

# A P P E N D I X

---

## B

### **The MSChart Control**

**M**SChart is actually a complicated application packaged as an ActiveX control. It can chart data in many different ways including graphic depictions of data points arranged in data sets. Let's say we need a graph of monthly sales over three years that contains three data sets, one for each year, and each data set is made up of 12 data points. The data can be plotted in many different ways and a few of them are shown in Figure B.1. As you can see, the MSChart control supports many graph types in two and three dimensions. Every item of the plot can be accessed and manipulated through code, so we turn our attention next to the objects exposed by the MSChart control, their properties, and methods.

To use the MSChart control in a project you must add the Microsoft Chart Control 6.0 component through the Components dialog box. When you place an instance of the MSChart control on the Form, it's automatically populated with random values. To change this behavior, set its *RandomFill* property to False. You should display random data on the control to simplify the design process and then populate the control with the actual data from within your code. The random data values are limited in the range 0 to 100; the data points are labeled R1, R2, and so on; and the data sets are labeled C1, C2, and so on.

## Basic Properties

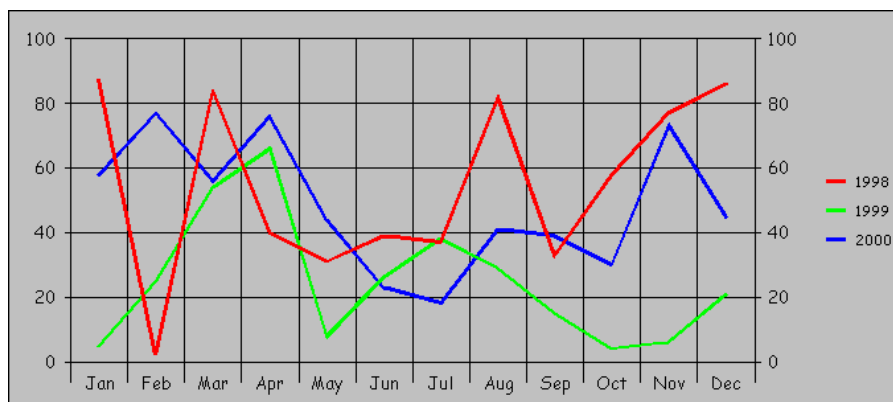
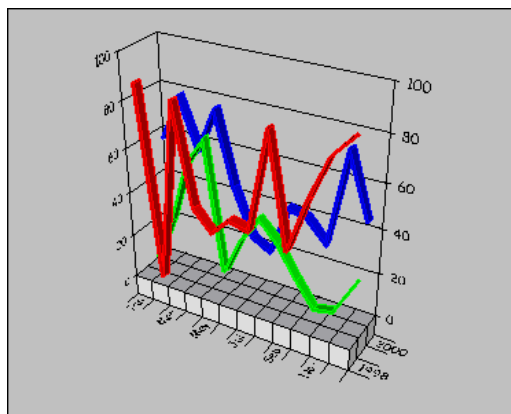
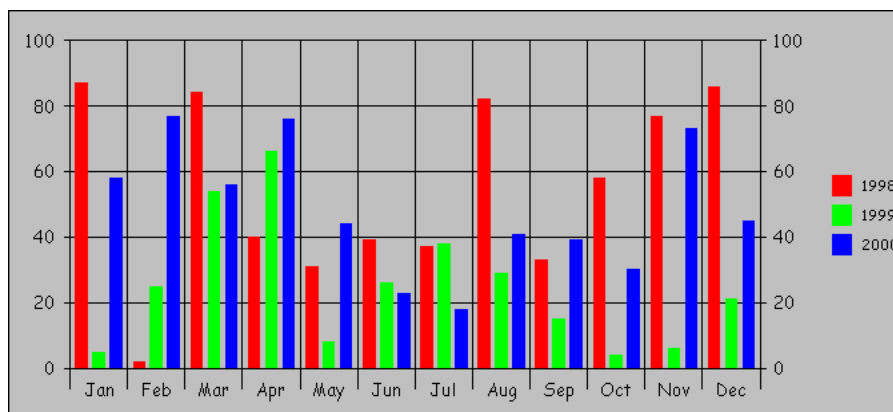
One of the first properties of the MSChart control you must set in the control's Properties window is the *chartType* property, which determines the type of the graph. Its value can be one of the constants shown in Table B.1. To size the MSChart control on the Form so that its data are easy to read, you must also set the values of the properties *ColumnCount* and *RowCount*, which are the number of columns and rows, respectively.

To see the effect of the various settings of the *chartType* property, place an instance of the MSChart control on a Form and try the various settings in the Properties window. The Chart project, described later on and found in this topic's folder on the CD, lets you change the type of the chart at runtime and set other related properties (rotation and illumination of 3D charts, for example). Many of the properties of the MSChart control discussed in this section are demonstrated by the Chart project. Open the Chart project and experiment with the chart type settings.

You should think of the MSChart control as a grid, only its data are displayed graphically. The rows of the grid are the data points and the columns of the grid are the data sets, or *series*. In Figure B.2 you see a MSChart control with three data sets (*RowCount* = 3) and 12 data points (*ColCount* = 12) in each set. Change the values of the Rows and Columns fields (which manipulate the *RowCount* and *ColCount* properties) to see how the chart's appearance is affected.

**FIGURE B.1:**

The same data set plotted in three different graph types

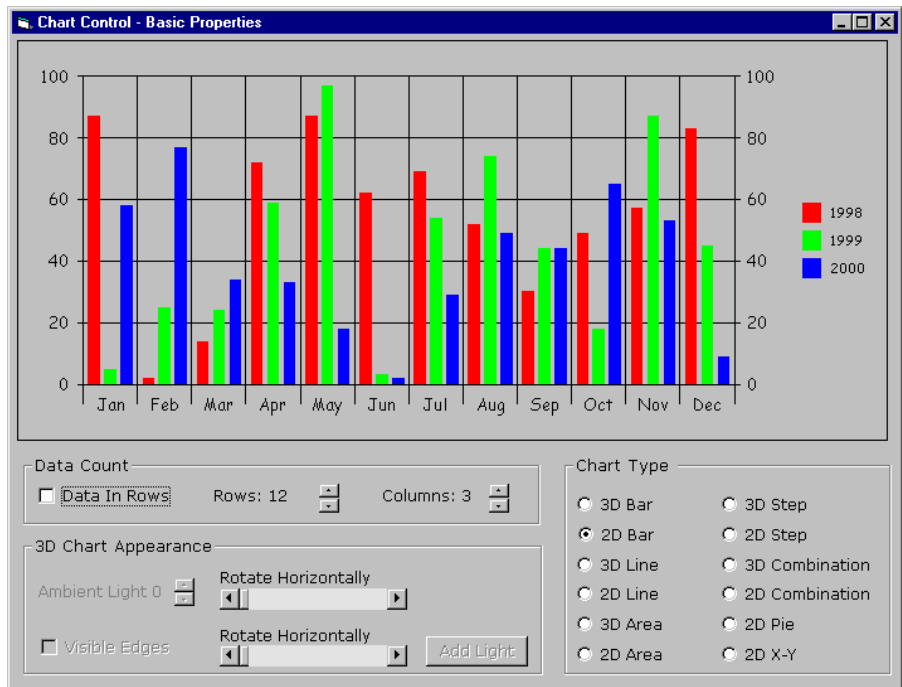


**TABLE B.1:** Values of the chartType Property

Constant	Value	Description
<i>vtChChartType3dBar</i>	0	3D bar graph
<i>vtChChartType2dBar</i>	1	2D bar graph
<i>vtChChartType3dLine</i>	2	3D line graph
<i>vtChChartType2dLine</i>	3	2D line graph
<i>vtChChartType3dArea</i>	4	3D area graph
<i>vtChChartType2dArea</i>	5	2D area graph
<i>vtChChartType3dStep</i>	6	3D step graph
<i>vtChChartType2dStep</i>	7	2D step graph
<i>vtChChartType3dCombination</i>	8	3D combination graph
<i>vtChChartType2dCombination</i>	9	2D combination graph
<i>vtChChartType2dPie</i>	14	2D pie chart
<i>vtChChartType2dXY</i>	16	2D X-Y graph

**FIGURE B.2:**

The Chart project demonstrates many of the basic properties of the MSChart control.



By default, the data sets are mapped to the columns of the control and data points to the rows of the control. You can swap data points and data sets by checking the box *Data In Rows*. This control manipulates the *DataSeriesInRow* property of the Plot object. This property accepts a Boolean value and is available at runtime only. By default, its value is False (the data series are mapped to the columns of the data grid). To access it, you must use the following expression:

```
MSChart1.Plot.DataSeriesInRow = True
```

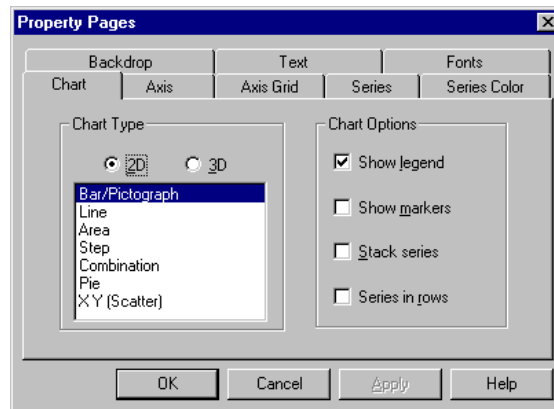
When you switch the data sets and data points, Visual Basic updates the labels and legends accordingly. Run the Chart project and check the option *Data In Rows*. The control will reverse rows and columns (data sets and data points) and change the legends of the graph.

## Using the MSChart Property Pages

The simplest way to set up a MSChart control is to use the control's property pages. Right-click the control on the Form and select Properties. The dialog box with the property pages shown in Figure B.3 will appear on your screen. On this dialog box you can manipulate many properties that relate to the appearance of the chart and you should use it to create the type of chart you want. Every time you change a few properties on the property pages, click the Apply button to see the effects of your changes on the control.

**FIGURE B.3:**

The property pages of the MSChart control let you specify the appearance of the control with point-and-click operations.



The MSChart control provides eight property pages, which are described next. The parameters you can set through these pages will be rather obvious, so I won't discuss them in detail.

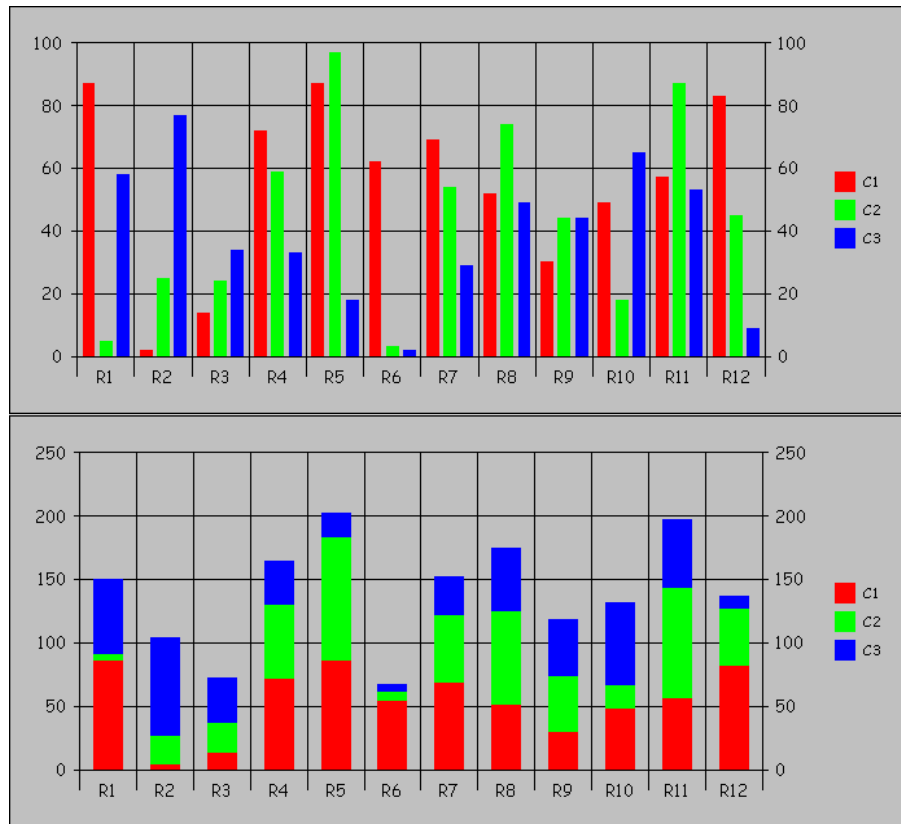
## Chart Page

Use this page to specify the chart's type and to determine whether:

- Data points should be identified with markers
- Multiple series should be stacked or plotted next to each other (see Figure B.4)
- Series correspond to rows (by default, series are mapped to columns)
- Series legends will be printed

**FIGURE B.4:**

Multiple series can be plotted next to each other (top) or stacked on top of each other (bottom).



## Axis Page

Use this page to set up the axes of the plot and the tick marks. On this window, select an axis (x-axis, y-axis, or second y-axis) in the Axis box and then set its properties.

## Axis Grid Page

Use this page to set the properties of the major and minor grids of the chart. Select the axis whose grid you want to set and then specify the properties of its major and minor grid lines.

## Series Page

In addition to the data points, the MSChart control can plot various statistics on the chart. The statistics are the Minimum, Maximum, Mean, and Standard Deviation. Select the series whose statistics you want to plot and check the boxes in front of the statistic you want to display on the chart. The last statistic is called Regression and it's a straight line that best describes the Data values of a series.

## Series Color Page

Use this page to set the appearance of each series in the plot. Select the series name in the Series box and set its properties. Some of the controls on this Form may be disabled, depending on the type of the chart.

## Backdrop Page

Every item in a chart can have its own background, which can be specified on this page. Select the item whose background you want to set and then set the background's property.

## Text Page

Use this page to set the text of the graph title, axis labels, and footnote. Select the item you want to set in the Property Name box and then set its text, alignment, and orientation.

## Fonts Page

The Fonts page is where you specify the font to be used in rendering the text of the various items of the chart. First, you must select the item whose font you want to set in the Property Name box (Title, Axis Labels and Axis Titles, Legends, and Footnote) and then set the characteristics of the font for this item.

The most important aspect of the data control is the data to be plotted, and you can't set them through the Properties dialog box. There are many methods for populating the MSChart control, and they are described in the following section.

## Accessing Data and Their Labels

One of the first operations of the MSChart control you must learn about is how to set and read its data. One method, which works through the Properties window but requires a lot of typing, is to set the *Column* and *Row* properties to the appropriate values and access the corresponding cell with the *Data* property. The value of the *Column* property identifies the current column in the data grid and it goes from 1 to *ColumnCount*. Likewise, the *Row* property identifies the current row and it goes from 1 to *RowCount*. Change the value of one of the properties, *Column* or *Row* (or both), in the Properties window and then locate the *Data* property. The current value should match the value of a data point on the graph. If *Row*=4 and *Column*=10, it's the fourth point of the tenth data set.

When you're specifying the control's data points, you can also specify the column and row labels. The column labels correspond to data series and the row labels to data points. To set the label of a data series, assign the appropriate value to the *Column* property (1 for the first series, 2 for the second, and so on). Then assign the string you want to appear as legend for the selected series to the *ColumnLabel* property. Likewise, to set the label of a data point, assign the row's number to the *Row* property and then assign the string you want to appear as legend for the selected data point to the *RowLabel* property.

In the Properties window of the MSChart control, you will see a few more label-related properties: *RowLabelCount*, *ColumnLabelCount*, *RowLabelIndex* and *ColumnLabelIndex*. You can specify more than one level of labels on the plot. If you set the *RowLabelCount* to a value larger than 1, you can specify additional labels for the rows of the plot. To select the level of a label, set the *RowLabelIndex* to a value and then assign the additional label to the *RowLabel* property. To produce the chart of Figure B.5 assign the following values to the properties listed here, in the same order:

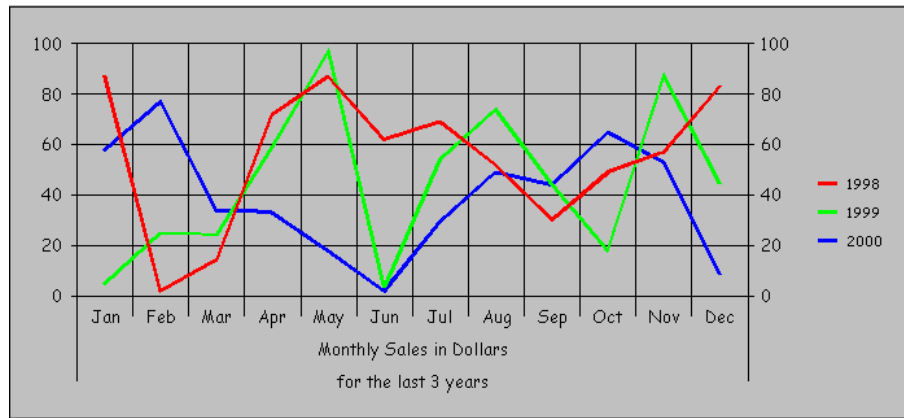
1. Set *RowLabelCount* to 3.
2. Set *RowLabelIndex* to 2 and then *RowLabel* to "Monthly Sales in Dollars" (without the quotes).
3. Set *RowLabelIndex* to 3 and then *RowLabel* to "for the last 3 years".

You can experiment on your own with the properties for the column labels and set multiple levels of labels for the chart's columns.



**FIGURE B.5:**

Use the RowLabel properties to set multiple levels of row labels.



Usually, the data to be plotted are set from within an application's code and not through the Properties window. After all, what use is it to always display the same data? You can use the process I just described from within your code to populate the grid, but it takes a lot of typing. To assign the value 99 to the second data point of the third series, you'd have to enter the following code:

```
MSChart1.Column = 3
MSChart1.Row = 2
MSChart1.Data = 99
```

The simplest method to populate the MSChart control from within your code is to declare an array with the appropriate dimensions and assign the Data values to its elements. To create three data sets with 12 elements each, declare an array with 12 columns and 3 rows as follows:

```
Dim arrData(1 To 12, 1 To 3) As Integer
```

Then, populate the array with the appropriate data. The following loop creates three data sets with random values in different ranges for the various data sets (100 to 300 for the first data set, 200 to 500 for the second data set, and 100 to 600 for the third data set):

```
For i = 1 to 12
    arrData(i, 1) = 100 + Int(Rnd() * 200) ' Series 1
    arrData(i, 2) = 200 + Int(Rnd() * 300) ' Series 2
    arrData(i, 3) = 100 + Int(Rnd() * 600) ' Series 3
Next
```

You must then size the MSChart control's data grid with the statements:

```
MSChart1.ColumnCount = 12
MSChart1.RowCount = 3
```

and finally assign the array with the data to the control's ChartData property:

```
MSChart1.ChartData = arrData
```

The array's dimensions must exactly match the numbers of columns and rows of the MSChart control. Notice that I declared both the lower and upper bounds of the array, to avoid a zero-based array. I could have also used the Option 1 statement in the Form's code and declared the array as:

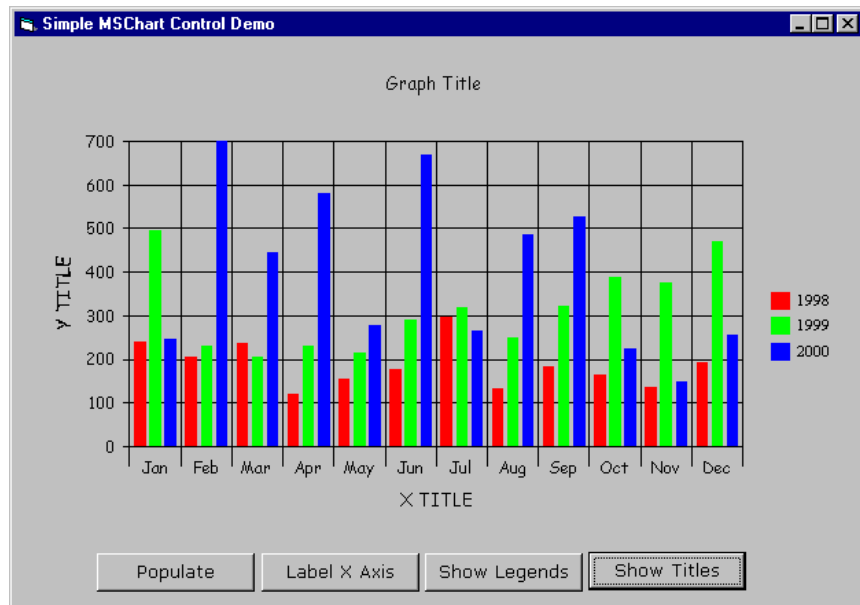
```
arrData(12, 3) As Integer
```

## VB6 at Work: The Chart1 Project

To demonstrate the techniques for populating a MSChart control and setting the labels with code, we're going to build the Chart1 project, which is shown in Figure B.6. If you don't want to enter the code shown next, open the Chart1 project, which you will find in this topic's folder on the CD.

**FIGURE B.6:**

The Chart1 project demonstrates some of the basic properties of the MSChart control.



Create a new project and place a MSChart control and a Command button on the Form. Name the Command button “Populate” and insert the following code in its Click event handler.

---

**Code B.1:      The Populate Button’s Code**

```
Private Sub btnPopulate_Click()  
    Dim arrData(1 To 12, 1 To 3)  
  
    For i = 1 To 12  
        arrData(i, 1) = 100 + Int(Rnd() * 200) ' Series 1  
        arrData(i, 2) = 200 + Int(Rnd() * 300) ' Series 2  
        arrData(i, 3) = 100 + Int(Rnd() * 600) ' Series 3  
    Next  
    MSChart1.RowCount = 12  
    MSChart1.ColumnCount = 3  
    MSChart1.ChartData = arrData  
End Sub
```

---

The vertical axis (y-axis) will be labeled properly based on the data values, but the horizontal axis labels will be generic (R1, R2, and so on). Let’s label the horizontal axis with the names of the months. If the first column of the *arrData* array contains strings, these strings will be used as labels. Place a new Command button on the Form, name it “Label X-Axis”, and insert the following code in its Click event handler.

---

**Code B.2:      The Label X-Axis Button’s Code**

```
Private Sub btnLabelX_Click()  
    Dim arrData(1 To 12, 1 To 4)  
  
    For i = 1 To 12  
        arrData(i, 1) = Left(MonthName(i), 3)  
        arrData(i, 2) = 100 + Int(Rnd() * 200) ' Series 1  
        arrData(i, 3) = 200 + Int(Rnd() * 300) ' Series 2  
        arrData(i, 4) = 100 + Int(Rnd() * 600) ' Series 3  
    Next  
    MSChart1.RowCount = 12  
    MSChart1.ColumnCount = 3  
    MSChart1.ChartData = arrData  
End Sub
```

---

There are no new statements in this code, except for the first statement in the loop, which assigns a string to the first element of each column. This string will become the label of the data set that corresponds to the column.

Setting the legends of the plot is more complicated. To set the label of a series, you must use the expression:

```
MSChart1.Plot.SeriesCollection(i).LegendText
```

where *i* is the index of the series. The Plot and SeriesCollection objects will be discussed later. Unlike axis labels, the plot's legends are not displayed by default. This behavior is controlled by the *ShowLegend* property. To display the legends shown in Figure B.6, place yet another Command button on the Form, name it "Show Legends", and insert the following code in its Click event handler.

---

**Code B.3:      The Show Legend Button's Code**

```
Private Sub btnnLegends_Click()  
    For i = 1 To MSChart1.ColumnCount  
        MSChart1.Plot.SeriesCollection(i).LegendText = i + 1997  
    Next  
    MSChart1.ShowLegend = True  
End Sub
```

---

The last few items you must set are the graph and axis titles. A title can be set with the control's *Title.Text* property. The axis titles are represented by the object `Plot.Axis(axisID).AxisTitle`, which will also be discussed later. The statements to display the titles of the graph shown in Figure B.6 are shown next.

---

**Code B.4:      The Show Titles Button's Code**

```
Private Sub btnnTitles_Click()  
    MSChart1.Title.Text = "Graph Title"  
    MSChart1.Plot.Axis(VtChAxisIdX).AxisTitle.Text = "X TITLE"  
    MSChart1.Plot.Axis(VtChAxisIdY).AxisTitle.Text = "Y TITLE"  
    MSChart1.Plot.Axis(VtChAxisIdY2).AxisScale.Hide = True  
End Sub
```

---

The last line hides the second y-axis, which is displayed by default.

The Chart1 project demonstrates all the basic properties of the MSChart control, and you can use the project to experiment with the control. The information presented so far is the absolute minimum you need to set up a MSChart control and populate it with your data set(s). However, the MSChart control provides many

properties and methods, which let you manipulate every aspect of its appearance from within your code. Its members are examined in the following sections.

## Graph Items

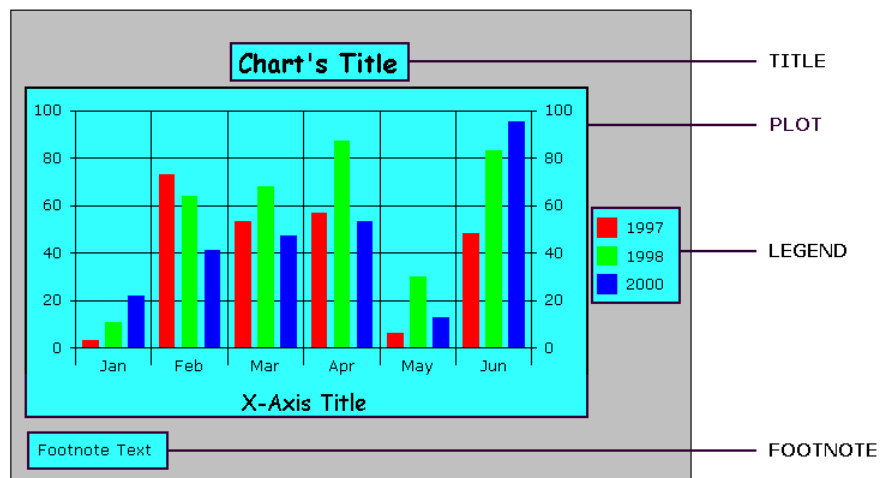
The MSChart control has more properties than any other control. Other than the data access controls, it probably has more properties than all the basic ActiveX controls combined. Every item on the chart, from the value and color of a data point to the location and alignment of the Labels, is exposed as an object which can be manipulated programmatically. I can't list all the properties and methods here, but most of these members can be easily located in the Object Browser, and their names usually reflect their role in programming the control. Instead, I will discuss the basic objects of a chart and the members you'll be using most often. I will also present a few examples to demonstrate how to manipulate the basic items of a chart, such as the data values being plotted, their titles and legends, and how to edit a chart's data interactively. After reading the information in this section, you should have a good understanding of the control's structure, and you'll be able to easily locate the object or member you need for a specific task in the Object Browser.

A graph is made up of the following objects, which represent distinct elements of the control's sections (see Figure B.7):

- **Plot** The main area of the control, where the plot appears
- **Title** The graph's titles
- **Footnote** The graph's footnote
- **Legend** The graph's legends

**FIGURE B.7:**

A MSChart control can be manipulated through the four objects shown in this figure.



Each one of these elements is an object with numerous properties and methods. The richest and most complicated object is the Plot object, which represents the plot area. In the following sections I will present the basic members of these objects so you can get an idea of the structure of the control. To make the following sections easier to read and to help you locate the information you need, I've prefixed the properties of each object and their subordinate objects/properties by the name of the object they belong to. For example, the Axis property of the Plot object is listed as Plot.Axis. Because this property is an object, its properties are listed under the Plot.Axis object and they are prefixed accordingly. The AxisGrid property of the Axis object is listed as Plot.Axis.AxisGrid, and so on.

## The Plot Object

The *Plot object* is the area with the plot and it exposes many properties and methods. As you will see, many of these properties are objects, and accessing the various parts of the plot requires typing long expressions.

**Plot.Axis(axisID)** The Axis object represents the graph's axes and exposes a number of properties, all of which are objects. Because a chart has two or three axes, the Axis object is a collection. Before you can call its properties, you must specify the desired axis either with an Index value or with the Item property. It's easier to specify an Index value in parentheses after the name of the Axis object, and this index should have one of the following constants:

*vtchAxisIdX*    *vtchAxisIdY2*    *vtchAxisIdY*    *vtchAxisIdZ*

A two-dimensional plot has an x- and y-axis, and optionally, a second y-axis (*vtchAxisIdY2*), which is parallel to the main y-axis and appears on the right-hand side of the chart. To get rid of this axis, use the Hide property, which is discussed shortly. 3D charts have a z-axis too, which is identified with the constant *vtchAxisIdZ*. To manipulate the chart's axes from within your code, use the following properties.

- **Plot.Axis(axisID).AxisGrid** The AxisGrid property determines the appearance of the chart's major and minor grids.
  - **Plot.Axis(axisID).AxisGrid.MajorPen** and **Plot.Axis(axisID).AxisGrid.MinorPen** These are the properties of the pens that are used to draw the major and minor grids of the chart. Both properties are Pen objects, and you'll find more information on this object (such as how to set the style of the grid lines and their colors) later on in the section "The Pen Object."

- **Plot.Axis(axisID).AxisScale** The *AxisScale* object determines the scale of the axes and it exposes four properties.

- **Plot.Axis(axisID).AxisScale.Hide** Set this property to *True* to hide an axis:

```
MSChart1.Plot.Axis(vtChAxisIdY2).Hide = True
```

The default value of the *Hide* property for all axes is *False*. This property is commonly used to get rid of the second y-axis.

- **Plot.Axis(axisID).AxisScale.Type** The *Type* property determines whether the axis is linear, logarithmic, or percentage-based. Its value can be one of the following constants:

```
vtchScaleTypeLinear vtchScaleTypeLogarithmic vtchScaleTypePercent
```

If the scale of an axis isn't linear, you'll probably want to set the values of the properties *LogBase* or *PercentBase*.

- **Plot.Axis(axisID).AxisScale.LogBase** The *LogBase* property specifies if the scale of a given axis is logarithmic. If the axis is logarithmic, you must assign the base of the logarithm to this property. The value of this property is usually 10.
  - **Plot.Axis(axisID).AxisScale.PercentBase** The MSChart control can plot the data values as percentages. The *PercentBase* property must be set to the type of percentage and can have one of the constants in Table B.2. The data points are plotted as percentages of the quantity specified by the constant.

**TABLE B.2:** Constants of the *PercentBase* Property

constant	TYPE OF PERCENTAGE PLOTTED
<i>vtchPercentAxisBasisMaxChart</i>	Maximum value on the chart
<i>vtchPercentAxisBasisMaxColumn</i>	Values in the column with the largest range
<i>vtchPercentAxisBasisMaxRow</i>	Values in the row with the largest range
<i>vtchPercentAxisBasisSumChart</i>	Sum of all values on the chart
<i>vtchPercentAxisBasisSumColumn</i>	Sum of all values in a column
<i>vtchPercentAxisBasisSumRow</i>	Sum of all values in a row

- **Plot.Axis(axisID).AxisTitle** The *AxisTitle* property is an object that represents the title of the respective axis. To access and manipulate the x-axis title from within your code, use the properties exposed by the object `Plot.Axis(vtchAxisIdX).AxisTitle`.
  - **Plot.Axis(axisID).AxisTitle.BackDrop** This object represents the background of the title. See the later section “The BackDrop Object” for a description of its properties.
  - **Plot.Axis(axisID).AxisTitle.Font** This is the *Font* property that determines the font to be used for rendering the title’s text. This property is a *Font* object that exposes the standard properties of a font (e.g., Name, Size, Bold).
  - **Plot.Axis(axisID).AxisTitle.Text** This property is the actual text of the axis title.
  - **Plot.Axis(axisID).AxisTitle.TextLayout** Use the properties exposed by this object to control the layout of the text (e.g., alignment, orientation). See the later section “The TextLayout Object” for a discussion of its properties.
  - **MSChart1.Plot.Axis(axisID).AxisTitle.vtFont** Use the properties and methods exposed by this object to specify the typeface and color that will be used to render the title’s text. It’s similar to the *Font* property, and its properties are discussed later in the section “The vtFont Object.”
- **Plot.Axis(axisID).Labels(i)** The *Labels* property is another collection that contains axis labels. Use the properties of this collection to control the appearance of the labels. The *Label* values are set automatically by the control and they depend on the values being plotted. However, you have complete control over the appearance of the labels with the following properties.
  - **Plot.Axis(axisID).Labels(i).BackDrop** This object represents the background of the label. See the later section “The BackDrop Object” for a description of its properties.
  - **Plot.Axis(axisID).Labels(i).Font** This object represents the font used to render the numbers on the plot’s tick marks.
  - **Plot.Axis(axisID).Labels(i).Format** This property lets you change the default formatting of the *Label* values. To display all values with two decimal digits even if they are zero (e.g., 10.50 or 12.00), set the *Format* property to the string “#.00”. Any setting you can use with the *Format()* function can also be assigned to the *Format* property.



- **Plot.Axis(axisID).Labels(i).TextLayout** Use the properties exposed by this object to control the layout of the text (e.g., alignment, orientation) on the tick marks of the plot. See the later section “The TextLayout Object” for a discussion of TextLayout object’s properties.
- **Plot.Axis(axisID).Labels(i).vtFont** This property specifies the font that will be used to render the legend text. See the later section “The vtFont Object” for more details.

**Plot.DataSeriesInRow** This is one of the few properties of the Plot object that is not an object. To switch the order of series and data points, set this property to True. The legends and the labels of the x-axis are also swapped automatically. The DataSeriesInRow property is False by default and the series (data sets) corresponds to the columns of the data grid.

**Plot.SeriesCollection(i)** The *SeriesCollection* property is a collection that contains all the data series in the control’s data grid. The first series is:

```
MSChart1.Plot.SeriesCollection(1)
```

and the last one is:

```
MSChart1.Plot.SeriesCollection(MSChart1.ColumnCount)
```

Each SeriesCollection object exposes a number of properties and methods that you can use to customize the appearance of the series on the plot.

- **Plot.SeriesCollection(i).DataPoints(j)** This collection contains all the data values in a series. The first value in the first series is:

```
MSChart1.Plot.SeriesCollection(1).DataPoints(1)
```

and the last one is:

```
MSChart1.Plot.SeriesCollection(1).DataPoints(MSChart1.RowCount)
```

- **Plot.SeriesCollection(i).LegendText** This property determines the legend of a specific series. To assign new legends to three series in a chart, use the following statements:

```
MSChart1.SeriesCollection(1).LegendText = "1999 Q1"
```

```
MSChart1.SeriesCollection(2).LegendText = "1999 Q2"
```

```
MSChart1.SeriesCollection(3).LegendText = "1999 Q3"
```

- **Plot.SeriesCollection(i).Pen** The *Pen* property is an object that lets you manipulate the properties of the pen used to plot a specific series. See the later section “The Pen Object” for more details.
- **Plot.SeriesCollection(i).SecondaryAxis** This property determines whether a specific series is mapped to the secondary axis of the chart, and it’s a True/False value.

## The Legend Object

The Legend object represents the area of the chart where the legends are printed. Through its properties you can manipulate the location of the legend area, the font of the chart's legends, and other similar properties. To access the values of the plot's legends (which correspond to the plot's series), you must use the property:

```
MSChart1.SeriesCollection(i).LegendText
```

where *i* is the desired legend's index. The basic properties and methods exposed by the Legend object are discussed next.

- **Legend.Backdrop** The *Backdrop* property is an object that represents the legend's background. See the later section "The BackDrop Object" for more information.
- **Legend.Location** The *Location* property is an object that exposes three properties that specify the location of the legend area on the chart:
  - **Legend.Location.LocationType** The *LocationType* property determines the location of the legend area and it can have one of the following values listed.

<i>vtchLocationTypeBottom</i>	<i>vtchLocationTypeBottomLeft</i>
<i>vtchLocationTypeBottomRight</i>	<i>vtchLocationTypeCustom</i>
<i>vtchLocationTypeLeft</i>	<i>vtchLocationTypeRight</i>
<i>vtchLocationTypeTop</i>	<i>vtchLocationTypeTopLeft</i>
<i>vtchLocationTypeTopRight</i>	

- **Legend.Location.Rect** Use the *Rect* property to specify the coordinates of the rectangle that encloses the legend area in pixels (if the *LocationType* property's value is *vtchLocationTypeCustom*). To specify the coordinates of this rectangle use the *Min* and *Max* properties, as follows:

```
MSChart1.Legend.Location.Rect.Min.X = minXCoordinate
MSChart1.Legend.Location.Rect.Min.Y = minYCoordinate
MSChart1.Legend.Location.Rect.Max.X = maxXCoordinate
MSChart1.Legend.Location.Rect.Max.Y = maxYCoordinate
```

Finally, you can make the legend visible or invisible, with the *Visible* property of the *Location* object, which has a *True/False* value.

- **Legend.TextLayout** This property specifies various options for the layout of the chart's legends. See the later section "The TextLayout Object" for more information.
- **Legend.vtFont** This property specifies the font that will be used to render the legend text. See the later section "The vtFont Object" for more information.

## The Footnote Object

This object represents the footnote of the graph and provides the following five properties.

- **FootNote.Backdrop** The *Backdrop* property is an object that represents the footnote's background. See the section "The BackDrop Object" later on for more information.
- **FootNote.Font** This property is a Font object that exposes the properties of the font that's used to render the footnote's text.
- **FootNote.Text** This property specifies the text of the footnote.
- **FootNote.TextLength** This property is a read-only variable that returns the length of the footnote's text.
- **FootNote.vtFont** This property is equivalent to the Font property and has the same properties and methods as the vtFont object, which is discussed later in the section "The vtFont Object."

## The Title Object

This object represents the graph's title. Its properties have different names but the same functions as those of the Footnote object listed previously; they include:

- **Title.BackDrop** The title's background; see the section "The BackDrop Object" for a discussion of its properties.
- **Title.Font** The title's font properties
- **Title.Text** The title's text
- **Title.TextLayout** The layout of the title's text; see the section "The Text-Layout Object" for a discussion of its properties.
- **Title.vtFont** An alternative property for specifying the title's font properties; see the section "The vtFont Object" for a discussion of its properties.

## The vtFont Object

The *vtFont* object represents a Font object and provides a set of members similar to the vbFont object. However, they are different than the Font's members. The vtFont object exposes the following eight properties.

- **vtFont.Name** The *Name* property is a String value that returns (or sets) the name of the typeface (e.g., "Comic Sans MS" or "Verdana").

- **vtFont.Size** This property specifies the typeface size in points.
- **vtFont.Style** This property specifies the typeface style, and it can have one of the following constants:

*vtFontStyleBold*    *vtFontStyleItalic*    *vtFontStyleOutline*

To apply more than one style, combine these constants with the Or operator:

```
vtFont.Style = VtFontStyleBold Or VtFontStyleItalic
```

- **vtFont.Effect** This property lets you apply the strikethrough or underline effect to a font, and it can have one of the following constants:

*vtFontEffectStrikeThrough*    *vtFontEffectUnderline*

You can specify both special effects for the font by combining the two constants with the Or operator:

```
vtFont.Effect = vtFontEffectStrikeThrough Or vtFontEffectUnderline
```

- **vtFont.vtColor** *vtColor* is a property of the *vtFont* object that lets you specify the color of the font. The *vtColor* object exposes the following properties.
  - **vtFont.vtColor.Automatic** The color of the font is determined automatically by the control and depends on the object to which the *vtColor* property applies.
  - **vtColor.Blue, vtColor.Green, vtColor.Red** These three properties determine the basic color components of the font's color. To specify a mid-magenta tone with the *vtColor* property, use the following statements:

```
MSChart1.Legend.VtFont.VtColor.Blue = 128
MSChart1.Legend.VtFont.VtColor.Green = 0
MSChart1.Legend.VtFont.VtColor.Red = 128
```

Notice that the MSChart control doesn't recognize Long Color values, as do most color properties of other ActiveX controls, and there's no property that returns the color of a font. You must retrieve all three Color values mentioned here.

- **vtFont.vtColor.Set** In addition to the properties mentioned here, the *vtColor* object provides the *Set* method, which sets the font's color. The *Set* method accepts three arguments, which are three integers in the range 0 to 255, and they represent the intensity of the three basic color components. The *Set* method is quite similar to Visual Basic's RGB() function and its syntax is:

```
vtFont.Set red, green, blue
```

## The TextLayout Object

The *TextLayout* object is used to specify various options for the layout of the text of the chart's title and/or footnote. Both the Title and Footnote objects have a *TextLayout* property, which is an object with the following four properties.

- **TextLayout.HorzAlignment** Use this property to specify how the legend is aligned horizontally. It can have one of the following values (their function is reflected in their name):

*vtHorizontalAlignmentCenter*      *vtHorizontalAlignmentFill*  
*vtHorizontalAlignmentFlush*      *vtHorizontalAlignmentLeft*  
*vtHorizontalAlignmentRight*

- **TextLayout.VertAlignment** This property specifies how the legend is aligned vertically. It can have one of the following values (their function is reflected in their names):

*vtVerticalAlignmentBottom*      *vtVerticalAlignmentCenter*  
*vtVerticalAlignmentTop*

- **TextLayout.Orientation** This property specifies the orientation of the legend. It can have one of the following values:

*vtOrientationDown*      *vtOrientationHorizontal*  
*vtOrientationUp*      *vtOrientationVertical*

- **TextLayout.WordWrap** This is a Boolean value that specifies if the text of the legend will be wrapped if it doesn't fit in the available area.

## The Backdrop Object

The *Backdrop* object represents the background of the various items on a chart, including the plot, legends, and axis titles. The background can be filled with a brush, it can have a frame, drop a shadow, and so on. All these features can be accessed through the following objects and properties:

- **Backdrop.Fill** The *Fill* property specifies how an object will be filled. It exposes the Brush and Style objects.
  - **Backdrop.Fill.Brush** The *Brush* property is a Brush object that determines how the object will be filled. Its properties are described later on in the section "The Brush Object."

- **Backdrop.Fill.Style** The *Style* property determines if the corresponding object will be filled with a brush. This property's valid values are the constants *vtFillStyleBrush* and *vtFillStyleNull*.
- **Backdrop.Frame** This is another object that represents the frame of the backdrop. It can be manipulated through the following three properties:
  - **Backdrop.Frame.FrameColor** and **Backdrop.Frame.SpaceColor** Use these two properties to specify the color of the frame. Both properties are objects that expose the same properties (Automatic, Blue, Green, and Red) and the Set method of the *vtFont* object's Color property.
  - **Backdrop.Frame.Style** This property determines the frame's style. It can have one of the following constants:

*vtFrameStyleDoubleLine*    *vtFrameStyleNull*    *vtFrameStyleSingleLine*  
*vtFrameStyleThickInner*    *vtFrameStyleThickOuter*

To see the effect of these settings on the frame's style, try them with a statement like the following one:

```
MSChart1.Plot.Backdrop.Frame.Style = vtFrameStyleThickInner
```

- **Backdrop.Frame.Width** This property specifies the frame's width in pixels.
- **Backdrop.Shadow** This property is also an object that determines if the backdrop will have a shadow and the style of the shadow. To create a shadow behind a frame, you must use the *Backdrop.Shadow.offset* property.
  - **Backdrop.Shadow.Offset** This property determines the offset of the shadow in pixels. To set the shadow's offset, use the *Offset* object's X and Y properties or the Set method. To specify a shadow depth of three pixels in both directions, use the statements:

```
MSChart1.Plot.Backdrop.Shadow.Offset.X = 3  
MSChart1.Plot.Backdrop.Shadow.Offset.Y = 3
```

The syntax of the equivalent Set method is:

```
MSChart1.Plot.Backdrop.Shadow.Offset.Set 3, 3
```

This statement will create a shadow behind the Plot area three pixels tall and three pixels wide.

## The Brush Object

The *Brush* object represents the brush used to draw various parts of the plot, such as the bars of a bar graph, the backdrop of a plot, or the legends. The brush can be solid or patterned and you can specify the pattern and color through the following four properties.

- **Brush.FillColor** This property lets you specify the brush's color. If the brush style is solid, the object is filled with this color. If it's patterned, the specified pattern is drawn on top of this color. The *FillColor* property is a *vtColor* object, and you can use the properties and methods discussed previously in the section "The *vtColor* Object" to manipulate the *FillColor* property.
- **Brush.Style** This property determines the fill pattern of the brush. It can have one of the following values:

*vtBrushStyleHatched*      *vtBrushStyleNull*  
*vtBrushStylePattern*      *vtBrushStyleSolid*

If the *Style* property is set to *vtBrushStylePattern*, you should use the *Index* property to specify the pattern's style.

- **Brush.Index** Use the *Index* property to specify a pattern other than solid to fill the object to which the Brush applies. The *Index* property can have one of the following constants:

<i>vtBrushPattern25Percent</i>	<i>vtBrushPatternBoldUpDiagonal</i>
<i>vtBrushPatternHorizontal</i>	<i>vtBrushPattern50Percent</i>
<i>vtBrushPatternBoldDownDiagonal</i>	<i>vtBrushPatternTrellis</i>
<i>vtBrushPattern75Percent</i>	<i>vtBrushPatternBoldVertical</i>
<i>vtBrushPatternInvertedTrellis</i>	<i>vtBrushPattern88Percent</i>
<i>vtBrushPatternUpDiagonal</i>	<i>vtBrushPatternGrid</i>
<i>vtBrushPattern94Percent</i>	<i>vtBrushPatternDownDiagonal</i>
<i>vtBrushPatternChecks</i>	<i>vtBrushPatternBoldHorizontal</i>
<i>vtBrushPatternVertical</i>	<i>vtBrushPatternWeave</i>

To see the effect of the various settings of the *Index* property use a statement like the following one to assign the constants mentioned in this paragraph to the *Index* property:

```
MSChart1.Plot.Backdrop.Fill.Brush.Index = VtBrushPattern50Percent
```

- **Brush.PatternColor** This is another `vtColor` object that determines the pattern's color. Use the `Automatic`, `Red`, `Green`, and `Blue` properties or the `Set` method to set the pattern color.

## The Pen Object

The *Pen* object represents the pen used to draw various items of the chart, such as the lines in a line plot, the lines and tick marks of an axis, or the major and minor grids (properties `MajorPen` and `MinorPen`, respectively). The *Pen* object provides five properties for controlling the pen's characteristics.

- **Pen.Cap** This property specifies the cap of the line segments used in line plots. It can have one of the following constants:

*vtPenCapButt*                      *vtPenCapRound*                      *vtPenCapSquare*

- **Pen.Join** This property specifies how successive line segments in line plots are joined to each other. It can have one of the following constants:

*vtPenJoinBevel*                      *vtPenJoinMiter*                      *vtPenJoinRound*

- **Pen.Style** This property specifies the style of the line. It can have one of the following values:

*vtPenStyleDashDit*                      *vtPenStyleDashDitDit*                      *vtPenStyleDashDot*  
*vtPenStyleDashDotDot*                      *vtPenStyleDashed*                      *vtPenStyleDitted*  
*vtPenStyleDotted*                      *vtPenStyleNative*                      *vtPenStyleNull*  
*vtPenStyleSolid*

- **Pen.vtColor** Use this property to set the color of the pen; it's another `vtColor` object. See the earlier section "The `vtColor` Object" for a description of its properties.
- **Pen.Width** Use this property to set the pen's width in pixels.

## Copying and Pasting Chart Data

The *MSChart* control provides two powerful methods for exchanging data with other applications: the *EditCopy* and *EditPaste* methods. The *EditCopy* method copies the contents of the control to the Clipboard, and the *EditPaste* method copies the contents of the Clipboard on the control. The two methods can copy

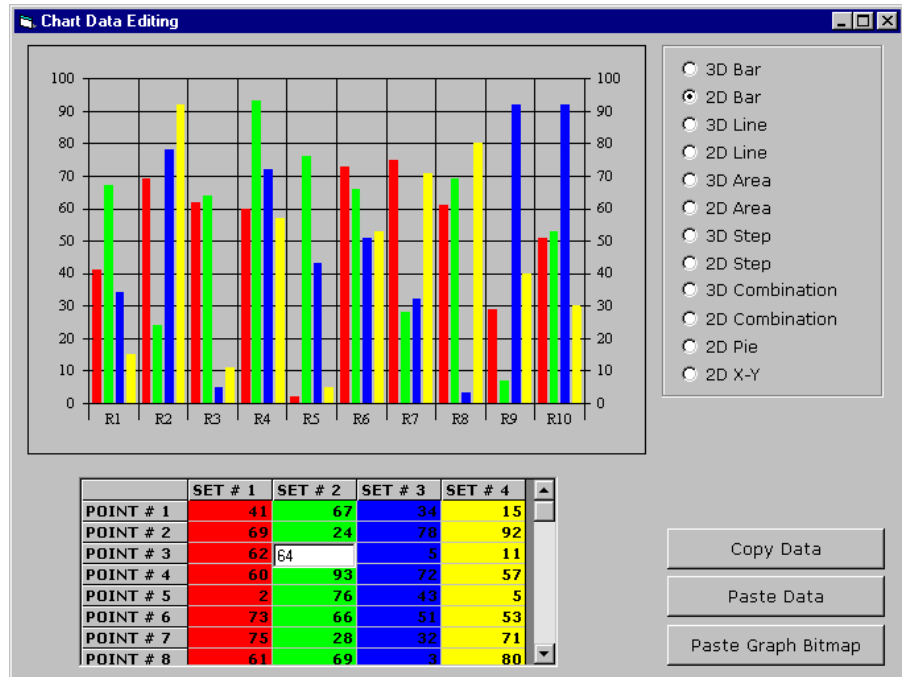


both Numeric data (including the labels) and bitmaps. Depending on the destination of the Paste command, the control's contents will be pasted either as text or as a metafile.

Figure B.8 shows the Form of the DataEdit Project, which is discussed in the following section. This application lets you edit the grid's data and copy it to the Clipboard.

**FIGURE B.8:**

MSChart control's data grid can be edited with the help of an MSFlexGrid control.



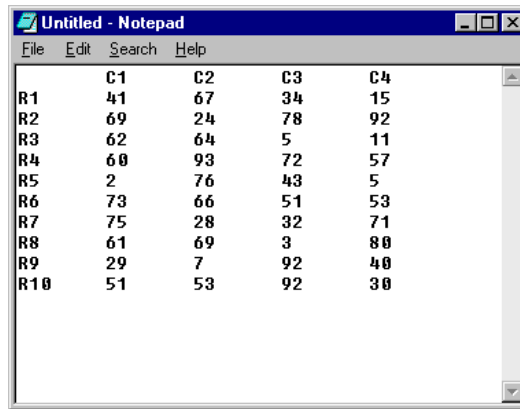
Let's discuss the code behind the Copy and Paste buttons. If you click the Copy Data button, the control's data are copied to the Clipboard with the statement:

```
MSChart1.EditCopy
```

If you start a text editor and issue the Paste command, the grid's contents will be pasted as text. Each row of the data grid will be placed on a new line and consecutive data points will be delimited with a Tab character. Figure B.9 shows how the contents of a MSChart control with four columns and 12 rows would look.

**FIGURE B.9:**

The data of a Chart control  
pasted as text



	C1	C2	C3	C4
R1	41	67	34	15
R2	69	24	78	92
R3	62	64	5	11
R4	60	93	72	57
R5	2	76	43	5
R6	73	66	51	53
R7	75	28	32	71
R8	61	69	3	80
R9	29	7	92	40
R10	51	53	92	30

Notice that the row and column labels are also copied along with the data. If you edit the data (without changing the structure), you can bring them back to the control with the Paste Data button. Change a few values in NotePad (you can also add or remove lines or columns), select all the data with the mouse or the Edit > Select All command, and copy them to the Clipboard with the Edit > Copy command. Since the data comes from a text editing application, the Clipboard contains plain text. The data can be pasted back on the control with the statement:

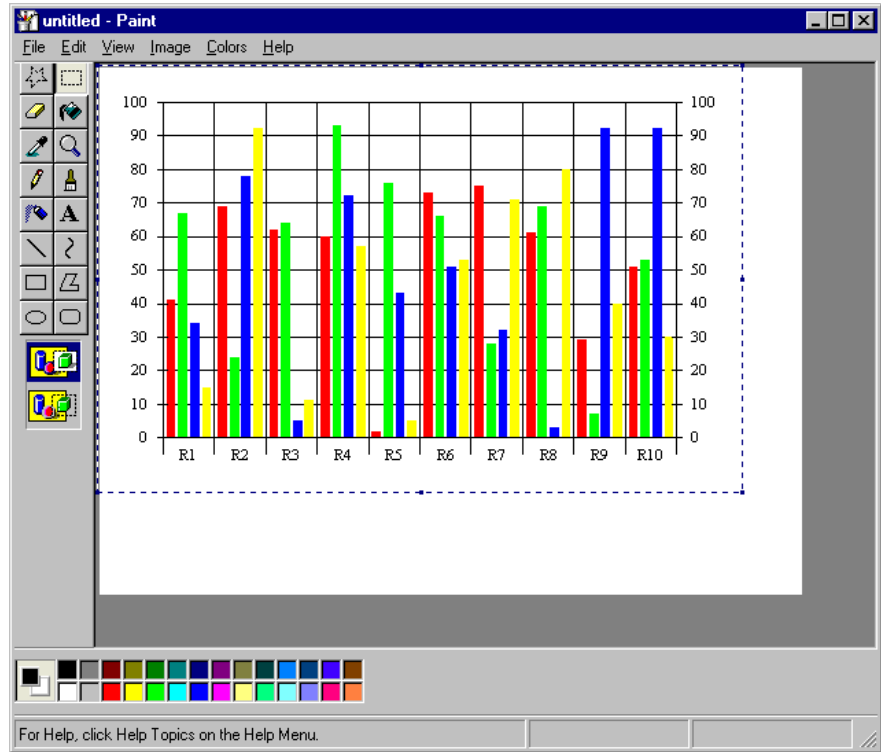
```
MSChart1.EditPaste
```

Click on the Copy Data button again, and this time, start a bitmap editing application like Paint and paste the data. The same data will be copied from the Clipboard as a bitmap (see Figure B.10). If you examine the contents of the Clipboard at this point, you'll see that they are described as text, bitmap, and metafile. Depending on the type of data the destination application recognizes, the chart will be copied as one of those types.

You can also use another application to generate the data grid and then copy and paste it on the MSChart control with the EditPaste method. For example, you can select a range of cells in Excel, copy them with Excel's Edit > Copy command, and paste them on the chart with the Paste Data button. You can also use Visual Basic's MSFlexGrid control to enter data and then plot them on a MSChart control. In the following section, we'll look at how you can couple a MSChart and a MSFlexGrid control to make them work in tandem. The MSFlexGrid control is the data entry and editing tool and the MSChart control displays the data graphically.

**FIGURE B.10:**

The bitmap of a chart  
pasted in Paint



## Editing Chart Data

You've learned how to populate the grid, edit data values from within your code, paste all the data values to a text editing application, and edit the data values, but there's no method for editing the data grid interactively. The MSChart control's internal data grid can't be edited directly. It's possible, however, to select points on the chart and edit them. To do so, you must program the *SeriesSelected* and *PointSelected* events, which are recognized by the MSChart control. As you may have noticed, when you click on a data point on the MSChart control for the first time, the entire series (data set) is selected. If you then click on a data point on the selected series, the point is selected. When a series is selected, the *SeriesSelected* event is fired and its handler's declaration is:

```
Private Sub MSChart1_SeriesSelected(Series As Integer, _
    MouseFlags As Integer, Cancel As Integer)
```

*Series* is an integer that identifies the selected series (the first series' index is 1) and the *MouseFlags* argument contains information about the status of the Con-

Control and Shift keys. Its values are *vtChMouseFlagControlKeyDown* (=8) if the Control key is down when the mouse is clicked and *vtChMouseFlagShiftKeyDown* (=4) if the Shift key is down. The *Cancel* argument is a True/False argument that can be set from within the subroutine's code to cancel the selection.

**TIP**

To let the user select a data point without selecting a series first, you can set the *AllowSeriesSelection* property to False.

When a data point is selected, the *PointSelected* event is fired. Its handler's declaration is:

```
Private Sub MSChart1_PointSelected(Series As Integer, _
    DataPoint As Integer, MouseFlags As Integer, Cancel As Integer)
```

Its arguments are the same as for the *SeriesSelected* event, except for the *DataPoint* argument, which reports the index of the selected point in the selected series (the first data point's index is 1). To access the value of the selected point on the chart, use the following statements:

```
MSChart1.Column = Series
MSChart1.Row = DataPoint
dataValue = MSChart1.Data
```

The variable *dataValue* holds the value of the selected point and you can adjust it from within your code accordingly. Then, update the chart by assigning the new value to the same data point:

```
MSChart1.Data = dataValue * 1.10
```

You can also use the *SeriesSelected* event to edit an entire data set. For instance, you can increase all the values in the series by a given percentage, change its color, and so on. To change the color of an entire series, you must access the selected series with the expression:

```
Dim thisSeries As Series
Set thisSeries = Mschart1.Plot.SeriesCollection(series)
```

Then, you must request the *Pen* object of the series and call its *Set* method to change the series color:

```
thisSeries.Pen.VtColor.Set(red, green, blue)
```

*Red*, *green* and *blue* are the three color components you would use with *RGB()* function to set a *Color* value.

## VB6 at Work: The DataEdit Project

Now we can look at the code of the DataEdit project, which uses a MSChart and a MSFlexGrid control to display the same data in different formats (the application's Form is shown in Figure B.8). The MSFlexGrid control contains Numeric data and it is plotted on the MSChart control. The two controls are coupled as follows:

1. When the application's Form is loaded, the MSChart control's (random) data are copied to the corresponding cells of the MSFlexGrid control. The columns of the grid are also colored with the same color as the corresponding series on the graph.
2. When the user selects a data point, a TextBox is displayed over the corresponding cell, so that the user can edit the cell's value in the TextBox control (this technique was discussed in detail in Chapter 9 in the section "The MSFlexGrid Control").
3. The user can commit the changes by pressing Enter or by moving the focus to another cell with the mouse. At this point, the cell's contents are copied to the MSChart control's data grid and the chart is updated automatically.

Let's start with the code that initializes the MSFlexGrid control, which is executed from within the Form's Load event:

---

**Code B.5: Copying the Chart's Data to a Grid Control**

---

```
Private Sub Form_Load()  
    Me.Show  
    Me.Refresh  
    MSFlexGrid1.Col = 0  
    MSFlexGrid1.ColWidth(0) = TextWidth("POINT # 999")  
    For irow = 1 To MSChart1.RowCount  
        MSFlexGrid1.Row = irow  
        MSFlexGrid1.Text = "POINT # " & irow  
    Next  
    For icol = 1 To MSChart1.ColumnCount  
        MSChart1.Column = icol  
        MSFlexGrid1.Col = icol  
        MSFlexGrid1.Row = 0  
        MSFlexGrid1.Text = "SET # " & icol  
        If MSChart1.ChartType <> VtChChartType2dLine Or _  
            MSChart1.ChartType <> VtChChartType3dLine Then  
            With MSChart1.Plot.SeriesCollection(icol)._  
                DataPoints(-1).Brush.FillColor  
                    clr = RGB(.Red, .Green, .Blue)  
            End With  
        End With  
    Next  
End Sub
```

```

Else
    With MSChart1.Plot.SeriesCollection(icol).Pen.VtColor
        clr = RGB(.Red, .Green, .Blue)
    End With
End If

For irow = 1 To MSChart1.RowCount
    MSChart1.Row = irow
    MSFlexGrid1.Row = irow
    MSFlexGrid1.Col = icol
    MSFlexGrid1.Text = MSChart1.Data
    MSFlexGrid1.CellBackColor = clr
Next
Next
' In case the chart is 3D, rotate it
MSChart1.Plot.View3d.Set 60, 30
End Sub

```

---

The MSFlexGrid control's size must match the size of the MSChart control's data grid. At design time, you must set the property MSFlexGrid1.Rows to the value of the property MSChart1.RowCount + 1 (to account for the fixed column of the grid) and the property MSFlexGrid1.Cols to the value of the property MSChart1.ColumnCount + 1 (to account for the fixed row).

The first loop in the previous listing sets the headings of the fixed column. Then comes a nested loop that scans the columns of the data grid and the rows within each column:

```

For icol = 1 To MSChart1.ColumnCount
    For irow = 1 To MSChart1.RowCount
        {process cell}
    Next
Next

```

In the outer loop, the code sets the current column's heading and assigns the color used to render the current data set to the matching column of the MSFlexGrid control. All types of charts use a Brush object to draw the chart, except for line plots, which use a Pen object. The long If ... Else structure extracts the color of the Pen or Brush object (depending on the type of the plot) and uses it to color the current column of the grid. Notice how I used the With structure to simplify the code. The expression for accessing the color of the Brush object is:

```
MSChart1.Plot.SeriesCollection(icol).DataPoints(-1).Brush.FillColor
```

To extract the three basic colors of the Brush object's Color value, you must call the properties:

```
MSChart1.Plot.SeriesCollection(icol).DataPoints(-1).Brush.FillColor.Red
MSChart1.Plot.SeriesCollection(icol).DataPoints(-1).Brush.FillColor.Green
MSChart1.Plot.SeriesCollection(icol).DataPoints(-1).Brush.FillColor.Blue
```

and then blend them with the RGB() function to obtain the desired Color value in a format that can be used to set the CellBackColor property of a cell on the MSFlexGrid control. Without the With structure, you'd have to enter three very long lines. The previous three lines can be compressed to the following:

```
With MSChart1.Plot.SeriesCollection(icol).DataPoints(-1).Brush.FillColor
    clr = RGB(.Red, .Green, .Blue)
End With
```

Notice the use of the Index value *-1* for the *DataPoints* array. We're interested in the Brush object, which is common for all data points in the series; we're not interested in specific Data values. The inner loop scans each Data value in the current column (series) of the data grid and assigns its value to the current cell of the MSFlexGrid control. It also sets the background of the cell to the color used by the MSChart control to render the current series. The code is fairly lengthy, but straightforward.

## Interactive Editing of Data Values

After the Form is loaded, the MSFlexGrid control contains the values of the MSChart control's data grid. To edit the control interactively, you must program the MSChart control's PointSelected event as follows:

---

### Code B.6: Preparing to Edit a Data Value

---

```
Private Sub MSChart1_PointSelected(Series As Integer, _
    DataPoint As Integer, MouseFlags As Integer, Cancel As Integer)
    MSChart1.Column = Series
    MSChart1.Row = DataPoint
    MSFlexGrid1.Row = MSChart1.Row
    MSFlexGrid1.Col = MSChart1.Column
    SetTextBox
End Sub
```

---

The first four lines locate the cell that corresponds to the Data value that was clicked on the MSChart control. Then, the code calls the SetTextBox subroutine, which prepares the hidden TextBox for editing. The SetTextBox subroutine's listing is shown next.

---

**Code B.7:      The SetTextBox() Subroutine**

```
Sub SetTextBox()  
    Text1.Left = MSFlexGrid1.Left + MSFlexGrid1.CellLeft  
    Text1.Top = MSFlexGrid1.Top + MSFlexGrid1.CellTop  
    Text1.Width = MSFlexGrid1.CellWidth  
    Text1.Height = MSFlexGrid1.CellHeight  
    Text1.Text = MSFlexGrid1.Text  
    Text1.SelStart = 0  
    Text1.SelLength = Len(Text1.Text)  
    Text1.Visible = True  
    Text1.SetFocus  
End Sub
```

---

This subroutine resizes and moves the TextBox control on top of the current cell of the MSFlexGrid control, assigns the cell's contents to the TextBox control, and moves the focus to it. The TextBox is then ready to accept user input. After the user is done editing the current cell, they can press the Enter key to commit the changes, or move the focus to another control altogether. In either case, the same code is executed. You must monitor the Enter key press from within the TextBox control's KeyPress event handler. If Enter is pressed, the program copies the new value to the MSChart control's data grid and makes the TextBox control invisible:

---

**Code B.8:      Committing the Changes**

```
Private Sub Text1_KeyPress(KeyAscii As Integer)  
    If KeyAscii = 13 Then  
        MSChart1.Data = Text1.Text  
        Text1.Visible = False  
    End If  
End Sub
```

---

The same actions take place from within the TextBox control's LostFocus event handler:

```
Private Sub Text1_LostFocus()  
    MSChart1.Data = Text1.Text  
    Text1.Visible = False  
End Sub
```



The user may also commit the changes by clicking on another cell. In this case, we must move the TextBox control on top of another cell and prepare it for editing. Here's the listing of the MSFlexGrid control's Click event handler:

```
Private Sub MSFlexGrid1_Click()  
    MSChart1.Row = MSFlexGrid1.Row  
    MSChart1.Column = MSFlexGrid1.Col  
    SetTextBox  
End Sub
```

The editing process will also end if the user scrolls the grid. If the contents of the grid are scrolled, the TextBox control will not be scrolled along with the other cells; it will end up on another cell. To avoid a potentially confusing situation, we must end the editing of the current cell from within the MSFlexGrid control's Scroll event:

```
Private Sub MSFlexGrid1_Scroll()  
    ' Remove this line to see what happens when  
    ' the FlexGrid control is scrolled  
    Text1_KeyPress (13)  
End Sub
```

We simply call the Text1\_KeyPress event, passing the ASCII value of the Enter key as argument to end the editing of the current cell. Run the DataEdit project and edit the chart interactively. Notice how you can select the point to edit either on the chart or grid. No matter how a point is selected, it's edited on the MSFlexGrid control.

DataEdit can become your starting point for many custom applications that combine two of the most powerful ActiveX controls. There are many similarities between the MSChart and MSFlexGrid controls. They both store data in a grid with rows and columns, but they present their data to the user in different formats, both equally useful.

The Copy and Paste buttons on the application's Form were discussed earlier in detail in the section "Copying and Pasting Chart Data." To complete the discussion of the project, I will show you the code behind these buttons.

---

**Code B.9: Copying and Pasting the Chart's Data**

```
Private Sub btnnCopy_Click()  
    MSChart1.EditCopy  
End Sub  
  
Private Sub btnnPaste_Click()  
    MSChart1.EditPaste
```

```
        Debug.Print Clipboard.GetFormat(vbCFBitmap)
        Debug.Print Clipboard.GetFormat(vbCFMetafile)
        Debug.Print Clipboard.GetFormat(vbCFRTF)
    End Sub

    Private Sub btnnPsteBMP_Click()
        frmGRAPHBitmap.Show
        frmGRAPHBitmap.Picture1.Picture = Clipboard.GetData()
    End Sub
```

---

The Paste Graph Bitmap button pastes the chart as a bitmap on another Form, which contains a PictureBox control. The PictureBox control's AutoSize property is set to True and the bitmap on the Clipboard is placed on the control with the GetData() method.

## Setting Chart Titles

A chart has several titles, which are listed next, along with the properties that allow you to access and manipulate them.

- **Graph Title** This is the chart's title, which appears by default above the chart. To access the chart's title, use the properties of the MSChart1.Title object.
- **X-axis Title** This is the title of the x-axis. To access the x-axis title, use the properties of the MSChart1.Plot.Axis(vtChAxisIdX) object.
- **Y-axis Title** This is the title of the y-axis. To access the y-axis title, use the properties of the MSChart1.Plot.Axis(vtChAxisIdY) object.
- **Y2-axis Title** This is the title of the second y-axis (the one on the right of the chart). To access the y2-axis title, use the properties of the MSChart1.Plot.Axis(vtChAxisIdY2) object.
- **Z-axis Title** This is the title of the z-axis, which applies to 3D charts only. To access the z-axis title, use the properties of the MSChart1.Plot.Axis(vtChAxisIdZ) object.
- **Legend** This object represents the plot's legends. To access the legends, use the properties of the MSChart1.Legend object.

All of the above properties expose a vtFont object as property, which you can use to manipulate the font for rendering the corresponding text. Except for the Legend object, all other objects have a Text property, which lets you specify the

corresponding title. The legend's text is determined by the data and it must be specified along with the data or through the LegendText property of the SeriesCollection object of the appropriate data series.

The Titles project, shown in Figure B.11, demonstrates the basic properties of the various objects that represent the titles of a chart. Open the Titles project (in this topic's folder on the CD) and examine the code that calls the Font common dialog box. The code uses the vtFont object of the various parts of the control to set the initial values on the Font common dialog box. After the user has made their selection(s) on the Font dialog box, the new values are extracted and assigned to the corresponding members of the vtFont object.

One item about this project's code I should mention here is that the Font common dialog box returns the selected font's color as a Long Color value. The code uses the functions Red(), Green(), and Blue() to extract the color components of the selected color. These functions are shown next:

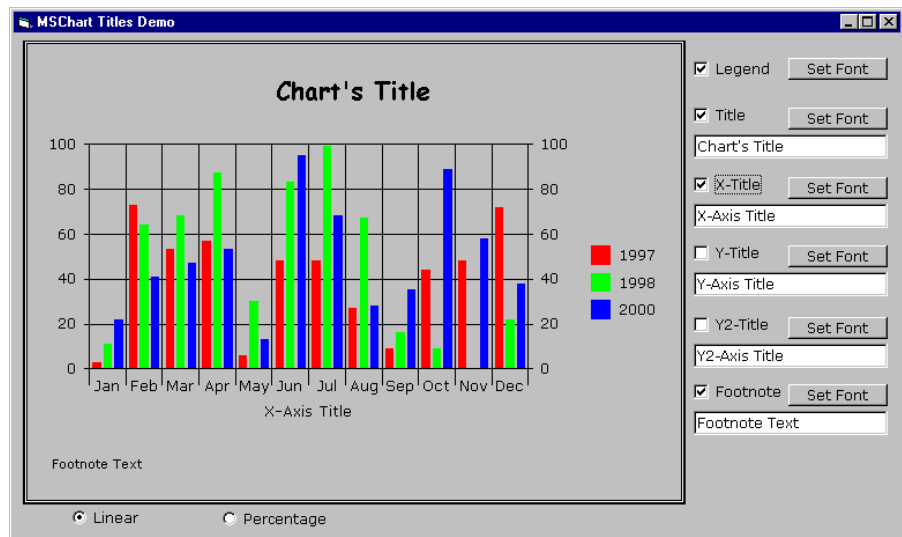
```
Public Function Red(ByVal rgb As Long) As Integer
    Red = &HFF& And rgb
End Function
```

```
Public Function Green(ByVal rgb As Long) As Integer
    Green = (&HFF00& And rgb) \ 256
End Function
```

```
Public Function Blue(ByVal rgb As Long) As Integer
    Blue = (&HFF0000 And rgb) \ 65536
End Function
```

**FIGURE B.11:**

The Titles project demonstrates the properties for manipulating the chart's titles.



## Rotating and Lighting 3D Charts

Three-dimensional charts can be rotated and illuminated in different ways. A 3D object can be arranged in the space by rotating it around its vertical and horizontal axes. If you rotate the object by the appropriate angle in the two directions, you can orient the object any way you like. To rotate a 3D chart, use the *View3D* property of the Plot object. The angle of rotation around the vertical axis is called *Rotation* and angle of rotation around the horizontal axis is called *Elevation*. To set these angles individually, use the Elevation and Rotation properties of the View3D object:

```
MSChart1.Plot.View3D.Rotation = 60  
MSChart1.Plot.View3D.Elevation = 45
```

Both angles must be specified in degrees. You can also set both angles with the *Set* method, whose syntax is shown here:

```
MSChart1.Plot.View3D.Set 60, 45
```

In addition to arranging a 3D chart in the space, you can also illuminate it with the members of the Light object. The *Light* object is a property of the Plot object, and it exposes several properties which let you manipulate the illumination of the chart programmatically. Before you can use the properties of the Light object, you must understand the illumination model used by the MSChart control, which is fairly simple. 3D charts can be illuminated with two types of light: ambient and directional light. The *ambient light* is uniform everywhere and illuminates the entire scene. The *directional light* comes from a source (a spot light) placed at a specific point in the space surrounding the chart, and it is directed toward the center of the chart.

To set the intensity of the ambient light, set the *AmbientIntensity* property to a value between 0 (no ambient light) and 1 (maximum intensity).

To illuminate a chart with directional light, you must create one or more light sources with the *LightSources* property of the Light object. The *LightSources* property is a collection to which you can add new light sources with the *Add* method:

```
MSChart1.Plot.Light.LightSources.Add (X, Y, Z, intensity)
```

X, Y and Z are the coordinates of the light source and the last argument is the intensity of the light source (a value between 0 and 1). The coordinates are not absolute units. If a chart is enclosed in a box with  $1 \times 1 \times 1$  dimensions and centered around the origin, select the coordinates of the light source so that you can control the angle at which it will illuminate the scene. To change the characteristics of an existing light source, use the *Set* method, which also allows you to set the color of the light source. The *Set* method's syntax is shown next:

```
MSChart1.Plot.Light.LightSources(1).Set red, green, blue
```

*Red*, *green*, and *blue* are the intensities of the basic color components (their value must be a number between 0 and 1).

Two related properties are the *EdgeVisible* and *EdgeIntensity* properties, which work together to highlight the edges of the chart. When *AmbientIntensity* is set to a low value to produce a dark chart, you can set the *EdgeVisible* property to *True* and the *EdgeIntensity* property to a high value to highlight the edges of the chart. Open the Chart project (in this topic's folder on the CD), set the chart's type to a 3D type, and then turn on and off the checkbox *Visible Edges* to see the effect of the *EdgeVisible* property. When the *Visible Edges* Option button is clicked, the following code is executed:

```
Private Sub Check1_Click()  
    ' Make edges visible  
    MSChart1.Plot.Light.EdgeIntensity = Check1.Value  
End Sub
```

The *Add Light* button adds a new spot light by manipulating the first member of the *LightSources* collection. The code in this button's *Click* event handler sets the color of the spotlight to a random value with the following statement:

```
Private Sub Command4_Click()  
    MSChart1.Plot.Light.LightSources(1).Set _  
        Rnd() * 10, Rnd() * 10, Rnd() * 10, Rnd()  
End Sub
```

The two scroll bars rotate 3D charts by calling the *Set* method of the *View3D* property. The first argument of the *Set* method is the *Rotation* angle and the second argument is the *Elevation* angle. Here's the code that's executed when the scroll bars change value:

```
Private Sub HScroll1_Change()  
    MSChart1.Plot.View3d.Set HScroll1.Value, HScroll2.Value  
End Sub  
  
Private Sub HScroll2_Change()  
    MSChart1.Plot.View3d.Set HScroll1.Value, HScroll2.Value  
End Sub
```

If you open the Object Browser in the Visual Basic IDE and select the *MSChartLib* library, you'll see that the *MSChart* control exposes many more objects and properties than can be discussed here. Armed with the information presented here, you should be able to figure out how to use the other objects and members of the control.