
    //Increase the line thickness
    c1Chart1.ChartGroups[0].ChartData.SeriesList[0].LineStyle.Thickness = 2;
    c1Chart1.ChartGroups[0].ChartData.SeriesList[0].SymbolStyle.Size = 10;
}
```

## Multiple Charts Tutorial

This section provides step-by-step instructions for programatically adding a **Bar** and **XYPlot** chart on the **C1Chart** control. The graph shows the **Bar** chart with a y-axis that represents the double values for the average precipitation and the x-axis that represents string values for each month from March till August. The **XYPlot** chart has a y2-axis that represents the double values for the average high temperature for each month from March till August and the x-axis that represents string values for each month from March till August.

The following chart illustration is shown before you begin creating the chart:



Multiple charts can be created programmatically using the following steps:

1. Add the following directive to declare the **C1.Win.C1Chart** namespace:

**To write code in Visual Basic**

Visual Basic

```
Imports C1.Win.C1Chart
```

**To write code in C#**

C#

```
using C1.Win.C1Chart;
```

2. Double click Form1 and add the following code in the **Form1\_Load** procedure to create the multiple charts:

**To write code in Visual Basic**

Visual Basic

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
    Dim cgroup As ChartGroup = c1Chart1.ChartGroups.Group0
    cgroup.ChartType = Chart2DTypeEnum.Bar
    'input the data through the series collection
    Dim cdsc As ChartDataSeriesCollection = cgroup.ChartData.SeriesList
    cdsc.Clear()

    'remove default data
    'create the series object from the collection and add data
    Dim cds As ChartDataSeries = cdsc.AddNewSeries()
    ' Add Data for ChartGroup0, Bar chart
    Dim MonthNames As String() = {"March", "April", "May", "June", "July",
    "August"}
    Dim AvgPrecip As Double() = {3.17, 3.01, 3.8, 4.12, 3.96, 3.38}
```

```
'create a label for the Bar chart data series
cds.Label = "Avg. Precip"
'Use the CopyDataIn method of the ChartdataArray object to copy the X and Y
value data into the data series
cds.X.CopyDataIn(MonthNames)
cds.Y.CopyDataIn(AvgPrecip)
'create variable for chart area

Dim carea As Cl.Win.ClChart.Area = c1Chart1.ChartArea
'Set axes titles for the ChartGroup0 (Bar)
carea.AxisX.Text = "Month"
carea.AxisY.Text = "Avg. Precip (in)"
'create and add the data for the XY chart in Group1
Dim cgroup2 As ChartGroup = c1Chart1.ChartGroups.Group1
cgroup2.ChartType = Chart2DTypeEnum.XYPlot
'input the bar chart data of group1 through the series collection
Dim cdsc2 As ChartDataSeriesCollection = cgroup2.ChartData.SeriesList
'create the series object from the second collection and add data
Dim cds2 As ChartDataSeries = cdsc2.AddNewSeries()
cds2.X.CopyDataIn(MonthNames)
cds2.Y.CopyDataIn(New Double() {49.5, 60.7, 70.8, 79.1, 82.7, 81.1})
cds2.Label = "Avg. High Temp"

'customize axes
'create new font for the X, Y and Y2 axes
Dim f As New Font("Arial", 10)
carea.AxisX.Font = f
carea.AxisY.Font = f
carea.AxisX.ForeColor = Color.SteelBlue
carea.AxisY.ForeColor = Color.SteelBlue
carea.AxisY2.ForeColor = Color.SteelBlue
carea.AxisY2.Font = f

'Set axes titles for the ChartGroup1 (XYPlot)
carea.AxisY2.Text = "Avg. High Temp (F)"
'set axis bounds
carea.AxisY.Min = 2.75
carea.AxisY2.Min = 30
carea.AxisY2.Max = 90
carea.AxisY.UnitMinor = 0.25

'rotate the axis X annotation
carea.AxisX.AnnotationRotation = 60

'add legend
c1Chart1.Legend.Visible = True

'add header
c1Chart1.Header.Visible = True
c1Chart1.Header.Text = "Pittsburgh Spring and Summer Season"
```

```

    'add visual effects
    Dim s As Style = carea.Style
    s.ForeColor = Color.White
    s.BackColor = Color.LightBlue
    s.BackColor2 = Color.Azure
    s.GradientStyle = GradientStyleEnum.Radial
    c1Chart1.ColorGeneration = ColorGeneration.Flow
End Sub

```

### To write code in C#

C#

```

private void Form1_Load(object sender, EventArgs e)
{
    ChartGroup cgroup = c1Chart1.ChartGroups.Group0;
    cgroup.ChartType = Chart2DTypeEnum.Bar;
    //input the data through the series collection
    ChartDataSeriesCollection cdsc = cgroup.ChartData.SeriesList;
    cdsc.Clear();

    //remove default data
    //create the series object from the collection and add data
    ChartDataSeries cds = cdsc.AddNewSeries();
    // Add Data for ChartGroup0, Bar chart
    string[] MonthNames = { "March", "April", "May", "June", "July",
"August" };
    double[] AvgPrecip = { 3.17, 3.01, 3.80, 4.12, 3.96, 3.38};
    //create a label for the Bar chart data series
    cds.Label = "Avg. Precip";
    //Use the CopyDataIn method of the ChartDataArray object to copy the X
and Y value data into the data series
    cds.X.CopyDataIn(MonthNames);
    cds.Y.CopyDataIn(AvgPrecip);

    //create variable for chart area
    C1.Win.C1Chart.Area carea = c1Chart1.ChartArea;
    //Set axes titles for the ChartGroup0 (Bar)
    carea.AxisX.Text = "Month";
    carea.AxisY.Text = "Avg. Precip (in)";
    //create and add the data for the XY chart in Group1
    ChartGroup cgroup2 = c1Chart1.ChartGroups.Group1;
    cgroup2.ChartType = Chart2DTypeEnum.XYPlot;
    //input the bar chart data of group1 through the series collection
    ChartDataSeriesCollection cdsc2 = cgroup2.ChartData.SeriesList;
    //create the series object from the second collection and add data
    ChartDataSeries cds2 = cdsc2.AddNewSeries();
    cds2.X.CopyDataIn(MonthNames);
    cds2.Y.CopyDataIn(new double[] { 49.5, 60.7, 70.8, 79.1, 82.7, 81.1});
    cds2.Label = "Avg. High Temp";
}

```

```
//customize axes
//create new font for the X, Y and Y2 axes
Font f = new Font("Arial", 10);
carea.AxisX.Font = f;
carea.AxisY.Font = f;
carea.AxisX.ForeColor = Color.SteelBlue;
carea.AxisY.ForeColor = Color.SteelBlue;
carea.AxisY2.ForeColor = Color.SteelBlue;
carea.AxisY2.Font = f;
//Set axes titles for the ChartGroup1 (XYPlot)
carea.AxisY2.Text = "Avg. High Temp (F)";
//set axis bounds
carea.AxisY.Min = 2.75;
carea.AxisY2.Min = 30;
carea.AxisY2.Max = 90;
carea.AxisY.UnitMinor = .25;

//rotate the axis X annotation
carea.AxisX.AnnotationRotation = 60;
//add legend
clChart1.Legend.Visible = true;

//add header
clChart1.Header.Visible = true;
clChart1.Header.Text = "Pittsburgh Spring and Summer Season";

//add visual Effects
Style s = carea.Style;
s.ForeColor = Color.White;
s.BackColor = Color.LightBlue;
s.BackColor2 = Color.Azure;
s.GradientStyle = GradientStyleEnum.Radial;
clChart1.ColorGeneration = ColorGeneration.Flow;
}
```

## Chart for WinForms Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET. By following the steps outlined in the help, you will be able to create various chart types by utilizing C1Chart's properties. You also will get a better feel of the capabilities of the **C1Chart** product through exploring its well designed interface and using its editors.

### Rotating the Y-Axis Title

To rotate the Y-Axis title, use the **C1Chart.ChartArea.AxisY.Rotation** enum like in the following example:

#### To write code in Visual Basic

Visual Basic

```
c1Chart2.ChartArea.AxisY.Rotation = RotationEnum.Rotate180
```

#### To write code in C#

C#

```
c1Chart2.ChartArea.AxisY.Rotation = RotationEnum.Rotate180;
```

### Rotating Data Labels

To rotate a datalabel for example to 90 degrees, use the following code:

#### To write code in Visual Basic

Visual Basic

```
dataSeries.DataLabel.Style.Rotation = RotationEnum.Rotate90
```

#### To write code in C#

C#

```
dataSeries.DataLabel.Style.Rotation = RotationEnum.Rotate90;
```

### Displaying the Data Label as a Percent in Pie Charts

To represent the sum of all series at a given point in a Pie chart, use the {YVAL} like in the following example:

#### To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load

    ' use light colors so all of the labels are easily read.
    C1Chart1.ColorGeneration = ColorGeneration.Flow
```

```

' maximize the ChartArea
ClChart1.ChartArea.Margins.SetMargins(0, 0, 0, 0)
ClChart1.ChartArea.Style.Border.BorderStyle = BorderStyleEnum.None
' Set chart type
ClChart1.ChartArea.Inverted = True
ClChart1.ChartGroups(0).ChartType = Cl.Win.ClChart.Chart2DTypeEnum.Pie
' Clear previous data
ClChart1.ChartGroups(0).ChartData.SeriesList.Clear()
' Add Data

Dim ProductNames As String() = {"Mortgage", "Car", "Food", "Day Care",
"Other", "Savings", "Utilities"}
Dim PriceX As Integer() = {2000, 1200, 500, 500, 450, 400, 350}

'get series collection
Dim dscoll As ChartDataSeriesCollection =
ClChart1.ChartGroups(0).ChartData.SeriesList
'populate the series
For i As Integer = 0 To PriceX.Length - 1
    Dim series As ChartDataSeries = dscoll.AddNewSeries()
    'Assign the prices to the Y Data series
    series.Y(0) = PriceX(i)
    'format the product name and product price on the legend

    series.Label = String.Format("{0} ({1:c})", ProductNames(i),
PriceX(i))

    series.DataLabel.Text = "#{TEXT}" & vbCrLf & vbCrLf & "#{YVAL}
({%YVAL:%})"

    series.DataLabel.Compass = LabelCompassEnum.RadialText
    series.DataLabel.Offset = -5
    series.DataLabel.Visible = True

Next
' show pie Legend
ClChart1.Legend.Visible = True
'add a title to the chart legend
ClChart1.Legend.Text = "Monthly Expenses"
End Sub

```

### To write code in C#

C#

```

private void Form1_Load(object sender, EventArgs e)
{
    // use light colors so all of the labels are easily read.
    clChart1.ColorGeneration = ColorGeneration.Flow;

    // maximize the ChartArea
    clChart1.ChartArea.Margins.SetMargins(0, 0, 0, 0);
    clChart1.ChartArea.Style.Border.BorderStyle = BorderStyleEnum.None;
    // Set chart type
    clChart1.ChartArea.Inverted = true;

```

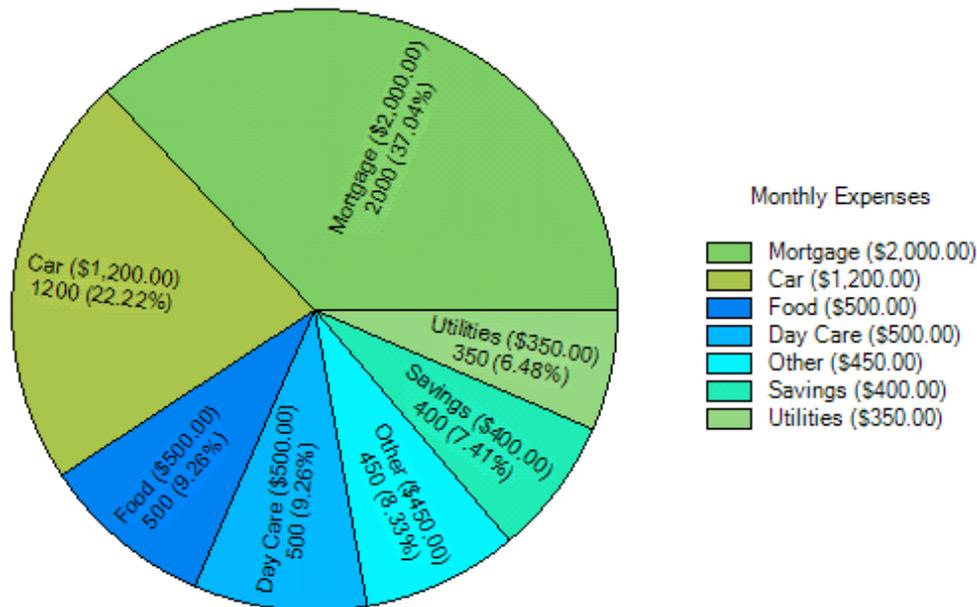
```
c1Chart1.ChartGroups[0].ChartType = Cl.Win.ClChart.Chart2DTypeEnum.Pie;
// Clear previous data
c1Chart1.ChartGroups[0].ChartData.SeriesList.Clear();
// Add Data

string[] ProductNames = { "Mortgage", "Car", "Food", "Day Care", "Other",
"Savings", "Utilities" };
int[] PriceX = {2000, 1200, 500, 500, 450, 400, 350 };

//get series collection
ChartDataSeriesCollection dscoll =
c1Chart1.ChartGroups[0].ChartData.SeriesList;
//populate the series
for (int i = 0; i < PriceX.Length; i++)
{
    ChartDataSeries series = dscoll.AddNewSeries();
    //Add one point to show one pie
    series.PointData.Length = 1;
    //Assign the prices to the Y Data series
    series.Y[0] = PriceX[i];
    //format the product name and product price on the legend
    series.Label = string.Format("{0} ({1:c})", ProductNames[i], PriceX[i]);
    series.DataLabel.Text = "{#TEXT}\r\n{#YVAL} ({%YVAL:%})";
    series.DataLabel.Compass = LabelCompassEnum.RadialText;
    series.DataLabel.Offset = -5;
    series.DataLabel.Visible = true;
}
// show pie Legend
c1Chart1.Legend.Visible = true;
//add a title to the chart legend
c1Chart1.Legend.Text = "Monthly Expenses";
}
```

## This task illustrates the following:

The data labels represent the point value and percentage of each slice or point on the Pie chart:



## Setting the Font Style for Data Labels

To set the font style for the data labels for example, to bold use the following code:

### To write code in Visual Basic

#### Visual Basic

```
label.Style.Font = New System.Drawing.Font("Arial", 10,
System.Drawing.FontStyle.Bold)
```

### To write code in C#

#### C#

```
label.Style.Font = new System.Drawing.Font("Arial", 10,
System.Drawing.FontStyle.Bold);
```

## Adding a Data Label on Top of Each Bar

To add a data label on top of each bar, use the following code:

### To write code in Visual Basic

#### Visual Basic

```
Dim sc As ChartDataSeriesCollection =
C1Chart1.ChartGroups(0).ChartData.SeriesList
Dim i As Integer
For i = 0 To sc.Count - 1
With sc(i).DataLabel
.Visible = True
.Compass = LabelCompassEnum.North
.Text = "{#YVAL}"

```

```
End With
Next
```

### To write code in C#

```
C#
ChartDataSeriesCollection sc = c1Chart1.ChartGroups[0].ChartData.SeriesList;
int i = 0;
for (i = 0; i <= sc.Count - 1; i++)
{
    var _with1 = sc[i].DataLabel;
    _with1.Visible = true;
    _with1.Compass = LabelCompassEnum.North;
    _with1.Text = "{#YVAL}";
}
```

## Wrapping Labels

You can manually make the text wrap by using the newline character (\n)

For an example that uses the newline character (\n) to wrap the data labels see [Displaying the Data Label as a Percent in Pie Charts](#).

## Adding a Transparent Label to Adjust the Gap Between the Values and the X-Axis

This code adds a transparent ValueLabel with the appearance of an arrow to the chart; since it is transparent, it is not visible to the end-user. By changing the ValueLabel.MarkerSize property, you can adjust the gap between the values and the x-axis.

### To write code in Visual Basic

```
Visual Basic
c1Chart1.ChartArea.AxisX.AnnoMethod = C1.Win.C1Chart.AnnotationMethodEnum.Mixed
Dim vl As ValueLabel = c1Chart1.ChartArea.AxisX.ValueLabels.AddNewLabel()
vl.Appearance = ValueLabelAppearanceEnum.TriangleMarker
vl.MarkerSize = 50
vl.Color = Color.Transparent
vl.NumericValue = 2
```

### To write code in C#

```
C#
ValueLabel vl = c1Chart1.ChartArea.AxisX.ValueLabels.AddNewLabel();
vl.Appearance = ValueLabelAppearanceEnum.TriangleMarker;
vl.MarkerSize = 50;
vl.Color = Color.Transparent;
vl.NumericValue = 2;
```

## Displaying both the Chart Legend and Chart Header

Both the **Header** and **Legend** have **Compass** and **Location** properties, and you can use one or both of these properties to change the position. The **Compass** property can be set to North, South, East, or West through the **CompassEnum**. By default, the **Header** and **Legend** will be centered within the Compass area. The **Location** property can be set to a **System.Drawing.Point**, which accepts x and y coordinates. Both the x and y coordinates can be set to -1 to use automatic positioning.

### Displaying the Header

The following example positions your chart header at the top. The point coordinates can be adjusted to fine tune the position:

#### To write code in Visual Basic

##### Visual Basic

```
c1Chart1.Header.Text = "My Chart Header"  
c1Chart1.Header.Compass = C1.Win.C1Chart.CompassEnum.North  
c1Chart1.Header.Location = New Point(-1, -1)  
c1Chart1.Header.Visible = True
```

#### To write code in C#

##### C#

```
this.c1Chart1.Header.Text = "My Chart Header";  
this.c1Chart1.Header.Compass = C1.Win.C1Chart.CompassEnum.East;  
this.c1Chart1.Header.Location = new Point(-1, -1);  
this.c1Chart1.Header.Visible = true;
```

### Displaying the Legend

The following example will position your chart legend to the left and you can adjust the point coordinates to fine tune the position.



**Note:** Remember the -1 just uses the default coordinate for the Compass setting, and one or both coordinates can have absolute positions. Just make sure that the point makes sense for the position. For example you probably do not want to set a very high y value for something that has Compass set to North, because the Chart will shrink to accommodate the header.

#### To write code in Visual Basic

##### Visual Basic

```
c1Chart1.Legend.Text = "My Chart Legend"  
c1Chart1.Legend.Compass = C1.Win.C1Chart.CompassEnum.West  
c1Chart1.Legend.Location = new Point(-1, -1)  
c1Chart1.Legend.Visible = True
```

#### To write code in C#

##### C#

```
this.clChart1.Legend.Text = "My Chart Legend";
this.clChart1.Legend.Compass = Cl.Win.ClChart.CompassEnum.West;
this.clChart1.Legend.Location = new Point(-1, -1);
this.clChart1.Legend.Visible = true;
```

## Displaying the Legends Vertically

Under C1Chart Style properties. A property called 'Orientation' can be set to either horizontal or vertical. Please make sure the orientation of the legend is set to 'vertical'.

## Getting the Slice of a Pie with a Click

You can get the slice of a Pie with a click using the following method:

### To write code in Visual Basic

#### Visual Basic

```
Dim seriesIndex = 0
Dim pointIndex = 0
Private Sub ClChart1_MouseMove(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles ClChart1.MouseMove
    Dim si, pi, d As Integer
    If ClChart1.ChartGroups(0).CoordToDataIndex(e.X, e.Y, _
        Cl.Win.ClChart.CoordinateFocusEnum.XandYCoord, si, pi, d) Then
        seriesIndex = si
        pointIndex = pi
    End If
End Sub
Private Sub ClChart1_MouseClick(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles ClChart1.MouseClick
    MsgBox(ClChart1.ChartGroups(0).ChartData(seriesIndex).Y(pointIndex))
End Sub
```

### To write code in C#

#### C#

```
var seriesIndex = 0;
var pointIndex = 0;
private void // ERROR: Handles clauses are not supported in C#
ClChart1_MouseMove(System.Object sender, System.Windows.Forms.MouseEventArgs e)
{
    int si = 0;
    int pi = 0;
    int d = 0;
    if (ClChart1.ChartGroups(0).CoordToDataIndex(e.X, e.Y,
Cl.Win.ClChart.CoordinateFocusEnum.XandYCoord, si, pi, d)) {
        seriesIndex = si;
        pointIndex = pi;
    }
}
```

```
private void // ERROR: Handles clauses are not supported in C#
C1Chart1_MouseClick(System.Object sender, System.Windows.Forms.MouseEventArgs e)
{
    Interaction.MsgBox(C1Chart1.ChartGroups(0).ChartData(seriesIndex).Y(pointIndex));
}
```

## Creating a Marker

To create a marker for your label, complete the following:

### To write code in Visual Basic

#### Visual Basic

```
c1Chart1.ChartArea.AxisX.AnnoMethod = C1.Win.C1Chart.AnnotationMethodEnum.Mixed
Dim vl As ValueLabel = c1Chart1.ChartArea.AxisX.ValueLabels.AddNewLabel()
vl.Appearance = ValueLabelAppearanceEnum.TriangleMarker
vl.MarkerSize = 50
vl.Color = Color.Transparent
vl.NumericValue = 2
```

### To write code in C#

#### C#

```
C1Chart1.ChartArea.AxisX.AnnoMethod = C1.Win.C1Chart.AnnotationMethodEnum.Mixed

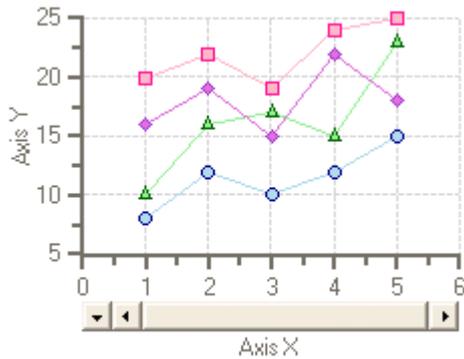
Dim markerX As C1.Win.C1Chart.ValueLabel =
C1Chart1.ChartArea.AxisX.ValueLabels.AddNewLabel()
markerX.NumericValue = 3
markerX.Moveable = True
markerX.MarkerSize = 15
markerX.GridLine = True
markerX.Color = Color.Blue
markerX.Appearance = C1.Win.C1Chart.ValueLabelAppearanceEnum.ArrowMarker
```

## Add Scrollbar to the X-Axis and Y-Axis

The following steps show how to add Scrollbars to the X and Y axis of the Chart

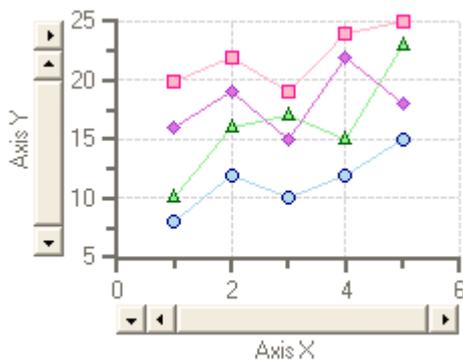
### Using the Properties Window:

1. In the C1Chart Properties window, expand **ChartArea** and expand the **AxisX** node.
2. Expand the **ScrollBar** node and set its **Visible** property to True.
3. Set the **AxisX.Min** property to 0 and **AxisX.Max** property to 6 so the scrollbar appears within the minimum and maximum values.
4. The default **AxisX** scrollbar is aligned to the center of the Axis X and appears within the AxisX minimum and maximum values.



5. In the C1Chart Properties window, expand the **AxisY** node.
6. Expand the ScrollBar node and set its **Visible** property to True.
7. Set the **AxisY.Min** property to 5 and **AxisY.Max** property to 25 so the scrollbar appears within the minimum and maximum values.

The default **AxisY** scrollbar is aligned to the center of the AxisY and appears within the AxisY minimum and maximum values.



### Using Code:

Add the following code to add scrollbars to the X-axis and Y-axis of your chart:

#### To write code in Visual Basic

##### Visual Basic

```
' Setup the AxisX scroll bar
'Note, that the cd variable represents the ChartData

c1Chart1.ChartArea.AxisX.ScrollBar.Min = cd.MinY
c1Chart1.ChartArea.AxisX.ScrollBar.Max = cd.MaxY
c1Chart1.ChartArea.AxisX.ScrollBar.Appearance = ScrollBarAppearanceEnum.Normal
c1Chart1.ChartArea.AxisX.ScrollBar.Visible = True
c1Chart1.ChartArea.AxisX.ScrollBar.Alignment = StringAlignment.Center

' Setup the AxisY scroll bar

c1Chart1.ChartArea.AxisY.ScrollBar.Min = cd.MinY
c1Chart1.ChartArea.AxisY.ScrollBar.Max = cd.MaxY
c1Chart1.ChartArea.AxisY.ScrollBar.Appearance = ScrollBarAppearanceEnum.Normal
c1Chart1.ChartArea.AxisY.ScrollBar.Visible = True
```

```
c1Chart1.ChartArea.AxisY.ScrollBar.Alignment = StringAlignment.Center
```

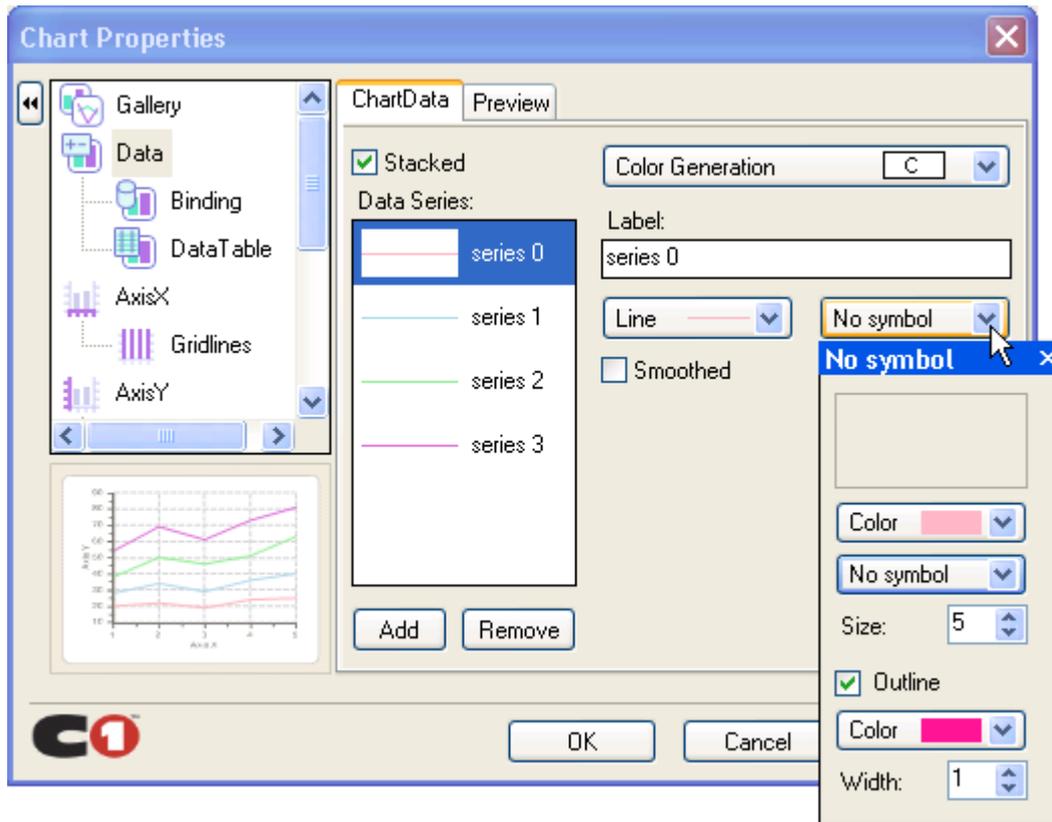
### To write code in C#

```
C#  
  
// Setup the AxisY scroll bar  
//Note, that the cd variable represents the ChartData  
  
c1Chart1.ChartArea.AxisX.ScrollBar.Min = cd.MinY;  
c1Chart1.ChartArea.AxisX.ScrollBar.Max = cd.MaxY;  
c1Chart1.ChartArea.AxisX.ScrollBar.Appearance = ScrollBarAppearanceEnum.Normal;  
c1Chart1.ChartArea.AxisX.ScrollBar.Visible = true;  
c1Chart1.ChartArea.AxisX.ScrollBar.Alignment = StringAlignment.Center;  
  
// Setup the AxisY scroll bar  
  
c1Chart1.ChartArea.AxisY.ScrollBar.Min = cd.MinY;  
c1Chart1.ChartArea.AxisY.ScrollBar.Max = cd.MaxY;  
c1Chart1.ChartArea.AxisY.ScrollBar.Appearance = ScrollBarAppearanceEnum.Normal;  
c1Chart1.ChartArea.AxisY.ScrollBar.Visible = true;  
c1Chart1.ChartArea.AxisY.ScrollBar.Alignment = StringAlignment.Center;
```

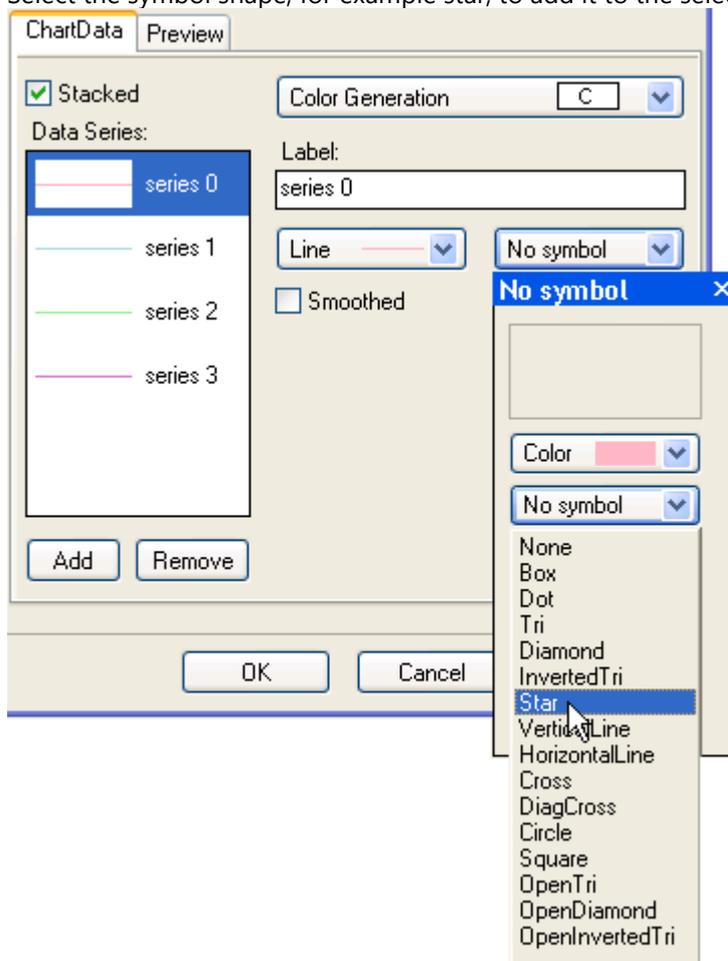
## Add Symbols to Data Series

To add a symbol to a data series using the Chart Properties designer, complete the following steps:

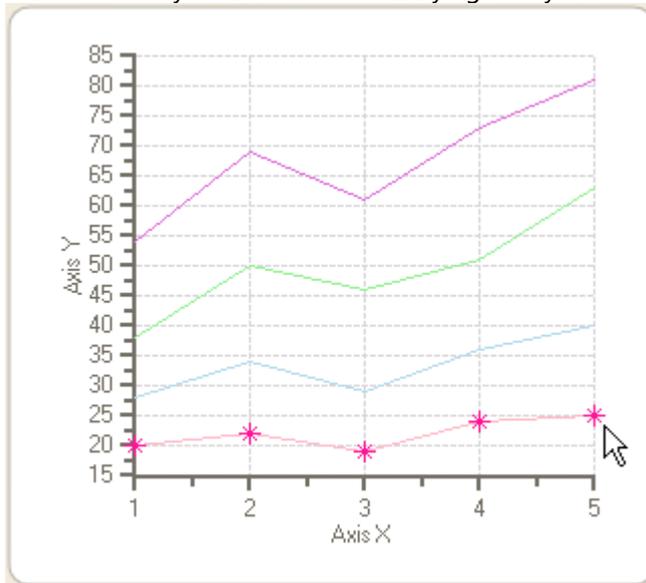
1. Right-click on the **C1Chart** control and select **Chart Properties** from C1Chart's context menu.
2. Select the **Data** element in the treeview pane of the **Chart Properties** designer.
3. Select a data series from the **Data Series** listbox.
4. Click on the **Symbol** dropdown arrow to open up the list of available symbols.



5. Select the symbol shape, for example star, to add it to the selected data series.



6. Click on the **Color** dropdown arrow and select Deep Pink.
7. Deselect the **Outline** checkbox to remove the Deep Pink outline from the star.
8. Increase the size of the star from 5 to 10.
9. Click **OK** once you are finished modifying the symbol's appearance.



## Add ToolTips to Chart Elements

The following topics show how to add tooltips to C1Chart's Header, Footer, and DataSeries objects.

## Add ToolTips to Chart's Points in the Data Series

To add ToolTips to the Chart's DataSeries, complete the following steps:

1. Add C1Chart to the Form.
2. Add the following directive to declare the **C1.Win.C1Chart** namespace:

### To write code in Visual Basic

```
Visual Basic
Imports C1.Win.C1Chart
```

### To write code in C#

```
C#
using C1.Win.C1Chart;
```

3. Add the following code in the **Form\_load** procedure to declare the dataseries and add the tooltips to the dataseries:

### To write code in Visual Basic

```
Visual Basic
C1Chart1.ToolTip.Enabled = True
Dim sc As ChartDataSeriesCollection =
```

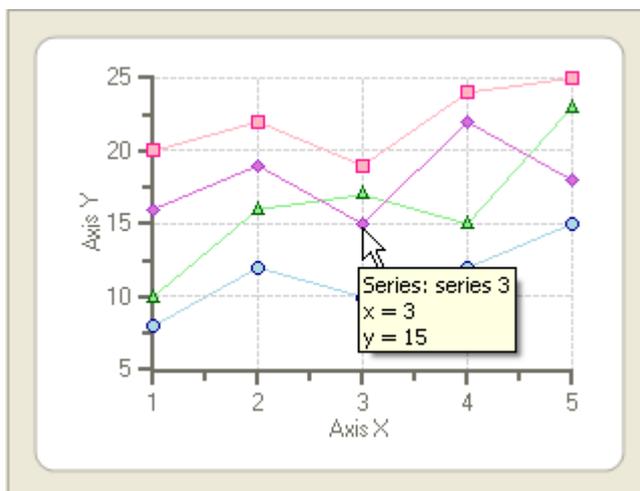
```
C1Chart1.ChartGroups(0).ChartData.SeriesList
For Each ds As ChartDataSeries In sc
    ds.ToolTipText = "Series: {#TEXT}" + ControlChars.Cr + ControlChars.Lf + "x
= {#XVAL}" + ControlChars.Cr + ControlChars.Lf + "y = {#YVAL}"
Next ds
```

## To write code in C#

```
C#
ChartDataSeriesCollection sc =
c1Chart1.ChartGroups[0].ChartData.SeriesList;
foreach (ChartDataSeries ds in sc)
    ds.ToolTipText = "Series: {#TEXT}" + '\r' + '\n' + "x = {#XVAL}" + '\r' +
'\n' + "y = {#YVAL}";
// Enable tooltip
c1Chart1.ToolTip.Enabled = true;
```

## This topic illustrates the following:

When you hover over the points in the data series on the chart area at run time, the tooltips appear for each point, like the following:



## Add ToolTips to Chart's Header and Footer

To add tooltips to the chart's header and footer, complete the following steps:

1. Add **C1Chart** to the Form.
2. Add the following directive to declare the **C1.Win.C1Chart** namespace:

### To write code in Visual Basic

```
Visual Basic
Imports C1.Win.C1Chart
```

### To write code in C#

C#

```
using Cl.Win.ClChart;
```

3. Add the following code in the Form\_load procedure to add the tooltips to the header and footer:

#### To write code in Visual Basic

Visual Basic

```
'Enable tooltip
c1Chart1.ToolTip.Enabled = True
c1Chart1.Header.ToolTipText = "This is header tooltip." + ControlChars.Cr +
ControlChars.Lf + "Second line."
c1Chart1.Footer.ToolTipText = "This is footer tooltip." + ControlChars.Cr +
ControlChars.Lf + "Second line."
```

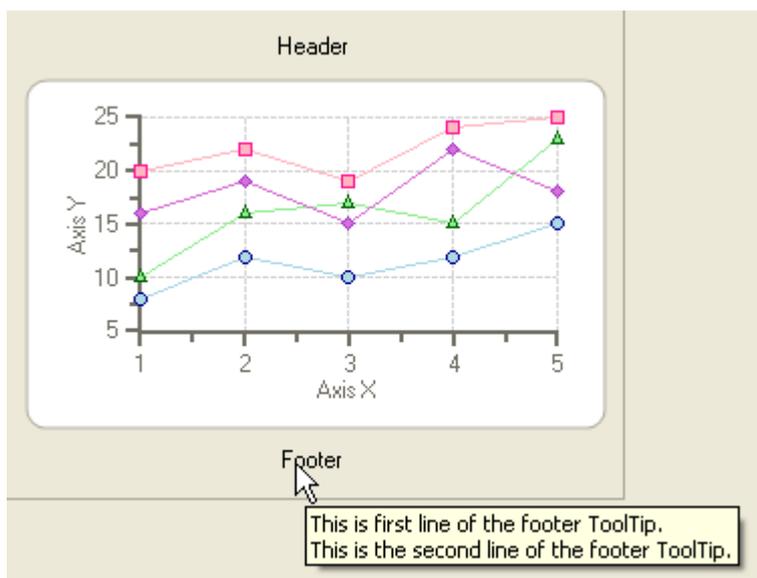
#### To write code in C#

C#

```
//Enable tooltip
c1Chart1.ToolTip.Enabled = true;
c1Chart1.Header.ToolTipText = "This is header tooltip.";
c1Chart1.Footer.ToolTipText = "This is first line of the footer ToolTip.\nThis
is the second line of the footer ToolTip.";
```

## This topic illustrates the following:

The ToolTips appear when you hover over Chart's Header or Footer at run-time, like the following:



## Add ToolTips to Axes

To add tooltips to the chart's axes, complete the following steps:

1. Add **C1Chart** to the Form.
2. Add the following directives to declare the C1.Win.Chart namespaces:

### To write code in Visual Basic

```
Visual Basic
Imports C1.Win.C1Chart
```

### To write code in C#

```
C#
using C1.Win.C1Chart;
```

3. Add the following code in the Form\_load procedure to add the chart's axes

### To write code in Visual Basic

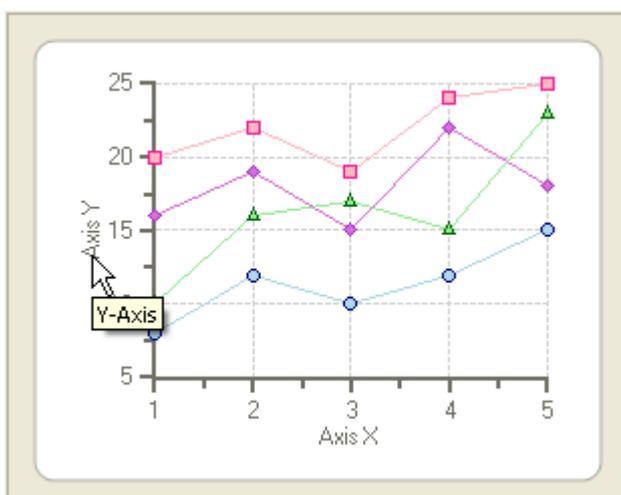
```
Visual Basic
'Enable tooltip
C1Chart1.ToolTip.Enabled = True
C1Chart1.ChartArea.AxisX.TooltipText = "X-Axis"
C1Chart1.ChartArea.AxisY.TooltipText = "Y-Axis"
```

### To write code in C#

```
C#
//Enable tooltip
c1Chart1.ToolTip.Enabled = true;
c1Chart1.ChartArea.AxisX.TooltipText = "X-Axis";
c1Chart1.ChartArea.AxisY.TooltipText = "Y-Axis";
```

## This topic illustrates the following:

When you hover over the x-axis and y-axis on the chart area at run time, the tooltips appear for the x-axis and y-axis, like the following:



## Adding Visual Effects to Chart Elements

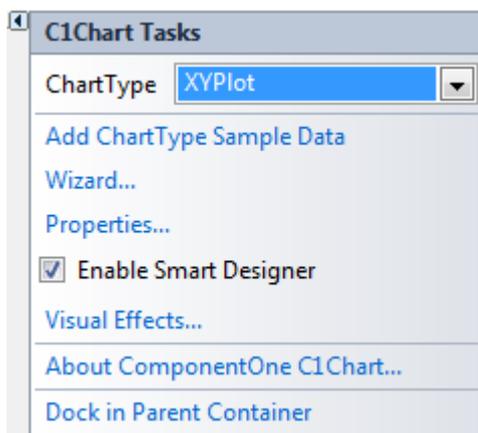
This section provides tasks for using the **Visual Effects** designer to visually enhance the Chart Header, Footer, and Data Series elements.

## Access the Visual Effects Designer

To access the **Visual Effects** designer, use the **C1Chart Tasks** menu (smart tag), the C1Chart context menu, or the Property grid editor.

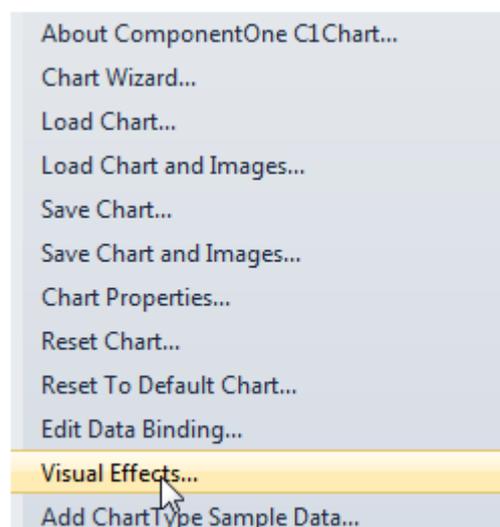
### C1Chart Tasks menu

Click the smart tag (📌) in the upper-right corner of **C1Chart** to open the **C1Chart Tasks** menu, and select **Visual Effects**.



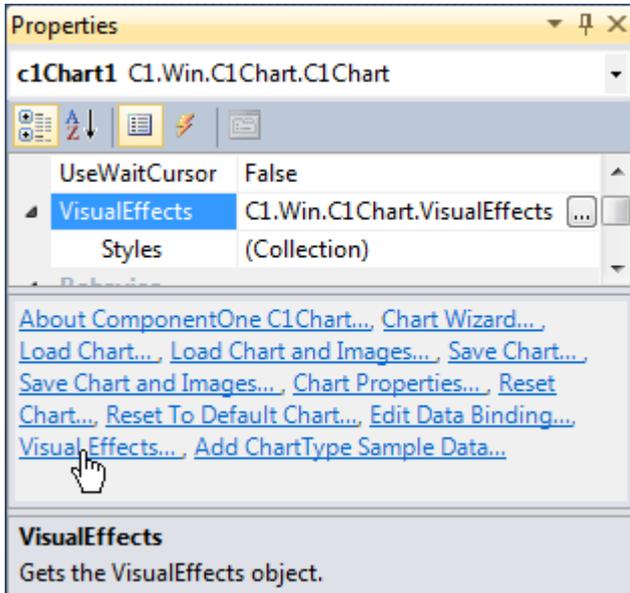
### Context Menu

Right-click on the **Chart2D** control and select **Visual Effects** from the context menu.



## Properties Window

Right-click on the **Chart2D** control and select **Properties**. Click on the **ellipsis** button next to **VisualEffects** in the Properties window or click on **Visual Effects** in the **Action** list area at the bottom of Visual Studio .NETs Properties window. The following image displays the **VisualEffects** items in C1Chart's Properties window.



## Customize Header and Footer

This section shows how to add light patterns, shapes, shadows, and preset styles to the Chart Header and Footer. In addition to light effects tasks this section shows how to enhance existing colors for the Chart Header by changing its Hue or increasing/decreasing brightness and saturation.

### Add a Light Pattern to the Chart Header and Footer

You can show repetitive light patterns in a chart element by setting the Scale property to a value less than 1. As you decrease the Scale, the light pattern repeats more. The values of the Scale property range from 0 to 1.

To add a light pattern to the Chart Header and Chart Footer elements, complete the following steps:

1. Add a Chart Header and Chart Footer.

For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).

2. Open the **Visual Effects** designer.

For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).

3. In the **Available Elements** list box, click on the boxes next to **Header** and **Footer**.
4. Click on the **Parameters** tab. Set the light **Shape** property to **Rectangle**.
5. Select the **Scale** property and move the **Scale** slider the left to 0.20 or enter .20 in the text box.

 **Note:** The **Scale** property is only applicable for the Rectangle light gradient.

The rectangular pattern is repeated for the Header element.



The rectangular pattern is repeated for the Footer element.



To add a light pattern to the Chart Header's and Footer's text element, complete the following steps:

1. Open the **Visual Effects** designer.

For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).

2. In the **Available Elements** list box, unselect the **Header** and **Text** check boxes and click on the boxes next to **Header.Text** and **Footer.Text**.
3. Click on the **Parameters** tab. Set the light **Shape** property to **Rectangle**.
4. Select the **Scale** property and move the Scale slider to **0.20** or enter **0.20** in the text box.

 **Note:** The **Scale** property is only applicable for the Rectangle light gradient.

The rectangular pattern is repeated for the Header's text element.



The rectangular pattern is repeated for the Footer's text element.



## Add a Light Shape to the Chart Header and Footer

To add a light shape to the Chart Header and Footer elements, complete the following steps:

1. Add a Chart Header and Chart Footer.

For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).

2. Open the **Visual Effects** designer.

For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).

3. In the **Available Elements** list box, click on the box next to **Header**, and then click on the box next to the **Footer**.
4. Click on the **Parameters** tab. Set the light **Shape** property to Ellipse. The Preview pane displays the ellipse shape.

 **Note:** Two additional properties (**Shift** and **Size**) appear for the Ellipse shape

5. To decrease the size of the ellipse shape, select the **Size** property and move the **Size** slider to left until 0.3.
6. To move the location of the ellipse shape from the center to the left corner, select the **Shift** property and slide the **Shift** slider to the left. As you move the slider to the left the value decreases and the ellipse light shape shifts.

The light ellipse shape appears at the corner of the Header element.



The light ellipse shape appears at the corner of the Footer element.



To add a light shape to the Chart Header's and Footer's text element, complete the following steps:

1. Add a Chart Header and Chart Footer.

For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer, see [Add a Chart Footer](#).

2. Open the **Visual Effects** designer.

For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).

3. In the **Available Elements** list box, unselect the **Header** and **Text** check boxes and click on the boxes next to **Header.Text** and **Footer.Text**.
4. Click on the **Parameters** tab. Set the light **Shape** property to **Ellipse**.

The ellipse shape appears on the Header's text.



The ellipse shape appears on the Footer's text.



## Add a Preset Style to the Chart Header and Footer

To add a preset style to the Chart Header and Footer elements, complete the following steps:

1. Add a Chart Header and Chart Footer.

For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).

2. Open the **Visual Effects** designer.

For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).

3. In the **Available Elements** list box, click on the box next to **Header.Text**, and then click on the box next to the **Footer.Text**.
4. Click on the **Presets** tab, and then select the third style in the first row. The Preview pane displays the selected preset style.

The selected preset style appears on the Header.



The selected preset style appears on the Header.



To add a preset style to the Chart Header's and Footer's text element, complete the following steps:

1. Add a Chart Header and Chart Footer.

For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).

2. Open the **Visual Effects** designer.

For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).

3. In the **Available Elements** list box, unselect the **Header** and **Text** check boxes and click on the boxes next to **Header.Text** and **Footer.Text**.
4. Click on the **Presets** tab, and then select the third style in the first row. The Preview pane displays the selected preset style.

The selected preset style appears on the Header's text.



The selected preset style appears on the Footer's text.



## Add a Shadow to the Chart Header and Footer

To add a shadow to the Chart Header and Footer elements, complete the following steps:

1. Add a Chart Header and Chart Footer.

For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).

2. Open the **Visual Effects** designer.

For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).

3. In the **Available Elements** list box, click on the box next to **Header**, and then click on the box next to the **Footer**.
4. Click on the **Parameters** tab, and then select the **Offset** property located in the Shadow group.
5. Slide the **Offset** slider to the right to 2.5 or enter the value 2.5 in the Offset's text box.

The Preview pane displays the shadow on the square.

6. To make the shadow appear less transparent, select the Transparency and slide the slider to 160.

The Preview pane displays the darker shadow.

A shadow appears on the Header.



A shadow appears on the Footer.



To add a shadow to the Chart Header's text element, complete the following steps:

1. Add a Chart Header and Chart Footer.

For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).

2. Open the **Visual Effects** designer.

For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).

3. In the **Available Elements** list box, unselect the **Header** and **Text** check boxes and click on the boxes next to **Header.Text** and **Footer.Text**.
4. Click on the **Parameters** tab, and then select the **Offset** property located in the Shadow group.
5. Slide the Offset slider to the right to 2.5 or enter the value 2.5 in the Offset's text box.

The Preview pane displays the shadow on the square.

6. To make the shadow appear less transparent, select the **Transparency** property and slide the slider to 160.

The Preview pane displays the darker shadow.

A shadow appears on the Header's text.



A shadow appears on the Footer's text.



## Adjust the Focus of the Light to the Chart Header and Footer

To adjust the focus of the light to the Chart Header and Footer elements, complete the following steps:

1. Add a Chart Header and Chart Footer. For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).
2. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
3. In the **Available Elements** list box, click on the box next to **Header.Text**, and then click on the box next to the **Footer.Text**.
4. Click on the **Parameters** tab, and then select the **Gradient** property located in the Light group to Triangle. The **Focus** property is added to the Light group properties and the default value is 0.1.
5. Select the **Focus** property and slide the slider to .55 so the focus for the light appears on both sides.

The Preview pane displays the position of the light focus on the square.

The triangular light is focused at both corners of the Header element.



The triangular light is focused at both corners of the Footer element.



To adjust the focus of the light to the Chart Header's text element and Chart Footer's text element, complete the following steps:

1. Add a Chart Header and Chart Footer. For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).
2. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
3. In the **Available Elements** list box, unselect the **Header** and **Text** check boxes and click on the boxes next to **Header.Text** and **Footer.Text**.
4. Click on the **Parameters** tab, and then select the Gradient property located in the Light group to Triangle. The **Focus** property is added to the Light group properties and the default value is 0.1.
5. Select the **Focus** property and slide the slider to 1.0 so the focus for the light appears on the opposite side.

The Preview pane displays the position of the light focus on the square.

The triangular light is focused at both ends of the Header's Text element.



The triangular light is focused at both ends of the Footer's Text element.



## Use the Color Sliders to Enhance an Existing Color for the Chart Header and Footer

To enhance the **Teal** back color for the Chart Header and Footer elements, complete the following steps:

1. Add a Chart Header and Chart Footer. For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).
2. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
3. In the **Available Elements** list box, click on the box next to **Header**, and then click on the box next to the **Footer**. The teal color is displayed slightly different in the Preview Pane because the default color scheme settings applied to the Header's existing color.
4. Click on the **Colors** tab, and then slide the HueShift slider to the value 14 so it has a bluer tone.
5. Slide the Saturation slider to the right and stop at 100. The Saturation is increased to 100% and the blue tone appears more pure and vivid.
6. Slide the Brightness slider to the right and stop at -22. This slightly increases the brightness of the color tone.

The new color appears on the Header element.

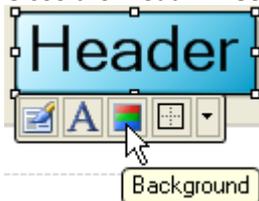


The new color appears on the Footer element.



As an option, to customize the color more, complete the following steps:

1. Close the **Visual Effects** designer and select the Chart Header on the form. The Header toolbar appears.



2. Click on the **Background** button and select the **Web** tab from the Color drop-down box.
3. Select **Blue** to add some Blue color to the existing color. Repeat steps 1-3 for the Footer too.

The new color for the Chart Header appears like the following.



The new color for the Chart Footer appears like the following.

## Footer

To create a custom color for the Chart Header's and Footer's text element, complete the following steps:

1. Add a Chart Header and Chart Footer, set their **ForeColor** to **Navy** and their **BackColor** to **DeepSkyBlue** make their text Bold. For details on how to add a Chart Header see, [Add a Chart Header](#). For details on how to add a Chart Footer see, [Add a Chart Footer](#).
2. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
3. In the **Available Elements** list box, unselect the **Header** and **Text** check boxes and click on the boxes next to **Header.Text** and **Footer.Text**.
4. Click on the **Colors** tab, then slide the Saturation slider to the right and stop at 100. The Saturation is increased to 100%.
5. Slide the Brightness slider to the right and stop at 24. This increases the brightness and makes the tone of color appear lighter.
6. Slide the HueShift slider all the way to the left so its value is zero.

The new enhanced color appears on the Header's text.

## Header

The new enhanced color appears on the Footer's text.

## Footer

If you would like to change the color so it appears pink, simply slide the HueShift slider to the right and stop at 66. The Header's text and Footer's appears like the following.

## Header

## Footer

## Customize Data Series

This section shows how to add light patterns, shapes, shadows, and preset styles to the Chart Data Series. In addition to light effects tasks this section shows how to create a custom color for the Chart Data Series using hue, brightness, and saturation. The examples in this section use the chart type, Pie, and the default colors for the data series.

## Add a Light Pattern to the Chart Data Series

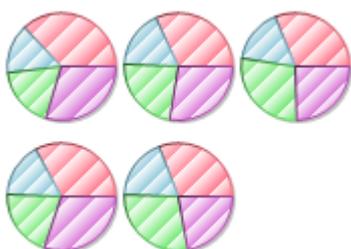
You can show repetitive light patterns in a chart element by setting the **Scale** property to a value less than 1. As you decrease the Scale, the light pattern repeats more. The values of the **Scale** property range from 0 to 1.

To create a light pattern for the Chart Data Series, complete the following steps:

1. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
2. In the **Available Elements** list box, click on the box next to **Default**. The default visual effects settings for the data series appear on the **Chart2D** control.
3. Click on the **Parameters** tab. Set the light **Shape** property to Rectangle.
4. Select the **Scale** property and move the Scale slider to 0.20 or enter 0.20 in the text box.

 **Note:** The **Scale** property is only applicable for the Rectangle light gradient.

The rectangular pattern is repeated for the data series in the Pie chart.



## Add a Light Shape to the Chart Data Series

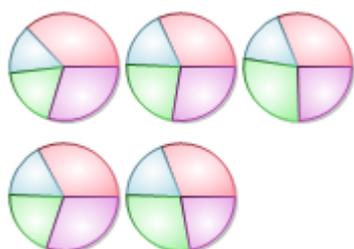
To create a light shape for the Chart Data Series, complete the following steps:

1. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
2. In the **Available Elements** list box, click on the box next to **Default**. The default visual effects settings for the data series appear on the **Chart2D** control.
3. Click on the **Parameters** tab. Set the light **Shape** property to Ellipse. The Preview pane displays the ellipse shape.

 **Note:** Two additional properties (**Shift** and **Size**) appear for the Ellipse shape.

4. To increase the size of the ellipse shape, select the **Size** property and move the **Size** slider to the right until 1.
5. To increase the intensity of the light, select the **Intensity** property and slide the Intensity slider to the right until you reach the end at 1.0. The intensity of the light increases.

The light ellipse shape appears at the center of the data series.



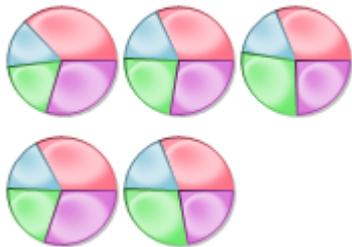
## Add a Preset Style to the Chart Data Series

To add a preset style to the Chart Data Series, complete the following steps:

1. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
2. In the **Available Elements** list box, click on the box next to **Default**. The default visual effects settings for the data series appear on the Chart2D control.
3. Click on the **Presets** tab, and then select the third style in the first row.

The Preview pane displays the selected preset style.

The selected preset style appears on the data series.



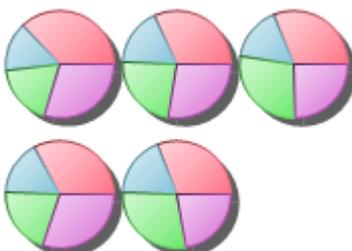
## Add a Shadow to the Chart Data Series

To create a shadow for the Chart Data Series, complete the following steps:

1. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
2. In the **Available Elements** list box, click on the box next to **Default**. The default visual effects settings for the data series appear on the Chart2D control.
3. Click on the **Parameters** tab, and then select the **Offset** property located in the Shadow group.
4. Slide the **Offset** slider to the right to 3 or enter the value 3 in the Offset's text box. The Preview pane displays the shadow on the square.
5. To make the shadow appear less transparent, select the Transparency and slide the slider to 160.

The Preview pane displays the darker shadow.

A shadow appears on the data series.



## Adjust the Focus of the Light for the Chart Data Series

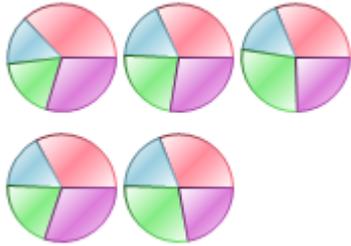
To adjust the focus of the light for the Chart Data Series, complete the following steps:

1. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
2. In the **Available Elements** list box, click on the box next to **Default**. The default visual effects settings for the data series appear on the Chart2D control.

3. Click on the **Parameters** tab, and then select the **Gradient** property located in the Light group to Triangle. The **Focus** property is added to the Light group properties.
4. Select the **Focus** property and slide the slider to .5 so the focus for the light appears in the center of the data series
5. Select the Intensity property and enter 1.0, to increase the intensity of the light.

The Preview pane displays the position of the light focus on the square.

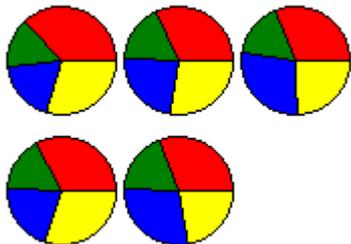
The triangular light is focused at the center of the data series elements.



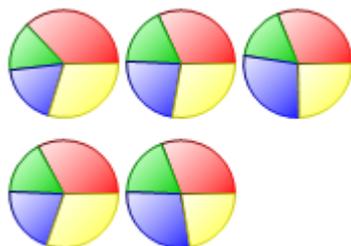
## Use the Color Sliders to Enhance an Existing Color for the Chart Data Series

To create a custom color for the Chart Data Series, complete the following steps:

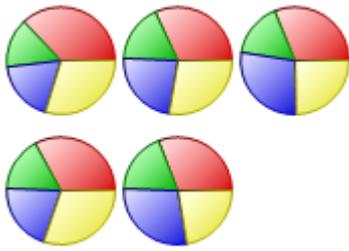
1. Open the **Chart Properties** designer. For details on how to access the **Chart Properties** designer see [Working with the Chart Properties Designer](#).
2. Set series 0 to red, series 1 to green, series 2 to blue, and series 3 to yellow. For details on how to apply color to the data series using the **Chart Properties** designer see [Modify the Appearance of the Data Series](#). The colors for each series appear like the following.



3. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
4. In the **Available Elements** list box, click on the box next to **Default**. The default visual effects settings for the data series appear on the Chart2D control.

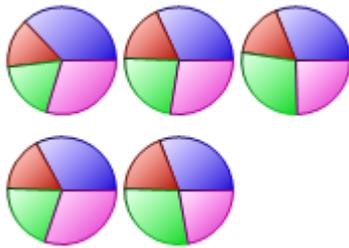


5. Click on the **Colors** tab, and then slide the Brightness slider to the value -55. The color appears less bright.
6. Slide the Saturation slider to the right and stop at 100. The Saturation is increased to 100% and the color tone appears more pure and vivid. The original colors for the data series are enhanced.



As an option, to change the colors for the data series, complete the following

- In the Colors tab, slide the HueShift slider to 247. Notice as you move the slider the colors change.

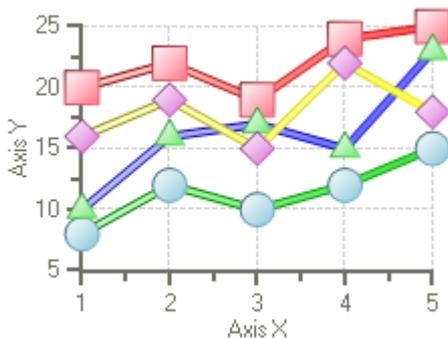


## Increase the Size of the Symbols in the Data Series

Adjust the **ScaleEffect** property to increase the size of the symbols for an XY plot.

To increase the size of the symbols in the data series symbols for an XY plot, complete the following steps:

1. Open the **Visual Effects** designer. For details on how to access the **Visual Effects** designer see [Access the Visual Effects Designer](#).
2. In the **Available Elements** list box, click on the box next to **Default**. The default visual effects settings for the data series appear on the Chart2D control.
3. Select the **Parameters** tab, then select the **ScaleEffect** property and slide the slider to 0.6 or enter 0.6. The **ScaleEffect** property increases the size of the symbols.



## Creating and Formatting Chart Elements Using the Properties Window

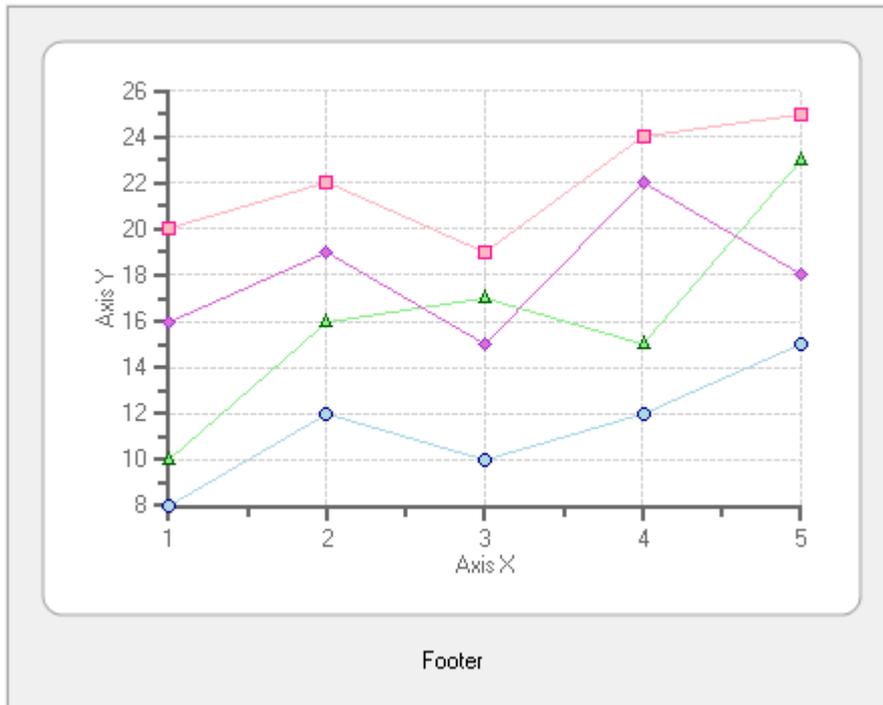
This section provides tasks for using the Properties Window to create and format chart elements.

## Add a Chart Footer using the Properties Window

To add a chart Footer using the C1Chart Properties window, complete the following steps:

1. Right-click on the **C1Chart** control and select **Properties** from the context menu. The Properties window for the **C1Chart** control will appear in the right pane.
2. Expand the **Footer** node and enter text for example, **Footer**, next to the **Text** property.
3. Set the **Visible** property to **True**.

The **Footer** element appears below the chart area. This is the default position for the chart **Footer** element.

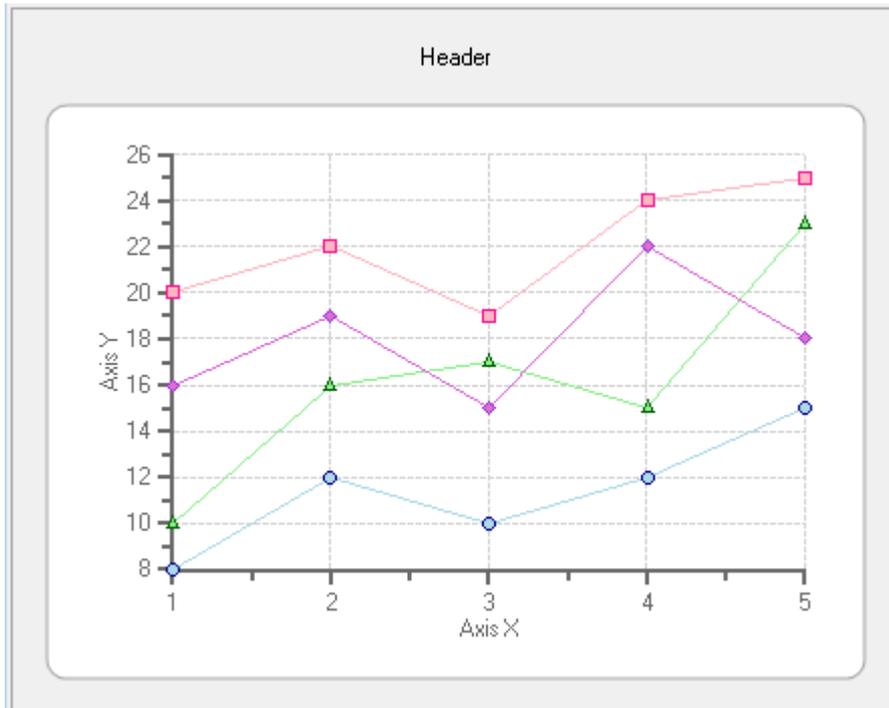


## Add a Chart Header using the Properties Window

To add a chart Header using the **C1Chart** Properties window, complete the following steps:

1. Right-click on the **C1Chart** control and select **Properties** from the context menu. The Properties window for the **C1Chart** control will appear in the right pane.
2. Expand the **Header** node and enter text for example, **Header**, next to the **Text** property.
3. Set the **Visible** property to **True**.

The **Header** element appears above the chart area. This is the default position for the chart **Header** element.

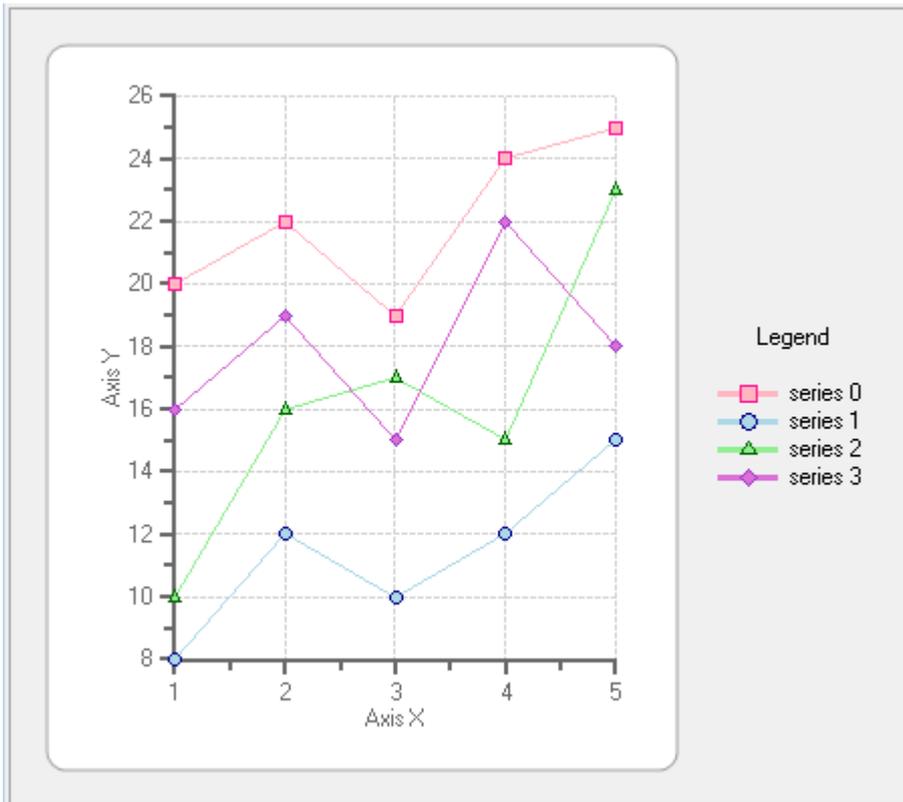


## Add a Chart Legend using the Properties Window

To add a chart Legend using the **C1Chart** Properties window, complete the following steps:

1. Right-click on the **C1Chart** control and select **Properties** from the context menu. The Properties window for the **C1Chart** control will appear in the right pane.
2. Expand the **Legend** node and enter text for example, **Legend**, next to the **Text** property.
3. Set the **Visible** property to **True**.

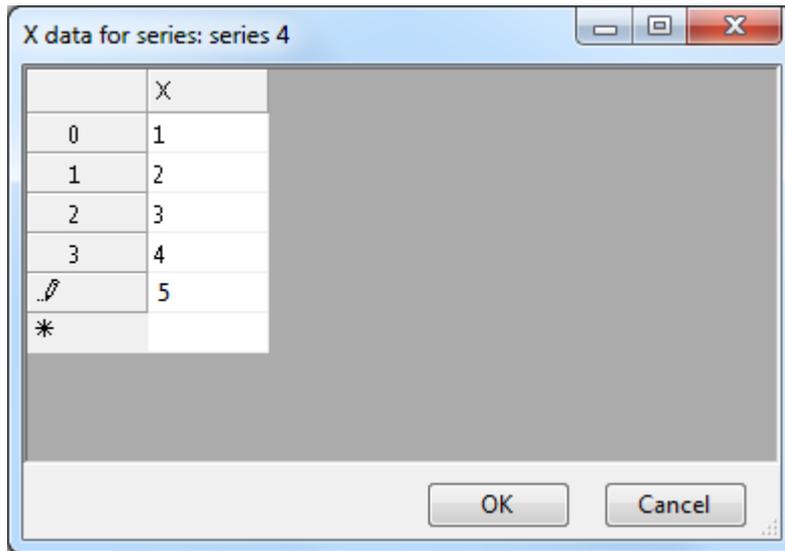
The **Legend** element appears to the right of the chart area or to the east of the chart area. This is the default position for the chart **Legend** element.



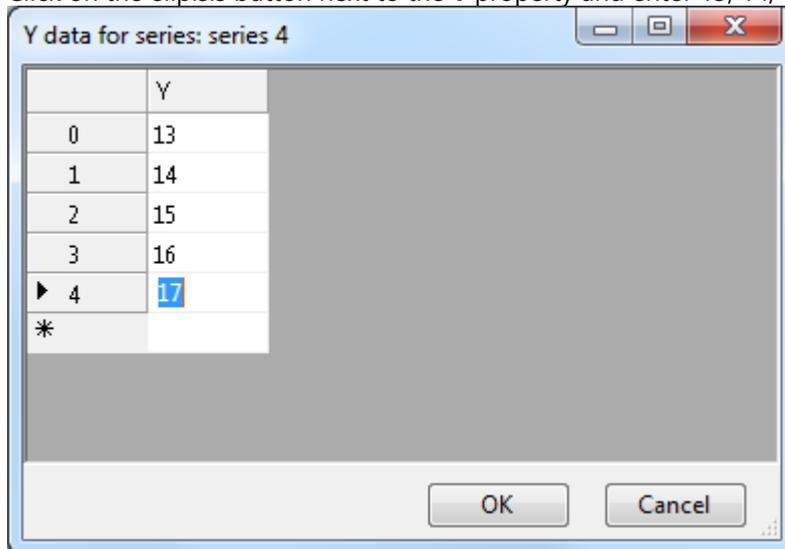
## Add Data Series and Data to the Chart using the Properties Window

To add Data Series using the **C1Chart** Properties window, complete the following steps:

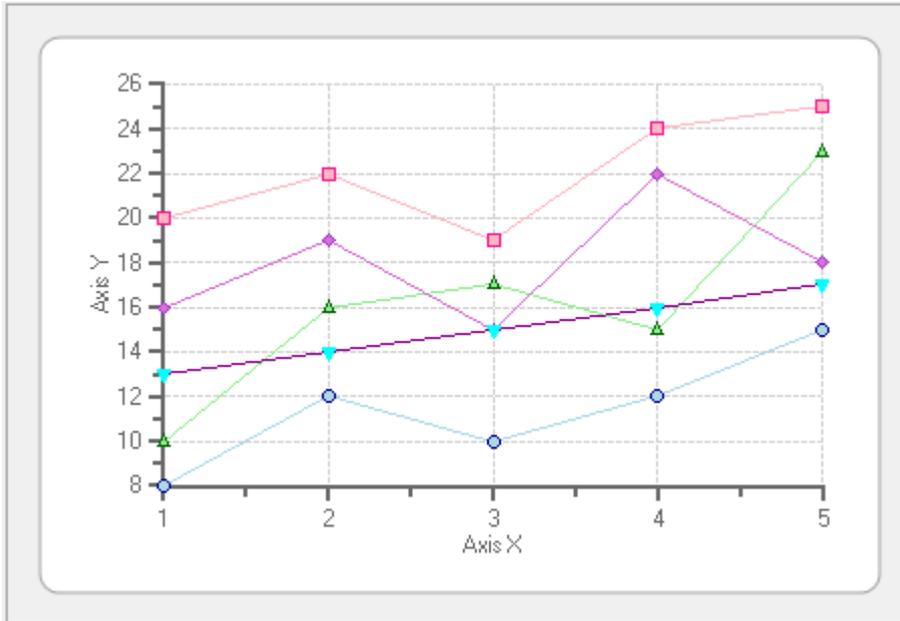
1. Right-click on the **C1Chart** control and select **Properties** from the context menu. The Properties window for the **C1Chart** control will appear in the right pane.
2. Under the **Misc** group in the **C1Chart Properties** window expand the **ChartGroups** node.
3. Expand **Group0** -> **ChartData** and click on the ellipsis button next to the **SeriesList** property.
4. Click the **Add** button once in the **ChartDataSeries Collection Editor** to add a new series to the **C1Chart** control.
5. Expand the **X** node and enter **5** next to the **Length** property.
6. Set the **DataType** property to **System.Single**.
7. Click on the ellipsis button next to the **X** property and enter 1, 2, 3, 4, and 5 for the X values.



8. Expand the **Y** node and enter **5** next to the **Length** property.
9. Set the **DataType** property to **System.Single**.
10. Click on the ellipsis button next to the **Y** property and enter 13, 14, 15, 16, and 17 for the **Y** values and click **OK**.



The new C1Chart series appears on the C1Chart control:

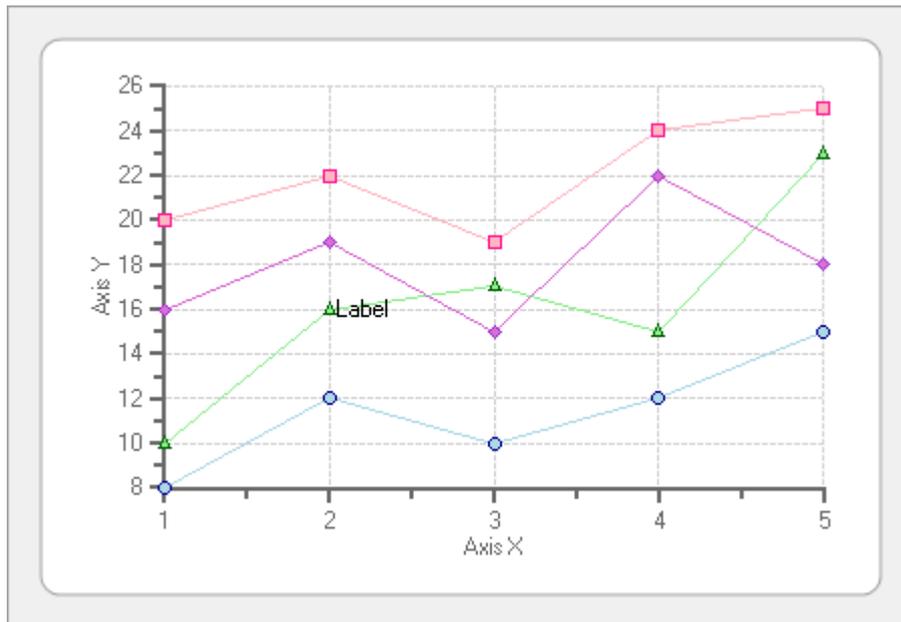


## Add Labels to the Chart using the Properties Window

To add labels to the C1Chart control using the **C1Chart** Properties window, complete the following steps:

1. Right-click on the **C1Chart** control and select **Properties** from the context menu. The Properties window for the **C1Chart** control will appear in the right pane.
2. Expand the **ChartLabels** and click on the ellipsis button next to **LabelsCollection**.
3. In the **Label Collection Editor**, click on the **Add** button to add a new label to the **C1Chart** control.
4. Select **DataIndex** from the **AttachMethod** property dropdown listbox. The **Attached By Data Index** command item gets the **DataIndex** member of the **AttachMethodEnum** and sets the value of the **Label.AttachMethod** property in the Label. It attaches the label to a specific data point on the plot area of the chart.
5. Expand the **AttachMethodData** node. Notice that the **Point Index** and **Series Index** properties each have a value associated with it. This is because the **DataIndex** attachment method attaches the labels by the data point. The **GroupIndex** value of **0** represents the **ChartGroup0**, the **PointIndex** value of **1** represents the second data point on the series, and the **SeriesIndex** value of **2** represents the **third** series on the chart.
6. Set the **Text** property to **Label**.
7. Click **OK** to save and close the **Label Collection Editor**.

The label appears on the second data point of the the third data series.



## Add Rotated Labels to the Chart using the Properties Window

To add labels to the C1Chart control using the **C1Chart** Properties window, complete the following steps:

1. Right-click on the **C1Chart** control and select **Properties** from the context menu. The Properties window for the **C1Chart** control will appear in the right pane.
2. Expand the **ChartLabels** and click on the ellipsis button next to **LabelsCollection**.
3. In the **Label Collection Editor**, click on the **Add** button to add a new label to the **C1Chart** control.
4. Select **DataIndex** from the **AttachMethod** property dropdown listbox. The **Attached By Data Index** command item gets the **DataIndex** member of the **AttachMethodEnum** and sets the value of the **Label.AttachMethod** property in the Label. It attaches the label to a specific data point on the plot area of the chart.
5. Expand the **AttachMethodData** node. Notice that the **Point Index** and **Series Index** properties each have a value associated with it. This is because the **DataIndex** attachment method attaches the labels by the data point. The **GroupIndex** value of **0** represents the **ChartGroup0**, the **PointIndex** value of **1** represents the second data point on the series, and the **SeriesIndex** value of **2** represents the **third** series on the chart.
6. Set the **Text** property to **RotatingLabel**.
7. Set the **Connected** property to **True**.
8. Set the **Offset** property to **10**.
9. Set the **RotationOverride** property to **60**.

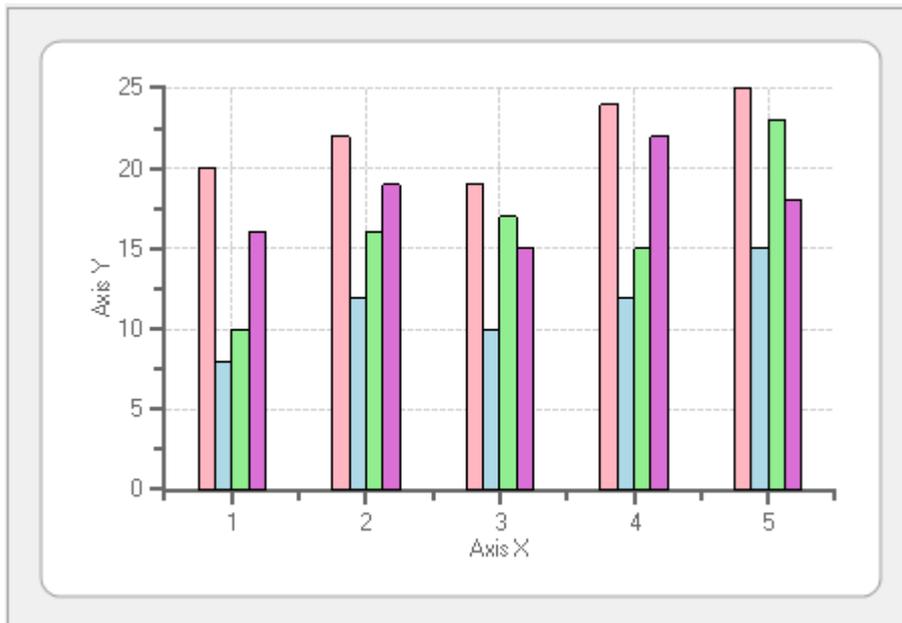
The label will rotate clockwise on a 60 degree angle once you create code to handle the animation.

## Choose a Chart Type using the Properties Window

To select a chart type using the **C1Chart** Properties window, complete the following steps:

1. Right-click on the **C1Chart** control and select **Properties** from the context menu. The Properties window for the **C1Chart** control will appear in the right pane.
2. Expand the **ChartGroups** node and then expand the **ChartGroup** you wish to change the chart type, **Group0** or **Group1**.
3. Click the **Chart type** drop-down arrow and choose the **Bar** chart type from the list.

The **Bar** chart type appears on the **C1Chart** control.

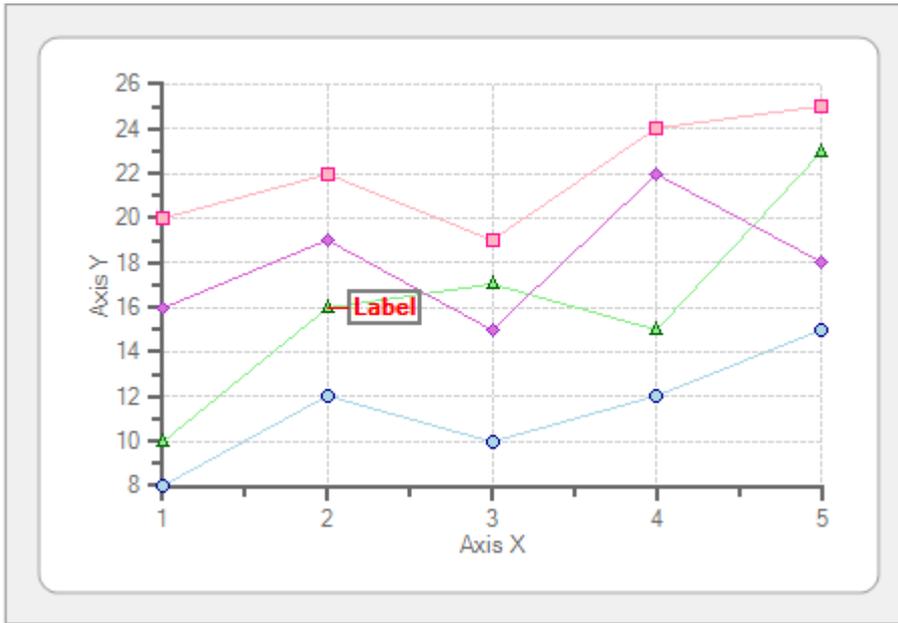


## Modify the Appearance of the Chart Labels using the Properties Window

To edit the chart labels using the **C1Chart** Properties window, complete the following steps:

1. Right-click on the **C1Chart** control and select **Properties** from the context menu. The Properties window for the **C1Chart** control will appear in the right pane.
2. Expand the **ChartLabels** and click on the ellipsis button next to **LabelsCollection**.
3. Select the existing label in the **Members** list and expand the **Style** node.
4. Expand the **Border** node and set the **BorderStyle** to **Solid**, **Color** to **Gray**, and **Thickness** to **2**.
5. Set the **ForeColor** to **Red** and click **OK**.

The label's appearance is modified:

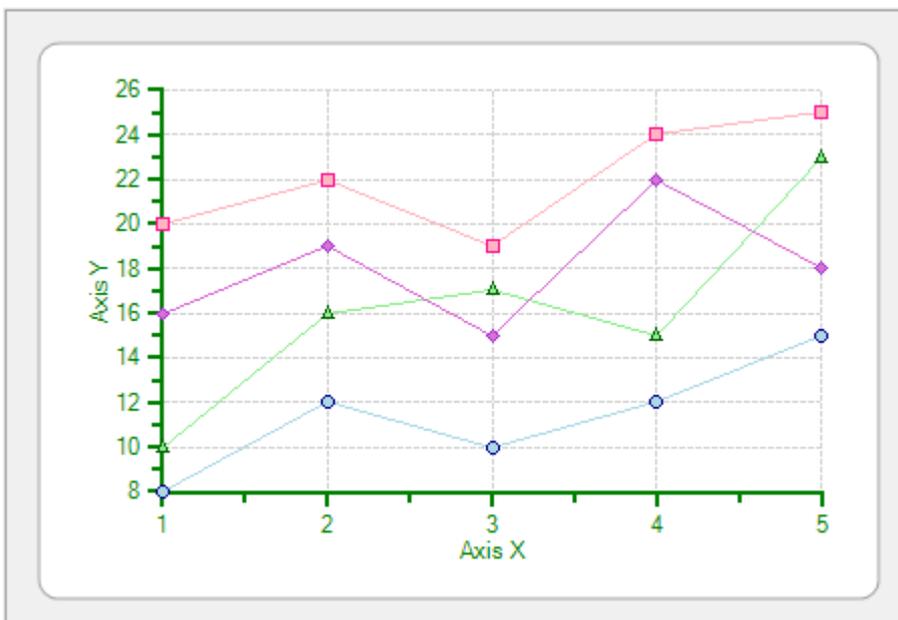


## Modify the X and Y Axis Appearance using the Properties Window

To modify the X and Y Axis appearance using the **C1Chart Properties** window, complete the following steps:

1. Right-click on the **C1Chart** control and select **Properties** from the context menu. The Properties window for the **C1Chart** control will appear in the right pane.
2. Expand the **ChartArea** node and then expand **AxisX**.
3. Click on the dropdown arrow next to the **ForeColor** property and select **Green** from the Custom tab.
4. Expand the **AxisY** node and Click on the dropdown arrow next to the **ForeColor** property and select **Green** from the **Custom** tab.

The X and Y axes forecolor are modified to green.



## Creating and Formatting Chart Elements Using the Smart Designer

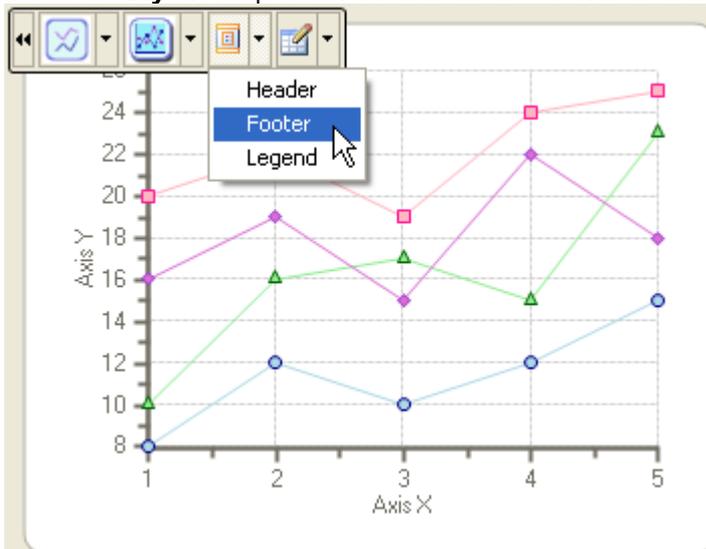
This section provides tasks for using the Smart Designer to create and format chart elements.

This section shows how to use the Smart Designer to create and format chart elements, or edit existing elements directly on the form. For example, rather than drilling down through the Chart's objects in the Properties window to set properties for each of the chart elements you can simply modify the chart elements directly on the form. A simple or complex chart can be created without using any code. It can all be done at design-time through the use of C1Chart's editors.

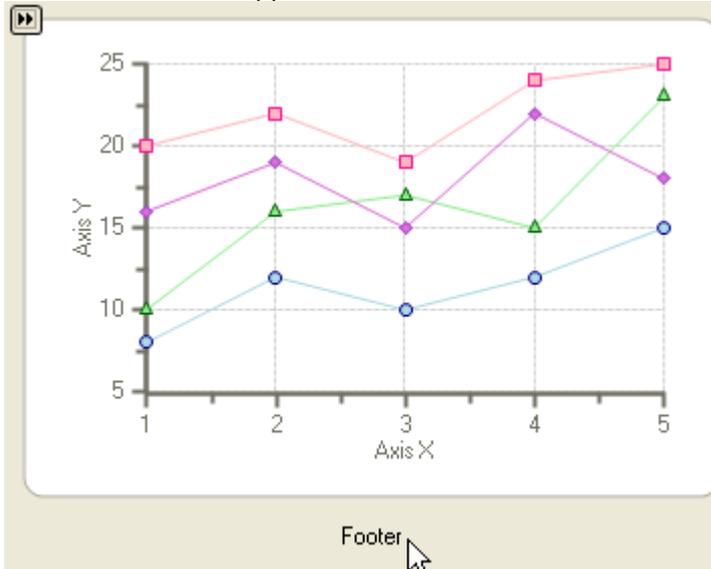
### Add a Chart Footer

To add a chart Footer using the C1Chart floating toolbar, complete the following steps:

1. Select the **C1Chart** control and click on the **Open** button  to open the C1Chart floating toolbar if it is not already open.
2. Select the **Layout** drop-down arrow from the C1Chart floating toolbar and choose the **Footer** item.



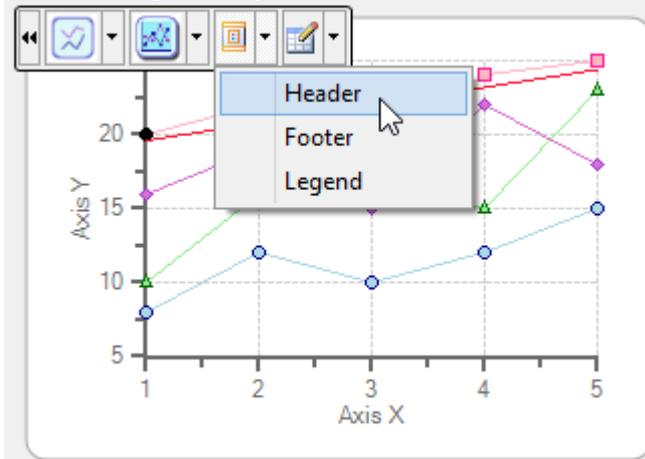
The **Footer** element appears below the chart area. This is the default position for the chart **Footer** element.



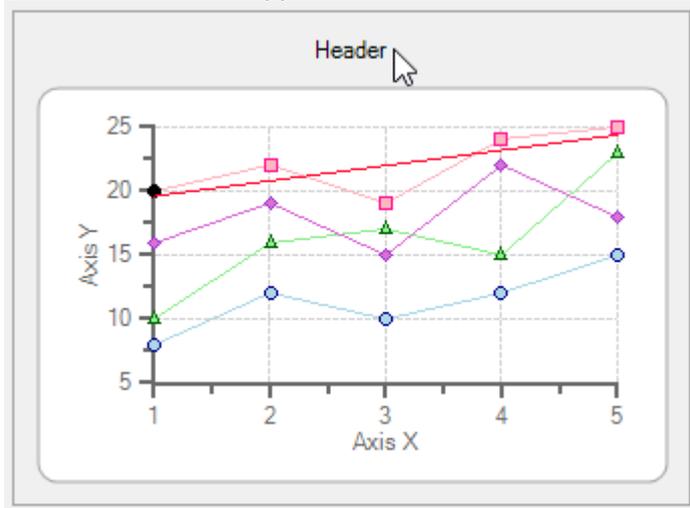
## Add a Chart Header

To add a chart header using the C1Chart floating toolbar, complete the following steps:

1. Select the **C1Chart** control and click on the **Open** button  to open the C1Chart floating toolbar if it is not already open.
2. Select the **Layout** drop-down arrow from the C1Chart floating toolbar and choose the **Header** item.



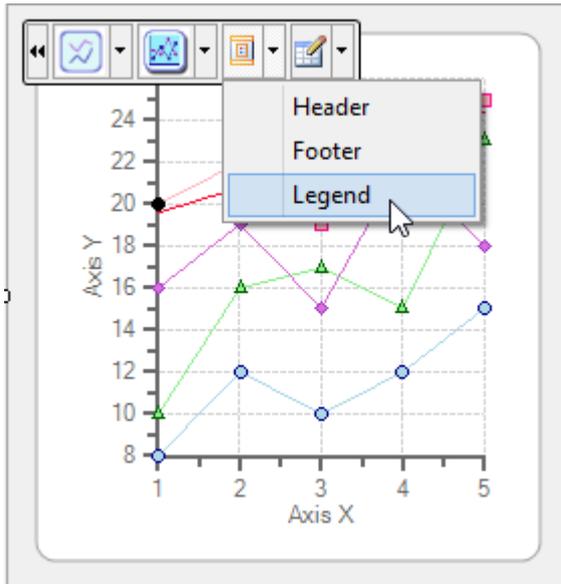
The **Header** element appears above the chart area. This is the default position for the chart **Header** element.



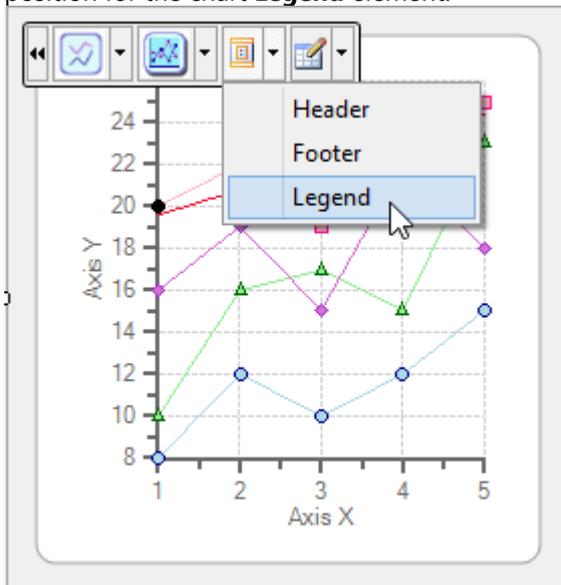
## Add a Chart Legend

To add a chart **Legend** using the C1Chart floating toolbar, complete the following steps:

1. Select the **C1Chart** control and click on the **Open** button  to open the C1Chart floating toolbar if it is not already open.
2. Select the **Layout** drop-down arrow from the C1Chart floating toolbar drop-down menu and choose the **Legend** item.



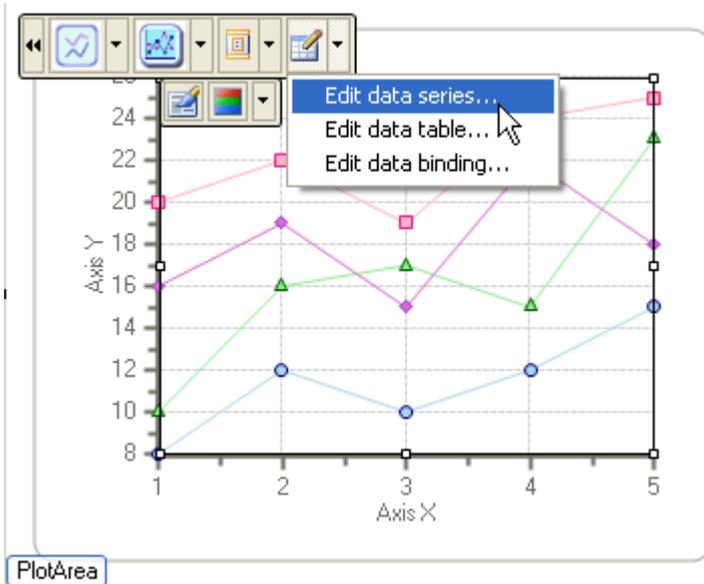
The **Legend** element appears to the right of the chart area or to the east of the chart area. This is the default position for the chart **Legend** element.



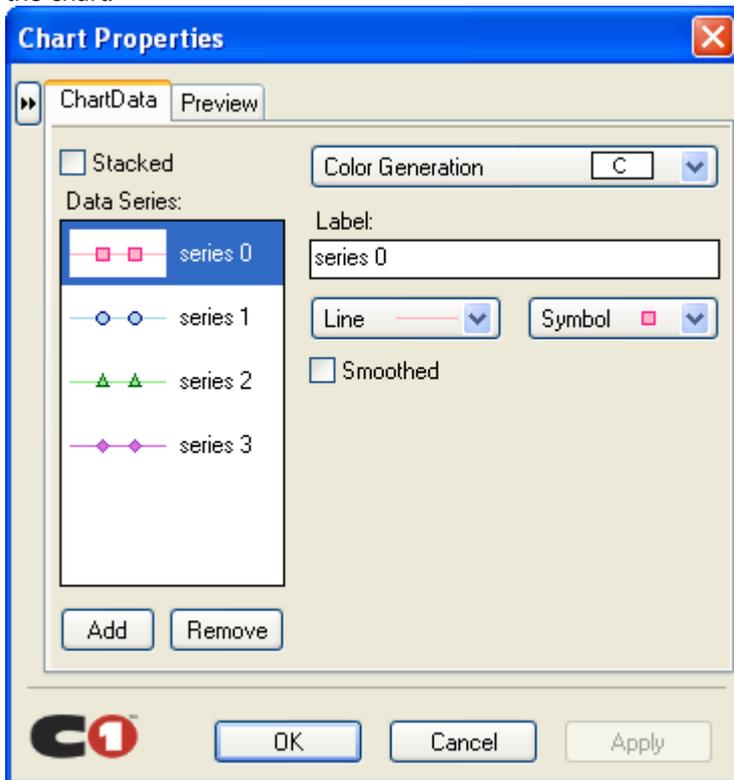
## Add Data Series to the Chart

To add data series to the chart using the C1Chart floating toolbar, complete the following steps:

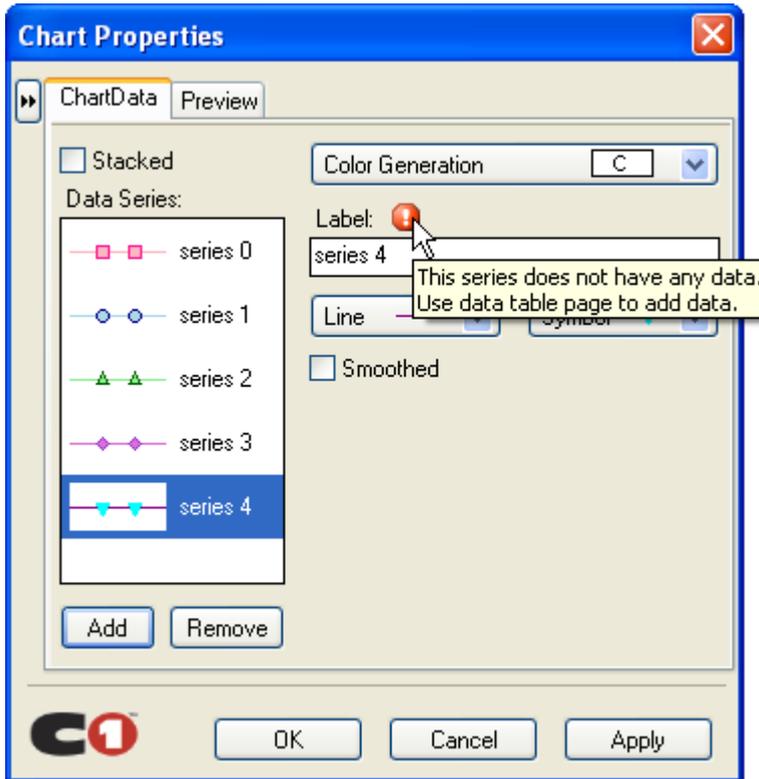
1. Select the **C1Chart** control and click on the **Open** button  to open the C1Chart floating toolbar if it is not already open.
2. Select the **Data** drop-down arrow from the C1Chart floating toolbar and choose the **Edit data series** item.



3. The **Chart Properties** designer appears for the ChartData series. Select the **Add** button to add a new series to the chart.



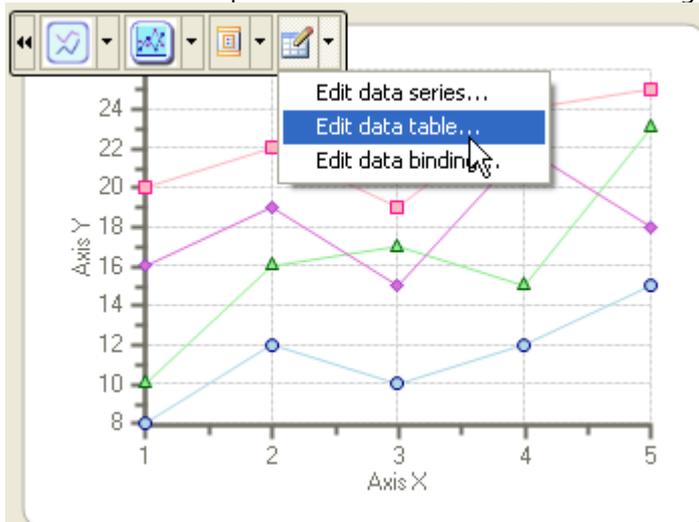
A blinking exclamation point appears to warn you that the data needs added to the new series.



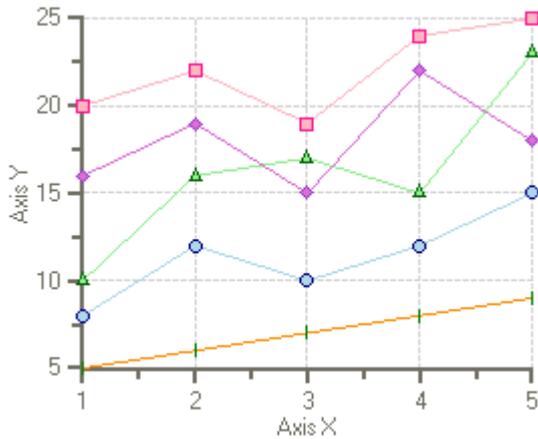
## Add Data to the Data Series

To add data to a data series using the C1Chart floating toolbar, complete the following steps:

1. Select the **C1Chart** control and click on the **Open** button  to open the C1Chart floating toolbar if it is not already open.
2. Select the **Data** drop-down arrow from the C1Chart floating toolbar and choose the **Edit data table** item.



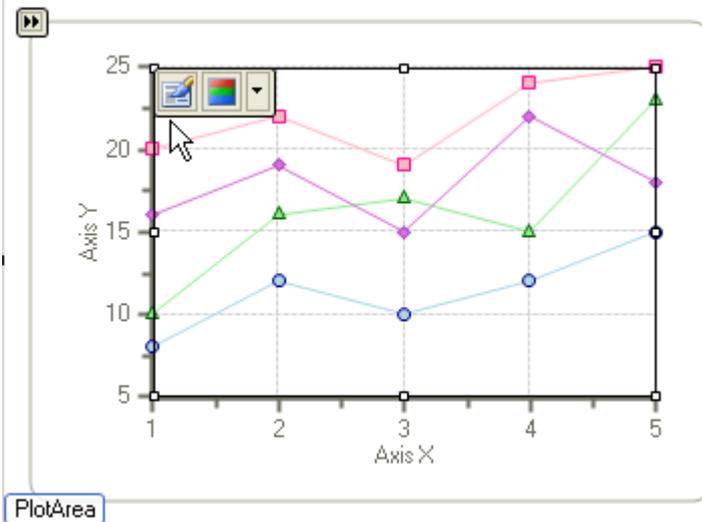
3. The **Chart Properties Data Table** editor appears. Locate the new series in the table and enter some arbitrary x and y values for the new series.
4. Once you finish entering the X and Y data values for the new series select **OK**. The new chart series appears on the chart.



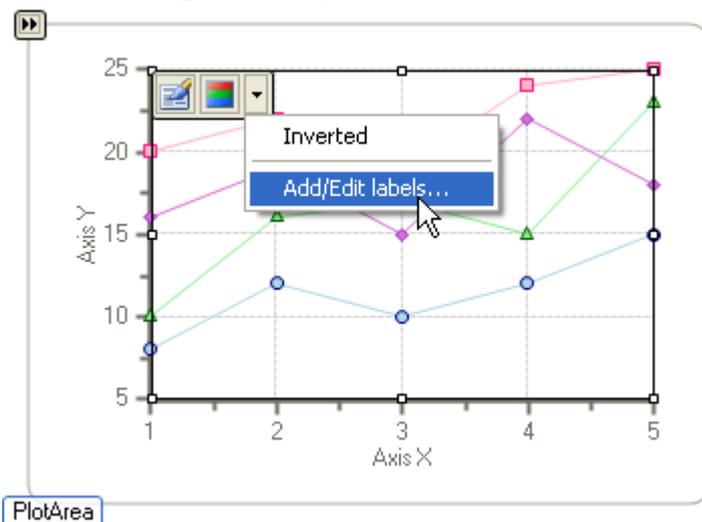
## Add Labels to the Chart

To add chart labels to the data series using the **Edit labels** editor, complete the following steps:

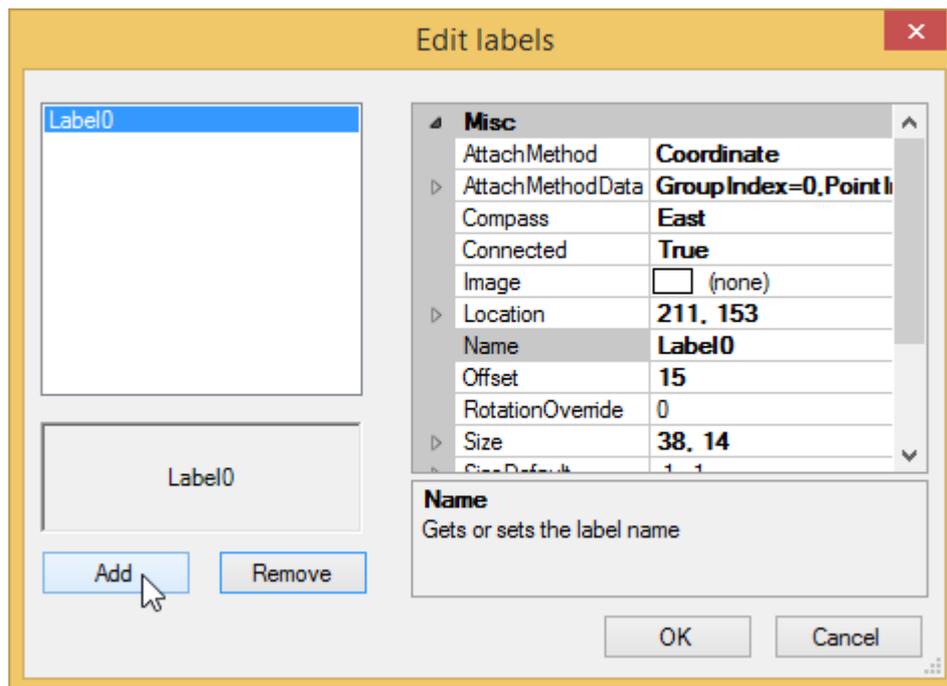
1. Slide your cursor inside the plot area to expose the **PlotArea** toolbar.



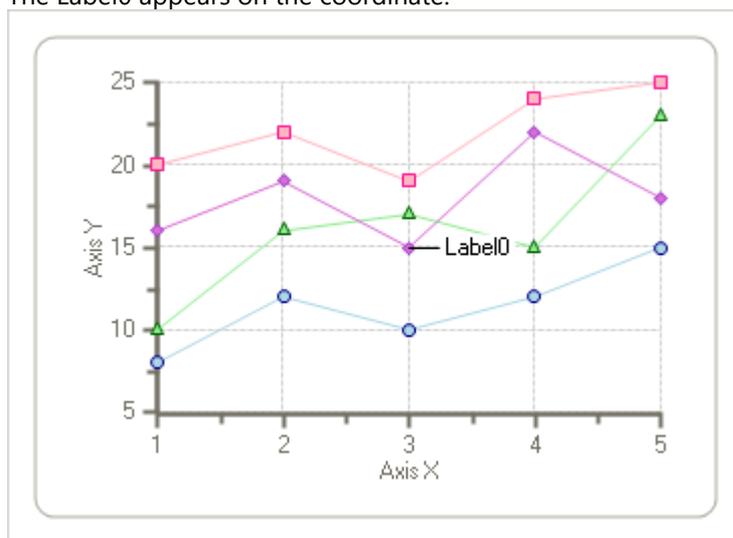
2. Select the **Background** drop-down button and choose **Add/Edit labels**.



3. Select the **Add** button in the **Edit labels** editor to add a new label. Repeat this to add more labels. If you would like to remove a label, select the label you would like to remove and click on the **Remove** button.



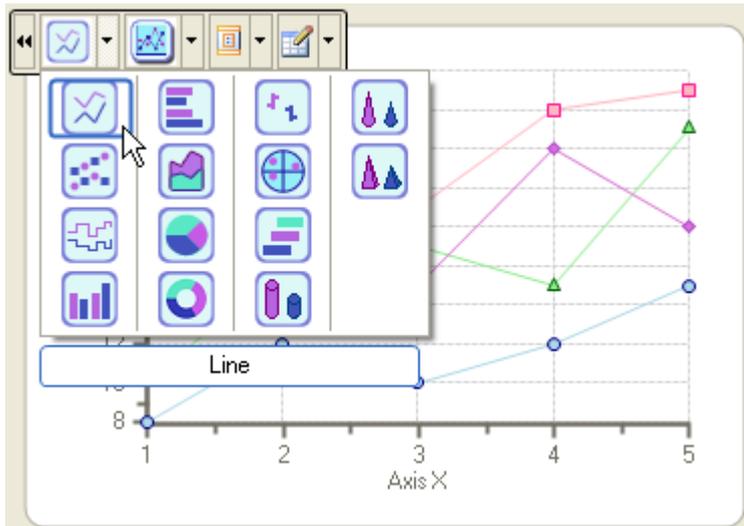
4. Select the **Close** button to close the **Edit labels** editor.  
The Label0 appears on the coordinate.



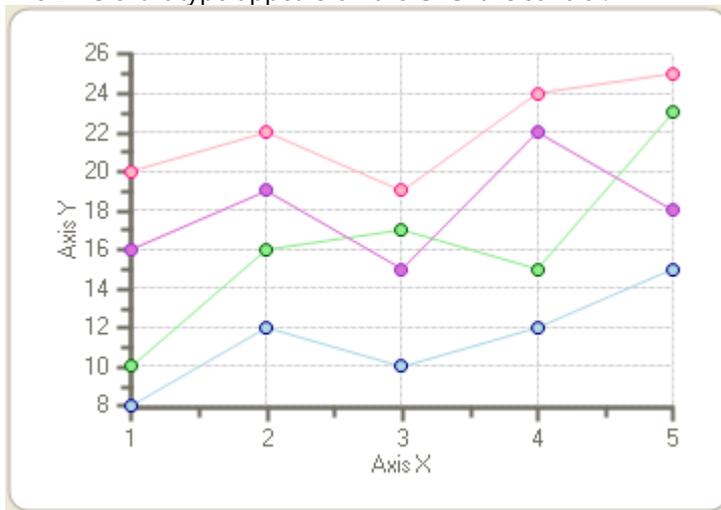
## Choose a Chart Type

To select a chart type through the C1Chart floating toolbar, complete the following steps:

1. Select the **C1Chart** control and click on the **Open** button  to open the C1Chart floating toolbar.
2. Click the **Chart type** drop-down arrow and choose the **Line** chart type from the list.



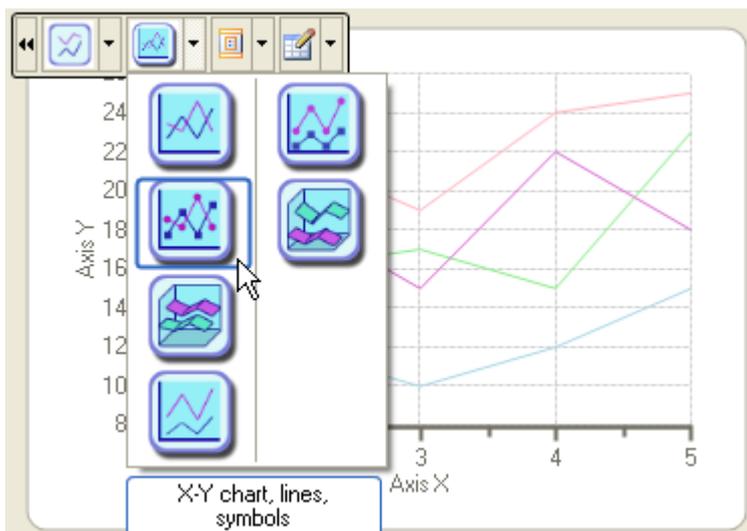
The **Line** chart type appears on the **C1Chart** control.



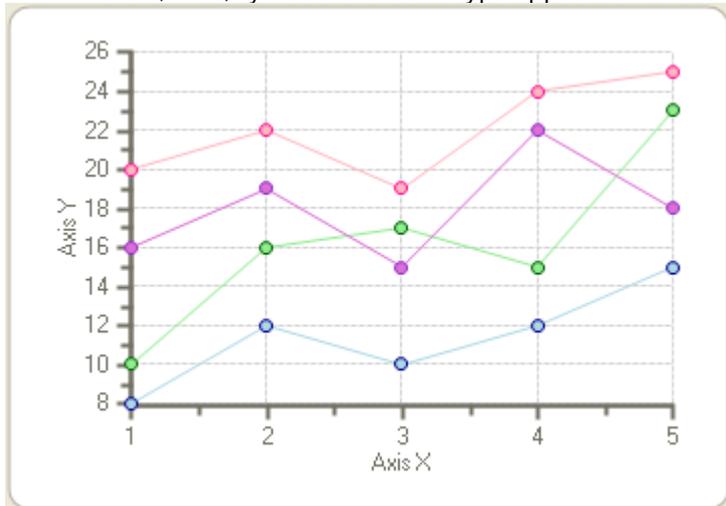
## Choose a Chart sub-type

To select a chart sub-type through the C1Chart floating toolbar, complete the following steps:

1. Select the **C1Chart** control and click on the **Open** button  to open the C1Chart floating toolbar if it is not already open.
2. Click the **Chart sub-type** drop-down arrow and choose the **X-Y chart, lines, symbols** chart sub-type.



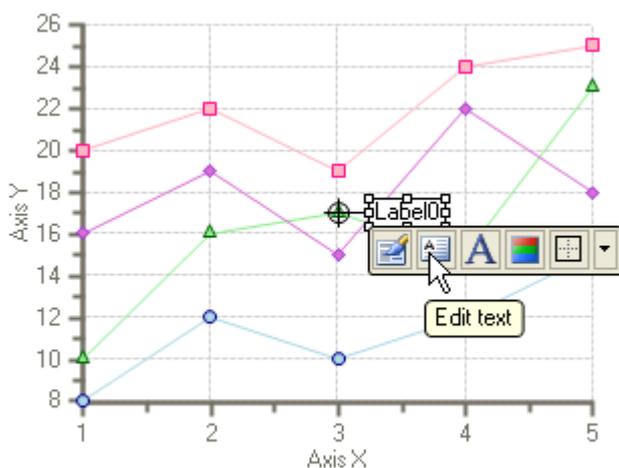
The X-Y chart, lines, symbols chart sub-type appears on the chart control.



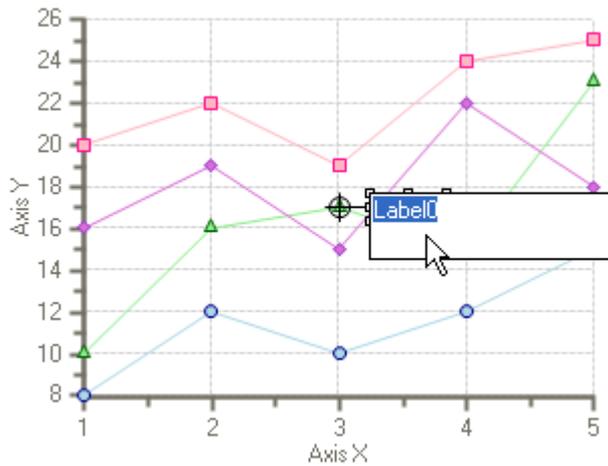
## Edit the Chart Labels

To edit the chart labels using the Label floating toolbar, complete the following steps:

1. Select the existing label on the chart and select the **Edit text** button from the Label floating toolbar.



2. The text box for the Label becomes editable.

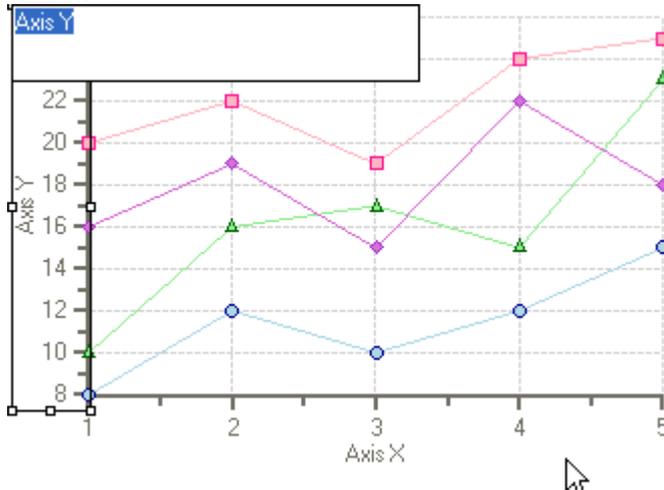


3. Place your cursor over the **Label0** and type its new label name.

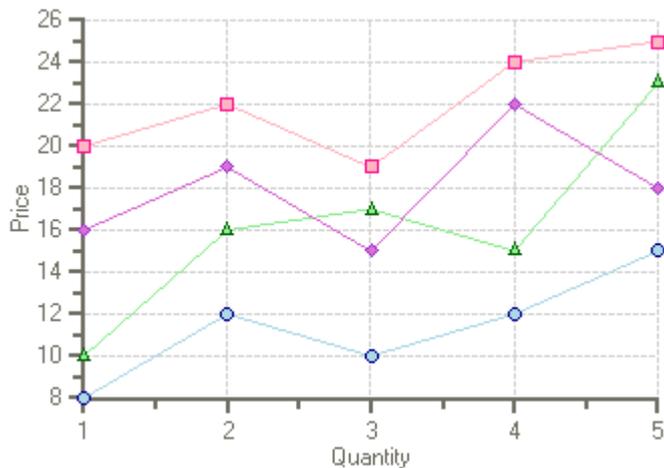
## Edit the X and Y Axis

To edit the Axis Y and Axis X label names directly on the form, complete the following steps:

1. Select **Axis Y** on the **C1Chart** control. Click again inside the Axis Y vertical box. The horizontal text box appears for Axis Y.



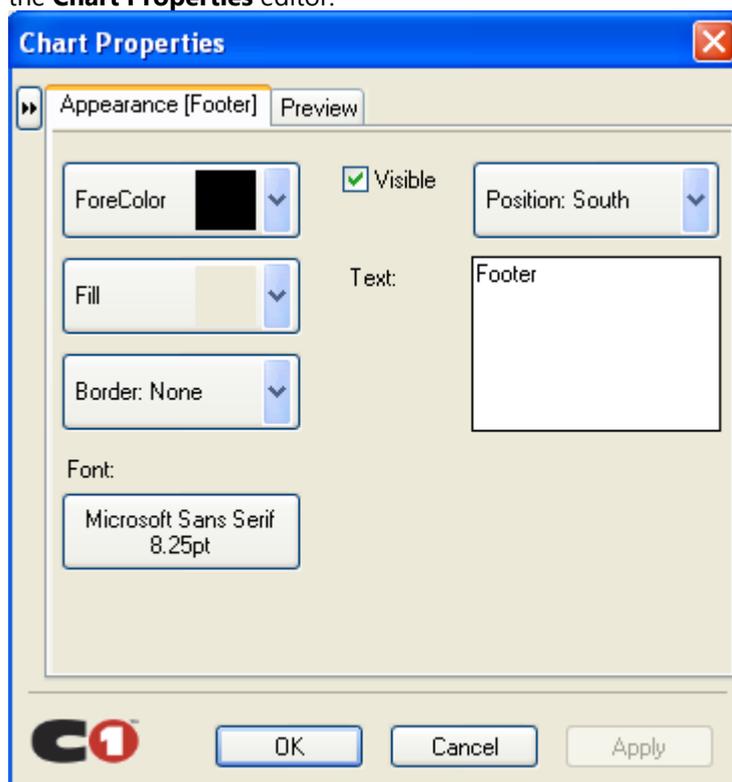
2. Type **Price** in the Axis Y textbox and press ENTER.
3. Select **Axis X** on the **C1Chart** control and click inside the horizontal box. The text box appears for the Axis X.
4. Type **Quantity** in the Axis X textbox and press ENTER.  
Price appears on the Y axis label and Quantity appears on the X axis label.



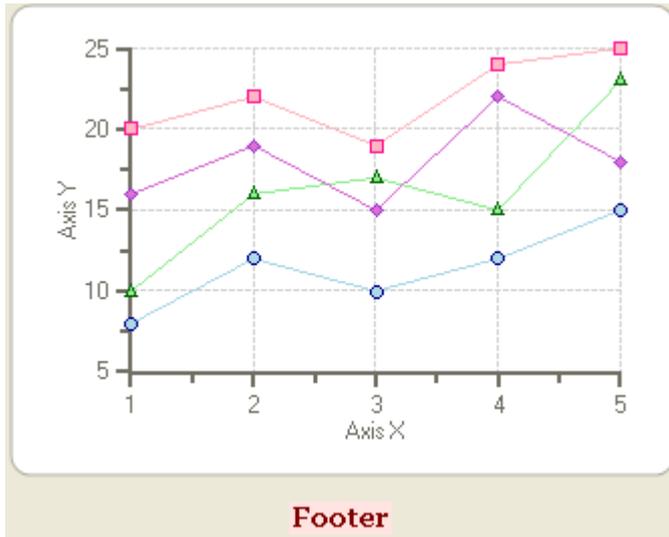
## Modify the Appearance of the Chart Footer

To modify the chart footer's appearance, complete the following steps:

1. Select the **Properties** button from the Footer's floating toolbar. The Appearance [Footer] properties appear in the **Chart Properties** editor.



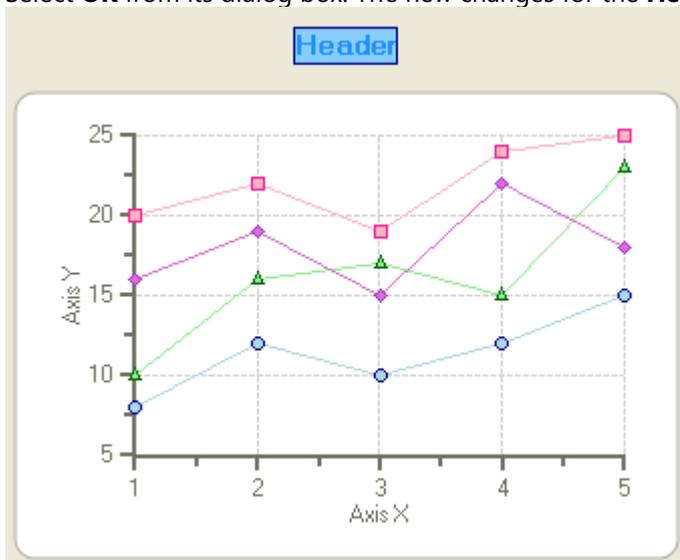
2. Modify its **ForeColor**, **Fill**, and **Border** properties to the following:
  - Set its ForeColor to **Maroon**.
  - Set its Fill color's style to **Gradient** and Color1 to **MistyRose** and Color2 to **RosyBrown**.
  - Set is Font type to **Georgia**, Font style to **Bold**, and Font size to **10**.
3. Select **OK** from its dialog box. The new changes for the **Footer** element appear on the chart.



## Modify the Appearance of the Chart Header

To modify the chart header's properties using the Header's floating toolbar, complete the following steps:

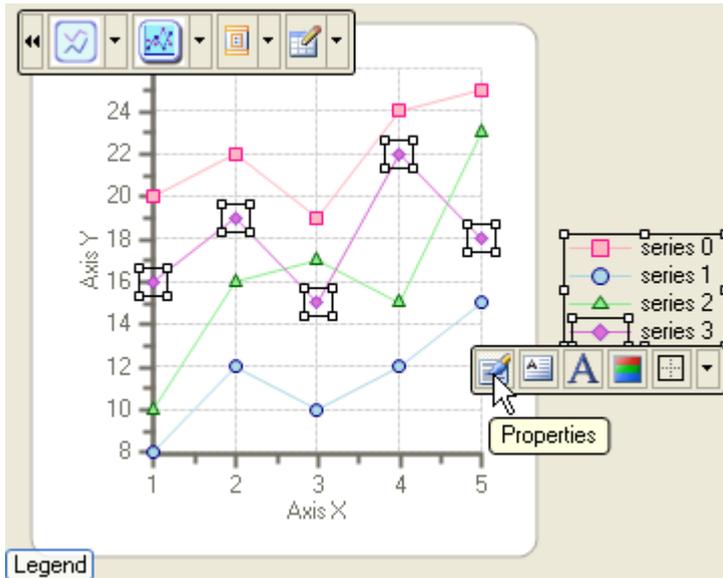
1. Select the **Chart Properties** button in the Header's floating toolbar.
2. Modify its **ForeColor**, **Fill**, and **Border** properties to the following:
  - Set its ForeColor to **Dodger Blue**.
  - Set its Fill color to **LightSkyBlue**.
  - Set its Border style to **Solid** and Border color to **DarkBlue**.
  - Set its Font style to **Bold** and Font size to **10**.
3. Select **OK** from its dialog box. The new changes for the **Header** element appear on the chart.



## Modify the Appearance of the Chart Legend

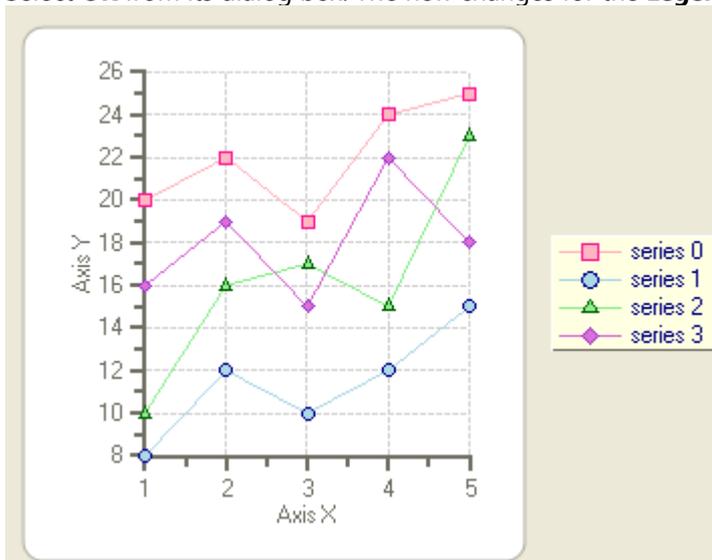
To modify the appearance of the chart **Legend** using the **Properties** button in the Legend's floating toolbar, complete the following steps:

1. Select the **Properties** button from the Legend's floating toolbar.



The Appearance Legend properties appear in the **Chart Properties** editor.

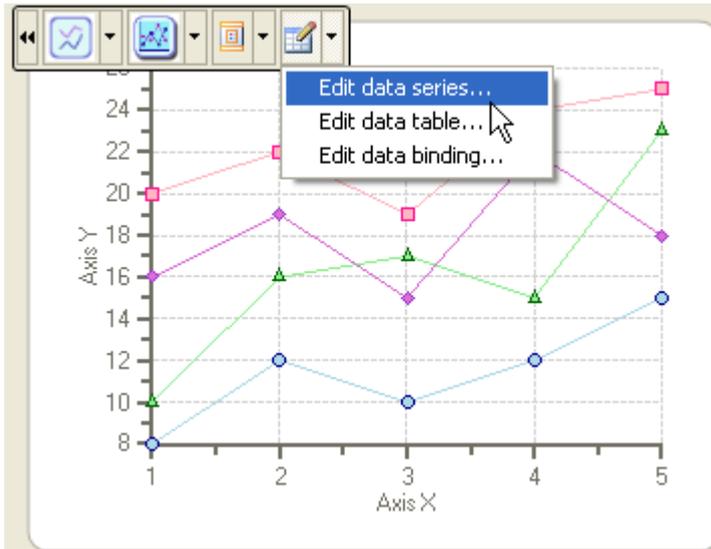
2. Modify its **ForeColor**, **Fill**, and **Border** properties to the following:
  - Set its **ForeColor** to **Navy**.
  - Set its **Fill** color to **Info**.
  - Set its **Border** to **Raised**.
3. Select **OK** from its dialog box. The new changes for the **Legend** element appear on the chart.



## Modify the Appearance of the Data Series

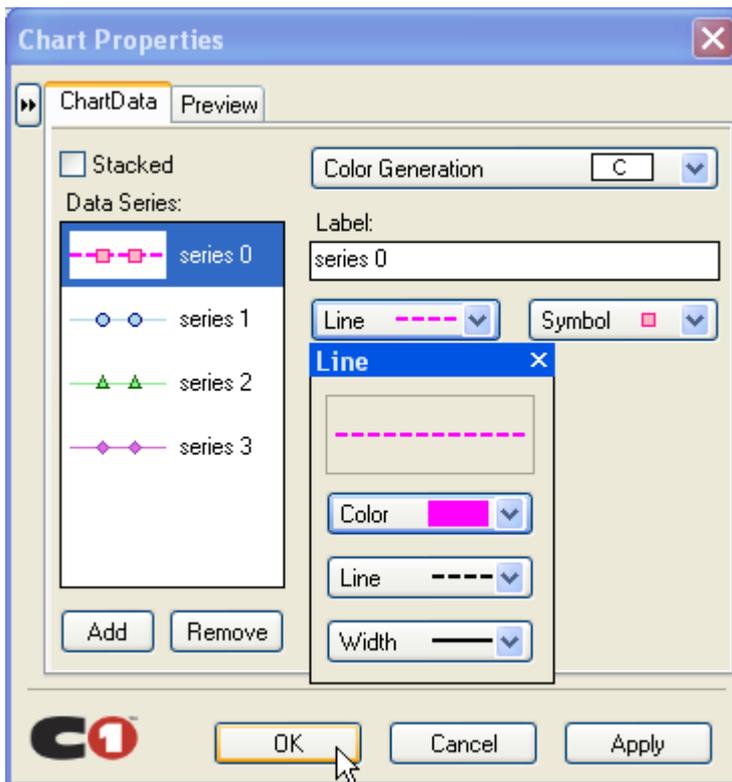
To modify the appearance of the data series using the C1Chart floating toolbar, complete the following steps:

1. Select the **C1Chart** control and click on the **Open** button  to open the C1Chart floating toolbar if it is not already open.
2. Select the **Data** drop-down arrow from the C1Chart floating toolbar and choose the **Edit data series** item.

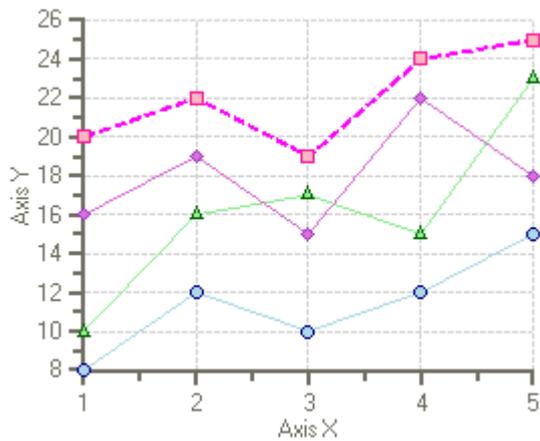


The **Chart Properties** designer appears for the data series.

3. Select **Series0** and click on the drop-down arrow for the **Line** property. Modify its Color, Line style, and Width style for its Line property to the following:
  - o Select **Fuchsia** from its **Color** drop-down list box.
  - o Select the **dashed line** style from its **Line** drop-down list box.
  - o Select **level 2** for its **Width** drop-down list box.



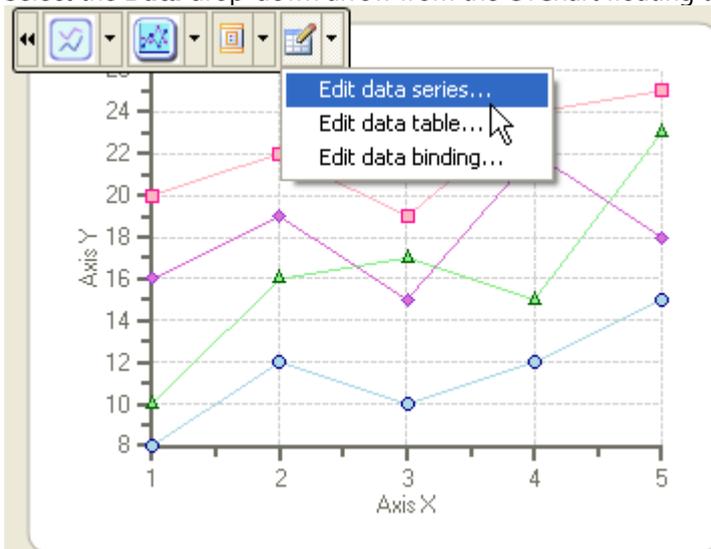
4. Select **OK** and the new changes for series 0 appears on the chart.



## Modify the Color Theme of the Data Series

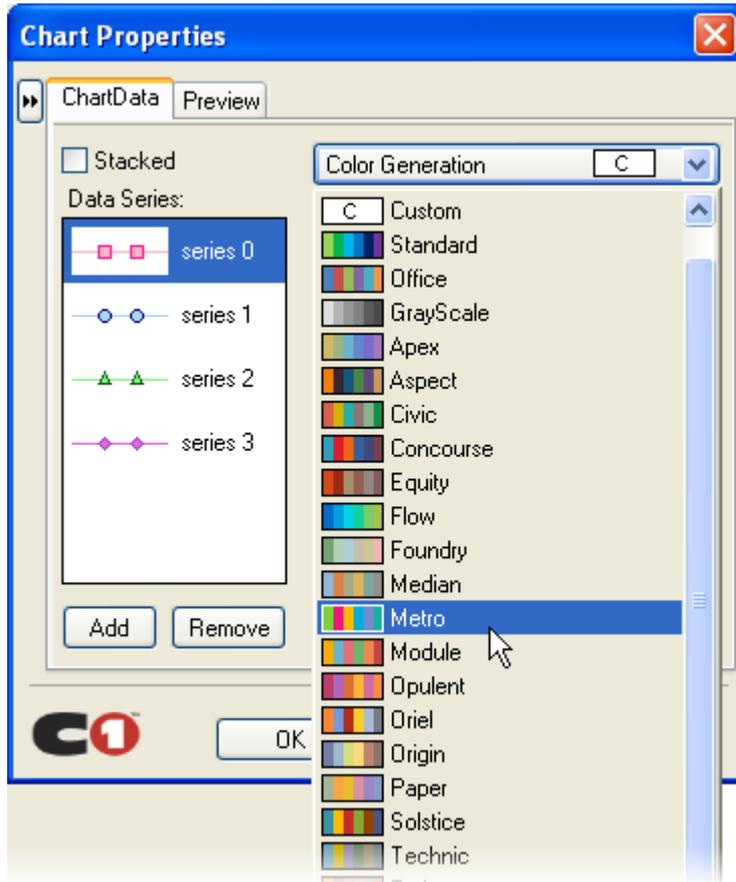
To modify the appearance of the chart data using the C1Chart floating toolbar, complete the following steps:

1. Select the **C1Chart** control and click on the **Open** button  to open the C1Chart floating toolbar if it is not already open.
2. Select the **Data** drop-down arrow from the C1Chart floating toolbar and choose the **Edit data series** item.



The **Chart Properties** designer appears for the data series.

3. With the **Series0** selected, click on the drop-down arrow for the **Color Generation** and select a color theme, **Metro**, for example:



The color theme applies to all data series.

4. Select **OK** and the changes appear on the chart.

## Attach Chart Labels

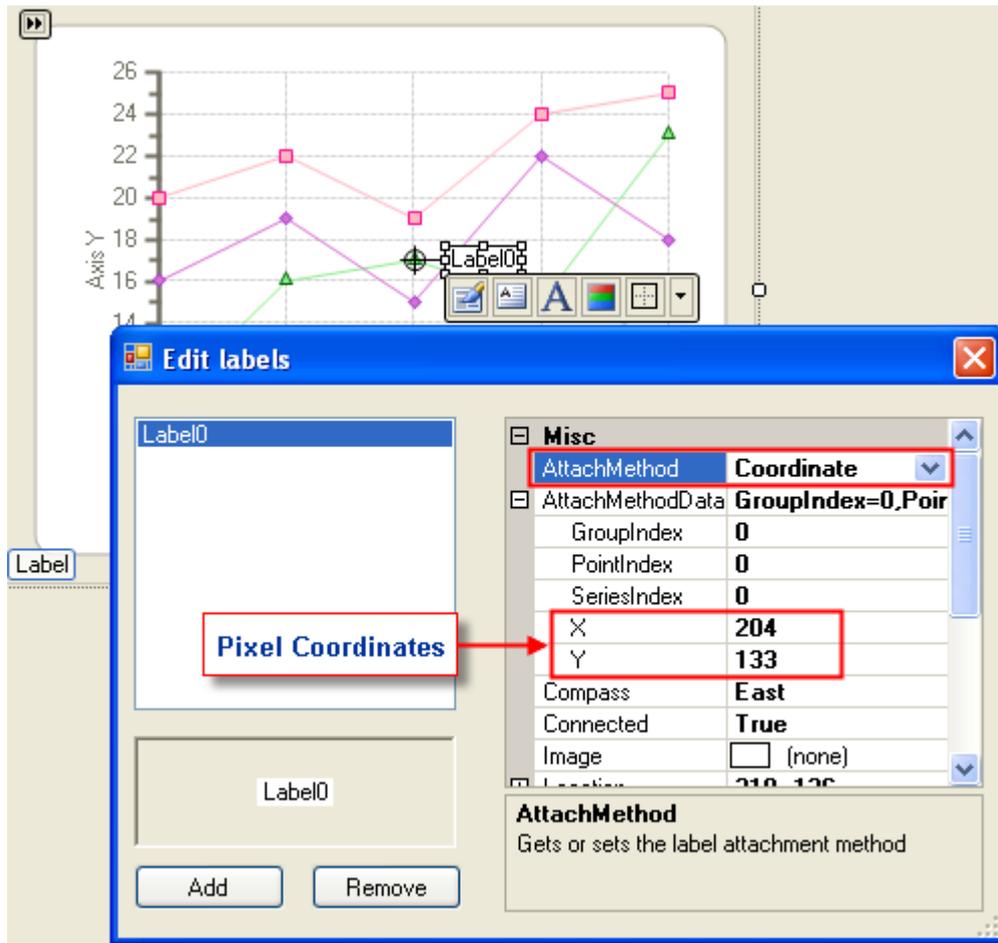
The Label floating toolbar includes three different attachment methods for the Chart labels. The following steps below show how to use each type of attachment method for the labels at design time. If you would like to see how to use these attachment methods through code, please see [Attaching the Chart Label by Pixel Coordinate](#), [Attaching the Chart Label by Data Coordinate](#), [Attaching the Chart Label by Data Point](#), or [Attaching the Chart Label by Data Point and Y Value](#).

## Attach By Coordinate

The Attached by Coordinate method item command gets the coordinate member of the AttachMethodEnum and sets the value of the `AttachMethod` property in the Label. It attaches the label anywhere on the chart. The following steps show how to attach the label using this method:

1. Select an existing Chart label and select the **Attached By Coordinate** item from its drop-down menu. If a label does not exist then add a label. For information about adding a chart label at design-time please see [Add Labels to the Chart](#).
2. Click on the **Properties** button of the **Label** floating toolbar and observe the following in its **Edit labels** editor:
  - o The AttachMethod is now set to Coordinate since the Attached by Coordinate item was chosen.
  - o Locate the X and Y coordinate values and notice that they are set to pixel values. This is because the Coordinate attachment method attaches the labels by the coordinate X and Y pixel values.

The following image illustrates the two observations:

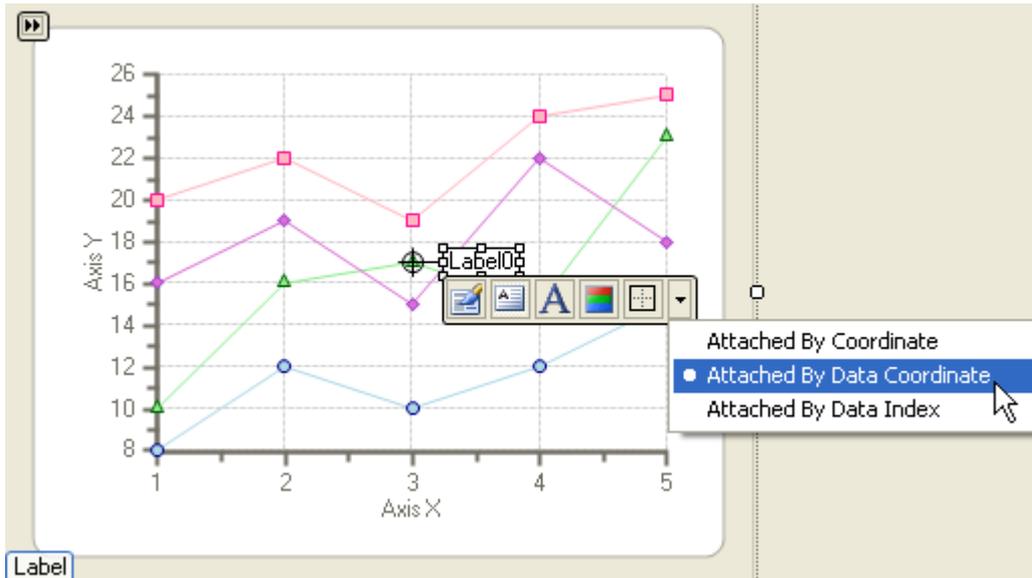


3. If you would like to specify a different position for the Label simply change the X and Y coordinate pixel value to where you would like the position of the label to lie on the chart.

## Attach By Data Coordinate

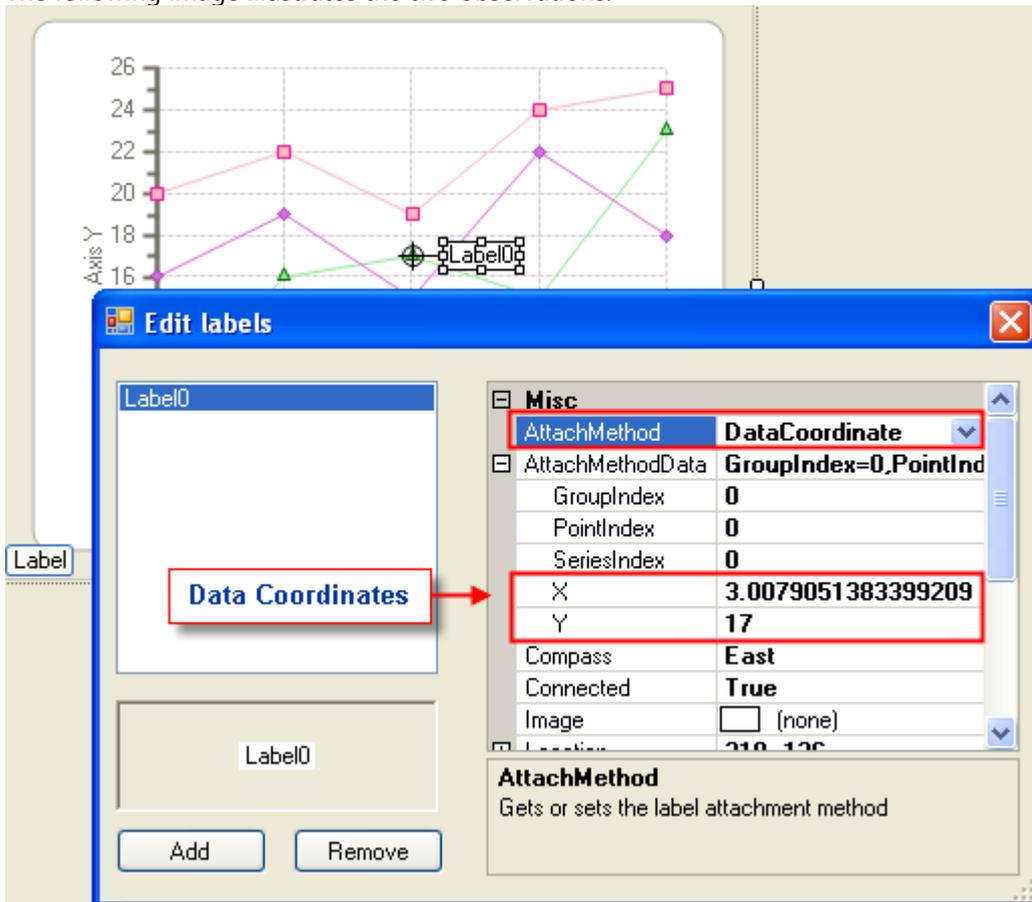
The Attached by Data Coordinate method item command gets the **DataCoordinate** member of the **AttachMethodEnum** and sets the value of the **AttachMethod** property in the Label. It attaches the label anywhere inside the plot area of the chart or to the specified coordinate location. To attach the label as a data coordinate, complete the following steps:

1. Select an existing Chart label and select the **Attached By Data Coordinate** item from its drop-down menu.



2. Click on the **Properties** button of the Label floating toolbar and observe the following in its **Edit labels** editor:
  - o The AttachMethod property is now set to DataCoordinate since the Attached by Data Coordinate item was chosen.
  - o Locate the X and Y coordinate values and notice that they are set to the values of the X and Y coordinate respectively. This is because the DataCoordinate attachment method attaches the labels by the defined coordinate X and Y values.

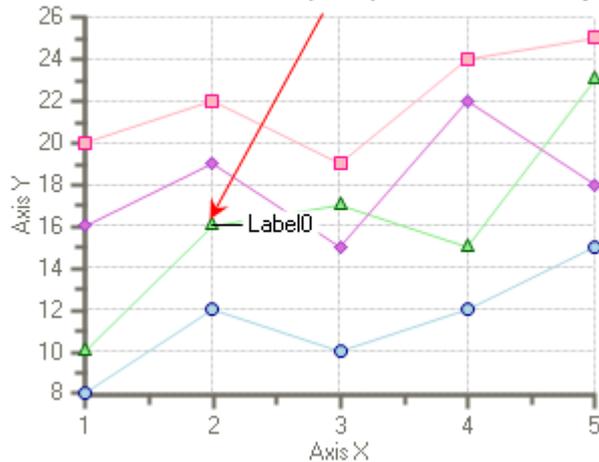
The following image illustrates the two observations:



3. If you would like the label to represent a different coordinate pair then simply change the X and Y coordinates in the **Edit labels** editor. For example, let's say we would like the label on the (2, 16) coordinate pair. In the **Edit labels** editor we change the X value to 2 and the Y value to 16. The label appears on the (2, 16) X-Y coordinate

pair. The image below shows the new position of the label.

**Label0 is located on the (2, 16) X-Y coordinate pair**

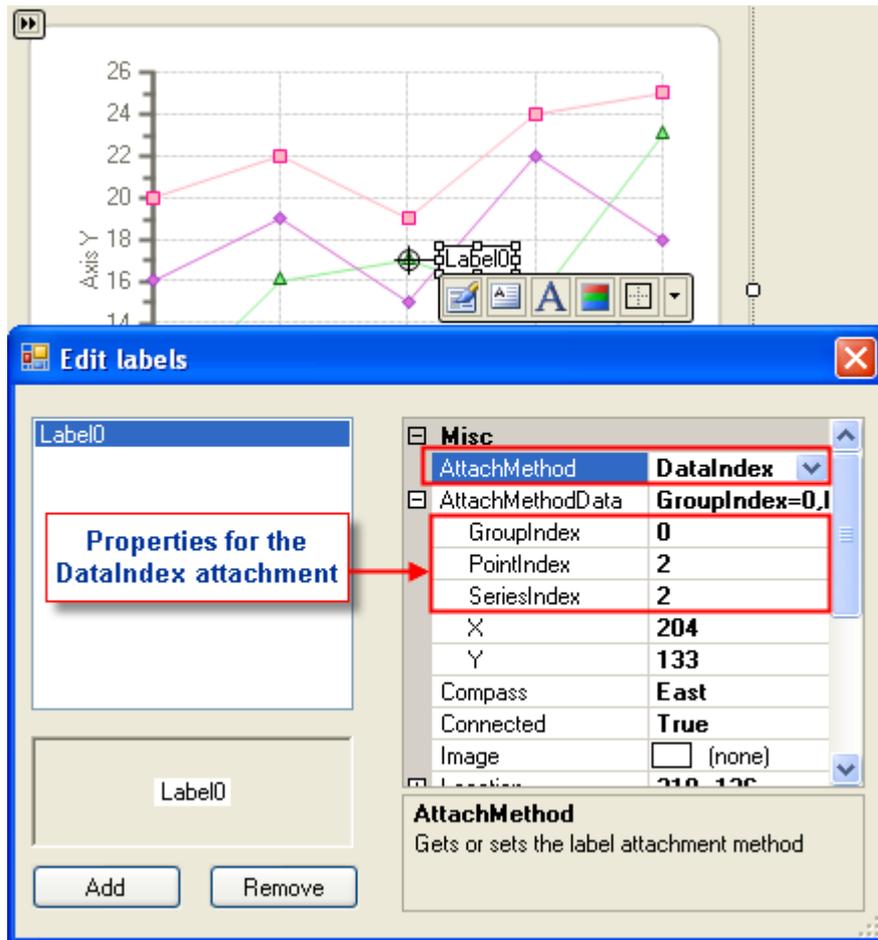


## Attach By Data Index

The **Attached By Data Index** command item gets the **DataIndex** member of the **AttachMethodEnum** and sets the value of the **Label.AttachMethod** property in the **Label**. It attaches the label to a specific data point on the plot area of the chart. The **GroupIndex**, **PointIndex**, and **SeriesIndex** properties can be specified. The following steps show how to attach the label using this method:

1. Select an existing Chart label and select the **Attached By Data Index** item from its drop-down menu.
2. Click on the **Properties** button of the **Label** toolbar and observe the following in its **Edit labels** editor:
  - o The **AttachMethod** is now set to **DataIndex** since the **Attached by Data Index** item was chosen.
  - o Expand the **AttachMethodData** object. Notice that the **Point Index** and **Series Index** properties each have a value associated with it. This is because the **DataIndex** attachment method attaches the labels by the data point. The **GroupIndex** value of 0 represents the **ChartGroup0**, the **PointIndex** value of 1 represents the second data point on the series, and the **SeriesIndex** value of 2 represents the third series on the chart.

The following image illustrates the two observations:



3. If you would like the label to represent a different data point then change the PointIndex and SeriesIndex properties in the **Edit labels** editor. For example, let's say we would like the label on series1, point3. In the **Edit labels** editor we change the SeriesIndex value to 1 and the PointIndex value to 3. The label appears on the series1, point3.

## Candle Chart Tasks

The following topics show tasks that apply to the Candle chart.

### Increasing the Width Size of the Candle Body

To make the width for the body of the candle chart wider, specify a larger value for the [Size](#) property like in the following code. Note, that the default value for the body width of the candle is 5.

#### To write code in Visual Basic

Visual Basic

```
c1Chart1.ChartGroups(0).ChartData.SeriesList(0).ChartSymbolStyle.Size = 10
```

#### To write code in C#

C#

```
c1Chart1.ChartGroups[0].ChartData.SeriesList[0].ChartSymbolStyle.Size = 10;
```

## Frequently Asked Questions

This section provides solutions to frequently asked 2D Chart questions.

### How do I change chart type?

To change the chart type, complete the following steps:

1. In the Properties window, expand the **ChartGroups** node.
2. Press the **ellipsis** button next to the **ChartGroupsCollection**. The **ChartGroups Collection Editor** appears.
3. Select the desired chart type from the **ChartType** property drop-down menu.

 **Note:** When using 2D Chart, the following chart types will be available: XY-Plot, Pie, Area, Polar, Radar, Bubble, HiLo, HiLoOpenClose, and Candle.

For more information about chart types, see [Specific 2D Charts](#).

### How do I change the way chart data is plotted?

To change the way the chart data is plotted, complete the following steps:

1. In the Properties window, expand the **ChartArea** node.
2. Expand the node of appropriate axis.
3. In **Grid Minor**, set **Max** and **Min** (maximum and minimum interval to display on axis) and set **UnitMajor** (numerical interval on axis).

For example, if **Max** is set to **20**, **Min** is set to **0**, and **UnitMajor** is set to **5**, every fifth number will display 0, 5, 10, 15, and 20 on the axis.

For more information, see [Axis Bounds](#).

### How do I change colors displayed in the chart?

#### To change series color:

1. Right-click on the chart control.
2. Select **Chart Properties**.
3. Choose **Data**.
4. Expand **SymbolStyle** node.
5. Select desired color from **Color** drop-down menu.

#### To change chart area color:

1. In the Properties window, expand the **ChartArea** node.
2. Expand **Style** node.
3. Select desired color from **BackColor** drop-down menu.

#### To change axis color:

1. In the Properties window, expand the **ChartArea** node.
2. Expand **Style** node.
3. Expand **Font** node.
4. Select desired color from **ForeColor** drop-down menu.

For more information, see [Axis Appearance](#).

## How do I change placement of the Legend?

To change the placement of the legend, complete the following steps:

1. In the Properties window, expand the **Legend** node.
2. Select East, West, North, or South from **Compass** drop-down menu.

For more information, see [Chart Legend](#).

## How do I add or modify a Border?

To add or modify a border, complete the following steps:

1. In the Properties window, expand the **ChartArea** node.
2. Expand **Style** node.
3. Expand **Border** node.
4. Select style type from the **BorderStyle** drop-down menu.
5. Select desired color from the **Color** drop-down menu.
6. Increase number of **Thickness** property to make border more prominent or decrease number to make border less prominent.

For more information, see [Chart Borders](#).

## How do I sync the X-axis of multiple charts?

In applications that show multiple charts, you can sync the ticks on the x-axis of each chart.

Use the [Margins](#) property to align the axis positions of several chart areas. For example, the following method arranges the x-axis positions for two charts. Note that it can also be modified for syncing three charts.

### To write code in Visual Basic

#### Visual Basic

```
Private Sub AdjustAxisPositions()  
    C1Chart1.ChartArea.Margins.Left = 10  
    C1Chart1.ChartArea.Margins.Right = 10  
    C1Chart2.ChartArea.Margins.Left = 10  
    C1Chart2.ChartArea.Margins.Right = 10  
    ' force redrawing  
    Dim img As Image = C1Chart1.GetImage()  
    If img IsNot Nothing Then  
        img.Dispose()  
    End If  
    img = C1Chart2.GetImage()  
    If img IsNot Nothing Then
```

```

        img.Dispose()
    End If
    Dim ch1_X As Rectangle = C1Chart1.ChartArea.AxisX.GetAxisRect()
    Dim ch2_X As Rectangle = C1Chart2.ChartArea.AxisX.GetAxisRect()
    Dim d As Integer = ch1_X.Left - ch2_X.Left
    If d > 0 Then
        C1Chart2.ChartArea.Margins.Left += d
    ElseIf d < 0 Then
        C1Chart1.ChartArea.Margins.Left -= d
    End If
    d = ch1_X.Right - ch2_X.Right
    If d > 0 Then
        C1Chart1.ChartArea.Margins.Right += d
    ElseIf d < 0 Then
        C1Chart2.ChartArea.Margins.Right -= d
    End If
End Sub

```

### To write code in C#

C#

```

void AdjustAxisPositions()
{
    c1Chart1.ChartArea.Margins.Left = 10;
    c1Chart1.ChartArea.Margins.Right = 10;
    c1Chart2.ChartArea.Margins.Left = 10;
    c1Chart2.ChartArea.Margins.Right = 10;
    // force redrawing
    Image img = c1Chart1.GetImage();
    if (img != null)
        img.Dispose();
    img = c1Chart2.GetImage();
    if (img != null)
        img.Dispose();
    Rectangle ch1_X = c1Chart1.ChartArea.AxisX.GetAxisRect();
    Rectangle ch2_X = c1Chart2.ChartArea.AxisX.GetAxisRect();
    int d = ch1_X.Left - ch2_X.Left;
    if (d > 0)
        c1Chart2.ChartArea.Margins.Left += d;
    else if (d < 0)
        c1Chart1.ChartArea.Margins.Left -= d;
        d = ch1_X.Right - ch2_X.Right;
    if (d > 0)
        c1Chart1.ChartArea.Margins.Right += d;
    else if (d < 0)
        c1Chart2.ChartArea.Margins.Right -= d;
}

```