Chart Types

Chart Types

Table of Contents

1 Chart Type Anatomy 101

Introduction

Chart types are constructed using several settings. Among them, the most significant are:

- **Chart Type**
- **Series Type**
- **Axis Scale**
- **GaugeType**

These properties work in concert to provide unprecedented flexibility.

Chart Types

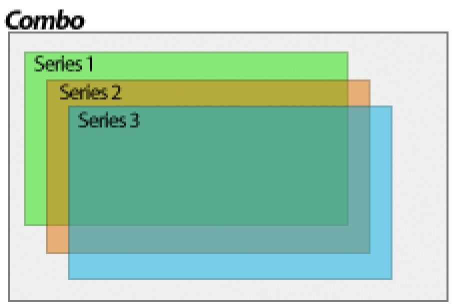
The most fundamental property, **Chart.Type**, determines the generic layout of series and in the case of pie, and radar, a type of chart. Axis scales further contribute to how elements are laid out, i.e. (Stacked). The series type determines how the series are drawn (line, bar, etc.).

```
[C#]
Chart.Type = ChartType.Radar;

[Visual Basic]
Chart.Type = ChartType.Radar
```

Combo

Combo Chart Type

This chart type supports all the series types. It is vertically oriented so the y axis (value axis) can only contain numeric or time values while the x axis can also contain names of elements (category axis).



Supported Series Types

- All

*Element values (properties) used on the (**Y Axis**).*

- YValue
- YValueStart
- YDateTime
- YDateTimeStart
- BubbleSize
- Complete
- (*Financial:* Open Close High Low)

*Element values used on the (**X Axis**).*

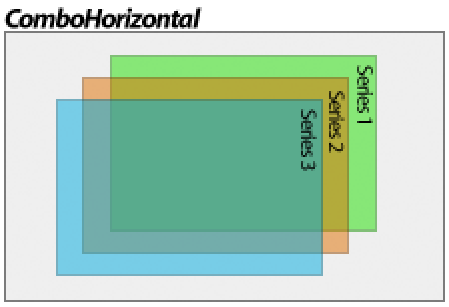
- XValue
- XDateTime
- Name

ComboHorizontal

Chart Types

ComboHorizontal Chart Type

This chart types is horizontally oriented so the x axis can only contain numeric or time values. It is the same as the Combo type except it's oriented horizontally.



Supported Series Types

- All *(New in version 3.4)*

Notice the X and Y properties are switched
*Element values used on the (**Y Axis**).*

- XValue
- XDateTime
- Name

*Element values used on the (**X Axis**).*

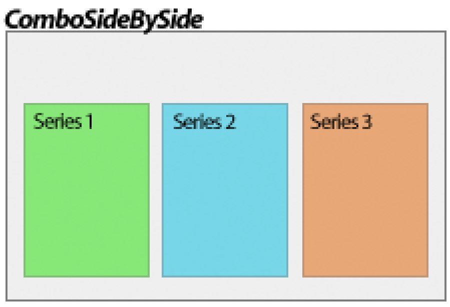
- YValue
- YValueStart
- YDateTime
- YDateTimeStart
- BubbleSize
- Complete
- Element.Parent (Gantt Dependencies)
- Element.InstanceID (Gantt Dependencies)
- Element.ParentID (Gantt Dependencies)
- (*Financial:* Open Close High Low)

Notice that the axes element values are reversed for this chart type, meaning the element XValue actually goes on the Y axis. The reason for this is so that if you change between vertical and horizontal chart types, the result will be the same.

ComboSideBySide

ComboSideBySide Chart Type

This chart type is similar to combo but series are placed side by side and the x axis tick labels show series names instead of element names. This layout can be achieved in a normal combo chart by transposing the data.



Supported Series Types

- All

*Element values used on the (**Y Axis**).*

- YValue

- YValueStart
- YDateTime
- YDateTimeStart
- BubbleSize
- Complete
- (*Financial*: Open Close High Low)

*Element values used on the (**X Axis**).*

- None (Series.Name us used)

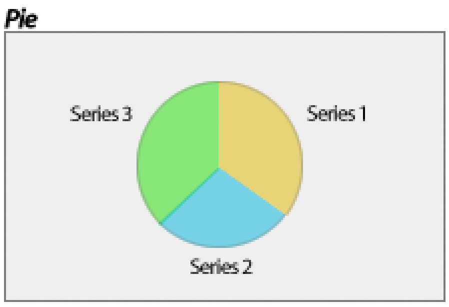
Notes

- SideBySide does not support **unclustered columns**.

Pie & Donut

Pie Chart Type

Shows a single pie consisting of all the series added to the chart where each slice represents a series.



Supported Series Types

- N/A

*Element values used on the (**Y Axis**).*

- YValue

*Element values used on the (**X Axis**).*

- N/A

Related properties:

- Element.ForceMarker
- Element.ExplodeSlice
- Chart.PieLabelMode
- Chart.ExplodedSliceAmount
- Chart.DonutHoleSize (Donut Only)
- Chart.XAxis.OrientationAngle

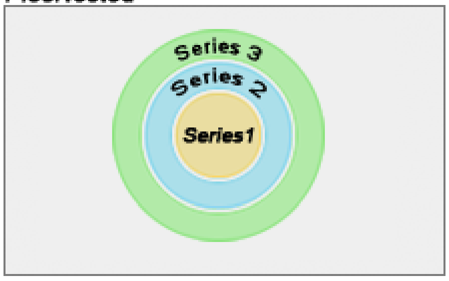
Notice that the series are slices when using this chart type, hence, if only a single series is used, the chart will have a single 100% slice. To see each element use the plural ChartType.Pies.

PiesNested

PiesNested Chart Type

Shows a series of nested pies.

PiesNested



Scale.Stacked
The width of each ring can also reflect the series total values when Chart.YAxis.Scale = Scale.Stacked is used. Alternatively Series.DefaultElement.BubbleSize can be used to specify ring widths and will be applied when these properties are set.

Scale.Normal
When Scale.Normal is used, the pies are not stacked, the ring width of each pie is measured from the middle point of the pies. This means that if two series have the same ring width, one will completely cover the other.

In this mode, SpacingPercentageNested is not applicable.

Supported Series Types

- N/A

*Element values used on the (**Y Axis**).*

- YValue
- Series.DefaultElement.BubbleSize (Control ring widths).

*Element values used on the (**X Axis**).*

- N/A

Related properties:

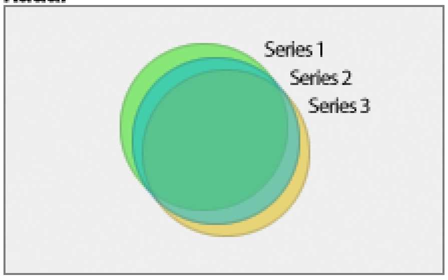
- Element.ForceMarker
- Chart.SpacingPercentageNested
- Chart.PieLabelMode

Radar

Radar Chart Type

Draws all series on a single radar. Both x and y axes are used. The x axis surrounds the radar and the y axis goes from the center to the outside.

Radar



Using this type will force each series to share a single x axis.

Supported Series Types

- Marker
- Line
- AreaLine
- Bubble
- Column

*Element values used on the (**Y Axis**).*

- YValue
- YValueStart
- YDateTime
- YDateTimeStart
- BubbleSize

*Element values used on the (**X Axis**).*

- Name
- XValue
- XDateTime

Related properties:

- Chart.RadarLabelMode
- Axis.RadarMode
- Chart.XAxis.OrientationAngle

Polar Charts
The radar chart can support standard spider and polar axes. If the x axis uses a category scale, the default will be a spider axis (resembling a spider web). If the axis uses a numeric or time axis, the type will default to Polar. The type can also be specified explicitly using:

```
[C#]
Chart.XAxis.RadarMode = RadarMode.Polar;

[Visual Basic]
Chart.XAxis.RadarMode = RadarMode.Polar
```

Lines on polar charts will appear as curves. This is a feature which helps determine the actual values of lines along along the width of the lines.

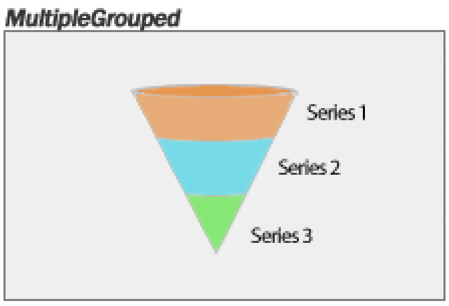
MultipleGrouped

MultipleGrouped Chart Type

Draws all series as elements of a single object such as types specified by SeriesTypeMultiple.

Supported Series Types (SeriesTypeMultiple)

- FunnelCone
- FunnelPyramid
- Pyramid
- Cone
- Pie
- Donut
- Gauge
- StackedBubble (Uses Element.YValue, not Element.BubbleSize)



*Element values used on the (**Y Axis**).*

- YValue

Related properties:

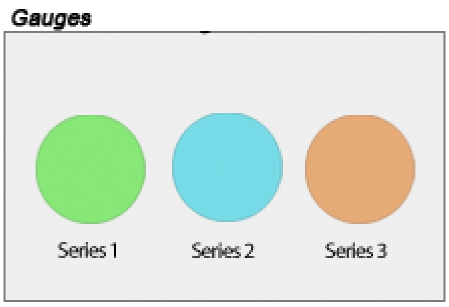
- Chart.FunnelNozzlePercentage
- Chart.FunnelNozzleWidthPercentage
- Chart.SpacingPercentage
- Chart.BubbleStackShadeAsOne
- Chart.BubbleCenterStack

Gauges

Gauges Chart Type

Shows each series as a gauge. Each can be a different GaugeType.

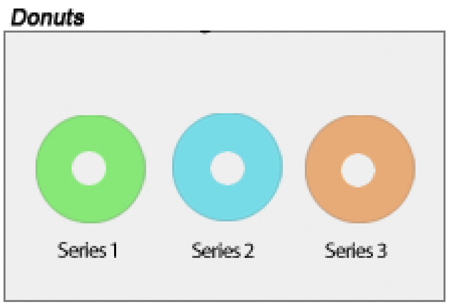
Please refer to the **Gauges Tutorial (Section 1.1)** for detailed documentation.



Donuts

Donuts Chart Type

The series are laid out in a way that allows for the maximum space usage. One donut for each series.



Supported Series Types

- N/A

Element values used on the (Y Axis).

- YValue
- Height (2D Only)

Element values used on the (X Axis).

- N/A

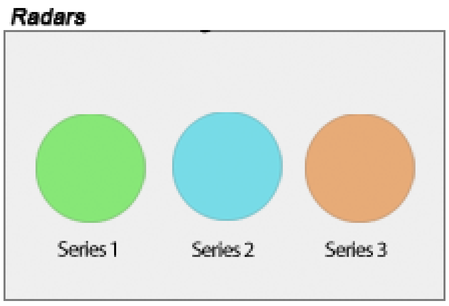
Related properties:

- Element.ForceMarker
- Element.ExplodeSlice
- Chart.PieLabelMode
- Chart.ExplodedSliceAmount
- Chart.DonutHoleSize
- Chart.XAxis.OrientationAngle
- Element.Outline

Radars

Radars Chart Type

Shows a radar for each series in the chart.



Supported Series Types

- Marker
- Line
- AreaLine
- Bubble
- Column

Element values used on the (Y Axis).

- YValue
- YValueStart
- YDateTime
- YDateTimeStart
- BubbleSize

Element values used on the (X Axis).

- Name
- XValue
- XDateTime

Related properties:

- Chart.RadarLabelMode
- Axis.RadarMode
- Chart.XAxis.OrientationAngle

Polar Charts
The radar chart can support standard spider and polar axes. If the x axis uses a category scale, the default will be a spider axis (resembling a spider web). If the axis uses a numeric or time axis, the type will default to Polar. The type can also be specified explicitly using:

```
[C#]
Chart.XAxis.RadarMode = RadarMode.Polar;

[Visual Basic]
Chart.XAxis.RadarMode = RadarMode.Polar
```

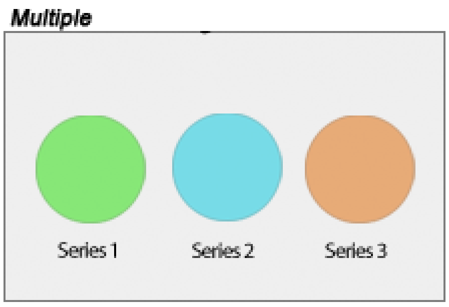
Lines on polar charts will appear as curves. This is a feature which helps determine the actual values of lines along along the width of the lines.

Multiple

Multiple Chart Type

Chart Types

Draws all series as individual objects. Different types are supported and can be specified by SeriesTypeMultiple for each series simultaneously. The benefit of this chart type is that it allows using different types such as pies, gauges, and the multiple types all on the same chart.



Supported Series Types (SeriesTypeMultiple)

- FunnelCone
- FunnelPyramid
- Pyramid
- Cone
- Pie
- Donut
- Gauge
- StackedBubble (Uses Element.YValue, not Element.BubbleSize)

*Element values used on the (**Y Axis**).*

- YValue

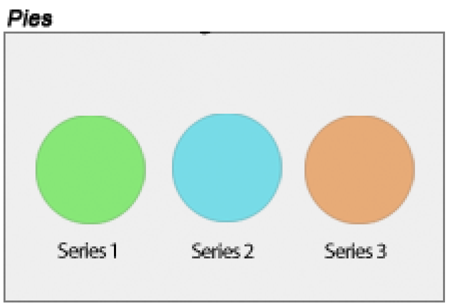
Related properties:

- Chart.FunnelNozzlePercentage
- Chart.FunnelNozzleWidthPercentage
- Chart.StackedSpacingPercentage
- Chart.BubbleStackShadeAsOne
- Chart.BubbleCenterStack

Pies

Pies Chart Type

Shows a single pie for each series in the chart.



Supported Series Types

- N/A

*Element values used on the (**Y Axis**).*

- YValue
- Height (2D Only)
- Length

*Element values used on the (**X Axis**).*

- N/A

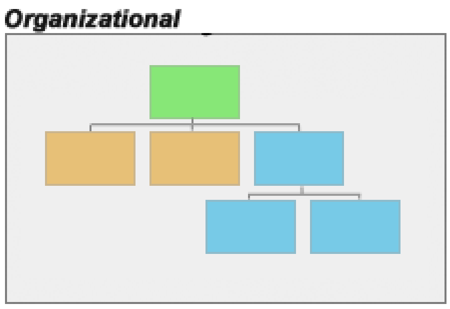
Related properties:

- Element.ForceMarker
- Element.ExplodeSlice

- Chart.PieLabelMode
- Chart.ExplodedSliceAmount
- Chart.XAxis.OrientationAngle
- Element.Outline

Organizational

Organizational Chart Type



Shows an organizational chart based on the elements in the charts series collection.

Only Element annotations are used with this chart type. Annotation text can be specified but if left unpopulated, the element name text will be used inside the annotations.

An organizational chart has a single element at the top with no parent. Every other element will have either ParentID or Parent property set to indicate which other element it is under. All elements can be in a single series, however, the Series.Line property is used to draw connections between elements so multiple series can be used to group elements that use the same connecting line style.

Elements with no names or annotation text can be used to create more complex layouts by drawing a straight line through the location of the element as demonstrated in sample Gallery/M10

This type is explained in more detail in the **Organizational Chart Tutorial (Section 1.2)**.

Supported Series Types

- N/A

*Element values used on the (**Y Axis**).*

- N/A

*Element values used on the (**X Axis**).*

- N/A

Related properties:

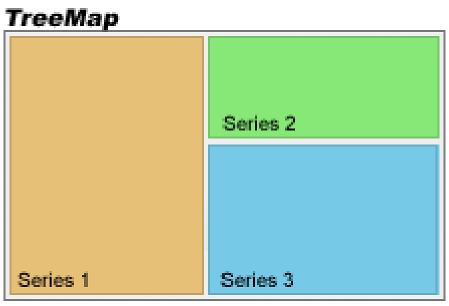
- Element.Annotation
- Element.Parent
- Element.InstanceID
- Element.ParentID
- Series.Line

TreeMap

TreeMap Chart Type

TreeMap charts are used to display hierarchial tree structure data with nested rectangles. TreeMap charts can be created using live data or manually. The boxes support all bar shading effects and smart label layout. Series.Box can be used to control the styling and header of each series box.

This type is explained in more detail in the **TreeMap Charts Tutorial (Section 1.4)**.



Supported Series Types

- N/A

*Element values used on the (**Y Axis**).*

- N/A

*Element values used on the (**X Axis**).*

- N/A

Related properties:

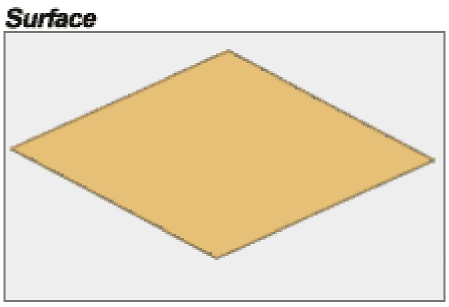
- Element.SmartLabel
- Element.YValue
- ChartArea.Padding
- Series.Box
- XAxis.Label

Surface

Surface Chart Type

The surface chart type enables visualizing 3d surface data. It provides a number of different visualizations and properties to tune them.

This type is explained in more detail in the **Surface Tutorial (Section 1.5)**.



Supported Series Types (SeriesTypeSurface)

- Surface
- Contour
- SurfaceX
- SurfaceY

Chart Properties

- Chart.View3D.AngleOfTilt
- Chart.View3D.AngleOfRotation
- Chart.View3D.EnableLighting
- Chart.ZAxis

Series Properties

- Series.ZAxis

- Series.Background
- Series.Line

*Element values (properties) used on the (**Y Axis**).*

- YValue

*Element values used on the (**X Axis**).*

- XValue

*Element values used on the (**Z Axis**).*

- ZValue

⚠ Other chart types such as **Bubble**, **Scatter**, and **Gantt** are obsolete but available for legacy support. They default to Combo, Combo, and ComboHorizontal respectively and set other properties to resemble the legacy results.

ChartType.**Financial** is also obsolete and unnecessary. It is provided only for backward compatibility as well.

Series Types

The next major contributor to chart types is the series type. The above chart type list includes a section on supported series types for each. Different series types can be specified for each series simultaneously using the following chart types:

- Combo
- ComboSideBySide
- ComboHorizontal
- Radar(s)
- Multiple
- MultipleGrouped
- Surface

This allows you to create virtually unlimited number of different chart types.

SeriesType enumeration

```
[C#]
mySeries.Type = SeriesType.Line;

[Visual Basic]
mySeries.Type = SeriesType.Line
```

- **Marker**
    - In 3D markers have no depth
    - Related Properties
        - **Element.Marker**
        - Element.YValue
        - Element.YDateTime

        - Series/Element.LegendEntry.Marker

- **Line**
    - Supported in 2D and 3D.
    - Each element can have a different color along the same line series.
    - In 2D elements will automatically show element markers.
    - Line caps can be applied to 2D lines through the Series.Line line cap properties.
    - Related Properties
        - **Series.Line** (2D)
        - Element.YValue
        - Element.YDateTime
        - **Element.Marker**

- **Spline**
    - Supported in 2D and 3D.
    - Each element can have a different color along the same spline series.
    - In 2D elements will automatically show element markers.
    - Line caps can be applied to 2D splines through the Series.Line line cap properties.
    - Related Properties
        - **Series.Line** (2D)
        - Element.YValue
        - Element.YDateTime
        - **Series.SplineTensionPercent**
        - **Element.Marker**

- **AreaLine**
    - Supported in 2D and 3D.
    - Each element can have a different color along the same line series.
    - In 2D elements will automatically show element markers.
    - Supports Stacked Axis Scales.
    - Related Properties
        - **Series.Line** (2D)
        - Element.YValue
        - Element.YDateTime
        - Element.YValueStart
        - Element.YDateTime
        - **Element.Marker**

- **AreaSpline**
    - Supports Stacked Axis Scales.
    - Related Properties

        - **Series.Line** (2D)
        - Element.YValue
        - Element.YDateTime
        - Element.YValueStart
        - Element.YDateTime
        - **Element.Marker**

- **Column & Bar**
    - Supports Stacked Axis Scales.
    - Supports shading effects.
    - Supports Element.Outline
    - Supports Element.Complete indicator which is useful with Gantt charts.
    - With ComboHorizontal, gantt dependencies can be used and are defined the same way as organizational chart relationships.

- **Cylinder**
    - Supports Stacked Axis Scales.
    - Supports Element.Outline
    - Supports 2D and 3D

- **Cone**
    - Supports Stacked Axis Scales.
    - Supports Element.Outline

- **Pyramid**
    - Supports Stacked Axis Scales.
    - Supports Element.Outline

- **Bubble**
    - Requires Element.BubbleSize values.
    - Supports shading effects.
    - Supports Element.Outline
    - Related Properties
        - **Element.BubbleSize**
        - **Chart.MaximumBubbleSize**

- **BubbleShape**
    - Requires Element.BubbleSize values.
    - Supports ShadingEffectMode.One
    - Supports 2D and 3D views.
    - Supports Element.Outline

- Related Properties
  - Element.ShapeType
  - **Chart.MaximumBubbleSize**
  - LegendEntry.ShapeType
  - **ShapeType Enumeration**

SeriesTypeFinancial enumeration

```
[C#]
mySeries.Type = SeriesTypeFinancial.CandleStick;

[Visual Basic]
mySeries.Type = SeriesTypeFinancial.CandleStick
```

- **CandleStick**
  - Element values used
    - Open or YValueStart
    - Close or YValue
    - High
    - Low
    - Complete
  - Supports column shading.
- **Bar**
  - When a open value is not provided this bar will be an HLC bar, if provided it will be a OHLC.
- **Open**
- **Close**
- **High**
- **Low**
- **HighLowArea**

SeriesTypeMultiple enumeration

- **FunnelCone**
  - Related Properties
    - **Chart.FunnelNozzlePercentage**
    - **Chart.FunnelNozzleWidthPercentage**
    - **Chart.SpacingPercentage**
- **FunnelPyramid**
  - Related Properties
    - Chart.FunnelNozzlePercentage
    - Chart.FunnelNozzleWidthPercentage
    - Chart.FunnelSpacingPercentage

- **Cone**
- **Pyramid**
- **Pie**
- **Gauge**
- **StackedBubble**
    - Related Properties
        - **Chart.BubbleCenterStack**
        - **Chart.BubbleStackShadeAsOne**

This series type (SeriesTypeMultiple) applies to charts using ChartType.Multiple & ChartType.MultipleGrouped.

SeriesTypeSurface enumeration

```
[C#]
mySeries.Type = SeriesTypeSurface.Contour;

[Visual Basic]
mySeries.Type = SeriesTypeSurface.Contour
```

- **Surface**
    - Element values used
        - XValue, YValue, ZValue
    - Related Properties
        - Series.InterpolationFillFactor
- **Contour**
    - Element values used
        - XValue, YValue, ZValue
    - Related Properties
        - Series.ContourCount
- **SurfaceX**
    - Element values used
        - XValue, YValue, ZValue
- **SurfaceY**
    - Element values used
        - XValue, YValue, ZValue
- Related Properties for **All Surface Types**
    - Series.DefaultElement.DrawToFloor
    - Series.ZAxis
    - Series.Use3D

Axis Scales

Further customization is achieved by specifying an axis scale. Besides controlling the quantitative scale type, scales also dictate how series behave. For example an axis scale can specify that columns are stacked.

See also: **Element Values & Axis Scales (Section 1.8.1)** | **Element Layout and Axes (Section 1.8)** | **Z Axis effect (Section 1.8.2)**

Conclusion

As you can see, the final chart type is fundamentally based on three settings. This mix and match concept may be more complicated than a single property, however, the flexibility it provides makes it well worth understanding.

1.1 Gauge Types

Introduction

A number of different gauge types are available within the gauge series type and can be specified by the Series.GaugeType property. Each series on a gauge chart can use a different gauge type setting.

```
[C#]
mySeries.GaugeType = GaugeType.Circular;

[Visual Basic]
mySeries.GaugeType = GaugeType.Circular
```

Gauges can also be used with ChartType.Multiple and ChartType.MultipleGrouped when Series.Type = SeriesTypeMultiple.Gauges.

Automatic Padding Feature

Generally many of these gauge types are used by themselves without a legend or other features. When the chart detects that the legend is not visible and the chart area is not drawn (by using ChartArea.ClearColors ()), it will automatically get rid of additional padding around the gauge.

Manual Sizing and Positioning (New in v4.4)

A static size can be applied to any gauge series by using the Series.Box.Position property and specifying a Size object.

```
[C#]
mySeries.Box.Position = new Size(100,100);

[Visual Basic]
mySeries.Box.Position = new Size(100,100)
```

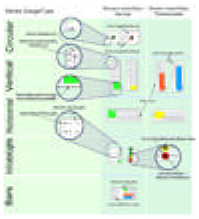
This feature also allows absolute positioning of each gauge when a Rectangle object is used instead.

```
[C#]
mySeries.Box.Position = new Rectangle(20,20,100,100);

[Visual Basic]
mySeries.Box.Position = new Rectangle(20,20,100,100)
```

Circular gauges height must be the same as their width, therefore, only the specified Width is used with this feature.

QuickStart Illustration

The following illustration shows which properties control certain aspects of gauge types.



Gauge type options (Series.GaugeType)

Circular
The circular type is the default gauge type used.

Custom Image Needles
This type also supports using custom images as needles along with the DynamicColor feature of ElementMarkers. For more information on using custom needles, see the **Element Markers (Section 1.7)**.

Supported Features

- All axis scale types except for stacked scales.
- Supports axis markers.
- Multiple needle types
- Multiple needles within a gauge
- An icon within the gauge using YAxis.LabelMarker
- Glass Shading when Chart.ClipGauges = false and Chart.Use3D = true

Related Properties

- YAxis.Line (border line properties)
- YAxis.LabelMarker
- YAxis.OrientationAngle
- YAxis.RangeAngle
- YAxis.SweepAngle
- Series.Background (background color)
- Series.NeedleType
- Series.GaugeBorderShape
- Series.Box
- Chart.ClipGauges
- Chart.Use3D (effect when not clipped)

Bars
This gauge type can be useful displaying element values when an axis scale is not necessary.

Supported Features

- All axis scale types.
- Axis minimum/maximum settings
- All bar ShadingEffectModes.
- The XAxis Tick and Element's Label is used along side of the bars.
- The tick and element labels can be aligned to right, left, or center (element label only).
- Segmented Bar type when using Series.Type = SeriesType.BarSegmented

Related Properties

- Series.Background (background color)
- Series.GaugeBorderShape
- Series.Box
- Series.Type (Bar or BarSegmented)
- Chart.ShadingEffectMode

Digital Readout
This gauge type can be used to display element values using a digital readout style.

Supported Features

- Digital readout font styles include Normal, Italic, Bold, and Bold Italic.
- The XAxis Tick Label is used along side of the readout.
- The axis tick and digital labels can be aligned to right or left.

Related Properties

- Series.Background (background color)
- Series.GaugeBorderShape
- Series.Box
- Series.Type (Bar or BarSegmented)
- Element.SmartLabel.Font (Used to specify digital readout font style)

IndicatorLights
These indicator LEDs can be used to convey the status of an element. If the element's value is < 0, the element's LED is grayed out, if the value is above zero, the element's color is used.

SmartPalette
Using smartPalettes with this type, the gray/colored feature based on value is deactivated. The smart palette is used to determine the element's color instead.

The size of the LEDs on a particular chart can be specified with Chart.DefaultElement.Marker.Size, however, by default the LED sizes are determined dynamically. The actual element markers however can be drawn separately on top of the LEDs when Element.ForceMarker is true as shown in sample Gallery/h018.aspx

Supported Features

- All bubble ShadingEffectModes.
- The XAxis Tick and Element's Label is used along side of the LEDs.
- The tick and element labels can be aligned to right, left, or center (element label only).
- Markers can be drawn on top of the LEDs.

Related Properties

- Series.Background (background color)
- Series.GaugeBorderShape
- Series.Box

- Chart.ShadingEffectMode
- Chart.DefaultElement.Marker.Size

Horizontal & Vertical (Includes Thermometers)
The horizontal and vertical gauge types are linear bar gauges with the YAxis but no XAxis. The y axis applies to both horizontal and vertical the same way, when the type is horizontal, the YAxis is the horizontal one and with vertical, the YAxis is vertical.

Normal & Thermometer styles
To use the thermometer variation use the following setting:

```
[C#]
Chart.DefaultSeries.GaugeLinearStyle = GaugeLinearStyle.Thermometer;

[Visual Basic]
Chart.DefaultSeries.GaugeLinearStyle = GaugeLinearStyle.Thermometer
```

Bar Widths
By default the element widths snap to the size of the gauge's mini chart area, however, the XAxis.SpacingPercentage or XAxis.StaticColumnWidth settings will be used if specified.

Supported Features

- All axis scale types except for stacked scales.
- Multiple elements for bars and thermometers.
- Supports axis markers.
- Thermometer GaugeLinearStyle
- All bar shading effect modes for thermometers and bars.
- Supports Columns BarSegmented and Marker Series Types.
  *Markers not available on thermometers.
- SubValues
- Virtually all axis features of combo types are available.
- ImageBarTemplate support

Related Properties

- YAxis.Line (border line properties)
- YAxis members
- Series.Background (background color)
- Series.GaugeBorderShape
- Series.GaugeLinearStyle
- Series.Box
- Chart.XAxis.SpacingPercentage
- Chart.XAxis.StaticColumnWidth
- Chart.ShadingEffectMode

Please see the Features > Gauge Type Tweaks section in samples for examples using gauge types.

1.2 Organizational Charts

Introduction

Organizational charts can be created using .netCHARTING by getting data from live databases or by adding it manually in code. Organizational nodes can be decorated and styled in many ways by using the element annotation properties. Using the label markup language; custom layout, images and other content can be organized inside each node. Rich interactivity features are provided to enable organizational chart drilldown, node expansion, and AJAX scrolling for large datasets. Other useful navigation features include drilldown breadcrumbs and indicators to accent the node appearance.

About Organizational Charts

As with other charts, each node in an organizational chart is represented by an element object. The element's annotation is used to define node styling and content. Annotation label text can be specified to define the node content, but if not set, the element name will be shown inside the annotations.

An organizational chart has a single element at the top of the hierarchy. Every other element will have either ParentID or Parent properties set to identify its parent element. All elements can be in a single series, however, the **Series.Line** property is used to style the connections between elements so multiple series can be used to group elements that use the same connecting line style.

Elements with no names or annotation labels can be used as invisible placeholders to create more complex layouts. When used, a straight line is drawn through the location of the element as demonstrated in sample Chart Type Gallery/Organizational/M10.

Data Acquisition

To retrieve an organizational chart from a database table, each row must have an ID and a ParentID. This shows an example of how an organizational chart gets its data. It also adds several attributes to each element from other table columns such as Name, Office, Email, etc..

C#

```
DataEngine de = new DataEngine(ConfigurationSettings.AppSettings
["DNCConnectionString"]);
de.SqlStatement = @"SELECT * FROM Employees";
de.DataFields =
"InstanceID=ID,InstanceParentID=PID,Name=Name,office,Department,Email,Phone,Picture";
Chart.SeriesCollection.Add(de.GetSeries());
```

VB.NET

```
Dim de As New DataEngine(ConfigurationSettings.AppSettings("DNCConnectionString"))
de.SqlStatement = "SELECT * FROM Employees"
de.DataFields =
"InstanceID=ID,InstanceParentID=PID,Name=Name,office,Department,Email,Phone,Picture"
Chart.SeriesCollection.Add(de.GetSeries())
```

Custom attributes can be shown in the annotation (node) labels using tokens such as %Office %Email and so on. If iterating the elements in code, attributes can be retrieved using code such as:

string value = element.CustomAttributes["office"];

💡 The attribute keys are case sensitive so if the above code does not work, use the debugger to evaluate the key values of the element.CustomAttributes collection.

For more information, see the **Custom Attributes Tutorial ('Custom Data Attributes' in the on-line documentation).**

Organizational SplitBy

Organizational nodes can be grouped into different series automatically based on the value of a specified database column. This code acquires the same data as the above, but it groups the nodes by 'Department'. Note that splitby is specified in the DataFields property.

C#

```
DataEngine de = new DataEngine(ConfigurationSettings.AppSettings["DNCConnectionString"]);
de.SqlStatement = @"SELECT * FROM Employees";
de.DataFields =
"InstanceID=ID,InstanceParentID=PID,Name=Name,office,Email,Phone,Picture,SplitBy=Departme
Chart.SeriesCollection.Add(de.GetSeries());
```

VB.NET

```
Dim de As New DataEngine(ConfigurationSettings.AppSettings("DNCConnectionString"))
de.SqlStatement = "SELECT * FROM Employees"
de.DataFields =
"InstanceID=ID,InstanceParentID=PID,Name=Name,office,Email,Phone,Picture,SplitBy=Departme
Chart.SeriesCollection.Add(de.GetSeries())
```

Samples OrgGroups & OrgGroups2 demonstrate this feature.

Adding Data Manually

When adding organizational elements manually, either the element InstanceParentID or Parent properties must be set. The InstanceParentID property takes a numeric id matching the parent element's InstanceID property. Alternatively, the parent element object can be specified for each child element's Parent property.

This sample demonstrates adding nodes manually without IDs. It is useful because it makes adding data with code easier.

C#

```
Element p1 = new Element("Margret Swanson");
Element vp1 = new Element("Mark Hudson");
Element vp2 = new Element("Chris Lysek");
vp1.Parent = p1;
vp2.Parent = p1;
Series s = new Series("", p1, vp1, vp2);
```

VB.NET

```
Dim p1 As New Element("Margret Swanson")
Dim vp1 As New Element("Mark Hudson")
Dim vp2 As New Element("Chris Lysek")
vp1.Parent = p1
vp2.Parent = p1
Dim s As New Series("", p1, vp1, vp2)
```

The following code snippet creates the same chart but using InstanceID and InstanceParentID properties to define the hirerarchy.

C#

```
Element p1 = new Element("Margret Swanson");
Element vp1 = new Element("Mark Hudson");
Element vp2 = new Element("Chris Lysek");
p1.InstanceID = 1;
vp1.InstanceID = 2;
vp3.InstanceID = 3;
vp1.InstanceParentID = 1;
vp2.InstanceParentID = 1;
Series s = new Series("", p1, vp1, vp2);
```

VB.NET

```
Dim p1 As New Element("Margret Swanson")
Dim vp1 As New Element("Mark Hudson")
Dim vp2 As New Element("Chris Lysek")
p1.InstanceID = 1
vp1.InstanceID = 2
vp3.InstanceID = 3
```

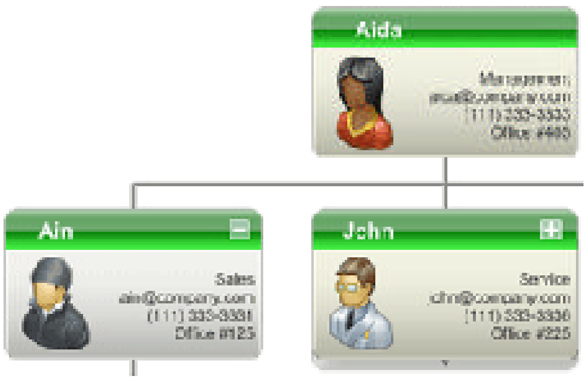
```
vp1.InstanceParentID = 1
vp2.InstanceParentID = 1
Dim s As New Series("", p1, vp1, vp2)
```

Styling Organizational Charts

The chart area padding controls the spacing between nodes and chart edges. The Chart.DefaultSeries.Line property can be used to define the default connecting line styling between nodes.

To specify a default annotation style, instantiate a new annotation object for the Chart.DefaultElement.Annotation property and modify those properties. They will automatically be used by all the annotations on the chart, unless otherwise specified.

Using the .netCHARTING markup language provides a way to control the layout of the text and images inside the annotations. The **markup language tutorial ('Label Layout Markup' in the on-line documentation)** explains how to use this markup. The text and markup can be specified with the Annotation.Label.Text property.



Annotations have two content areas. When a string is set with Annotation.HeaderLabel.Text, that content is placed in a header which allows separate background styling.

- Annotations try to maintain a certain width to height ratio by default which may result in odd rectangles. However, setting Annotation.DynamicSize = false will make the annotation size based on the content.
- Annotations support a sizing option where only the height or width is provided and the other side sized automatically. This can be done by specifying a size with a value of 0 for the side that should size automatically:
  For example: Annotation.Size = new Size(100,0)
  Width will be 100 and the height will be automatically determined.

See also:

- **Annotation Class** - annotation properties and usage.
- **Box Tutorial (Styling) ('Box Tutorial' in the on-line documentation)** - basics of box styling.
- **Label Tutorial ('Using labels' in the on-line documentation)** - Token and attribute usage.

This sample code demonstrates styling the default annotation of a chart:

C#

```
Chart.DefaultElement.Annotation = new Annotation();
Chart.DefaultElement.Annotation.Padding = 5;
Chart.DefaultElement.Annotation.Size = new Size(75, 30);
Chart.DefaultElement.Annotation.Background.ShadingEffectMode =
ShadingEffectMode.Background2;
```

VB.NET

```
Chart.DefaultElement.Annotation = New Annotation()
Chart.DefaultElement.Annotation.Padding = 5
Chart.DefaultElement.Annotation.Size = New Size(75, 30)
Chart.DefaultElement.Annotation.Background.ShadingEffectMode =
ShadingEffectMode.Background2
```

Iterating elements to specify styling individually is done in the same way except element.Annotation would be used for each object instead of Chart.DefaultElement.Annotation. However, the annotation must be instantiated for each element before accessing its properties.

Organizational Chart Navigation

Organizational chart nodes can be fairly large and with many of them on a chart, it is difficult to view them all at the same time. When a large number of nodes is loaded into a chart, the chart will try to render them on a larger image and scale it down to fit the chart area. This can result in nodes showing up very small.

A number of methods and features are provided to facilitate navigating large and complex organizational chars in a simple and intuitive way. This functionality is implemented with a lower level API that is more flexible and allows creating highly customized navigation experiences.

ASP.NET AJAX Scrolling

The most easy to use feature to facilitate navigating large organizational charts, is by using the AJAX Zoomer. Simply use `Chart.Zoomer.Enabled = true` to enable this feature. This feature behaves differently with organizational charts. There are no context menu options for zooming in or out. It allows the entire set to be loaded into a chart, and if it generates an image too large for the chart area, drag scrolling and scrollbars become available.

Sample: Feature Gallery/Chart Type Tweaks/OrgExpansionDB uses this feature.

Organizational Node DrillDown

This concept provides the ability to click on a node, and navigate into an organizational hierarchy subset without displaying all the nodes at the same time. The **Series.Trim** method provides this ability, and works by returning a subset of nodes based on the root ID of an element, and the number of subsequent levels of child nodes to include in the result. Annotation hotspots are used with a URL to postback the ID of the clicked annotation. Then the page must get this ID and pass it to the trim method in order to display the correct subset of nodes.

Sample: Feature Gallery/Chart Type Tweaks/OrgDrillDownDB demonstrates organizational drilldown with a live database.

Drilldown Breadcrumbs

Drilldown breadcrumbs provide a way to keep track of the location within an organizational hierarchy when using drilldown features. Breadcrumbs are common and look something like this:

Animals / Feline / Cheetah

Samples such as OrgDrilldown demonstreate how to use this feature in a drilldown context. The sample OrgBreadcrumbs uses this method to get an array of breadcrumb strings, and display them as annotations.

Consider the following code creating a breadcrumb path

```
string path = series.GetElementBreadcrumbs(elementID, elementCrumb, lastElementCrumb, separator);
```

The above usage specifies the elementID of the last element or 'Cheetah' in the above example. The elementCrumb is a label template that can use element tokens such as %Name. This would be applied to the underlined crumbs 'Animals' and 'Feline'. Common usage would require these nodes are linked. Using this string as the elementCrumb template will create links for those crumbs:

```
string elementCrumb = "<block url='?id=%id' fColor='blue' fStyle='underline'>%Name";
```

The lastElementCrumb can be just the name with no links or styling to indicate it is not clickable:

```
string lastElementCrumb = "<block>%Name";
```

The separator also uses the block tag to terminate any block tags previously added by the elementCrumb template.

```
string separator = " <block>/ ";
```

The resulting string can be added anywhere on the chart, like in the chart title box, chart area label, or in a floating annotation amongst others.

Organizational Node Attributes

The Series.Trim method also analyzes the dataset and optionally populates element attributes that indicate the level in the hierarchy and node type such as whether elements have parent or child nodes not shown in the resulting subset of nodes.

Sample *OrgLevels* uses the trim method to not trim the nodes but just to analize the dataset and populate organizational level attributes for each element. Then they are colored based on the hierarchical level.

This table describes the attributes added by the Trim method.

Attribute Name	Values and Descriptions
NodeType	**EndNodeWithChildren** - Indicates this element's children are trimmed from the result but child elements exist. **RootWithParents** - Indicates it is the root element in the result but parent nodes exist and were trimmed away. **Root** - The root of the result and dataset. **NoChildren** - Node has no children in the result or in the dataset.
OrganizationalLevel	A number ranging from **1 to n** levels in the data set indicating the level of each particular element.

Organizational Indicators

The NodeType attributes can also be used to style the annotations by drawing an accent above or below the node. For example, if the element's NodeType attribute value is 'RootWithParents', an accent is drawn above the annotation, and if the value is 'EndNodeWithChildren', it is drawn it below it. This is demonstrated in sample OrgIndicators where these attributes are set manually to display the accent, and in sample OrgDBDrilldown where they are supplied by the trim method.

Organizational Node Expansion

This feature allows the user to click on any node to expand or collapse its children, thereby exploring the hirerarchy while displaying only the expanded nodes. This is similar to drilldown, however, it requires that a comma delimited list of expanded node IDs is maintained by the page code or application. Another overload of the Trim method can take this list of IDs and return only the nodes that should be visible.

Sample: Feature Gallery/Chart Type Tweaks/OrgExpansionDB demonstrates implementing this feature using a live database.

Combining Navigational Features

When using organizational expansion, it is useful to enable the AJAX zoomer so that when the expanded nodes create a large image, scrolling is enabled to help navigate around (Sample: OrgExpansionDB).

Because organizational charts with many elements can result is huge images, making the nodes smaller with less information can help fit much more of a hierarchy onto a chart. The additional information can be displayed in a tooltip. With label markup support for tooltips, the same string and layout used for annotations can be used in tooltips (Sample: OrgDBRichTooltips).

1.3 Financial Charts

Financial charts may create some confusion, never the less, making them works on the same principal as any other chart. There is, however, a difference in acquiring data.

Data Acquisition

First let's get data from a database which already has open, close, high, and low values using the DataEngine. In order to populate OCHL (open, close, high, low) element properties the DataEngine.GetFinancialSeries() method is used instead of GetSeris().

```
[C#]
DataEngine de = new DataEngine(myConnectionString);
de.QueryString = "SELECT timeCol ,openCol, closeCol, highCol, lowCol, timeCol FROM
de.DataFields = "open=openCol,close=closeCol,high=highCol,low=lowCol,xValue=timeCo

SeriesCollection stockSC = de.GetFinancialSeries();
stockSC[0].Type = SeriesTypeFinancial.CandleStick;
Chart.SeriesCollection.Add(stockSC);
```

```
[Visual Basic]
Dim de As New DataEngine(myConnectionString)
de.QueryString = "SELECT timeCol ,openCol, closeCol, highCol, lowCol, timeCol FROM
de.DataFields = "open=openCol,close=closeCol,high=highCol,low=lowCol,xValue=timeCo

Dim stockSC As SeriesCollection = de.GetFinancialSeries()
stockSC(0).Type = SeriesTypeFinancial.CandleStick
Chart.SeriesCollection.Add(stockSC)
```

For the next example, we'll get our data from a database that has only the price column where the entries represent a price snapshot for every hour of the day over several months. We'll want to create the same type of chart as the above but obviously the data is not formatted in this manner. Fortunately the DataEngine is capable of grouping the data properly and calculating the high, low, open, and close prices. The DataEngine will know that this is the result we're going for when we use GetDataFinancial() instead of GetData().

```
[C#]
DataEngine de = new DataEngine(myConnectionString);
de.DateGrouping = TimeInterval.Days;
de.QueryString = "SELECT timeCol, price FROM stockPrice";

SeriesCollection stockSC = de.GetFinancialSeries();
stockSC.Type = SeriesTypeFinancial.CandleStick;
Chart.SeriesCollection.Add(stockSC);
```

```
[Visual Basic]
Dim de As New DataEngine(myConnectionString)
de.DateGrouping = TimeInterval.Days
de.QueryString = "SELECT timeCol, price FROM stockPrice"

Dim stockSC As SeriesCollection = de.GetFinancialSeries()
stockSC.Type = SeriesTypeFinancial.CandleStick
Chart.SeriesCollection.Add(stockSC)
```

For more information on data acquisition, see: **Data Tutorials (on-line documentation)**.

Setting up Charts

ChartType.Financial
Please note that this chart type should not be used. It is there for backward compatibility. Using this chart type, a single series can be used where the element price properties are set as well as the volume value, however, the functionality will be limited. Instead use ChartType.Combo. The following example will show how to create the volume chart area using the newer more capable method.

```
[C#]
DataEngine de = new DataEngine(myConnectionString);
de.QueryString = "SELECT timeCol, volume, FROM stockVolume";
de.DateGrouping = TimeInterval.Day;
SeriesCollection volSC = de.GetData();
// Create a new chart area, add the volume data and add the area to the chart.
ChartArea vca = new ChartArea(volSC);
```

```
vca.HeightPercentage = 30;
Chart.ExtraChartAreas.Add(vca);

[Visual Basic]
Dim de As New DataEngine(myConnectionString)
de.QueryString = "SELECT timeCol, volume, FROM stockVolume"
de.DateGrouping = TimeInterval.Day
Dim volSC As SeriesCollection = de.GetData()
// Create a new chart area, add the volume data and add the area to the chart.
Dim vca As New ChartArea(volSC)
vca.HeightPercentage = 30
Chart.ExtraChartAreas.Add(vca)
```

For more information on using and manipulating ChartAreas, see: **Multiple Chart Areas (on-line documentation).**

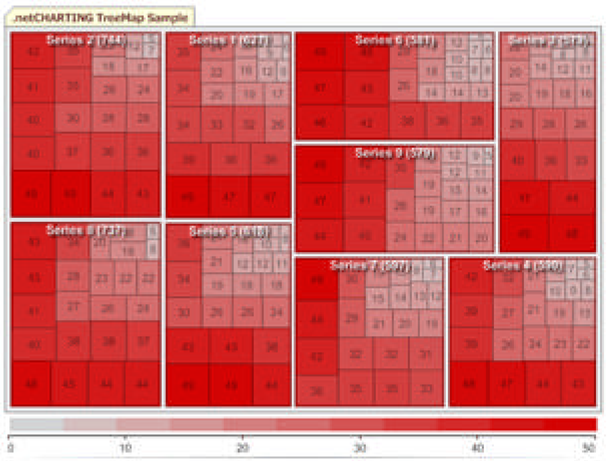
1.4 TreeMap Charts

Introduction

TreeMap charts are used to display hierarchial tree structure data with nested rectangles. TreeMap charts can be created manually or by using live data.

Features

- Series Labeling - A label can be drawn inside the series rectangle, or a header can be applied to the series rectangle with all header related features supported.
- Element Labelig - Element labels can dynamically adjust to fit better, and many common label properties are supported.
- Color Ranges - Elements can be colored based on their values using a specified range of colors.
- Color Range Swatches - The legend can display the values and their respective colors for reference.
- Shading Effects - The element rectangles can use all the same shading effects as regular bar and column charts.



A TreeMap Chart

To use the treemap chart type, the following code can be used.

```
[C#]
Chart.Type = ChartType.TreeMap;

[Visual Basic]
Chart.Type = ChartType.TreeMap
```

Series Spacing

With a single series on the chart, the spacing is ignored. With multiple series however, the ChartArea.Padding property specifies the space between the series boxes and chart area edges in pixels. The following code will create a 5 pixel gap between the series and chart area.

```
[C#]
Chart.ChartArea.Padding = 5;

[Visual Basic]
```

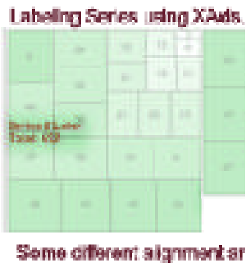
```
Chart.ChartArea.Padding = 5
```

Series Labeling

The series rectangle can be labeled in two ways on a TreeMap chart. A label can be drawn inside the series rectangle or the series rectangle can utilize a box header label.

Series.Box.Label

The first method involves using the Series.Box.Label property which will be rendered inside the series bounds. Tokens in this label's text will be evaluated for each series and will apply to all of them.



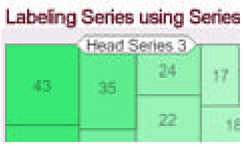
```
[C#]
Chart.DefaultSeries.Box.Label.Text = "%Name %YSum";
Chart.DefaultSeries.Box.Label.LineAlignment = StringAlignment.Far;

[Visual Basic]
Chart.DefaultSeries.Box.Label.Text = "%Name %YSum"
Chart.DefaultSeries.Box.Label.LineAlignment = StringAlignment.Far
```

Series.Box.Header.Label

The second method involves using the series box header label. In effect, the series rectangle is treated like the box content and the header is used to label it. The box in this context does not support corner styling or shadows.

The box header can place the series label outside the series bounds and can take up more space, however, it helps prevent the label from overlapping the elements and possibly hiding some data. Using *Header.VerticalAlignment* = *EdgeAlignment.Edge* is useful in this situation, as it does not intrude too much onto the series rectangle and minimizes the series label use of space.

See the **Box Tutorial (on-line documentation)** for more information about manipulating the box header. It offers a robust array of options that affect the way the header is drawn.



```
[C#]
Chart.DefaultSeries.Box.Header.Label = "%Name %YSum";

[Visual Basic]
Chart.DefaultSeries.Box.Header.Label = "%Name %YSum"
```

Series tokens can be used with both labeling methods and will be evaluated with the series information. See **Tokens Reference ('Tokens' in the on-line documentation)** for a full list of available tokens.

It is also possible to use both types of labels simultaneously. For example, one label can be used to describe the series name and the other to display values or totals.

Element Labeling

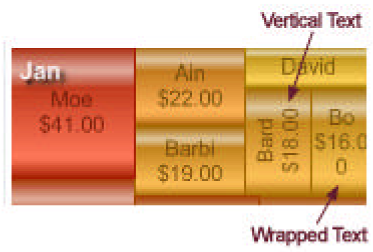
Element labels behave much like any other element labels using all the same applicable properties. If there is not enough space to fit the label properly, the chart will attempt to draw the label vertically, or wrap the text so it can be seen.

The following SmartLabel properties can be used.

- AbsolutePosition
- Alignment
- AutoWrap
- DynamicDisplay
- ForceVertical
- Hotspot

- LineAlignment
- Padding
- Text
- Truncation
- And common visual enhancements: Color/Font/GlowColor/OutlineColor/Shadow/Type

```
[C#]
Chart.DefaultElement.ShowValue = true;
Chart.DefaultElement.SmartLabel.Text = "%Name %YValue";
Chart.DefaultElement.SmartLabel.DynamicDisplay = false;

[Visual Basic]
Chart.DefaultElement.ShowValue = true
Chart.DefaultElement.SmartLabel.Text = "%Name %YValue"
Chart.DefaultElement.SmartLabel.DynamicDisplay = false
```



Because SmartLabel overwrites the base label's Alignment property of type StringAlignment with an Alignment property of type LabelAlignment, you must cast the smart label to a label object and then set the base alignment property like so:

((Label)Chart.DefaultElement.SmartLabel).Alignment = StringAlignment.Near;

Coloring Elements

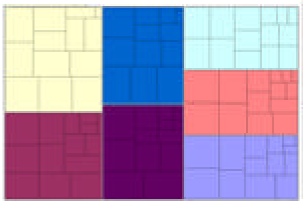
By default, all elements of a series will have the same color. This coloring is the default behavior for all chart types, and TreeMap uses the same generic algorithms. (See Figure 1)

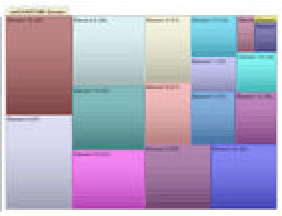
For this chart type it can be useful to color each element with a different color. This is done by setting either Series.Palette or Series.PaletteName properties. (See Figure 2).

A much more useful coloring method is to use color ranges to automatically color elements based on their values. This allows an easy way to quickly understand the values of elements based on their colors. (See Figure 3).

When using color ranges, a legend color swatch may be used to allow the user to match element colors with their approximate values.

The **Color Ranges tutorial ('Smart Palettes' in the on-line documentation)** describes how to use color ranges to apply to elements and how to create the legend color swatches.


Default coloring


Color per element.

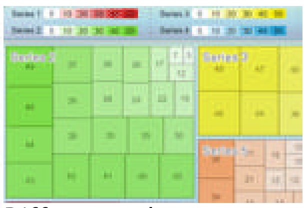

Different color ranges per series.

Fig 1 Fig 2 Fig 3

1.5 Surface Type

Introduction

The surface chart type enables visualizing 3d surface data. It provides a number of different visualizations and properties to tune them. The surfaces can be colorized based on z values utilizing smartPalettes. Any conventional series can be converted to surface data and charted.

The initial settings for a surface chart can look like so:

```
[C#]
chart1.Type = ChartType.Surface;
chart1.Use3D = true;
chart1.LegendBox.Visible = false;
chart1.View3D.AngleOfTilt = 45;
chart1.View3D.AngleOfRotation = 45;

[Visual Basic]
chart1.Type = ChartType.Surface
chart1.Use3D = True
chart1.LegendBox.Visible = False
chart1.View3D.AngleOfTilt = 45
chart1.View3D.AngleOfRotation = 45
```

QuickStart Tip: The sample **SurfaceDesigner (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceDesigner)** was designed to demonstrate all the related settings and provide a way to design a surface chart without writing code. The sample generates the code necessary to create those visuals.

Data Population

The surface chart type only supports a single data set at this time. There are two ways to populate series with surface data. One way is to add elements with the correct, x, y, and z values like with any other chart type. The other way is to supply arrays of these values to the Series.FromSurfaceData() method.

Samples:

SurfaceFromElements (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceFromElements) - Demonstrates populating data with elements

Surface__ - Nearly all others populate data with arrays

Performance Tip: Using arrays to populate the chart will reduce overhead and result in better performance over using multiple elements to define the data.

Surface Data From regular series.

Category Series to Surface Data

A category series is one where elements use names (strings) as x axis values. The method SeriesCollection.GetSurfaceData() can be used to convert a SeriesCollection of category series for surface charts. When converting, the element names are mapped to the x axis, series names are mapped to the y axis and y values become the z component of surface data.

SeriesCollection.GetSurfaceData(out List<string> xLabels, out List<string> yLabels)

The above method will return a series with surface data and populate the label lists with element and series names that can be used on the axes.

The following code demonstrates how the lists can be used on axes.

```
[C#]
List<string>  xLabels = new List<string>();
List<string>  yLabels = new List<string>();
Series s = GetSurfaceData(out xLabels, out yLabels);
 Chart.XAxis.ClearValues = true;
 for (int i = 0; i < xLabels.Count; i++)
 {
    Chart.XAxis.ExtraTicks.Add(new AxisTick(i, xLabels[i]));
 }
 Chart.YAxis.ClearValues = true;
 for (int i = 0; i < yLabels.Count; i++)
 {
    Chart.YAxis.ExtraTicks.Add(new AxisTick(i, yLabels[i]));
 }

[Visual Basic]
Dim xLabels As List(Of String) = New List(Of String)()
Dim yLabels As List(Of String) = New List(Of String)()
Dim s As Series = GetSurfaceData(xLabels, yLabels)
Chart.XAxis.ClearValues = True
For i As Integer = 0 To xLabels.Count - 1
  Chart.XAxis.ExtraTicks.Add(New AxisTick(i, xLabels(i)))
Next i
Chart.YAxis.ClearValues = True
For i As Integer = 0 To yLabels.Count - 1
  Chart.YAxis.ExtraTicks.Add(New AxisTick(i, yLabels(i)))
Next i
```

Alternatively, by passing the chart axes to this method, they will be populated automatically.

SeriesCollection.GetSurfaceData(Axis xAxis, Axis yAxis)

Sample:
SurfaceFromCategoryData (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceFromCategoryData) - Demonstrates converting category data into surface data.

Time Series to Surface Data

The chart provides an easy way to create surface data from any regular time based series by grouping them based on a time interval. When using the Series.GetSurfaceData method on a populated series, an x axis date grouping interval can be specified and the method will return a series containing surface data along with an array of dates to represent the x axis tick values. This enables any series with elements that use XDateTime property to be plotted on a surface chart.

Samples:
SurfaceTimeDB (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceTimeDB) - Demonstrates populating a surface chart with live date based data.
SurfaceDateGrouping (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceDateGrouping) - Demonstrates populating a surface chart with live date based data offering an option to select the date grouping interval.

3D View Angles & Lighting

The properties under Chart.View3D provide control over the view angles and lighting. The chart can be rotated between 0 and 180 degrees and tilted between 0-90 degrees. Lighting emulates a light source located at the top of the z axis where the x and y axis minimum values are located.

```
[C#]
chart1.View3D.AngleOfTilt = 45;
chart1.View3D.AngleOfRotation = 45;
chart1.View3D.EnableLighting = true;

[Visual Basic]
chart1.View3D.AngleOfTilt = 45
chart1.View3D.AngleOfRotation = 45
chart1.View3D.EnableLighting = true
```

Chart.Use3D can be set to true to make the surface chart draw in 3D. Each series has its own Use3D setting as well. This enables mixing 2D and 3D visuals on the same chart.

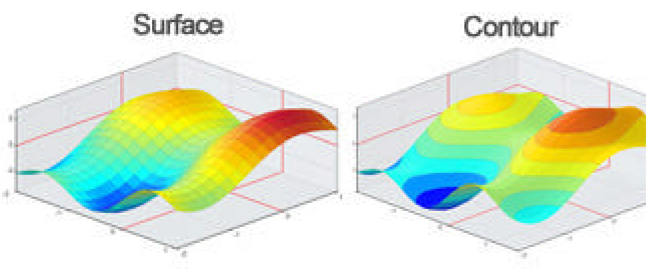
Series Data Visuals

The surface type offers a flexible API with an endless array of setting combinations that control the visualization.

Four base surface types

Surface charts utilize the SeriesTypeSurface enum members to define the series type when used with the Series.Type property. The following four surface types are available.

```
[C#]
series1.Type = SeriesTypeSurface.Contour;


[Visual Basic]
series1.Type = SeriesTypeSurface.Contour
```



Surface: The surface type draws a 3D surface or a 2D heatmap. It supports the Series.InterpolationFillFactor property which accepts an integer specifying the number of different x and y color sections to split a single tile into. Surface supports 2D and 3D as well as 2D in 3D mode.

Contour: The contour type draws a 2D or 3D surface partitioning the plane into a number of contours defined by Series.ContourCount. Contour supports 2D, 3D, and 2D in 3D mode.

SurfaceX & SurfaceY: These types are similar to surface except they connect points over only the x or y axis and no surface is filled unless the DrawToFloor option is used. When used with the DrawToFloor option, they resemble AreaLines.

Series layering

Only data from the first series is plotted on a surface chart, however, multiple series can be added with different settings to create more complex visuals. The additional series will render using the first series' data but with their own styling and type settings.

This code can be used to add a 2D contour visualization to a 3D surface chart.

```
[C#]
Series s1 = new Series();
s1.Type = SeriesTypeSurface.Contour;
s1.Use3D = false;
Chart.SeriesCollection.Add(s1);

[Visual Basic]
Series s1 = new Series()
s1.Type = SeriesTypeSurface.Contour
s1.Use3D = false
Chart.SeriesCollection.Add(s1)
```

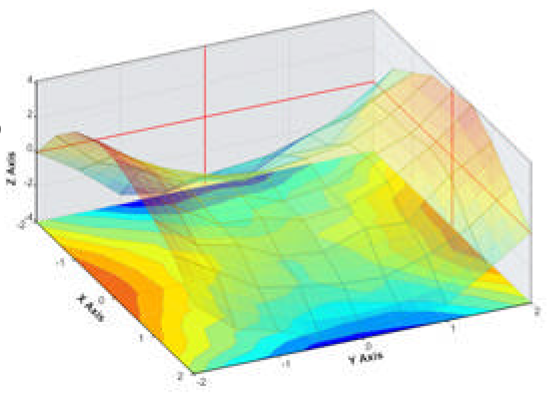


Chart with multiple series.

Series Type Variations/Options

The following properties are available per surface series types:

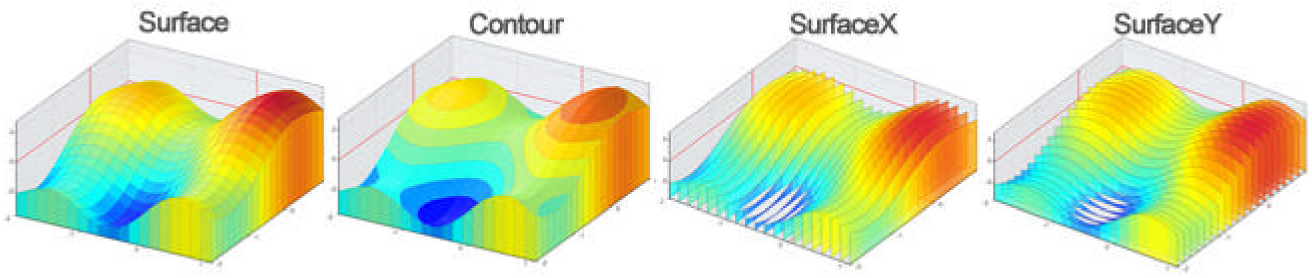
	ContourCount	InterpolationFillFactor	DrawToFloor	Use3D
Surface	☐	✓	✓	✓
Contour	✓	☐	✓	✓
SurfaceX	☐	☐	✓	✓
SurfaceY				

Chart Types

	☐	☐	✓	✓

Series.ContourCount	The contourCount setting specifies the number of contours to divide the surface into when used with SeriesTypeSurface.Contour. The z value range is divided into this number of contours evenly.
Series.InterpolationFillFactor	This property is used with SeriesTypeSurface.Surface. When the setting is higher than 1, the tiles connecting the surface points are split and draws multiple interpolated colors. A setting of 2 divides the tile into 4 different color sections for example (2x2).
DefaultElement.DrawToFloow	When this setting is set to true, 3D surface and contour series fill the outside bounds of the surface connecting them to the chart area floor. It can help enhance visualizing the outside edges of a surface shape. This setting is not applicable with 2D series. When used with SurfaceX or SurfaceY, the visual resembles AreaLines. See image below.
Series.Use3D	This setting indicates whether a series is drawn in 2D or 3D.

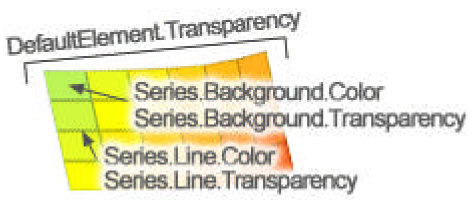
Types with DrawToFloor Enabled



Coloring Surface Series

By default, a surface or contour series will use a single color defined by the chart palette or default element of the series. It is much more useful to color surface charts with dynamic colors based on the z value. The smartPalettes enables this functionality. See the **SmartPalette tutorial ('Smart Palettes' in the on-line documentation)** for more info and options.

Legends
Generally, legend boxes are not useful with surface charts so it's best to disable them. Instead color swatches can be used which can be added to any label on the chart.



Styling API Properties

Surfaces are drawn by connecting elements, then filling sections and drawing the outlines. When dynamic colors are used, static colors can overwrite the styling of lines and fills independently. The fill and line transparencies can be set independently as well. Lines can also use other styling specified with Series.Line properties such as DashStyle. Series.Background is used to control the fill through color and transparency settings.

Following are some examples of color and transparency tasks and code necessary to achieve them.

Creating a smart palette from data and a list of colors:

```
[C#]
Chart.SmartPalette = series1.GetSmartPalette(ElementValue.ZValue,Color.Green,Color

[Visual Basic]
Chart.SmartPalette = series1.GetSmartPalette(ElementValue.ZValue,Color.Green,Color
```

Apply uniform transparency for the surface:

```
[C#]
series1.DefaultElement.Transparency = 50;

[Visual Basic]
series1.DefaultElement.Transparency = 50
```

Use a solid color for the fill:

```
[C#]
series1.Background.Color = Color.White;

[Visual Basic]
series1.Background.Color = Color.White
```

Tip: Use line transparency of about 70% on solid surface and contour series to achieve a smooth look.

Palette Transparency Features

The SmartPalette also supports transparent colors enabling sections of a surface to be translucent or transparent. A number of samples are available to demonstrate these features.

This sample demonstrates using a palette that includes a transparent color. Color transparency is interpolated between it and solid colors within the palette resulting in a smooth visual.

Sample: **Gallery P043 (http://www.dotnetcharting.com/gallery/view.aspx?id=Gallery/P043)**

When the smart palette does not encompass a value, those sections will not be rendered.

Sample: **SurfaceViewsTransparent (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceViewsTransparent)**

This functionality can be used to render a heatmap on top of an image.

Sample: SurfaceImageHeatmap (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceImageHeatmap)

Axes and surfaces

The axes of a 3D chart support most of the same features as combo charts. Different scales like logarithmic, inverted, scale breaks, etc. are all supported on surface charts. Surface charts support only numeric axis scales at this time however, time and category scales can be easily emulated as described below and in the Data Population section above. 2D surface charts support only numeric scale also, but they support all other numeric axis features like axisMarkers and range ticks because the core combo chart rendering is used for the chart area and axes. Some features like axis markers are not supported in the 3D view but custom ticks can be used to simulate single value axis markers by changing the tick's gridline properties.

Time / Category Axis Scales

Surface charts support only numeric axis scales at this time, however, other axis features allow replacing the axis labels with strings which enables simulating any type of axis necessary. In order to use string axis labels relating to surface data, create an array holding the text labels and use the array index value instead of the string to specify the x values for the data. Then iterate the string array adding axis ticks with the array index as the value and corresponding string as the tick label text. This will replace the numeric ticks on the axis with the correct labels.

Sample: **SurfaceAxisLabels (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceAxisLabels)** - Demonstrates category scale type axes.

The method Series.GetSurfaceData() outputs a list of DateTime objects that can be mapped to the axis ticks and is demonstrated in the following samples:

Samples:
SurfaceTimeDB (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceTimeDB) - Demonstrates populating a surface chart with live date based data.
SurfaceDateGrouping (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceDateGrouping) - Demonstrates populating a surface chart with live date based data offering an option to select the date grouping interval.

For more information, see the Data Population section above.

Axis Scales

By default, axes with a Scale.Normal setting behave slightly different than they do with combo charts. With surface charts, the axis range snaps to data to remove any gaps whereas with combo charts the axis range maximum values are rounded up to the nearest interval.

The supported scales are:

- Normal
- Range
- Logarithmic

Z Axis

The z axis of a surface chart behaves the same as the x and y axes. It can be accessed through the Series.ZAxis and Chart.ZAxis properties.

ScaleBreaks

ScaleBreaks are supported, they are drawn as a line with a zigzag in the middle. This line styling can be controlled through Axis.ScaleBreakLine.

Sample: SurfaceAxisScaleBreak (http://www.dotnetcharting.com/gallery/view.aspx?id=SurfaceAxisScaleBreak)

1.6 ErrorBars and other SubValues

Introduction

.netCHARTING has an advanced system for handing error bars and other sub value types that are used in many statistics charts. Element and series classes have properties that quickly set sub values, however, there is a more powerful system underneath that controls and helps manage these sub elements.

SubValue Class Constructors

Each sub value is represented by a class that is instantiated by a number of sub value constructors that define its value.

SubValue offsets and percent values will be based on the element's yValue.

Constructor	Type	Description
FromHighLowValue (highValue,lowValue)	Range	Creates a range sub value from specified high and low values.
FromPlusMinusOffset (plusOffset,minusOffset)	Relative Range	Creates a range sub value from specified high and low offsets.
FromPlusMinusOffset (o)	Relative Range	Creates a range sub value from specified offset above and blow the element's yValue.
FromPlusMinusPercent (hPercent,lPercent)	Relative Range	Creates a range sub value from specified percentages above and below the element's yValue.
FromPlusMinusPercent (percent)	Relative Range	Creates a range sub value from the specified percentage above and below the element's yValue.
	Relative Single	Creates a subValue from the specified offset. (-2) will be

FromOffset(o)	Value	below the element's yValue and (2) will be above.
FromPercent(p)	Relative Single Value	Creates a subValue from the specified percentage. (-50) will be below the element's yValue and (50) will be above.
FromValue(v)	Single Value	Creates a subValue from the specified value.

Adding SubValues

Each element contains a **SubValueCollection** under the Element.**SubValues** property. Adding a sub value can be done simply be adding it to that collection.

```
[C#]
Element e = new Element();
e.SubValues.Add(SubValue.FromOffset(5));

[Visual Basic]
Dim e As new Element()
e.SubValues.Add(SubValue.FromOffset(5))
```

The element e now contains a subValue which is equivalent to e.YValue + 5.

SubValues Appearance

There are a few options available that determine how a subVlaue will be drawn with an element. These options are specified with the **SubValueType** enumeration.

- SubValueType.ErrorBar
  A traditional error bar stemming from the top of an element.
  NOTE: An idential error bar drawn on top and bottom of the element is achieved when the element values specify a range (YValue and YValueStart is set).

- SubValueType.Marker
  A marker or two markers for range sub values.
- SubValueType.Line
  Similar to error bars but without the vertical line.

This code snippet demonstrates how the appearance properties can be used.

```
[C#]
SubValue sv = SubValue.FromOffset(5);
sv.Type = SubValueType.Marker;
sv.Marker.Type = ElementMarkerType.Circle;
sv.Marker.Color = Color.Red;
SubValue sv2 = SubValue.FromOffset(5);
sv2.Type = SubValueType.Line;
sv2.Line.Color = Color.Black;
sv2.Line.DashStyle = DashStyle.Dash;

[Visual Basic]
Dim sv As SubValue = SubValue.FromOffset(5)
sv.Type = SubValueType.Marker
sv.Marker.Type = ElementMarkerType.Circle
sv.Marker.Color = Color.Red
Dim sv2 As SubValue = SubValue.FromOffset(5)
sv2.Type = SubValueType.Line
sv2.Line.Color = Color.Black
sv2.Line.DashStyle = DashStyle.Dash
```

Default Settings

SubValues can be quickly added and manipulated for all elements within a series or series collection. When a SubValue is added to a series' default element it is also added to all subsequent elements automatically.

This code snippet automatically adds a subValue to all elements on the chart.

```
[C#]
Chart.DefaultSeries.DefaultElement.SubValues.Add(SubValue.FromOffset(5));

[Visual Basic]
Chart.DefaultSeries.DefaultElement.SubValues.Add(SubValue.FromOffset(5))
```

SubValues default appearance can also be specified quickly by using the element's DefaultSubValue properties.

```
[C#]
Chart.DefaultSeries.DefaultElement.DefaultSubValue.Line.DashStyle = DashStyle.Dash

[Visual Basic]
Chart.DefaultSeries.DefaultElement.DefaultSubValue.Line.DashStyle = DashStyle.Dash
```

Limitations

- SubValues are not available with financial series types.
- Not available with stacked scales.
- Time scales don't support error bars.

1.7 Element Markers

Introduction
The SeriesType.Marker series type draws an element marker for each element. The properties of these markers can be accessed through Element.Marker.

Tips:

1. Using Chart.DefaultElement.Marker allows setting all element marker properties simultaneously.
2. When using some series types like line or spline in 2D mode, markers are automatically drawn. They can be turned off by setting Chart.DefaultElement.Marker.Visible = false
3. Markers can always be drawn on any series / chart types by specifying Element.ForceMarker = true.

Element Markers as Shapes
Element Markers can appear in a number of different shapes specified by the ElementMarkerType enumeration.

ElementMarkerType

- Square
- Triangle
- TriangleUpsideDown
- Circle
- Diamond

- ArrowUp
- ArrowDown
- FourPointStar
- FivePointStar
- SizPointStar
- SevenPointStar
- Split (Finance Related)
- ReverseSplit (Finance Related)
- Merger (Finance Related)
- Dividend (Finance Related)
- Spinoff (Finance Related)

Element Markers as Images
Element markers can also use custom images loaded from disk instead of the pre-defined shapes.

```
[C#]
myElement.Marker = new ElementMarker("image.png");

[Visual Basic]
myElement.Marker = New ElementMarker("image.png")
```

Using Dynamic Colors with custom images
The DynamicImageColor feature allows colorizing custom marker images to match the colors of elements they represent. The sections of the image that are colorized are based on a specified DynamicColor.

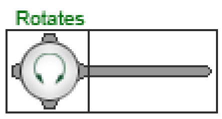
```
[C#]
myElement.Marker.DynamicImageColor = Color.FromArgb(123,123,123);

[Visual Basic]
myElement.Marker.DynamicImageColor = Color.FromArgb(123,123,123)
```

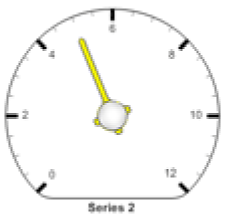
The Png image format allows variable transparency meaning, a pixel can have a specific color as well as an alpha part which indicates the pixel's transparency from 0(transparent) to 255(solid). The actual color of a particular pixel does not have to change along a fade to a transparent color. The dynamic color feature is capable of maintaining the original transparency of each pixel while replacing it with the element's color, provided the pixel's color matches or closely resembles the specified dynamic color.

DynamicImageColorTolerance

Many times, images don't use solid colors, instead they use antialiasing which blurs hard edges creating a smoother looking transition. By specifying a tolerance (0-100) the dynamic color will cover a wider range of colors based around the main dynamic color. At the same time, it will maintain the difference in deviation from the main color, therefore, the element's color will not be applied to colors within the tolerance range the same way. The color will vary to replicate the original images variations such as slight highlights.

Custom Circular Gauge Needles
The element marker can also be used with the circular gauge and a custom image to allow custom gauge needles. The following illustration shows a sample custom needle image and resulting chart.





```
[C#]
Chart.YAxis.GaugeNeedleType = GaugeNeedleType.UseMarker;
Chart.DefaultElement.Marker = new ElementMarker("../../images/needle2.png");
Chart.DefaultElement.Marker.DynamicImageColor = Color.FromArgb(255, 255, 255);

[Visual Basic]
Chart.YAxis.GaugeNeedleType = GaugeNeedleType.UseMarker
Chart.DefaultElement.Marker = New ElementMarker("../../images/needle2.png")
Chart.DefaultElement.Marker.DynamicImageColor = Color.FromArgb(255, 255, 255)
```

1.8 Element Layout Control

Element Layout Control

Introduction
This section will describe the following:

- **Sizing Columns**
  How axes influence the element behavior and appearance.
- **Clustering Columns**
  How axes control column and cylinder clustering.

Besides the obvious, an axis can also control how elements drawn on its scale behave. These behavioral features are determined by the axis that shows element names or x values. For a vertical combo chart this would mean the x axis and y axis for horizontal combo.

Column and Cylinder Widths

- **SpacingPercentage**
  Gets or sets a percentage (0 - 100) which indicates the spacing between columns, cylinders or groups thereof.
- **StaticColumnWidth**
  Gets or sets the static bar width in pixels.

Example: This code specifies the column or cylinder widths for a ChartType.Combo chart.

```
[C#]
Chart.XAxis.SpacingPercentage = 30; // Default is 16
// When this property is set, it takes precedence over spacing percentage.
Chart.XAxis.StaticColumnWidth = 20;

[Visual Basic]
Chart.XAxis.SpacingPercentage = 30 ' Default is 16
' When this property is set, it takes precedence over spacing percentage.
Chart.XAxis.StaticColumnWidth = 20
```

Tip: The column width control also defines the tool tip hotspot width of area line series types.

Clustering Columns / Cylinders
This feature enables columns in 3D modes to cluster (draw side by side) or not (draw one in front of the

other). The default behavior is to cluster and column must be clustered in 2D mode.

```
[C#]
Chart.Use3D = true;
Chart.XAxis.ClusterColumns = false;

[Visual Basic]
Chart.Use3D = True
Chart.XAxis.ClusterColumns = False
```

Other options include

- **Position**
  The axis positions when 2 or more axes are drawn on the same side of a ChartArea.
- **ReverseSeriesPositions**
  Indicates whether the positions of series bound to this axis are reversed without reversing legend positions.
- **ReverseSeries**
  Indicates whether the positions of series bound to this axis are reversed.
- **ReverseStack**
  Indicates whether the order of stacked elements is reversed.

1.8.1 Elements And Axis Scales

Introduction

While there are many options for scales, .netCHARTING can automatically determine the appropriate scales based on your data. This tutorial will demonstrate how element data influences axis scales.

The Y Axis (value axis)

We will call this the y axis but by '*value axis*' we don't literally mean (Y) axis. With ChartType.ComboHorizontal for instance we would be referring to the x axis. For all others however it is the y axis. The element values that influence this axis are

- YValue
- YValueStart
- YDateTime
- YDateTimeStart

[New in v5.0]
The elements of a series can be excluded from the axis scale range using Series.ExcludeFromAxisRange = true

The automatically chosen axis scales here are either **Normal** or **Time**. It is determined by whether the YValue (numeric) or YDateTime (time) values are specified for each element.

The following table shows value settings of these element properties that the chart engine will consider not set.

- YValue & YValueStart
  Setting: double.NaN
- YDateTime & YDateTimeStart
  Setting: DateTime.MinValue

The X Axis (Category / Value Axis)

Y Axis using ChartType.ComboHorizontal

The x axis is very powerful, it can operate just like the y axis *value axis* as well as a *category axis*. If the elements have names (Element.Name) specified, the axis scale will be a category scale. However, if the elements have either XValue or XDateTime properties set, the appropriate value axis scale will be chosen.

The element properties that influence the x axis type are:

- Name
- XValue
- XValueStart
- XDateTime
- XDateTimeStart

⚠Even if all numeric or time values are provided, setting any of the element's Name properties will yield a category axis scale and values will become names for elements without specified names.

The data engine will always provide elements with both, names and values if the names represent numeric or time values. By default this creates a category scale. To use a value scale instead, set the axis scale property to the appropriate scale. Scale.Normal for numeric and Scale.Time for date time values.

⚠Setting the scale through DefaultAxis.Scale will not produce the same result.

```
[C#]
Chart.XAxis.Scale = Scale.Time;

[Visual Basic]
Chart.XAxis.Scale = Scale.Time
```

Smart Category Axis

A *smart category axis* scale is one that contains elements with names and values but the names are string representations of the value properties. The 'smart' part is that despite it being a category (string) axis, the elements will be sorted in sequence based on the values they represent.

To use this axis type you must use the DataEngine and specify 'XAxis=valueDBColumn' in its DataFields property.

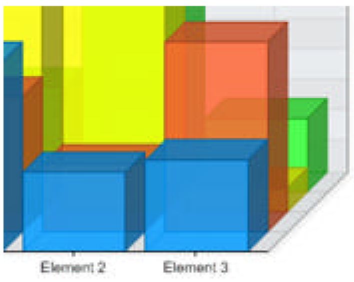
1.8.2 Element Layout Control Advanced

Z-Axis Effect

Using multiple axes with series can yield some interesting results you may not be aware of. We'll explore a situation that simulates a z axis. A basic z axis can be simulated by setting the x axis ClusterColumns property to false:

```
[C#]
Chart.XAxis.ClusterColumns = false;

[Visual Basic]
Chart.XAxis.ClusterColumns = false
```

This may show a chart that looks like this:



Gantt chart tip:

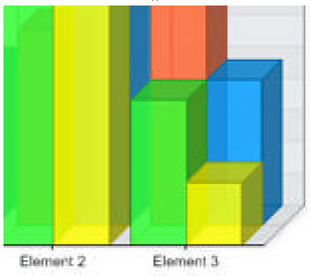
The CluseterColumns feature is only available when Chart.Use3D = true, however, it is still possible to use this feature in 2D by using 3D but emulating a 2D chart by setting the Chart.Depth property to zero. This trick may be most useful when creating Gantt charts which require element columns from different series (but using the same element names) to occupy the same horizontal space (with vertical 'Combo') or vertical space with ComboHorizontal charts.

Using X Axes

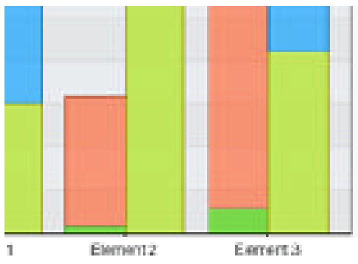
This works well however if we wanted a z axis with two steps and two clustered series on each step we will have to use two X axes. This time we will also omit setting the cluster columns property to false.

```
[C#]
Axis a2 = new Axis();
mySC[0].XAxis = a2;
mySC[1].XAxis = a2;
a2.Clear();

[Visual Basic]
Dim a2 As New Axis()
mySC(0).XAxis = a2
mySC(1).XAxis = a2
a2.Clear()
```

We "Clear()" the second axis so that only the original one is visible.



Tip: This method also allows overlapping columns in 2D mode which is useful in Gantt charts:



Using Y Axes

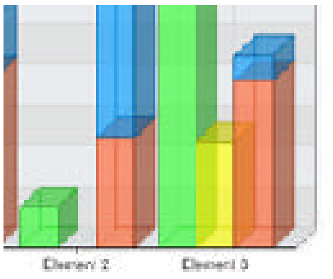
Y axes will typically not affect the z axis. For example using the following code where we give two of the four series a new y axis and set the scale to stacked:

```
[C#]
```

Axis a2 = new Axis();
mySC[0].YAxis = a2;
mySC[1].YAxis = a2;
a2.Scale = Scale.Stacked;

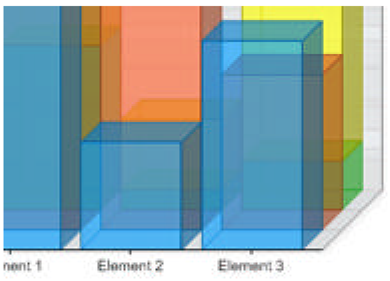
```
[Visual Basic]
```

Dim a2 As New Axis()
mySC(0).YAxis = a2
mySC(1).YAxis = a2
a2.Scale = Scale.Stacked
Will yield this result:



Notice that the new y axis is stacked but still the main x axis manages the clustered - unclustered layout. Therefore if we wanted to uncluster the columns we could have multiple z axis steps and some may be stacked while other wont.

```
[C#]
Chart.XAxis.ClusterColumns = false;

[Visual Basic]
Chart.XAxis.ClusterColumns = false
```



Please note that using multiple value axes while hiding one of them may result in one going out of sync with the other. This will result in the bars indicating incorrect values (according to the visible value axis). This issue can be resolved by synchronizing the visible with the invisible axes.

For more information see the **scale synchronization (on-line documentation)** tutorial.

Sample: AxisDualScales.aspx

1.9 Micro Charts

Introduction

MicroCharts are miniature versions of actual charts, designed to help viewers understand complex relationships between data in repetitive scenarios such as data grids or InfoGrids. MicroCharts are optimized to use very little space and are ideal for data grids, info grids or use as parts of labels within charts. MicroCharts are defined using a markup language inside the chart label text. This means that anywhere a label is used, a microChart can easily be embedded. With the use of tokens, microCharts can be setup using default properties and will automatically represent the label's parent objects token values. InfoGrids are similar to traditional data grids but are generated entirely within chart labels. MicroCharts and InfoGrids are two unique technologies that are ideally suited for each other; MicroCharts enrich InfoGrids with concise data visualizations. Please see the **tutorial on InfoGrids ('InfoGrids' in the on-line documentation)** for additional information on data grid and dashboard usage.

Basic Usage

To use a microChart in a label, use a syntax that starts with <Chart: then the type of microChart and other options ending with a closing '>' character. Parameters must be specified using a format like *param='value'* using single quotes around the value as shown below:

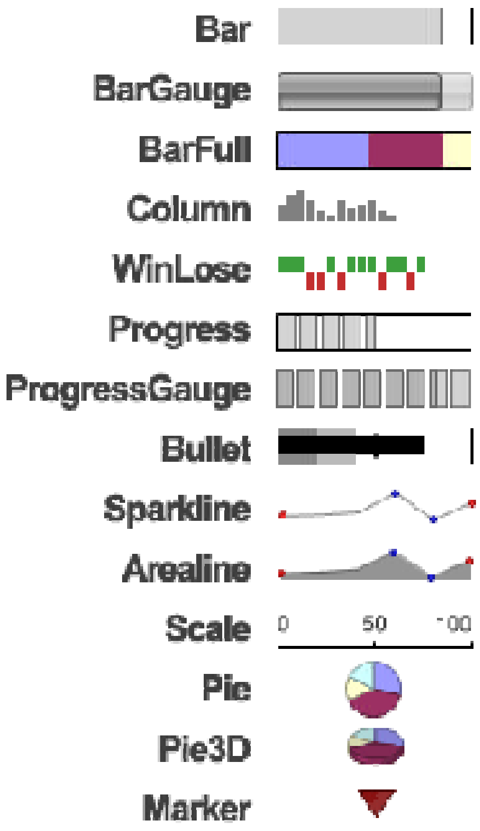
```
[C#]
Chart.TitleBox.Label.Text = "<Chart:Bar value='5' max='10'>";

[Visual Basic]
Chart.TitleBox.Label.Text = "<Chart:Bar value='5' max='10'>"
```

MicroChart Types

Each microChart type has a number of options available. Similar types work alike but all share the same syntax.

- **Bar**
  <Chart:Bar value='5' max='10'>
  Draws a basic bar. Usually used with a max attribute so it lines up with other bars in a grid format.
- **BarGauge**
  <Chart:BarGauge value='5' max='10'>
  This type is similar to Bar except a full width translucent version of the bar is drawn underneath like GaugeType.Bars on normal Charts.
- **BarFull**
  <Chart:BarFull values='5,4,3' >
  Values specified for this type are placed on a 100% stacked axis.
- **Column**
  <Chart:Column values='5,4,3' >
  Creates a series of columns based on specified values. This type is different (in appearance) than regular combo column charts because it's optimized for small sizes and does not support shading effects.
- **WinLose**
  <Chart:WinLose values='1,1,1,1,-1,-1,-1'>
  This type is similar to Column except it only takes values -1,0,or 1 to represent a loss, tie, or win respectively. It is also useful in showing the status of something like a server functioning normally or being down over time.
- **Progress**
  <Chart:Progress value='5' max='10'>
  This type is similar to Bar except the bar is drawn in segments.
- **ProgressGauge**
  <Chart:ProgressGauge value='5' max='10'>
  This type is similar to BarGauge except the bar is drawn in segments.
- **Bullet**
  <Chart:Bullet values='20,18,8,13' max='25'>
  The bullet type shows a value (bar) a target (vertical line segment) and AxisMarker bands indicate additional target or status.
- **Sparkline**
  <Chart:Sparkline values='5,4,3' >
  A sparkline is similar to a line chart but without the extras like grid lines.
- **Arealine**
  <Chart:AreaLine values='5,4,3' >
  Similar to an AreaLine chart but without the extras like axes.
- **Scale and ScaleB**

<Chart:Scale min='0' max='100'>
The scale micro chart is designed to be used in a header of a column that will show micro charts based on that scale range. By specifying the same width for the scale and subsequent charts, it will ensure the charts in that column line up with each other and the scale. Using a scale also requires that the same max parameter value is set for all the charts so the values of each chart correspond to the values indicated by the scale. In some cases, the min parameter may also be required. ScaleB works the same as Scale, except it faces upward instead of facing down like Scale does.



- **Pie**
  <Chart:Pie values='5,4,3' >
  Simple pie microChart
- **Pie3D**
  <Chart:Pie3D values='5,4,3' >
  Same as pie but uses the 3D effect.
- **Marker**
  <Chart:Marker type='square' color='crimson'>
  Draws any available ElementMarkerType. In addition, the circle (default) marker type also supports bubble shading effects when specified.
- **Image**
  <Chart:Image src='image.png' rotate='45'>
  Drawn the specified image. Options such as image rotation and sizing are possible with this tag.
- **Spacer**
  <Chart:Spacer size='230x15'>
  Acts as a transparent placeholder or spacer. It is useful in a number of scenarios to control layout.

A sample is available for each type of microChart demonstrating all the options available for those given types. These samples are located in the gallery under Chart Types/MicroCharts.

Available Parameters

- Value
  Used by micro charts that take a single value such as Bar, BarGauge, Progress, ProgressGauge
- Values
  Specify multiple numeric values for the micro chart.
- Color
  Specifies a color for micro charts that display a single value. For some cases it is used for a simpleColor feature described in more detail below.
- Colors
  Used to specify the colors used on the chart.
- Min / Max
  Specifies the minimum and maximum scale values.
- Width / Height
  Specifies the width and height of the micro chart in pixels.
- Size
  Can be used to set the size easier than using width and height separately. Examples: size='100x20' or size='20' (for pies or markers). Setting the size parameter with a single value applies it to both width and height.
- Shading
  Takes a number between 1 and 7 and specifies the shading effect mode to use on the micro chart.

- AxisMarker
  Takes a single to draw a reference line or two values that specify a range to highlight similar to range axis markers on regular charts. Sparkline, AreaLine, and Column support this feature.
- Type
  Specifies the ElementMarkerType for use with a marker micro chart.
- Url
  Specifies a clickable URL for the microchart.
- Tooltip
  Specifies a tooltip that will appear when hovering over the micro chart.
- Src
  Image source path. Used with Chart:Image type.
- Rotate
  Specifies image rotation in degrees. Used with Chart:Image type.

Sizing MicroCharts

All microCharts have predefined default sizes. Types like Pie, Pie3D use 30x30 and Marker uses 20x20. All other except for Column and WinLose use 100x20. The Column and WinLose types size automatically depending on the number of values specified. This way stacking multiple column microCharts with different number of elements will allow the columns to be the same width. Column types will also automatically left align to support this. The column widths can be controled by specifying only the height parameter, however, using both width and height or just width is also supported.

Shading MicroCharts

A number of micro charts support shading effect modes. These can be specified with the 'shading' parameter which takes a number from 1 to 7 indicating the shading effect mode to use. The micro charts that support shading are:

- Bar
- BarGauge
- Progress
- ProgressGauge
- BarFull
- Bullet
- Pie
- Marker - (With markers, only the default 'Circle' marker type supports shading)

Setting Axis Range

The scale micro chart provides a means to describe other horizontal based micro charts. When other micro charts are stacked vertically with a scale in the header row then must sync the scales in order to visualize the information correctly. The scale range can be specified through the Min and Max attributes. The default value of the Min attribute is always 0. The following micro chart types support the min/max axis range attributes:

- Bar
- BarGauge
- Progress
- ProgressGauge
- Bullet
- Scale

The scale can also be used with other microCharts in some situations. For example, if using SparkLines to represent sales in january, a static scale with two labels: "Jan", "Feb" can be used to indicate the range represented by the sparklines.

Chart Types

Values and Colors

While most parameters apply to every type of microChart in a similar fashion, the value(s) and color(s) parameters vary depending on the type of microChart used. The following grid matrix explains how value and color parameters are used with the corresponding type of microChart.

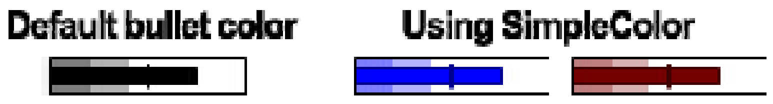
NOTE: While value and color parameters take a single string value, the values and colors parameters (plural) accept a comma delimited list of the same values.

Type	Value / Values	Colors
Bar	Value parameter indicates the bar value.	Color: Sets the bar color. Colors: 1st color sets the bar 2nd color sets the background
BarGauge	Value parameter indicates the bar value.	Color: Sets the bar color.
Progress	Value parameter indicates the bar value.	Color: Sets the bar color. Colors: 1st color sets the bar 2nd color sets the background
ProgressGauge	Value parameter indicates the bar value.	Color: Sets the bar color.
BarFull	Values parameter is used to pass a comma delimited list of values.	Colors: a list of colors for each bar. Color: **SimpleColor Feature**
Bullet	Values: the sequence of values specify: Value, Target, 1st, and 2nd axis marker positions. If the 3rd value is larger than the 4th. The axis marker colors fill in the opposite direction.	Colors: 1st Color: Bar 2nd Color: Target 3rd Color: Shade 1 4th Color: Shade 2 5th Color: Background Color: **SimpleColor Feature**
SparkLine	Values parameter is used to pass a comma delimited list of values.	Colors 1st Color: Line 2nd Color: Line ends 3rd Color: Max/Min 4th Color: AxisMarker The word 'Transparent' can be used to omit specifying a color in the colors array.
AreaLine	Values parameter is used to pass a comma delimited list of values.	Colors 1st Color: Line 2nd Color: Line ends 3rd Color: Max/Min 4th Color: AxisMarker The word 'Transparent' can be used to omit specifying a color in the colors array.
Scale & ScaleB	Value: sets the default axis tick label text. ex. '<% Value,Currency>' Values: a comma delimited list of string labels to place on the axis. ex: 'Bad,Good,Great' Because each value will be processed by the chart itself, markup can be used as well such as including images. This is demonstrated in the MicroScaleOptions sample.	Color: Specifies the color for axis and labels Colors: 1st color: labels 2nd color: axis line
Pie	Values parameter is used to pass a comma delimited list of values.	Colors reflect element colors Color: **SimpleColor Feature**
Pie3D	Values parameter is used to pass a comma delimited list of values.	Colors reflect element colors Color: **SimpleColor Feature**
Marker	N/A	Color: marker color.

Column	Values parameter is used to pass a comma delimited list of values.	Color: (+)columns Colors: 1st: (+)columns 2nd: (-)columns 3rd: AxisMarker
WinLose	Values parameter is used to pass a comma delimited list of values. The options are -1 = lose, 0 = tie, and 1 = win.	Color: (win)columns Colors: 1st: (win)columns 2nd: (loss)columns 3rd: AxisMarker

SimpleColor Feature

By setting the color parameter for charts that support this feature, multiple colors will be created for all elements based on the specified color. This can allow the ability to quickly apply a primary color of your page to ensure the MicroCharts fit.



Using MicroCharts

Because micro charts can be used anywhere a label appears on a chart, there are an endless number of possibilities, however, below are a number of scenarios in which microCharts can be especially useful.

Scenario	**Example**
In Labels on charts	Code: Chart.TitleBox.Label.Text = "<Chart:Bar value='5' max='10'>";
In Element Labels	Sample: Dashboards And InfoGrids/ChartMicroPie
On Axis Tick labels	Sample: Dashboards And InfoGrids/ChartMicroAxis
In legend box entries	Sample: Legend Box/LegendSparkLines
In Empty legend box headerLabel.	Sample: Dashboards And InfoGrids/AnnotationInfoGrid1
Inside annotations	Sample: Dashboards And InfoGrids/InfoGridMulti
By themselves as a chart control.	Sample: Dashboards And InfoGrids/InfoGrid1
Inside asp.net DataGrid	Sample: Dashboards And InfoGrids/DataGridDash

MicroChart Helpers

A number of tokens are available to help making microCharts easier. For example, the sample [Dashboards And InfoGrids/ChartMicroSubValues] uses the element token %SubValueList which can be used in the element label inside a microChart tag. Other useful tokens include:

Type	**Tokens**
Element	%SubValueList - a comma delimited list of sub values.
Series	%YValueList - a list of element y values
SeriesCollection	%YSeriesSumList - a list of of each series y value sum. %YGroupSumList - a list of y value sums for each element group. %ColorList - a list of colors used by the series in the collection.

Extending MicroChart functionality.

If the provided micro chart types do not meet the requirements of your project, custom micro charts can be built from scratch and embedded into chart labels in a similar fashion. Sample: Features/Legend Box/EmbeddedLegendCharts demonstrates this scenario. The sample inserts a Progress type MicroChart into the legend box similar to sample ('LegendCharts' located in the same section), except, the code to generates the indicator chart is stored in a separate file which can be modified.

Index