



Website: [www.chestysoft.com](http://www.chestysoft.com)

Email: [info@chestysoft.com](mailto:info@chestysoft.com)

## **csASPNetGraph - Version 1.1**

### **.NET Component for Drawing Bar Charts, Pie Charts & Line Graphs**

This is a .NET component that dynamically generates 2D bar charts, pie charts, simple line graphs and scatter diagrams. The resulting graph can be streamed to a web browser from an ASP.NET application, saved to disk or exported as a Bitmap or MemoryStream structure. The file formats supported are gif, png, bmp, tif and jpg. Images exported in gif format can support a transparent colour to help in fitting the graph to a web page.

This is a non visual component designed for use in ASP.NET but it can be used in a compiled Windows Forms application. It is simple to export the graph image as a Bitmap and pass this to a PictureBox control to view it.

A free, fully functional trial version of csASPNetGraph is available. This trial version has a built in expiry date that causes it to stop working after that time. This is the only functional limitation between the trial and full versions. This means that you can fully test if this component is suitable for your application before considering whether to license the full version.

### **Using these Instructions**

These instructions are divided into a number of sections covering the different types of graph. There are quick links to each section. A full Table of Contents is available on the next page and an index listing all commands in alphabetical order is included at the back for easy reference.

The component has a large number of properties that control the appearance of the graphs. When these properties are not specified in code they will take their default values and these defaults have been selected with the aim of producing a legible graph with the minimum of code.

Click on one of the links below to go directly to the section of interest:

- [Getting Started](#)
- [Pie Charts](#)
- [Bar Charts](#)
- [Line Graphs](#)
- [Stacked Bar Charts](#)
- [Streaming an Image to the Browser](#)
- [Alphabetical List of Commands](#)

# TABLE OF CONTENTS

<b>1. GETTING STARTED .....</b>	<b>4</b>
1.1. USING CSASPNETGRAPH IN ASP.NET .....	4
1.2. USING CSASPNETGRAPH IN VISUAL STUDIO .....	4
1.3. THE TRIAL VERSION .....	5
1.4. THE LICENCE FILE .....	5
1.5. DEFINING COLOURS IN METHODS AND PROPERTIES .....	6
1.6. FONT PROPERTIES .....	6
1.7. ENUMERATED PROPERTIES .....	7
<b>2. PIE CHARTS .....</b>	<b>9</b>
2.1. PIE CHART METHODS .....	9
2.2. PIE CHART PROPERTIES .....	9
2.3. PIE CHART EXAMPLE .....	10
<b>3. BAR CHARTS .....</b>	<b>12</b>
3.1. BAR CHART METHODS .....	12
3.2. BAR CHART PROPERTIES .....	12
3.3. BAR CHART EXAMPLE .....	13
<b>4. LINE GRAPHS .....</b>	<b>15</b>
4.1. LINE GRAPH METHODS .....	15
4.2. LINE GRAPH PROPERTIES .....	15
4.3. LINE GRAPH EXAMPLE .....	16
<b>5. STACKED BAR CHARTS .....</b>	<b>18</b>
5.1. STACKED BAR CHART METHODS .....	18
5.2. STACKED BAR CHART PROPERTIES .....	18
5.3. STACKED BAR CHART EXAMPLE .....	19
<b>6. MISCELLANEOUS SETTINGS .....</b>	<b>21</b>
6.1. PROPERTIES TO DEFINE THE AXES OF BAR AND LINE GRAPHS .....	21
6.2. LEGEND PROPERTIES .....	22
6.3. CHANGING THE SIZE OF A GRAPH .....	22
6.4. STANDARD TEXT AND ANNOTATIONS .....	22
6.4.1. <i>Labels (Axis Values and Pie Chart Labels)</i> .....	22
6.4.2. <i>Title Text</i> .....	23
6.4.3. <i>The Prefix, Suffix and ShowSeparator Properties</i> .....	23
6.5. DECIMAL PLACES .....	24
6.6. ADDING EXTRA TEXT AND ANNOTATIONS TO A GRAPH .....	24
6.6.1. <i>AddText and AddExtraLine</i> .....	24
6.6.2. <i>Adding Text to Line Graphs</i> .....	24
6.7. SUBSTITUTING AXIS LABELS .....	26
6.8. PLOTTING DATES OR TIMES .....	26
6.9. DISPLAYING PERCENTAGES ON BAR AND PIE CHARTS .....	28
6.10. RANDOM COLOURS .....	28
6.11. MISCELLANEOUS DISPLAY PROPERTIES AND FUNCTIONS .....	29
6.12. THE DUMMY GRAPH FUNCTION .....	29
<b>7. PLOTTING LINES ON BAR CHARTS .....</b>	<b>31</b>
7.1. BAR CHART EXAMPLE WITH A PLOTTED LINE .....	31
<b>8. DISPLAYING OR EXPORTING THE GRAPH IMAGE .....</b>	<b>33</b>
<b>9. DISPLAYING GRAPHS IN A WEB BROWSER .....</b>	<b>34</b>
<b>10. IMAGE MAPS .....</b>	<b>35</b>

10.1.	IMAGE MAP METHODS .....	35
10.2.	IMAGE MAP PROPERTIES .....	35
10.3.	GENERATING AN IMAGE MAP .....	35
<b>11.</b>	<b>SAMPLE GRAPHS .....</b>	<b>37</b>
<b>12.</b>	<b>OTHER PRODUCTS FROM CHESTYSOFT .....</b>	<b>38</b>
<b>13.</b>	<b>ALPHABETICAL LIST OF COMMANDS.....</b>	<b>39</b>

# 1. Getting Started

## 1.1. Using csASPNetGraph in ASP.NET

The DLL file Chestyssoft.csASPNetGraph.dll (or Chestyssoft.csASPNetGraphTrial.dll for the trial version) must be placed in the \bin folder for the web application. The ASP.NET machine account must have Read and Execute permission on the DLL file. For the full version, the licence file csASPNetGraph.GraphClass.lic must also be copied to the \bin folder. The trial version does not have a licence file.

The permissions on the \bin folder should not allow access for the anonymous internet user, IUSR\_machine\_name. This is to prevent users from downloading DLLs or components.

The ASP.NET script must import the namespace csASPNetGraph (csASPNetGraphTrial for the trial version). The class name for the component is GraphClass.

### Creating the component instance in VB.NET:

```
<%@ Page language="vb" debug="true" %>
<%@ Import Namespace = "csASPNetGraph" %>
<%
.
Dim Graph As New GraphClass
.
.
.
%>
```

### Creating the component instance in C#:

```
<%@ Page language="c#" debug="true" %>
<%@ import Namespace = "csASPNetGraph" %>
<%
.
GraphClass Graph = new GraphClass();
.
.
.
%>
```

Both these examples show use with the full version of the component. If any font properties need to be set, the System.Drawing namespace must also be imported.

The script that runs the csASPNetGraph component will usually export binary output, and it will be called from inside an IMG tag in another web page. This process is described fully in Section 8.

## 1.2. Using csASPNetGraph in Visual Studio

The DLL file Chestyssoft.csASPNetGraph.dll (or Chestyssoft.csASPNetGraphTrial.dll for the trial version) can be placed in any folder. For the full version, the licence file csASPNetGraph.GraphClass.lic must also be copied to the same folder. The trial version does not have a licence file.

In Visual Studio, select Customize Toolbox from the Tools menu, click on the tab for .NET Framework Components, and select the Browse button. Find the DLL and this will add the component to the General section of the Toolbox. It is displayed using the class name of GraphClass. Once in the

Toolbox, the component can be added to a form. It is not a visual component and does not appear on the form itself. The properties are not available for editing in the Properties window and must be set from code. Graphs can be displayed in a Windows form by exporting as a bitmap and displaying in a PictureBox.

### Displaying a graph in a VB.NET Windows Form:

The following code will produce a simple pie chart with two data items and display it in a PictureBox:

```
GraphClass1.ClearData
GraphClass1.AddData("Item 1", 10, "FF0000")
GraphClass1.AddData("Item 2", 5, "00FF00")
PictureBox1.Image = GraphClass1.GraphToBitmap
```

The PictureBox must be the same size as the graph, which is 400 x 300 pixels when the default properties are used. The *GraphToBitmap* method draws the graph using the properties and data items that have been set and returns a Bitmap structure, which can be read straight into the PictureBox as shown.

### Displaying a graph in a C# Windows Form:

The code is almost identical to the VB.NET example. C# is case sensitive and the default class names start with a lower case letter:

```
graphClass1.ClearData();
graphClass1.AddData("Item 1", 10, "FF0000");
graphClass1.AddData("Item 2", 5, "00FF00");
pictureBox1.Image = graphClass1.GraphToBitmap();
```

## 1.3. The Trial Version

The trial version of the component is supplied as a separate DLL called Chestyssoft.csASPNetGraph.dll. This trial version is fully functional but it has an expiry date, after which time it will stop working. An exception will be raised if an attempt is made to create the object after the expiry date and the description of the exception will make it clear that the trial period has expired.

The expiry date can be found by reading the *Version* property.

**Version** - String, read only. This returns the version information and for the trial, the expiry date.

Example in ASP.NET (VB):

```
Dim Graph As New GraphClass
Response.Write(Graph.Version)
```

Visit the Chestyssoft web site for details of how to buy the full version - [www.chestysoft.com](http://www.chestysoft.com)

The trial version has the namespace csASPNetGraphTrial, compared with csASPNetGraph in the full version. After upgrading to the full version the scripts using the component will need to have this namespace name changed.

## 1.4. The Licence File

The full version of csASPNetGraph is supplied with a licence file, csASPNetGraph.GraphClass.lic. This file must be present in the same directory as the component DLL in order to use the component from ASP.NET or at design time in a Windows Forms project. It is not required with a compiled Windows Forms project and it must not be distributed with a compiled application.

## 1.5. Defining Colours in Methods and Properties

csASPNetGraph takes colour values as 6 character hexadecimal strings, in the same way that HTML defines colours. This is used in preference to the System.Drawing.Color structure partly to make it easier to integrate into a web application and because it will often be used in a text editor which will not be able to check syntax.

The colour string is in the form of 6 hexadecimal characters representing the RGB colours, "RRGGBB". So red is "FF0000", blue is "0000FF" and white is "FFFFFF". These strings are not case sensitive.

Functions are provided to convert between these 6 character strings and System.Drawing.Color values.

<b>StringToColor</b> ( <i>InString</i> As String) - Color. Takes a string of the format "RRGGBB" and returns the equivalent Color value.	
<b>ColorToString</b> ( <i>InColor</i> As Color) - String. Takes a Color value and returns the equivalent string in the format "RRGGBB".	

## 1.6. Font Properties

The appearance of text used in the graphs is controlled by a number of font properties, which use the System.Drawing.Font structure. The individual properties of a Font are read only so a new Font must be created and the font name, size and style are specified inside the constructor. The System.Drawing namespace must be imported in order to create a Font.

In both VB.NET and C# the namespace is imported by adding the following line to the script:

```
<%@ import Namespace = "System.Drawing" %>
```

Here is an example of setting the *LabelFont* property, which controls the appearance of the text used to annotate pie charts and bar charts.

### VB.NET

```
Graph.LabelFont = New Font("Arial", 10, FontStyle.Bold)
```

The Font constructor is overloaded and this variation sets the font name, the size and the style. It is also acceptable to miss the style if the regular style is required:

```
Graph.LabelFont = New Font("Arial", 10)
```

### C#

```
Graph.LabelFont = new Font("Arial", 10, FontStyle.Bold);
```

or with a regular style:

```
Graph.LabelFont = new Font("Arial", 10);
```

Refer to the .NET Framework documentation for details of the Font class and its constructor.

## 1.7. Enumerated Properties

Some of the properties and methods use enumerated values, and these are all integers. This is to simplify coding when using a text editor. These values include the graph type, the image type when exporting a graph, text alignment options, and line point and grid styles for line graphs.

The enumerated values are listed here. Refer to the property and method descriptions for full details.

### GraphType

0	-	Pie chart
1	-	Vertical bar chart (default)
2	-	Horizontal bar chart
3	-	Line graph
4	-	Vertical stacked bar chart
5	-	Horizontal stacked bar chart

### Image Type (used in *GraphToFile* and *GraphToStream* methods)

0	-	BMP
1	-	GIF
2	-	PNG
3	-	JPG
4	-	TIF

### Text Alignment (used in *TitleTextAlign*, *LineGraphTextAlign* and *LegendAlign*)

0	-	Top left
1	-	Top centre
2	-	Top right
3	-	Baseline left
4	-	Baseline centre
5	-	Baseline right
6	-	Bottom left
7	-	Bottom centre
8	-	Bottom right

### Pen Style (used in the *GridStyle* property and *AddExtraLine* method)

0	-	Solid
1	-	Dotted
2	-	Dashed

### PointStyle

0	-	None
1	-	Dot
2	-	Circle
3	-	Diagonal cross
4	-	Vertical cross

### StackedTextAlign

0	-	Centre horizontal
1	-	Centre vertical
2	-	Left horizontal
3	-	Left vertical
4	-	Right horizontal
5	-	Right vertical

**DateTimeFormat** (used when values are plotted on line graphs as dates or times)

0	-	DateTime
1	-	Date
2	-	Time



## 2. Pie Charts

Pie charts are drawn by adding data items using the *AddData* method. Each data item has a name, a value and a colour and each item will be shown as a separate sector on the pie chart. An optional legend can be generated showing the names and colours of each data item. Properties are available to specify the position of the pie and its diameter, as well as options for showing the data values and/or the labels (data item names). Note that if the pie sectors are small these text annotations may interfere with each other. By default the data items are shown starting at the bottom of the chart and extending anticlockwise. The *StartAngle* property allows the starting point to be changed.

The *GraphType* property must be set to zero to produce a pie chart.

When multiple graphs are drawn using the same instance of the object, call the *ClearData* method to remove the data items and start again.

### 2.1. Pie Chart Methods

<b>AddData</b> ( <i>Name</i> As String, <i>Value</i> As Double, <i>Colour</i> As String) - Each data item in a pie chart has a <i>Name</i> - which may be displayed on the chart or in the legend, a <i>Value</i> - which must be a positive numeric value, and a <i>Colour</i> - which is a six character hexadecimal string giving the RGB value of the colour (eg "ff0000" is red).
--

A separate call to *AddData* must be made for each data item in the chart.

Example:

```
Graph.AddData("Item 1", 17, "ff0000")
Graph.AddData("Item 2", 28, "00ff00")
Graph.AddData("Item 3", 5, "0000ff")
```

This would add three items of data, to be displayed in red, green and blue respectively.

The graph itself is produced with a call to one of the following methods: *GraphToBrowser*, *GraphToBitmap*, *GraphToFile*, *GraphToStream*.

### 2.2. Pie Chart Properties

The properties described here are specific to pie charts. The default values are shown in brackets.

<b>GraphType</b>	-	Integer property. This specifies the type of graph produced. Set to 0 for a pie chart.
<b>CenterX</b>	-	Integer. X-coordinate of the centre of the pie, measured from the left. (175)
<b>CenterY</b>	-	Integer. Y-coordinate of the centre of the pie, measured from the top. (150)
<b>PieDia</b>	-	Integer. Diameter of the pie in pixels. (200)
<b>StartAngle</b>	-	Single. Position of the first pie segment measured in degrees, anticlockwise, with zero at the bottom of the pie. (0)
<b>PieOffset</b>	-	Integer. The spacing between the labels and the pie chart edge. It does not represent any specific units but a larger value gives a larger spacing. (10)

There are other properties listed in other sections of these instructions that can also be used to control the appearance of a pie chart, such as the annotations and legend.

After adding data and setting any required properties, the graph is produced using one of the methods described below. The *GraphType* property must also be set.

## 2.3. Pie Chart Example

The following examples shows how a pie chart can be drawn using ASP.NET.

### VB.NET

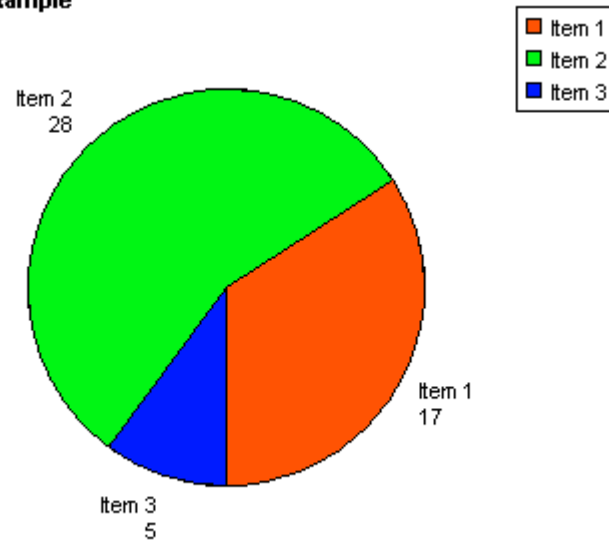
```
<%@ Page language="vb" debug="true" %>
<%@ Import Namespace = "csASPNetGraph" %>
<%
Response.Buffer = true
Response.Expires = 0
Response.Clear
Dim Graph As New GraphClass
Graph.Title = "Pie Chart Example"
Graph.AddData("Item 1", 17, "ff0000")
Graph.AddData("Item 2", 28, "00ff00")
Graph.AddData("Item 3", 5, "0000ff")
Graph.GraphType = 0
Graph.GraphToBrowser(1)
%>
```

### C#

```
<%@ Page language="c#" debug="true" %>
<%@ import Namespace = "csASPNetGraph" %>
<%
Response.Buffer = true;
Response.Expires = 0;
Response.Clear();
GraphClass Graph = new GraphClass();
Graph.Title = "Pie Chart Example";
Graph.AddData("Item 1", 17, "ff0000");
Graph.AddData("Item 2", 28, "00ff00");
Graph.AddData("Item 3", 5, "0000ff");
Graph.GraphType = 0;
Graph.GraphToBrowser(1);
%>
```

In this example all the data values are hard coded and the default property values are used. It is written for the full version of the component. Use the namespace "csASPNetGraphTrial" if the trial version is used. *GraphType* is set to 0 for a pie chart and the *ImageType* parameter of *GraphToBrowser* is 1 for a GIF image.

**Pie Chart Example**



The *Title* property has been set, with the title using the default settings for font and position.

## 3. Bar Charts

Bar charts are drawn by adding data items using the *AddData* method. Each data item has a name, a value and a colour and each item will be shown as a separate bar on the bar chart. An optional legend can be generated showing the names and colours of each data item. A large number of properties are available to control the size, shape and appearance of bar charts. These include options for displaying negative values, adjusting the size and calibration of the axes and displaying a legend.

There are two variations of bar chart. The bars can be shown vertically or horizontally and the type of bar chart is defined by the *GraphType* property. *GraphType* must be set to 1 for a vertical bar chart (the default value) or 2 for a horizontal bar chart.

An optional linear regression trend line can be shown on bar charts by setting the *ShowTrendLine* property to true.

When multiple graphs are drawn using the same instance of the object, call the *ClearData* method to remove the data items and start again.

### 3.1. Bar Chart Methods

<b>AddData</b> ( <i>Name</i> As String, <i>Value</i> As Double, <i>Colour</i> As String) - Each data item in a bar chart has a <i>Name</i> - which may be displayed on the chart or in the legend, a <i>Value</i> - which must be a numeric value, and a <i>Colour</i> - which is a six character hexadecimal string giving the RGB value of the colour (eg "ff0000" is red).
---

A separate call to *AddData* must be made for each data item in the chart.

Example:

```
Graph.AddData ("Item 1", 17, "ff0000")
Graph.AddData ("Item 2", 28, "00ff00")
Graph.AddData ("Item 3", 5, "0000ff")
```

This would add three items of data, to be displayed in red, green and blue respectively.

The graph itself is produced with a call to one of the following methods: *GraphToBrowser*, *GraphToBitmap*, *GraphToFile*, *GraphToStream*.

### 3.2. Bar Chart Properties

The properties described here are specific to bar charts. The default values are shown in brackets.

<b>GraphType</b> - Integer property. This specifies the type of graph produced. Set to 1 for a bar chart with vertical bars and to 2 for a bar chart with horizontal bars.
<b>BarWidth</b> - Integer. Width of each bar in pixels. Auto calibrate if zero. (0)
<b>BarGap</b> - Integer. Space between bars in pixels. Auto calibrate if zero. (0)
<b>LabelVertical</b> - Boolean. Controls the orientation of the names of each data item written under each bar. (false)
<b>ShowBarTotal</b> - Boolean. When true the data value will be shown above each bar. (false)
<b>BarTotalVertical</b> - Boolean. Controls the orientation of the data values above each bar. (false)
<b>ShowTotalifZero</b> - Boolean. Determines whether zero data values are shown when <i>ShowBarTotal</i> is true. (true)

The following properties control the trend line.

<b>ShowTrendLine</b>	-	Boolean. When true a linear regression trend line will be drawn on the bar chart. (false)
<b>TrendLineColor</b>	-	String. The colour of the trend line as a 6 digit hexadecimal string. ("000000")
<b>TrendLineWidth</b>	-	Integer. The thickness of the trend line, in pixels. (1)
<b>TrendLineName</b>	-	String. The name that will appear in the legend to describe the trend line. If this is an empty string the trend line will not be displayed in the legend. (null)

As of version 1.1 of csASPNetGraph it is possible to plot lines onto bar charts. This is described in Section 7 of these instructions.

There are a large number of properties that can be used with bar charts that are described in other sections of these instructions.

### 3.3. Bar Chart Example

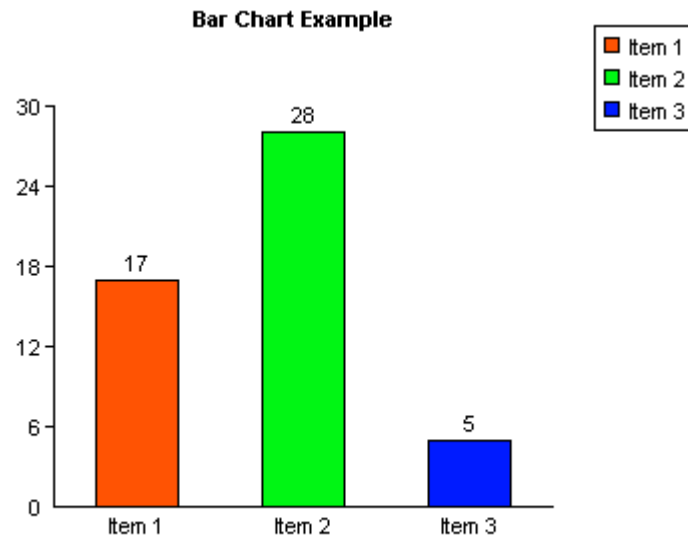
The following example shows how a bar chart can be drawn using ASP.NET.

#### VB.NET

```
<%@ Page language="vb" debug="true" %>
<%@ Import Namespace = "csASPNetGraph" %>
<%
Response.Buffer = true
Response.Expires = 0
Response.Clear
Dim Graph As New GraphClass
Graph.Title = "Bar Chart Example"
Graph.TitleX = 120
Graph.ShowBarTotal = true
Graph.AddData("Item 1", 17, "ff0000")
Graph.AddData("Item 2", 28, "00ff00")
Graph.AddData("Item 3", 5, "0000ff")
Graph.GraphType = 1
Graph.GraphToBrowser(1)
%>
```

#### C#

```
<%@ Page language="c#" debug="true" %>
<%@ import Namespace = "csASPNetGraph" %>
<%
Response.Buffer = true;
Response.Expires = 0;
Response.Clear();
GraphClass Graph = new GraphClass();
Graph.Title = "Bar Chart Example";
Graph.TitleX = 120;
Graph.ShowBarTotal = true;
Graph.AddData("Item 1", 17, "ff0000");
Graph.AddData("Item 2", 28, "00ff00");
Graph.AddData("Item 3", 5, "0000ff");
Graph.GraphType = 1;
Graph.GraphToBrowser(1);
%>
```



In this example all the data values are hard coded and the default property values are used. It is written for the full version of the component. Use the namespace "csASPNetGraphTrial" if the trial version is used.

The *Title* property has been set and *TitleX* has been used to move the title text into a central position. *ShowBarTotal* is true which causes the value of each data item to appear above each bar.

Refer to Section 6 for the full range of properties that are shared with other graph types, such as setting the axes, positioning the legend and additional annotation.

## 4. Line Graphs

The line graphs drawn by csASPNetGraph are simple 2-D graphs on a single set of X and Y axes. Multiple lines can be displayed on one graph if the lines are separate colours. By default the points are hidden and the lines joining the points are shown. Changing the *PointStyle* property will make the points appear and setting *ShowLine* to false will hide the graph lines and this arrangement can be used to draw a type of scatter diagram.

The *GraphType* property must be set to 3 to produce a line graph.

When multiple graphs are drawn using the same instance of the object, call the *ClearData* method to remove the data items and start again.

### 4.1. Line Graph Methods

**AddPoint** (*X* As Double, *Y* As Double, *Colour* As String, *Name* As String) - This command adds a point to be plotted on a line graph. *X* and *Y* are the coordinates of the point, which must be numeric values. *Colour* is a six character hexadecimal string giving the RGB colour value of the line. *Name* is a string and it will be displayed if a legend is used. When the graph is plotted all the points that are the same colour will be joined by straight lines going from point to point in the order they were added.

A separate call to *AddPoint* must be made for each point on the graph.

Example:

```
Graph.AddPoint(0, 0, "ff0000", "Red Line")
Graph.AddPoint(30, 30, "ff0000", "")
Graph.AddPoint(0, 0, "00ff00", "Green Line")
Graph.AddPoint(30, 20, "00ff00", "")
```

This would draw two lines, one in red and the other in green. Note that the name used in the legend is the first value of *Name* for each colour so an empty string can be used for the remaining entries.

The graph itself is produced with a call to one of the following methods: *GraphToBrowser*, *GraphToBitmap*, *GraphToFile*, *GraphToStream*.

### 4.2. Line Graph Properties

<b>GraphType</b>	-	Integer. Set to 3 for a line graph. (1)
<b>LineWidth</b>	-	Integer. Width of the graph lines in pixels. (1)
<b>ShowLine</b>	-	Boolean. Determines whether points are shown joined together. Set to false for a scatter diagram. (true)
<b>PointStyle</b>	-	Integer. Determines the type of point that will be drawn. 0 - none, 1 - dot, 2 - circle, 3 - diagonal cross, 4 - vertical cross. This property applies to all the graph lines. (0)
<b>PointSize</b>	-	Integer. Size of the plotted point. (2)
<b>XValuesVertical</b>	-	Boolean. Determines the orientation of the x-axis values. When true, the values are written up the page from bottom to top. (true)
<b>HideHGrid</b>	-	Boolean. Hides the horizontal grid lines, when <i>ShowGrid</i> is true. (false)
<b>HideVGrid</b>	-	Boolean. Hides the vertical grid lines, when <i>ShowGrid</i> is true. (false)
<b>PrefixX</b>	-	String. This value will be shown in front of the numeric values on the x-axis. (null)

<b>PrefixY</b> - (null)	String. This value will be shown in front of the numeric values on the y-axis.
<b>SuffixX</b> - (null)	String. This value will be shown after the numeric values on the x-axis.
<b>SuffixY</b> - (null)	String. This value will be shown after the numeric values on the y-axis.
<b>ShowSeparatorX</b> - (false)	Boolean. When true, commas will be used to separate thousands on the x-axis.
<b>ShowSeparatorY</b> - (false)	Boolean. When true, commas will be used to separate thousands on the y-axis.

There are a large number of properties that can be used with line graphs that are described in other sections of these instructions.

### 4.3. Line Graph Example

The following example shows how a line graph can be drawn using ASP.NET.

#### VB.NET

```
<%@ Page language="vb" debug=true %>
<%@ Import Namespace = "csASPNNetGraph" %>
<%@ import Namespace = "System.Drawing" %>
<%
Response.Buffer = true
Response.Expires = 0
Response.Clear
Dim Graph As New GraphClass
Graph.ShowGrid = true
Graph.YAxisText = "Y Axis"
Graph.XAxisText = "X Axis"
Graph.AxisTextFont = New Font("Arial", 8, FontStyle.Bold)
Graph.AddPoint(0, 0, "ff0000", "Red Line")
Graph.AddPoint(30, 30, "ff0000", "")
Graph.AddPoint(0, 0, "00ff00", "Green Line")
Graph.AddPoint(30, 20, "00ff00", "")
Graph.GraphType = 3
Graph.GraphToBrowser(1)
%>
```

#### C#

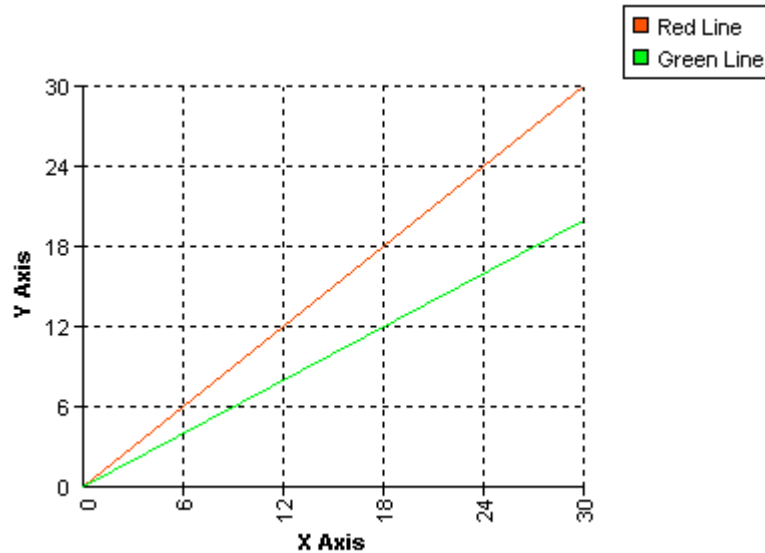
```
<%@ Page language="c#" debug=true %>
<%@ import Namespace = "csASPNNetGraph" %>
<%@ import Namespace = "System.Drawing" %>
<%
Response.Buffer = true;
Response.Expires = 0;
Response.Clear();
GraphClass Graph = new GraphClass();
Graph.ShowGrid = true;
Graph.YAxisText = "Y Axis";
Graph.XAxisText = "X Axis";
Graph.AxisTextFont = new Font("Arial", 8, FontStyle.Bold);
Graph.AddPoint(0, 0, "ff0000", "Red Line");
```



```

Graph.AddPoint(30, 30, "ff0000", "");
Graph.AddPoint(0, 0, "00ff00", "Green Line");
Graph.AddPoint(30, 20, "00ff00", "");
Graph.GraphType = 3;
Graph.GraphToBrowser(1);
%>

```



In this example all the data values are hard coded and the default property values are used. It is written for the full version of the component. Use the namespace "csASPNetGraphTrial" if the trial version is used.

The grid is displayed by setting *ShowGrid* to true. This uses the default style and colour settings. The axes are labelled by setting the *XAxisText* and *YAxisText* properties and the text for these labels has been specified as Arial font, size 8 and bold.

## 5. Stacked Bar Charts

Stacked bar charts are a variation of bar chart that allow the data to be grouped. The bars that make up a group are shown end on or "stacked".

There are two variations of stacked bar chart. The bars can be shown vertically or horizontally and the type of chart is defined by the *GraphType* property. *GraphType* must be set to 4 for a vertical stacked bar chart or 5 for a horizontal stacked bar chart.

When multiple graphs are drawn using the same instance of the object, call the *ClearData* method to remove the data items and start again.

### 5.1. Stacked Bar Chart Methods

**AddGroupedData** (*Group* As String, *Name* As String, *Value* As Double, *Colour* As String) - Each data item belongs to a *Group* and all items belonging to a *Group* will appear as part of the same bar. The *Group* may be shown under each bar. Each data item has a *Name* and this may be displayed in the legend. *Value* must be a positive number. *Colour* is a six character hexadecimal string giving the RGB value of the colour representing the data item. Data items with the same name should have the same colour although no checks are made to enforce this.

A separate call to *AddGroupedData* must be made for each data item in the graph.

Example:

```
Graph.AddGroupedData("January", "Red things", 17, "ff0000")
Graph.AddGroupedData("January", "Blue things", 28, "0000ff")
Graph.AddGroupedData("February", "Red things", 5, "ff0000")
Graph.AddGroupedData("February", "Blue things", 14, "0000ff")
```

This would add the data to draw two bars, one marked "January", and the other marked "February". Each bar would be split into two areas, one red and the other blue. If the legend is to be displayed it would have a red square marked "Red things" and a blue square marked "Blue things".

The graph itself is produced with a call to one of the following methods: *GraphToBrowser*, *GraphToBitmap*, *GraphToFile*, *GraphToStream*.

### 5.2. Stacked Bar Chart Properties

Some of the properties used for ordinary bar charts are also used by stacked bar charts, so refer to the section on bar chart properties for a description.

The properties which are unique to stacked bar charts control how the numeric values are shown for each data item, if they are to be displayed. These properties are described below.

<b>GraphType</b>	-	Integer property. This specifies the type of graph produced. Set to 4 for a stacked bar chart with vertical bars and to 5 for a stacked bar chart with horizontal bars.
<b>ShowStackedValue</b>	-	Boolean. When true each data item will have its value displayed next to the bar or inside the bar, depending on the value of <i>StackedTextAlign</i> . The following properties define the font and appearance. (false)
<b>StackedTextFont</b>	-	Font. Font used for the data values. ("Arial", Size 8)
<b>StackedTextBGColor</b>	-	String. Background colour of the data value text as a 6 digit hexadecimal string. ("ffffff")

<b>StackedTextColor</b>	-	String. Colour of the data value text as a 6 digit hexadecimal string. ("000000")
<b>StackedTextTransparent</b>	-	Boolean. When true the data values have a transparent background. (true)
<b>StackedTextAlign</b>	-	Integer. Determines the position and orientation of the data value relative to the bars. 0 - Centre Horizontal, 1 - Centre Vertical, 2 - Left Horizontal, 3 - Left Vertical, 4 - Right Horizontal, 5 - Right Vertical. (0)

The horizontal values mean the text reads from left to right, the vertical values read from bottom to top and the centre values show the text inside the bar. Note that when bars are horizontal the data values will only be shown horizontally and an alignment of left or right means above or below respectively.

### 5.3. Stacked Bar Chart Example

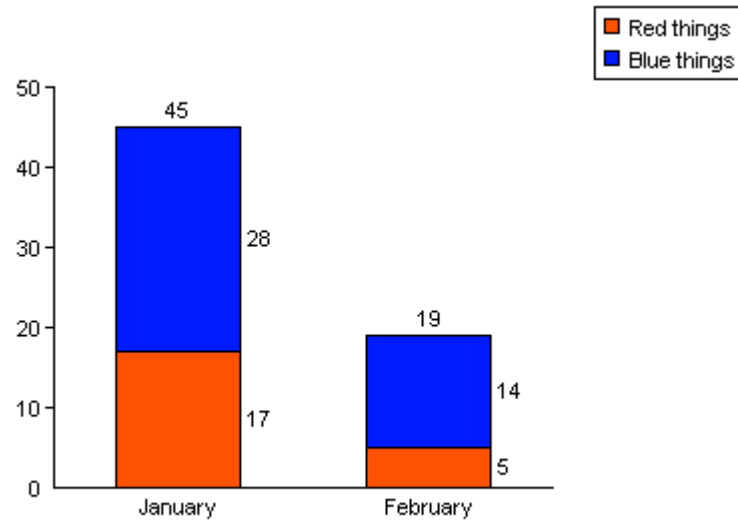
The following example shows how a stacked bar chart can be drawn using ASP.NET.

#### VB.NET

```
<%@ Page language="vb" debug="true" %>
<%@ Import Namespace = "csASPNETGraph" %>
<%
Response.Buffer = true
Response.Expires = 0
Response.Clear
Dim Graph As New GraphClass
Graph.ShowBarTotal = true
Graph.ShowStackedValue = true
Graph.StackedTextAlign = 4
Graph.AddGroupedData("January", "Red things", 17, "ff0000")
Graph.AddGroupedData("January", "Blue things", 28, "0000ff")
Graph.AddGroupedData("February", "Red things", 5, "ff0000")
Graph.AddGroupedData("February", "Blue things", 14, "0000ff")
Graph.GraphType = 4
Graph.GraphToBrowser(1)
%>
```

#### C#

```
<%@ Page language="c#" debug="true" %>
<%@ import Namespace = "csASPNETGraph" %>
<%
Response.Buffer = true;
Response.Expires = 0;
Response.Clear();
GraphClass Graph = new GraphClass();
Graph.ShowBarTotal = true;
Graph.ShowStackedValue = true;
Graph.StackedTextAlign = 4;
Graph.AddGroupedData("January", "Red things", 17, "ff0000");
Graph.AddGroupedData("January", "Blue things", 28, "0000ff");
Graph.AddGroupedData("February", "Red things", 5, "ff0000");
Graph.AddGroupedData("February", "Blue things", 14, "0000ff");
Graph.GraphType = 4;
Graph.GraphToBrowser(1);
%>
```



In this example all the data values are hard coded and the default property values are used. It is written for the full version of the component. Use the namespace "csASPNetGraphTrial" if the trial version is used.

The numbers above and to the side of the bars are not shown by default. The total above the bar is shown because *ShowBarTotal* has been set to true and the individual values are shown because *ShowStackedValue* is true. *StackedTextAlign* specifies where the values are displayed relative to the bars.

## 6. Miscellaneous Settings

### 6.1. Properties to Define the Axes of Bar and Line Graphs

The following properties apply to graphs with axes, such as bar charts and line graphs. They set the length of each axis including any negative amount, the position of the graph origin and they define the way each axis is calibrated.

Default property values are shown in brackets.

<b>OriginX</b>	-	Integer. X - coordinate of the origin, measured from the left of the image. (50)
<b>OriginY</b>	-	Integer. Y - coordinate of the origin, measured from the top of the image. (250)
<b>MaxX</b>	-	Integer. Length of the x-axis in pixels. (250)
<b>MaxY</b>	-	Integer. Length of the y-axis in pixels. (200)
<b>XAxisNegative</b>	-	Integer. Length of the negative x-axis in pixels. The length of the positive x-axis will be ( $MaxX - XAxisNegative$ ). Only line graphs can have a negative x-axis. (0)
<b>YAxisNegative</b>	-	Integer. Length of the negative y-axis in pixels. The length of the positive y-axis will be ( $MaxY - YAxisNegative$ ). (0)
<b>XTop</b>	-	Double. Maximum value on the x-axis. Auto calibrate if zero. (0)
<b>XGrad</b>	-	Double. Distance between the graduations on the x-axis as plotted values, not pixels. Auto calibrate if zero. (0)
<b>YTop</b>	-	Double. Maximum value on the y-axis. Auto calibrate if zero. (0)
<b>YGrad</b>	-	Double. Distance between the graduations on the y-axis as plotted values, not pixels. Auto calibrate if zero. (0)
<b>XOffset</b>	-	Double. The starting value on the x-axis. Set this when the range of values on the x-axis do not start from zero. (0)
<b>YOffset</b>	-	Double. The starting value on the y-axis. Set this when the range of values on the y-axis do not start from zero. (0)
<b>ShowGrid</b>	-	Boolean. When true, grid lines will be drawn on the graph level with the graduations on the axes. For bar charts only horizontal grid lines are shown. (false)
<b>GridStyle</b>	-	Integer. Determines the line type used for the grid lines. 0 - solid, 1 - dot, 2 - dash. (1)
<b>GridColor</b>	-	String. Colour of the grid lines as a 6 digit hexadecimal string. ("000000")
<b>XMarkSize</b>	-	Integer. The length of the graduation mark on the x-axis of a line graph. (4)
<b>YMarkSize</b>	-	Integer. The length of the graduation mark on the y-axis. (4)
<b>PlotAreaColor</b>	-	String. Background colour of the area enclosed by the axes as a 6 digit hexadecimal string. ("ffffff")
<b>ShowPlotBorder</b>	-	Boolean. When true, the plot area is enclosed by a rectangular border the colour of <i>GraphPen</i> . (false)
<b>XAxisText</b>	-	String. Label to display along the x-axis. (null)
<b>YAxisText</b>	-	String. Label to display along the y-axis. (null)
<b>AxisTextFont</b>	-	Font. Font to use for the axis text. ("Arial", Size 8)
<b>AxisTextBGColor</b>	-	String. Colour of the background to the axis text as a 6 digit hexadecimal string. ("ffffff")
<b>AxisTextColor</b>	-	String. Colour of the axis text. ("000000")
<b>AxisTextTransparent</b>	-	Boolean. When true the axis text has a transparent background. (false)

## 6.2. Legend Properties

The following properties control the appearance and position of the legend. They apply to all types of graph.

<b>LegendX</b>	-	Integer. X-coordinate of the legend box. (320)
<b>LegendY</b>	-	Integer. Y-coordinate of the legend box. (20)
<b>LegendAlign</b>	-	Integer. This specifies the part of the legend box to which <i>LegendX</i> and <i>LegendY</i> measure. 0 - Top Left, 1 - Top Centre, 2 - Top Right, 3 - Middle Left, 4 - Middle, 5 - Middle Right, 6 - Bottom Left, 7 - Bottom Centre, 8 - Bottom Right. (0)
<b>LegendVertical</b>	-	Boolean. When true the legend is a vertical list, when false it is a horizontal list. (true)
<b>LegendFont</b>	-	Font. Font used for the legend text. ("Arial", Size 8)
<b>LegendBGColor</b>	-	String. Background colour of the legend area as a 6 digit hexadecimal value. ("ffffff")
<b>LegendColor</b>	-	String. Colour of the text and lines in the legend area as a 6 digit hexadecimal value. ("000000")
<b>Square</b>	-	Integer. Size of the coloured square in the legend box. (8)
<b>Padding</b>	-	Integer. Spacing inside the legend box. (5)
<b>ShowLegend</b>	-	Boolean. Determines whether the legend is displayed. (true)
<b>ShowLegendBox</b>	-	Boolean. Determines whether the rectangle is displayed around the legend. (true)
<b>LegendInvert</b>	-	Boolean. The legend entries are usually shown in the order the data is entered. Setting <i>LegendInvert</i> to true reverses the order. Does not apply to line graphs. (false)
<b>LegendHideEmptyNames</b>	-	Boolean. Set to true to prevent legend entries from displaying when the label is an empty string. This option allows empty bars to be added to bar charts as spacers without affecting the legend. In line graphs it can allow a line to be hidden from the legend if no string is used in the <i>AddPoint</i> command. (false)

## 6.3. Changing the Size of a Graph

The size of the image produced by csASPNetGraph is defined by the *Width* and *Height* properties.

<b>Width</b>	-	Integer. Width of the graph image in pixels. Default value 400.
<b>Height</b>	-	Integer. Height of the graph image in pixels. Default value 300.

There are separate properties to cover the size and position of each feature on the graph and so other properties will need to be changed in addition to *Width* and *Height* to produce a different size of graph.

For a pie chart the pie diameter and position are determined by *PieDia*, *CenterX* and *CenterY*. For other types of graph the plotted area is located by *OriginX* and *OriginY*. The overall length of the axes is defined by *MaxX* and *MaxY*. On all types of graph the legend is located by *LegendX* and *LegendY*.

## 6.4. Standard Text and Annotations

This section describes the properties that control the standard text that annotates all the types of graph.

### 6.4.1. Labels (Axis Values and Pie Chart Labels)

The following properties apply to the values shown along the axes of bar charts and line graphs, as well as the annotation labelling each pie chart sector.

<b>LabelFont</b>	-	Font. Font used for the labels. ("Arial", Size 8)
<b>LabelBGColor</b>	-	String. Background colour of the label text as a 6 digit hexadecimal string. ("ffffff")
<b>LabelColor</b>	-	String. Colour of the label text as a 6 digit hexadecimal string. ("000000")
<b>LabelTransparent</b>	-	Boolean. When true the label text has a transparent background. (false)

The following properties determine whether the labels are displayed.

<b>ShowNumbers</b>	-	Boolean. Determines whether numerical values are shown on the axes for bar and line graphs, or next to the sectors on pie charts. (true)
<b>ShowLabel</b>	-	Boolean. Determines whether the names of each data item are displayed on bar and pie charts. (true)

### 6.4.2. Title Text

A line of text can be added as a title. By default this is drawn at the top left corner of the image in bold but it can be positioned as required and the font can be specified.

<b>Title</b>	-	String. Single line of text to be displayed if required. The properties controlling the font and position are described below. (null)
<b>TitleX</b>	-	Integer. X-coordinate of the title text alignment point. (0)
<b>TitleY</b>	-	Integer. Y-coordinate of the title text alignment point. (0)
<b>TitleFont</b>	-	Font. Font used for the title. ("Arial", Size 8, Bold)
<b>TitleBGColor</b>	-	String. Background colour of the title text as a 6 digit hexadecimal string. ("ffffff")
<b>TitleColor</b>	-	String. Colour of the title text as a 6 digit hexadecimal string. ("000000")
<b>TitleTransparent</b>	-	Boolean. When true the title text has a transparent background. (false)
<b>TitleTextAlign</b>	-	Integer. This specifies the alignment point of the title text which is the position within the text string that is located by <i>TitleX</i> and <i>TitleY</i> . 0 - Top Left, 1 - Top Centre, 2 - Top Right, 3 - Baseline Left, 4 - Baseline Centre, 5 - Baseline Right, 6 - Bottom Left, 7 - Bottom Centre, 8 - Bottom Right. (0)

### 6.4.3. The Prefix, Suffix and ShowSeparator Properties

The properties described in this section control the formatting of numerical values on all types of graphs. *Prefix* and *Suffix* allow strings or single characters to be added at the start or end of all values, allowing currency symbols or other units to be displayed. *ShowSeparator* toggles the display of the thousands separator symbol as defined in the regional settings.

<b>Prefix</b>	-	String. This value will be shown in front of all numeric values. It may be used to display currency symbols, for example. (null)
<b>Suffix</b>	-	String. This value will be added at the end of each numeric value. It may be used to display a units symbol. (null)
<b>ShowSeparator</b>	-	Boolean. When true, the numeric values will be formatted to separate thousands using the separator defined by the regional settings. This is usually a comma or a dot. (false)

In a line graph, these properties will be applied to values on both the x and y axes, which might not be required. Properties have been provided in the section on line graphs to allow a prefix, suffix or separator to be applied to a single axis.

## 6.5. Decimal Places

The number of decimal places displayed in each graph is specified by the *Decimals* property, which defaults to zero. This property must be set to a non-zero value for decimal places to show in plotted values.

<b>Decimals</b>	-	Integer. The number of decimal places shown for numeric values. To display a fixed number of decimal places prefix the value with a "-" sign. (0)
For example, to show currency values with 2 fixed decimal places, set <i>Decimals</i> to -2. This will display 2 decimal places with trailing zeroes where appropriate.		
The symbol used for the decimal point will be taken from the regional settings on the server.		

## 6.6. Adding Extra Text and Annotations to a Graph

csASPNetGraph contains functionality to add extra text and lines to the graph to provide customised annotation.

### 6.6.1. AddText and AddExtraLine

*AddText* and *AddExtraLine* will draw text and lines on any type of graph and these features are positioned using pixel coordinates, measured from the top left of the graph image.

<b>AddText</b>	( <i>Text</i> As String, <i>X</i> As Integer, <i>Y</i> As Integer, <i>Angle</i> As Single)	-	<i>Text</i> is the text string, <i>X</i> and <i>Y</i> are the coordinates measured from the top left of the graph image and <i>Angle</i> is the rotation angle of the text. This text font is defined by the property <i>TextFont</i> .
<b>AddExtraLine</b>	( <i>X1</i> As Integer, <i>Y1</i> As Integer, <i>X2</i> As Integer, <i>Y2</i> As Integer, <i>Thickness</i> As Integer, <i>PenStyle</i> As Integer, <i>Colour</i> As String)	-	This adds a line to the graph. <i>X1</i> , <i>Y1</i> are the coordinates of the starting point and <i>X2</i> , <i>Y2</i> are the coordinates of the end point. <i>Thickness</i> is the line thickness in pixels where 0 or 1 both give a line of 1 pixel width. <i>PenStyle</i> is an integer value defining the line type, 0 - solid, 1 - dot, 2 - dash. <i>Colour</i> is a 6 character string for colour of the line.

Each of the functions above can be called multiple times to display multiple text strings or lines.

The following properties define the font and colour used by the *AddText* command. All the text drawn by *AddText* has the same properties.

<b>TextFont</b>	-	Font. Font to use for the extra text drawn using <i>AddText</i> . ("Arial", Size 8)
<b>TextTransparent</b>	-	Boolean. When true the extra text has a transparent background. (false)
<b>TextColor</b>	-	String. Colour of the extra text. ("000000")
<b>TextBGColor</b>	-	String. Colour of the background to the extra text as a 6 digit hexadecimal string. ("ffffff")

### 6.6.2. Adding Text to Line Graphs

Text can be added to line graphs using the *AddLineGraphText* method and the text is positioned using the graph coordinates, making it easy to annotate the graph near the plotted points. Related properties give options for drawing a border around the text and a leader line to another point.

<b>AddLineGraphText</b>	( <i>Text</i> As String, <i>X</i> As Single, <i>Y</i> As Single, <i>Angle</i> As Single)	-	This function adds a text string to be added to a line graph. <i>Text</i> is the text string. <i>X</i> and <i>Y</i> are the coordinates of the string relative to the line graph origin. <i>Angle</i> is the angle of rotation of the string.
-------------------------	--	---	---



The following properties apply to all the text strings added by *AddLineGraphText*. The default values are in brackets.

<b>LineGraphTextFont</b> -	Font. The font used by the line graph text. ("Arial", Size 8)
<b>LineGraphTextBGColor</b> -	String. Background colour of the text as a 6 digit hexadecimal string. ("ffffff")
<b>LineGraphTextColor</b> -	String. Colour of the text. ("000000")
<b>LineGraphTextAlign</b> -	Integer between 0 and 8. Alignment of the text relative to the point (X, Y). 0 – Top Left, 1 – Top Centre, 2 – Top Right, 3 – Baseline Left, 4 – Baseline Centre, 5 – Baseline Right, 6 – Bottom Left, 7 – Bottom Centre, 8 – Bottom Right. (0)
<b>LineGraphTextXOffset</b> -	Integer. Horizontal distance in pixels between the point (X, Y) and the point where the text is actually drawn. (0)
<b>LineGraphTextYOffset</b> -	Integer. Vertical distance in pixels between the point (X, Y) and the point where the text is actually drawn. (0)
<b>LineGraphTextTransparent</b> -	Boolean. When true the line graph text has a transparent background. (false)
<b>LineGraphTextBorder</b> -	Boolean. Determines whether a border is drawn around the text. This is only drawn when the text is at an angle of 0 or 90 degrees. (false)
<b>LineGraphTextLeader</b> -	Boolean. Determines whether a leader line is drawn between the point (X, Y) and the alignment point of the text. (false)
<b>LineGraphBorderColor</b> -	String. Colour of the border around the text. ("000000")
<b>LineGraphLeaderColor</b> -	String. Colour of the leader line. ("000000")

A typical use of line graph text is to display the values next to a plotted point. *AddLineGraphText* would be called for each point that requires a value to be displayed. *LineGraphTextXOffset*, *LineGraphTextYOffset* and *LineGraphTextAlign* could be set to position each text item a specified distance from the plotted point.

For example, if *X* and *Y* are variables, the following two lines might be used inside a loop to plot a point and display the *X* and *Y* values at that point:

```
Graph.AddPoint(X, Y, "ff0000", "Red Line")
Graph.AddLineGraphText("(" & X & ", " & Y & ")", X, Y, 0)
```

The following property settings would be outside the loop because they apply globally to all the line graph text. The value for *LineGraphTextYOffset* will move each item up by 30 pixels. There will be a box drawn around each item and a leader line will connect the point *X, Y* with the text alignment point. The text is aligned "Bottom Centre" so that the middle of the text will be immediately above the point it marks.

```
Graph.LineGraphTextAlign = 7
Graph.LineGraphTextYOffset = 30
Graph.LineGraphBorder = true
Graph.LineGraphLeader = true
```

There is no corresponding function for placing text on bar charts or pie charts. Any additional annotation must be placed using *AddText* or *AddExtraLine*.

## 6.7. Substituting Axis Labels

It is possible to replace the numeric values on the axes with specified string values. In the case of bar charts and stacked bar charts this means the y-axis values. On line graphs the values on both axes can be replaced with strings. There are several reasons why this might be done but generally it is to achieve a labelled axis that cannot be produced in the normal way.

The properties *UseXAxisLabels* or *UseYAxisLabels* are set to true and then values on the axis are replaced by calling the *AddXValue* or *AddYValue* methods.

<b>UseXAxisLabels</b>	-	Boolean property. Set to true when text values are to be used on the x-axis instead of numbers. (false)
<b>UseYAxisLabels</b>	-	Boolean property. Set to true when text values are to be used on the y-axis instead of numbers. (false)
<b>AddXValue</b> ( <i>X</i> As Double, <i>Label</i> As String)	-	This method replaces the numeric value <i>X</i> on the x-axis of a line graph with the string value <i>Label</i> . <i>UseXAxisLabels</i> must be true for it to have any effect. Note that the value <i>X</i> must be on the x-axis and only then will <i>Label</i> be able to replace it.
<b>AddYValue</b> ( <i>Y</i> As Double, <i>Label</i> As String)	-	This method replaces the numeric value <i>Y</i> on the y-axis with the string value <i>Label</i> . <i>UseYAxisLabels</i> must be true for it to have any effect. Note that the value <i>Y</i> must be on the y-axis and only then will <i>Label</i> be able to replace it.
<b>ClearAxisLabels</b> ( )	-	This method removes the values added by the previous methods.

Example:

```
Graph.UseXAxisLabels = true
Graph.AddXValue(0, "Jan")
Graph.AddXValue(1, "Feb")
Graph.AddXValue(2, "Mar")
```

When the graph is plotted, the values 0, 1 and 2 on the x-axis will be replaced with the strings "Jan", "Feb" and "Mar". It is important to realise that it is the axis labels that are replaced with strings, not the plotted values.

## 6.8. Plotting Dates or Times

Some line graphs require the display of dates or times along one axis. Often the most effective way to achieve this is to use axis label substitution as described above. The data may be simplified before plotting, for example by plotting years or months as integer values, with a starting point of zero.

It is possible to use csASPNetGraph to plot values as *DateTime* values and the corresponding date or time will be displayed on the axis. This is done by setting either the *UseXAxisDates* or *UseYAxisDates* properties to true. The integer part of a *DateTime* counts the number of days that have elapsed since 30th December 1899 and the decimal part represents the time of the day. This makes calibration of the axis difficult because modern dates are large numbers and the data points will be relatively close together. When used along the x-axis, it will be necessary to set *XOffset*, *XTop* and *XGrad* to specify the first point, the last point, and the interval of the calibration marks. The *DateTimeFormat* property determines whether the values are treated as a combined date and time, a date or a time.

The *AddPoint* method requires the data point to be added as Double values (double precision real numbers). To convert a *DateTime* value to a Double use the *DateTime.ToOADate* method.

**For example, in VB.NET**

```
Dim TempDate As New DateTime(2000, 01, 01)
```

```
Graph.AddPoint(TempDate.ToOADate, 10, "FF0000", "Example")
Graph.AddPoint(DateTime.Parse("02-01-2000").ToOADate, 20, "FF0000",
" ")
```

This shows how to create a `DateTime` and convert it to a `Double`, and also how to convert a string into a `DateTime` and then to a `Double`.

<b>UseXAxisDates</b>	-	Boolean. Set to true when values on the x-axis are to be plotted as <code>DateTime</code> values. (false)
<b>UseYAxisDates</b>	-	Boolean. Set to true when values on the y-axis are to be plotted as <code>DateTime</code> values. (false)
<b>DateTimeFormat</b>	-	Integer, value 0, 1 or 2. This property determines whether the values shown on the axis are a combined date/time, a date or a time. 0 - date/time, 1 - date, 2 - time. (0)

The exact formatting is determined by the properties *DateFormatString* and *TimeFormatString*. The full range of possible format strings are not described here.

<b>DateFormatString</b>	-	String. Specifies the display format of a date. When null the system settings for a short date are used. (null)
<b>TimeFormatString</b>	-	String. Specifies the display format of a time. When null the system settings for a long time are used. (null)

For date formatting the letters *y*, *m* and *d* are used for year, month and day. The number of each letter determines how many characters are used for each quantity and the characters in between specify the delimiter.

Example

```
"dd:mm:yy"      -      01:07:05
"dd-mmm-yyyy"   -      01-Jul-2005
```

For time formatting the letters *h*, *n*, *s* and *z* are used for hours, minutes, seconds and milliseconds. The suffix "am/pm" will use a 12 hour display and will append "am" or "pm" as appropriate.

Example

```
"hh:nn:ss"      -      14:05:37
"h:nn:ss am/pm" -      2:05:37 pm
```

Characters inside either format string enclosed in single or double quotes are displayed literally.

The following VB.NET example shows how some dates can be plotted.

```
Dim Graph As New GraphClass()
Dim TempDate As New DateTime(2000, 01, 01)
Graph.AddPoint(TempDate.ToOADate, 10, "FF0000", "Example")
Graph.AddPoint(DateTime.Parse("02-01-2000").ToOADate, 20, "FF0000",
" ")
Graph.AddPoint(DateTime.Parse("03-01-2000").ToOADate, 18, "FF0000",
" ")
Graph.AddPoint(DateTime.Parse("04-01-2000").ToOADate, 15, "FF0000",
" ")
```

```

Graph.AddPoint(DateTime.Parse("05-01-2000").ToOADate, 7, "FF0000",
"")
Graph.UseXAxisDates = true
Graph.XOffset = TempDate.ToOADate
Graph.XGrad = 1
Graph.XTop = Graph.XOffset + 4
Graph.DateTimeFormat = 1
Graph.DateFormatString = "dd-mmm-yy"

```

## 6.9. Displaying Percentages on Bar and Pie Charts

There is an option to show the data in pie and bar charts as percentages of the total values rather than absolute values. Set the *ShowPercent* property to true for this type of graph.

<b>ShowPercent</b>	-	Boolean. When true on bar charts and pie charts, the data values will be shown as percentages of the total. (false)
--------------------	---	---

Example:

```

Graph.AddData("Item 1", 1, "ff0000")
Graph.AddData("Item 2", 2, "00ff00")
Graph.AddData("Item 3", 5, "0000ff")
Graph.ShowPercent = true
Graph.Decimals = 1

```

When these values are displayed in a pie chart or line graph they will be shown as 12.5%, 25% and 62.5%. Do not use negative values if *ShowPercent* is true. They will be treated as positive.

## 6.10. Random Colours

Bar charts and pie charts can use a system of random colours instead of specifying the colour in the *AddData* command. To implement this, set *UseRNDColor* to true before calling *AddData*. A colour value must still be specified in *AddData* as a dummy parameter to prevent an input error, but a random colour will be used. The random colours consist of 211 web safe colours in a pseudo random sequence, the starting point of which is determined by the *RNDColor* property, which takes a value between 0 and 210 with a default of zero. The same sequence of colours will always be produced for a given value of *RNDColor*. The value of *RNDColor* needs to be set before using the *AddData* method.

<b>UseRNDColor</b>	-	Boolean. When true, pie charts and bar charts will use a random colour instead of the colour specified in the <i>AddData</i> command. (false)
--------------------	---	---

<b>RNDColor</b>	-	Integer in the range 0 to 210. The starting point in the sequence of pseudo random colours used when <i>UseRNDColor</i> is true. (0)
-----------------	---	--

Example:

```

Graph.UseRNDColor = true
Graph.RNDColor = 25
Graph.AddData("Item 1", 17, "")
Graph.AddData("Item 2", 28, "")
Graph.AddData("Item 3", 5, "")

```

The colours are not truly random and two identical colours will not appear next to each other in the sequence. Random colours cannot be used on line graphs and stacked bar charts because with these types of graph the colour plays a role in organising the data.

## 6.11. Miscellaneous Display Properties and Functions

The properties described in this section apply to the overall appearance.

<b>BGColor</b>	-	String. This is the background colour for the graph image as a 6 digit hexadecimal string. On a line graph there is an additional background colour for the plotted area defined by <i>PlotAreaColor</i> . ("ffffff")
<b>GraphPen</b>	-	String. The colour of the lines used to draw the axes, the outline of bars and pie charts. ("000000")
<b>ClearData ( )</b>	-	This method removes any data values or points that have been added ready to produce a new graph. It is rarely needed in an ASP.NET script where the class is created new each time, but it is important in a Windows Forms application where the class instance is reused.

## 6.12. The Dummy Graph Function

Sometimes it is useful to find information about the graph that will be plotted. The *DummyGraph* function goes through the process of producing the graph without exporting an image. This will set a number of properties which can be read and then the graph can be produced properly later. These properties include the position of the first and last bar in a bar chart, the number of pixels per graph unit in each direction, and the calibration details of each axis when automatic calibration has been used. These properties can be used with *AddText*, and *AddExtraLine* for annotating a graph. They can be used with *AddXValue* and *AddYValue* when the labels are substituted on the axes, as described in Section 6.7.

The *ImageMapText* property is set when *DummyGraph* is called. This is described in Section 9, "Image Maps".

<b>DummyGraph ( )</b>	-	Function which performs the calculations needed to produce a graph. A number of read only properties are set by this function call.
-----------------------	---	---

The following properties are set by a call to *DummyGraph*, or when any other graph drawing command is called, such as *GraphToBrowser*, *GraphToFile* etc.

<b>XPixelsPerUnit-</b> the X direction.	Real, read only. Returns the number of pixels used for every graph unit in the X direction.
<b>YPixelsPerUnit-</b> the Y direction.	Real, read only. Returns the number of pixels used for every graph unit in the Y direction.
<b>FirstBarPos</b>	- Integer, read only. Returns the distance, in pixels, between the left side of the image and the middle of the first bar, in a bar chart or stacked bar chart using vertical bars. When the bars are horizontal it is the distance between the top of the image and the middle of the first bar.
<b>LastBarPos</b>	- Integer, read only. Returns the distance, in pixels, between the left side of the image and the middle of the last bar, in a bar chart or stacked bar chart using vertical bars. When the bars are horizontal it is the distance between the top of the image and the middle of the last bar.
<b>ReadBarGap</b>	- Integer, read only. Returns the size, in pixels, of the gap between bars in a bar chart. This will be different from the <i>BarGap</i> property if <i>BarGap</i> was zero for auto calibration.
<b>ReadBarWidth</b>	- Integer, read only. Returns the width, in pixels, of the bars in a bar chart. This will be different from the <i>BarWidth</i> property if <i>BarWidth</i> was zero for auto calibration.
<b>ReadXGrad</b>	- Real, read only. Returns the size of the graduations on the x axis of line graphs. This will be different from the <i>XGrad</i> property if <i>XGrad</i> was zero for auto calibration.

**ReadYGrad** - Real, read only. Returns the size of the graduations on the y axis. This will be different from the *YGrad* property if *YGrad* was zero for auto calibration.

**ReadXTop** - Real, read only. Returns the maximum value on the x axis of line graphs. This will be different from the *XTop* property if *XTop* was zero for auto calibration.

**ReadYTop** - Real, read only. Returns the maximum value on the y axis. This will be different from the *YTop* property if *YTop* was zero for auto calibration.

## 7. Plotting Lines on Bar Charts

Version 1.1 of csASPNetGraph introduced a feature where data points and lines can be added to bar charts. The *AddPoint* method is used to specify each point in the same way as for a line graph. The *ShowLinesWithBars* property must be set to true and at least one value must have been entered using *AddPoint*. Multiple lines can be drawn by using more than one colour. The name used for the first point on each line will appear in the legend. This legend entry can be hidden by using an empty string for the name and setting *LegendHideEmptyNames* to true.

**ShowLinesWithBars** - Boolean. When true, data points and lines will be shown on bar charts if any points have been defined using *AddPoint*. The first point using a particular colour will be centred on the first bar on the graph, the second point for that colour will be centred on the second bar, and so on. The X parameter for *AddPoint* is not used to position the point. The properties *ShowLine*, *LineWidth*, *PointSize* and *PointStyle* all control the appearance of the points and lines. (false)

### 7.1. Bar Chart Example With a Plotted Line

The following example shows how a bar chart can be drawn with a plotted line, using ASP.NET.

#### VB.NET

```
<%@ Page language="vb" debug="true" %>
<%@ Import Namespace = "csASPNetGraph" %>
<%
Response.Buffer = true
Response.Expires = 0
Response.Clear
Dim Graph As New GraphClass
Graph.Title = "Bar Chart Example"
Graph.TitleX = 120
Graph.ShowBarTotal = true
Graph.AddData("Item 1", 17, "ff0000")
Graph.AddData("Item 2", 28, "00ff00")
Graph.AddData("Item 3", 5, "0000ff")
Graph.ShowLinesWithBars = true
Graph.AddPoint(0, 5, "ff00ff", "Line")
Graph.AddPoint(0, 20, "ff00ff", "")
Graph.AddPoint(0, 2, "ff00ff", "")
Graph.LineWidth = 3
Graph.PointSize = 4
Graph.PointStyle = 1
Graph.GraphType = 1
Graph.GraphToBrowser(1)
%>
```

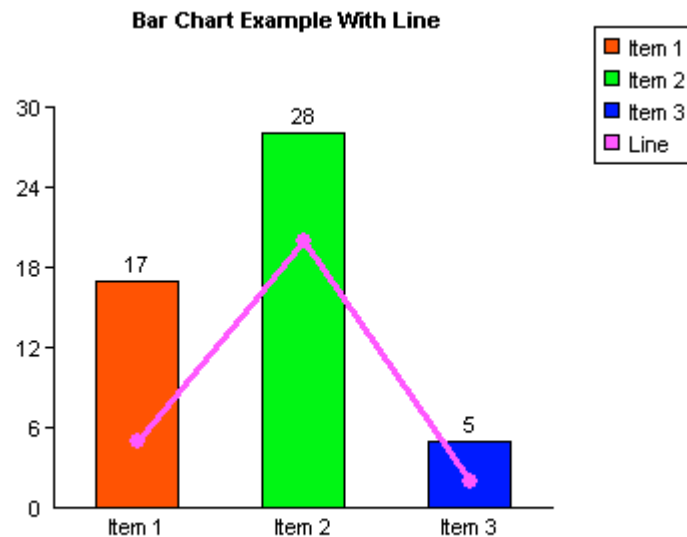
#### C#

```
<%@ Page language="c#" debug="true" %>
<%@ import Namespace = "csASPNetGraph" %>
<%
Response.Buffer = true;
Response.Expires = 0;
Response.Clear();
GraphClass Graph = new GraphClass();
Graph.Title = "Bar Chart Example";
Graph.TitleX = 120;
Graph.ShowBarTotal = true;
Graph.AddData("Item 1", 17, "ff0000");
Graph.AddData("Item 2", 28, "00ff00");
```

```

Graph.AddData("Item 3", 5, "0000ff");
Graph.ShowLinesWithBars = true;
Graph.AddPoint(0, 5, "ff00ff", "Line");
Graph.AddPoint(0, 20, "ff00ff", "");
Graph.AddPoint(0, 2, "ff00ff", "");
Graph.LineWidth = 3;
Graph.PointSize = 4;
Graph.PointStyle = 1;
Graph.GraphType = 1;
Graph.GraphToBrowser(1);
%>

```



This example uses the same values as the bar chart example in Section 3. It is written for the full version of the component. Import the namespace "csASPNNetGraphTrial" if the trial version is used.

The *ShowLinesWithBars* property is set to true and *AddPoint* has been used to add three data points. The X parameter for *AddPoint* has been set to zero for all three points although any could have been used because it is ignored for this type of graph. The first point is drawn half way along the first bar, the second is drawn half way along the second bar, etc. The maximum number of points that can be drawn for each line is the same as the number of bars. It would be possible to draw another line of three points by calling *AddPoint* three more times and specifying another colour.

Note that the values of the data points apply to the same y-axis as the bar values. It is not possible to draw a second y-axis with different values.



## 8. Displaying or Exporting the Graph Image

After adding the data and setting any required properties, the graph is generated using one of the methods described below.

**GraphToBrowser** (*ImageType* As Integer) - This is used in ASP.NET to stream the graph image to the browser. It sets the content type to the appropriate value and sends the image through Response.BinaryWrite in the format specified by *ImageType*. It is important that the script using *GraphToBrowser* contains no other output and no HTML tags.

*ImageType* can be one of the following values, 0 - BMP, 1 - GIF, 2 - PNG, 3 - JPG, 4 - TIF.

**GraphToBitmap** ( ) - This returns the graph image as a Bitmap. It can be used in a Windows Forms application to copy the graph into a visual control, or it can be used if further image editing is required.

**GraphToFile** (*Filename* As String) - The graph image is saved to disk where *Filename* is the physical path to the file, complete with the extension. The extension determines the file format.

In ASP.NET applications, the ASPNET Machine Account must have Write permission in the destination directory when *GraphToFile* is used.

**GraphToStream** (*ImageType* As Integer) - This returns the graph image as a MemoryStream. *ImageType* specifies the image format.

**GraphToArray** (*ImageType* As Integer) - This returns the graph image as an array of bytes. *ImageType* specifies the image format. This format can be used by the Response.BinaryWrite command.

When the graph image is exported in GIF format, a single colour can be specified as transparent. The *TransColor* and *Transparent* properties must be set.

**TransColor** - String. Transparent colour when the graph is exported as a GIF. ("ffffff")

**Transparent** - Boolean. When true an exported GIF will use transparency and the transparent colour will be *TransColor*. (false)

## 9. Displaying Graphs in a Web Browser

An ASP.NET page will return HTML output by default. An HTML page is formatted text which can include spaces to display images. The images themselves are not part of the HTML but are separate files, the location of which is specified inside the <IMG> tag. An ASP.NET page can be an image if the ContentType is set to "image/gif" and the binary data of the image is output using the Response.BinaryWrite command. The image is generated by placing the path to the script inside the <IMG> tag.

For example, this page will display the image produced by "chart.asp":

```
<html>
<head><title>HTML page containing an image</title></head>
<body>

</body>
</html>
```

Graph.asp may look like this:

```
<%@ Page language="vb" debug=true %>
<%@ Import Namespace = "csASpNetGraph" %>
<%
Response.Buffer = true
Response.Expires = 0
Response.Clear
Dim Graph As New GraphClass
Graph.Title = "Example Pie Chart"
Graph.AddData("Item 1", 17, "ff0000")
Graph.AddData("Item 2", 28, "00ff00")
Graph.AddData("Item 3", 5, "0000ff")
Graph.GraphType = 0
Graph.GraphToBrowser(1)
%>
```

The *GraphToBrowser* method sets the content type to match the image type parameter. In this case it is "image/gif", because the parameter is 1. The *GraphToBrowser* method is the equivalent of:

```
Response.ContentType = "image/gif"
Response.BinaryWrite(Graph.GraphToArray(1))
```

When the first HTML page is loaded it looks for the image at "graph.asp", runs the script and is sent a stream of binary data in GIF format, so the browser displays the image. It is not possible to place the BinaryWrite command inside the IMG tag to produce the image, it must be in a separate file.

It is useful to know that parameters can be passed to the ASP image script using the URL string and this can be read using Request.QueryString and used somewhere in the script.

Example:

```

```

Other parameters can be passed, for example, a user ID for opening a database, or some other value to assist in plotting the graph. Remember that URL parameters are visible to a user if they view the source code, so no sensitive information should be passed.

## 10. Image Maps

The HTML code required to make an image map to accompany a graph can be generated using csASPNetGraph, as described below.

### 10.1. Image Map Methods

**AddMapArea** (*URL* As String, *Hint* As String, *Extra* As String) - This method creates an <area> tag for the image map. *URL* is the URL for the href attribute, *Hint* is the string for the title attribute and *Extra* is a string that will be written into the tag before the closing ">" and it can contain any other attributes such as the target or onMouseOver. Usually *AddMapArea* will be called once for each data item in the chart.

If an empty string is used for the *URL* parameter, no href attribute will be used.

**SetImageMapParams** (*Name* As String, *DataArea* As Boolean, *MapLabel* As Boolean, *Legend* As Boolean) - This method sets some parameters that will be used when generating the image map. If it is not called, default values will be used. *Name* is the image map name and will default to "map1" if not called. *DataArea*, *MapLabel* and *Legend* are all Boolean values to determine which map areas are drawn. If *DataArea* is true an area tag will be written for the pie segment, bar or data point, depending on the graph type. If *MapLabel* is true an area tag will be written for the text label next to the pie chart or under the bar. If *Legend* is true an area tag will be written for the coloured square in the legend box, and also for the text in the legend box. All three parameters are true by default so four area tags will be written for each *AddMapArea* call. The *MapLabel* and *Legend* parameters are ignored for line graphs and stacked bar charts because map areas are never created for the labels or legends of these types of graphs.

### 10.2. Image Map Properties

**ImageMapExtra** - String. The value of this property will be written into the HTML of the image map between the last automatically generated <area> tag and the closing </map> tag. Set it to include any additional area tags or a default area.

**ImageMapText** - String, read only. This is the HTML code for the image map. It has no value until the graph has been drawn. Examples of commands which draw a graph are *GraphToBrowser*, *GraphToFile*, *GraphToBitmap* and *DummyGraph*.

### 10.3. Generating an Image Map

There is a complication when using csASPNetGraph to produce an image map dynamically and it requires a change to the sequence of events. As usual, two scripts are needed to generate and display the graph. One is the "outer" page, which is HTML containing an <img> tag. The other is the "embedded" page, which runs the component, produces the graph and returns binary data to be interpreted as an image by the browser. This is described more fully in the previous section.

The image map must go in the "outer" page because it is HTML code, so the component must be called in this page. There are ways of doing this but each has some disadvantages.

A simple method is to write the graph to a file and link to the file. Here is an example of a script that does this.

```
<%@ Page language="vb" debug=true %>
<%@ Import Namespace = "csASPNetGraph" %>
<%
Dim Graph As New GraphClass
Graph.Title = "Example Pie Chart"
```

```

Graph.AddData("Item 1", 17, "ff0000")
Graph.AddData("Item 2", 28, "00ff00")
Graph.AddData("Item 3", 5, "0000ff")
Graph.AddMapArea("url1.asp", "Data 1", "target_blank")
Graph.AddMapArea("url2.asp", "Data 2", "")
Graph.AddMapArea("url3.asp", "Data 3", "")
Graph.GraphType = 0
Graph.GraphToFile(Server.MapPath("./temp.gif"))
%>
<html>
<head>
<title>Test with image map</title>
</head>
<body>
<%
Response.Write(Graph.ImageMapText)
%>

</body>
</html>

```

This technique works when the graph image can be shared by different users, but if the graph is unique to each user, a unique file name would be required and some way would need to be found to delete the file after use. A possible solution would be to use file names in a numeric sequence and store the current value as an Application variable. This sequence could reset after a specific number of files so that each user gets a different file name from the previous user, but after a certain length of time the file names get reused and the files overwritten.

Another approach is to repeat the graph generating code so that it is called in both the outer page and the embedded page. This could be impractical if the code is complex. The advantage to this method would be that there would be no temporary file to delete.

The example shown above is for a pie chart with three data items. Note that the calls to *AddMapArea* are matched with *AddData* methods in order. If a fourth *AddMapArea* command was used it would be ignored. The first *AddMapArea* command uses the Extra parameter to add a target attribute to the <area> tag. The other *AddMapArea* commands do not use this parameter and so an empty string must be used to maintain correct syntax.

There is no call to *SetImageMapParams* so the defaults are used. The image map is called "map1" and the hotspots on the image are the pie sectors, the labels, and the legend entries.

Image maps can be produced with line graphs and stacked bar charts but the legend and labels are not used as hot spots. On a line graph, the hot spot is an area around each data point which is double the radius of the plotted point. On a stacked bar chart, each data area within a bar is a hot spot.

Note that the <map> tag will have an id attribute with the same value as the name attribute. The <area> tag will have an alt attribute with the same value as the title attribute. This enables the HTML to validate.

## 11. Sample Graphs

A selection of graphs are shown on our demonstration web site to show how different property settings affect the appearance - <http://demo.chestysoft.com/demos.aspx>. There is also a downloadable example using a MS Access database as well examples using image maps.

Bar Charts - <http://demo.chestysoft.com/aspnet/aspnetgraph/barcharts.aspx>

Pie Charts - <http://demo.chestysoft.com/aspnet/aspnetgraph/piecharts.aspx>

Line Graphs - <http://demo.chestysoft.com/aspnet/aspnetgraph/linegraphs.aspx>

Stacked Bar Charts - <http://demo.chestysoft.com/aspnet/aspnetgraph/stackedbarcharts.aspx>

Database Example - <http://demo.chestysoft.com/aspnet/aspnetgraph/databasedemo.aspx>

Image Map 1 - <http://demo.chestysoft.com/aspnet/aspnetgraph/imagemap.aspx>

Image Map 2 - <http://demo.chestysoft.com/aspnet/aspnetgraph/imagemap2.aspx>

There is also a VB.NET Visual Studio Example.

VS/VB.NET - <http://www.chestysoft.com/aspnetgraph/vsvbdemo.asp>

## 12. Other Products From Chestysoft

Visit the Chestysoft web site for details of other COM objects.

### ActiveX Controls

- [csXImage](#) - An OCX control to display, edit and scan images.
- [csXGraph](#) - An OCX control to draw pie charts, bar charts and line graphs.
- [csXThumbUpload](#) - Upload multiple files by HTTP or FTP with previews and image edits.
- [csXPostUpload](#) - Uploads batches of files from a client to a server using an HTTP post.
- [csXMultiUpload](#) - Select and upload multiple files and post to a server using HTTP.

### ASP Components

- [csImageFile](#) - Resize, edit and create images in ASP.
- [csDrawGraph](#) - Component to draw pie charts, bar charts and line graphs.
- [csASPGif](#) - Create and edit animated GIFs.
- [csIniFile](#) - Read and Edit Windows style inifiles.
- [csASPUpload](#) - Process file uploads through a browser.
- [csASPZipFile](#) - Create zip files and control binary file downloads.
- [csFileDownload](#) - Control file downloads with an ASP script.
- [csFTPQuick](#) - ASP component to transfer files using FTP.

### ASP.NET

- [csNetUpload](#) - ASP.NET component for saving HTTP uploads.
- [csNetDownload](#) - ASP.NET class to control file downloads.

### Web Hosting

We can offer ASP/ASP.NET enabled web hosting with our components installed. [Click for details.](#)

## 13. Alphabetical List of Commands

Command	Page no.	Command	Page no.
AddData (Bar Charts)	12	LegendInvert	22
AddData (Pie Charts)	9	LegendVertical	22
AddExtraLine	24	LegendX	22
AddGroupedData	18	LegendY	22
AddLineGraphText	24	LineGraphBorderColor	25
AddMapArea	35	LineGraphLeaderColor	25
AddPoint	15	LineGraphTextAlign	25
AddText	24	LineGraphTextBGColor	25
AddXValue	26	LineGraphTextBorder	25
AddYValue	26	LineGraphTextColor	25
AxisTextBGColor	21	LineGraphTextFont	25
AxisTextColor	21	LineGraphTextLeader	25
AxisTextFont	21	LineGraphTextTransparent	25
AxisTextTransparent	21	LineGraphTextXOffset	25
BarGap	12	LineGraphTextYOffset	25
BarTotalVertical	12	LineWidth	15
BarWidth	12	MaxX	21
BGColor	29	MaxY	21
CenterX	9	OriginX	21
CenterY	9	OriginY	21
ClearAxisLabels	26	Padding	22
ClearData	29	PieDia	9
ColorToString	6	PieOffset	9
DateFormatString	27	PlotAreaColor	21
DateTimeFormat	27	PointSize	15
Decimals	24	PointStyle	15
DummyGraph	29	Prefix	23
FirstBarPos	29	PrefixX	15
GraphPen	29	PrefixY	16
GraphToArray	33	ReadBarGap	29
GraphToBitmap	33	ReadBarWidth	29
GraphToBrowser	33	ReadXGrad	29
GraphToFile	33	ReadXTop	30
GraphToStream	33	ReadYGrad	30
GraphType (Bar Charts)	12	ReadYTop	30
GraphType (Line Graphs)	15	RNDColor	28
GraphType (Pie Charts)	9	SetImageMapParams	35
GraphType (Stacked Bar)	18	ShowBarTotal	12
GridColor	21	ShowGrid	21
GridStyle	21	ShowLabel	23
Height	22	ShowLegend	22
HideHGrid	15	ShowLegendBox	22
HideVGrid	15	ShowLine	15
ImageMapExtra	35	ShowLinesWithBars	31
ImageMapText	35	ShowNumbers	23
LabelBGColor	23	ShowPercent	28
LabelColor	23	ShowPlotBorder	21
LabelFont	23	ShowSeparator	23
LabelTransparent	23	ShowSeparatorX	16
LabelVertical	12	ShowSeparatorY	16
LastBarPos	29	ShowStackedValue	18
LegendAlign	22	ShowTotalifZero	12
LegendBGColor	22	ShowTrendLine	13
LegendColor	22	Square	22
LegendFont	22	StackedTextAlign	19
LegendHideEmptyNames	22	StackedTextBGColor	18

Command	Page no.	Command	Page no.
StackedTextColor	19	TrendLineName	13
StackedTextFont	18	TrendLineWidth	13
StackedTextTransparent	19	UseRNDColor	28
StartAngle	9	UseXAxisDates	27
StringToColor	6	UseXAxisLabels	26
Suffix	23	UseYAxisDates	27
SuffixX	16	UseYAxisLabels	26
SuffixY	16	Version	5
TextBGColor	24	Width	22
TextColor	24	XAxisNegative	21
TextFont	24	XAxisText	21
TextTransparent	24	XGrad	21
TimeFormatString	27	XMarkSize	21
Title	23	XOffset	21
TitleBGColor	23	XPixelsPerUnit	29
TitleColor	23	XTop	21
TitleFont	23	XValuesVertical	15
TitleTextAlign	23	YAxisNegative	21
TitleTransparent	23	YAxisText	21
TitleX	23	YGrad	21
TitleY	23	YMarkSize	21
TransColor	33	YOffset	21
Transparent	33	YPixelsPerUnit	29
TrendLineColor	13	YTop	21