Visual Basic: Windows Controls

Visual Studio 6.0

Node Object, Nodes Collection

See Also Example Properties Methods Events

- A **Node** object is an item in a **TreeView** control that can contain images and text.
- A **Nodes** collection contains one or more **Node** objects.

Syntax

treeview.**Nodes**

treeview.**Nodes.Item(***index***)**

The syntax lines above refer to the collection and to individual elements in the collection, respectively, according to standard collection syntax.

The **Node** object and **Nodes** collection syntax have these parts:

Part	Description
treeview	An object expression that evaluates to a TreeView control.
index	Either an integer or string that uniquely identifies a member of a **Nodes** collection. The integer is the value of the **Index** property; the string is the value of the **Key** property.

Remarks

Nodes can contain both text and pictures. However, to use pictures, you must associate an **ImageList** control using the **ImageList** property.

Pictures can change depending on the state of the node; for example, a selected node can have a different picture from an unselected node if you set the **SelectedImage** property to an image from the associated **ImageList**.

The **Visible** property of the **Node** object is read-only at run time.

Visual Basic Reference

Visual Studio 6.0

OLE Container Control

See Also Example Properties Methods Events

The **OLE** container control enables you to add insertable objects to the forms in your Visual Basic applications. With the **OLE** container control, you can:

- Create a placeholder in your application for an insertable object. At run time you can create the object that is displayed within the **OLE** container control or change an object you placed within the **OLE** container control at design time.
- Create a linked object in your application.
- Bind the **OLE** container control to a database using the **Data** control.

You either create the object at design time using the Insert Object dialog box (which contains the commands Insert Object, Paste Special, and so on), or at run time by setting the appropriate properties.

When you move an **OLE** container control on a form using the **ObjectMove** method, the **Height** and **Width** property values of the object may be slightly different after the move. This is because the parameters to the **ObjectMove** method are pixel values converted to the current form's scaling mode. The conversion from pixels to twips and back doesn't always result in identical values.

Using the OLE Container Control's Pop-up Menus

Each time you draw an **OLE** container control on a form, the Insert Object dialog box is displayed. Use this dialog box to create a linked or embedded object. If you choose Cancel, no object is created.

At design time, click the **OLE** container control with the right mouse button to display a pop-up menu. The commands displayed on this pop-up menu depend on the state of the **OLE** container control as shown in the following table:

Command	**Enabled in pop-up menu when**
Insert Object	Always enabled.
Paste Special	**Clipboard** object contains a valid object.
Delete Embedded Object	**OLE** container control contains an embedded object.
Delete Linked Object	**OLE** container control contains a linked object.
Create Link	**SourceDoc** property is set.
Create Embedded Object	**Class** or **SourceDoc** property is set.

An **OLE** container control can contain only one object at a time. You can create a linked or embedded object in several ways:

- Use the Insert Object or Paste Special dialog boxes (run time or design time).
- Set the **Class** property in the Properties window, click the **OLE** container control with the right mouse button, and then select the appropriate command (design time only).
- Use the appropriate method of the **OLE** container control.

Finding Class Names

You can get a list of the class names available to your application by selecting the **Class** property in the Properties window and clicking the Properties button.

Note The Insert Object dialog box doesn't display a list of class names. This dialog box displays user-friendly names for each class of object, which are generally longer and more easily understood.

Visual Basic: RichTextBox Control

Visual Studio 6.0

OLEObject Object

See Also Example Properties Methods Events

An **OLEObject** object represents an insertable object within a **RichTextBox** control.

Remarks

The **RichTextBox** control enables you to add insertable objects to an RTF file. Insertable objects are represented by the **OLEObject** object.

Each embedded **OLEObject** object counts as one character in the **RichTextbox** control. That is, when counting from the beginning of the control, the object will take up one character position. OLE objects can be highlighted, cut or copied by the user, but they do not support any of the Selxxx properties (i.e. SelBold, SelItalic, etc.).

Each object supports a context menu that contains the standard Cut, Copy, Paste, and Delete commands, as well as Open and Edit. Open will cause the objects application to be opened in its own window so that the object can be edited. Edit will cause the object to in-place activate if the object supports in-place activation.

You can manually add **OLEObject** objects to the **OLEObjects** collection at run time by using the **Add** method, or by dragging an object from the Windows Explorer into the **RichTextBox** control.

Visual Basic: RichTextBox Control

Visual Studio 6.0

OLEObjects Collection

See Also Example Properties Methods Events

An **OLEObjects** collection contains a collection of **OLEObject** objects.

Syntax

object.**OLEObjects(***index***)**

object.**OLEObjects.Item(***index***)**

The **OLEObjects** collection syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	The value of either the Index property or the Key property which uniquely identifies the OLEObject object.

Remarks

Every embedded OLE object created in the **RichTextBox** control is represented in the **OLEObjects** collection. You can manually add objects to the **OLEObjects** collection at run time by using the **Add** method, or by dragging an object from the Windows Explorer into the **RichTextBox** control.

The **OLEObjects** collection is a standard collection and supports the **Add**, **Item**, and **Remove** methods, as well as the **Count** property.

Visual Basic Reference

Visual Studio 6.0

OptionButton Control

See Also Example Properties Methods Events

An **OptionButton** control displays an option that can be turned on or off.

Syntax

OptionButton

Remarks

Usually, **OptionButton** controls are used in an option group to display options from which the user selects only one. You group **OptionButton** controls by drawing them inside a container such as a **Frame** control, a **PictureBox** control, or a form. To group **OptionButton** controls in a **Frame** or **PictureBox**, draw the **Frame** or **PictureBox** first, and then draw the **OptionButton** controls inside. All **OptionButton** controls within the same container act as a single group.

While **OptionButton** controls and **CheckBox** controls may appear to function similarly, there is an important difference: When a user selects an **OptionButton**, the other **OptionButton** controls in the same group are automatically unavailable. In contrast, any number of **CheckBox** controls can be selected.

Visual Basic: Windows Controls

Visual Studio 6.0

Panel Object

See Also Example Properties Methods Events

A **Panel** object represents an individual panel in the **Panels** collection of a **StatusBar** control.

Remarks

A **Panel** object can contain text and a bitmap which may be used to reflect the status of an application.

Use the **Panels** collection to retrieve, add, or remove an individual **Panel** object.

To change the look of a panel, change the properties of the **Panel** object. To modify the properties at design-time, you can change the properties of the **Panel** object in the Panels tab of the Properties Page. At run-time, you can change the **Panel** object properties in code.

Visual Basic: Windows Controls

Visual Studio 6.0

Panels Collection

See Also Example Properties Methods Events

A **Panels** collection contains a collection of **Panel** objects.

Syntax

statusbar.**Panels**(*index*)

The **Panels** collection syntax has these parts.

Part	Description
statusbar	An object expression that evaluates to a **StatusBar** control.
Index	An integer or string that uniquely identifies the object in the collection. The integer is the value of the **Index** property of the desired **Panel** object; the string is the value of the **Key** property of the desired **Panel** object.

The **Panels** collection is a 1-based array of **Panel** objects. By default, there is one **Panel** object on a **StatusBar** control. Therefore, if you want three panels to be created, you only need to add two objects to the **Panels** collection.

The **Panels** property returns a reference to a **Panels** collection.

To add a **Panel** object to a collection, use the **Add** method for **Panel** objects at run time, or the Panels tab on the Properties Page of the **StatusBar** control at design time.

Each item in the collection can be accessed by its **Index** property or its **Key** property. For example, to get a reference to the third **Panel** object in a collection, use the following syntax:

```
Dim pnlX As Panel
Set pnlX = StatusBar1.Panels(3)   ' Reference by index number.
   ' or
Set pnlX = StatusBar1.Panels("Third")   ' Reference by unique key.
   ' or
Set pnlX = StatusBar1.Panels.Item(3)   ' Use Item method.
```

Visual Basic Reference

Visual Studio 6.0

ParentControls Collection

See Also Example Properties Methods Events

A collection that allows access to controls in another control's container.

Syntax

ParentControls(*index***)**

The placeholder *index* represents an integer with a range from 0 to `ParentControls.Count - 1`.

Visual Studio 6.0

Visual Basic: MSChart Control

Pen Object

See Also Example Properties Methods Events

Describes the color and pattern of lines or edges on a chart.

Syntax

Pen

Visual Basic: MSChart Control

Pen Object, GuideLinePen Property Example

The following example sets the pen attributes for a two-dimensional xy chart series. The **GuideLinePen** property returns a reference to a **Pen** object.

```
Private Sub Command1_Click()
   ' Set Guide Lines for 2D XY chart Series 1.
   MSChart1.ChartType = VtChChartType2dXY
   MSChart1.Plot.SeriesCollection.Item(1) _
   .ShowGuideLine(VtChAxisIdX) = True
   With _
   MSChart1.Plot.SeriesCollection.Item(1).GuideLinePen
      ' Set Pen attributes.
      .VtColor.Set 255, 255, 0
      .Width = 10
      .Style = VtPenStyleDashDot
      .Join = VtPenJoinRound
      .Cap = VtPenCapRound
   End With
End Sub
```

Visual Basic Reference

Visual Studio 6.0

PictureBox Control

See Also Example Properties Methods Events

A **PictureBox** control can display a graphic from a bitmap, icon, or metafile, as well as enhanced metafile, JPEG, or GIF files. It clips the graphic if the control isn't large enough to display the entire image.

Syntax

PictureBox

Remarks

You can also use a **PictureBox** control to group **OptionButton** controls and to display output from graphics methods and text written with the **Print** method.

To make a **PictureBox** control automatically resize to display an entire graphic, set its **AutoSize** property to **True**.

To create animation or simulation, you can manipulate graphics properties and methods in code. Graphics properties and events are useful for run-time print operations, such as modifying the format of a screen form for printing.

A **PictureBox** control can also act as a destination link in a DDE conversation.

The PictureBox and **Data** controls are the only standard Visual Basic controls that you can place in the internal area of an MDI form. You can use it to group controls at the top or bottom of the internal area to create a toolbar or status bar.

Note The Unisys Corporation has a patent that it alleges covers certain aspects of GIF-LZW compression technology, which the PictureBox and Image controls use. Microsoft Corporation obtained a license to the Unisys LZW patents in September, 1996. Microsoft's license does not, however, extend to software developers or third parties who use any Microsoft toolkit, language development, or operating system products to provide GIF read/write and/or any other LZW capabilities in their own products (for example, by way of DLLs and APIs).

If your commercial application uses this control (and thus, the LZW technology), you may wish to obtain an independent legal opinion on the effect of the patent, or contact Unisys USA at http://www.unisys.com/ for more information.

Visual Basic: PictureClip Control

Visual Studio 6.0

PictureClip Control

See Also Example Properties Methods Events

The **PictureClip** control allows you to select an area of a source bitmap and then display the image of that area in a form or picture box. **PictureClip** controls are invisible at run time.

File Name

PICCLP32.OCX

Class Name

PictureClip

Remarks

The **PictureClip** control provides an efficient mechanism for storing multiple picture resources. Instead of using multiple bitmaps or icons, create a source bitmap that contains all the icon images required by your application. When you need to display an individual icon, use the **PictureClip** control to select the region in the source bitmap that contains that icon.

For example, you could use this control to store all the images needed to display a toolbox for your application. It is more efficient to store all of the toolbox images in a single **PictureClip** control than it is to store each image in a separate picture box. To do this, first create a source bitmap that contains all of the toolbar icons. The picture at the top of this topic is an example of such a bitmap.

Note For international localized applications, resource files are sometimes a more useful way to store bitmaps. For more information, refer to Chapter 16, "International Issues," in the Visual Basic *Programmer's Guide*.

You can specify the clipping region in the source bitmap in one of two ways:

- Select any portion of the source bitmap as the clipping region. Specify the upper-left corner of the clipping region using the **ClipX** and **ClipY** properties. Specify the area of the clipping region using the **ClipHeight** and **ClipWidth** properties. This method is useful when you want to view a random portion of a bitmap.
- Divide the source bitmap into a specified number of rows and columns. The result is a uniform matrix of picture cells numbered 0, 1, 2, and so on. You can display individual cells using the **GraphicCell** property. This method is useful when the source bitmap contains a palette of icons that you want to display individually, such as in a toolbar bitmap.

Load the source bitmap into the **PictureClip** control using the **Picture** property. You can load only bitmap (.bmp) files into the **PictureClip** control.

Distribution Note When you create and distribute applications that use the **PictureClip** control, you should install PicClp32.ocx in the customer's Microsoft Windows System or System32 subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

Visual Basic: PictureClip Control

Clip Example (PictureClip Control)

Visual Basic Example

The following example displays a **Clip** image in a picture box when the user specifies X and Y coordinates and then clicks a form. **First** create a form with a **PictureBox**, a **PictureClip** control, and two TextBox controls. At design time, use the Properties sheet to load a valid bitmap into the **PictureClip** control.

```
Private Sub Form_Click ()
   Dim SaveMode As Integer
   ' Save the current ScaleMode for the picture box.
   SaveMode = Picture1.ScaleMode
   ' Get X and Y coordinates of the clipping region.
   PicClip1.ClipX = Val(Text1.Text)
   PicClip1.ClipY = Val(Text2.Text)
   ' Set the area of the clipping region (in pixels).
   PicClip1.ClipHeight = 100
   PicClip1.ClipWidth = 100
   ' Set the picture box ScaleMode to pixels.
   Picture1.ScaleMode = 3
   ' Set the destination area to fill the picture box.
   PicClip1.StretchX = Picture1.ScaleWidth
   PicClip1.StretchY = Picture1.ScaleHeight
   ' Assign the clipped bitmap to the picture box.
   Picture1.Picture = PicClip1.Clip
   ' Reset the ScaleMode of the picture box.
   Picture1.ScaleMode = SaveMode
End Sub
```

Visual Studio 6.0

Visual Basic: MSChart Control

Plot Object

See Also Example Properties Methods Events

Represents the area upon which a chart is displayed.

Syntax

Plot

Remarks

The **Plot** object allows you to program the following objects:

1. Axis objectrepresents the *x*, *y*, and *z* axis of the chart. The *z* axis is only visible on 3D charts.
2. BackDrop objectthe area behind the axes.
3. Light objectthe ambient and edge light of the plot.
4. LocationRect objectthe location of the plot.
5. SeriesCollection objectthe collection of series sets.
6. View3D objectthe elevation and rotation of the 3D image.
7. Wall objectthe area behind the plot.

Visual Basic: MSChart Control

Plot Object Example

The following example sets the chart viewing distance and axis division spacing.

```
Private Sub Command1_Click()
   ' Change the chart type to 3D Bar.
   MSChart1.ChartType = VtChChartType3dBar
   With MSChart1.Plot
      ' Changes 3d bar chart's viewing.
      .DepthToHeightRatio = 2
      .WidthToHeightRatio = 2
      ' Changes the spacing between divisions on the
      ' X-Axis.
      .xGap = 0
      ' Changes the spacing between divisions on the
      ' Z-Axis.
      .zGap = 0.8
   End With
End Sub
```

Visual Studio 6.0

Visual Basic: MSChart Control

PlotBase Object

See Also Example Properties Methods Events

The area beneath a chart.

Syntax

PlotBase

Visual Basic: MSChart Control

PlotBase Object Example

The following example sets the chart base parameters on a three-dimensional bar chart.

```
Private Sub Command1_Click()
   ' Change the chart type to 3D.
   MSChart1.ChartType = VtChChartType3dBar
   With MSChart1.Plot.PlotBase
      ' Change the base height.
      .BaseHeight = 20
      ' Use the pattern style for base.
      .Brush.Style = VtBrushStylePattern
      .Brush.Index = VtBrushPatternHorizontal
      .Brush.FillColor.Set 255, 160, 160
      .Brush.PatternColor.Set 180, 180, 255
      .Pen.Style = VtPenStyleSolid
      .Pen.VtColor.Set 72, 72, 255
   End With
End Sub
```

Visual Basic Reference

Visual Studio 6.0

Printer Object, Printers Collection

See Also Example Properties Methods Events

The **Printer** object enables you to communicate with a system printer (initially the default system printer).

The **Printers** collection enables you to gather information about all the available printers on the system.

Syntax

Printer

Printers(*index***)**

The *index* placeholder represents an integer with a range from 0 to `Printers.Count-1`.

Remarks

Use graphics methods to draw text and graphics on the **Printer** object. Once the **Printer** object contains the output you want to print, you can use the **EndDoc** method to send the output directly to the default printer for the application.

You should check and possibly revise the layout of your forms if you print them. If you use the **PrintForm** method to print a form, for example, graphical images may be clipped at the bottom of the page and text carried over to the next page.

The **Printers** collection enables you to query the available printers so you can specify a default printer for your application. For example, you may want to find out which of the available printers uses a specific printer driver. The following code searches all available printers to locate the first printer with its page orientation set to Portrait, then sets it as the default printer:

```
Dim X As Printer
For Each X In Printers
   If X.Orientation = vbPRORPortrait Then
      ' Set printer as system default.
      Set Printer = X
      ' Stop looking for a printer.
      Exit For
   End If
Next
```

You designate one of the printers in the **Printers** collection as the default printer by using the **Set** statement. The preceding example designates the printer identified by the object variable X, the default printer for the application.

Note If you use the **Printers** collection to specify a particular printer, as in `Printers(3)`, you can only access properties on a read-only basis. To both read and write the properties of an individual printer, you must first make that printer the default printer for the application.

Visual Basic: Windows Controls

Visual Studio 6.0

ProgressBar Control

See Also Example Properties Methods Events

The **ProgressBar** control shows the progress of a lengthy operation by filling a rectangle with chunks from left to right.

Syntax

ProgressBar

Remarks

The **ProgressBar** control monitors an operation's progress toward completion.

A **ProgressBar** control has a range and a current position. The range represents the entire duration of the operation. The current position represents the progress the application has made toward completing the operation. The **Max** and **Min** properties set the limits of the range. The **Value** property specifies the current position within that range. Because chunks are used to fill in the control, the amount filled in only approximates the **Value** property's current setting. Based on the control's size, the **Value** property determines when to display the next chunk.

The **ProgressBar** control's **Height** and **Width** properties determine the number and size of the chunks that fill the control. The more chunks, the more accurately the control portrays an operation's progress. To increase the number of chunks displayed, decrease the control's **Height** or increase its **Width**. The **BorderStyle** property setting also affects the number and size of the chunks. To accommodate a border, the chunk size becomes smaller.

You can use the **Align** property with the **ProgressBar** control to automatically position it at the top or bottom of the form.

Tip To shrink the chunk size until the progress increments most closely match actual progress values, make the **ProgressBar** control at least 12 times wider than its height.

The following example shows how to use the **ProgressBar** control, named ProgressBar1, to show the progress of a lengthy operation of a large array. Put a **CommandButton** control and a **ProgressBar** control on a form. The **Align** property in the sample code positions the **ProgressBar** control along the bottom of the form. The **ProgressBar** control displays no text.

```
Private Sub Command1_Click()
   Dim Counter As Integer
   Dim Workarea(250) As String
   ProgressBar1.Min = LBound(Workarea)
   ProgressBar1.Max = UBound(Workarea)
   ProgressBar1.Visible = True

'Set the Progress's Value to Min.
   ProgressBar1.Value = ProgressBar1.Min

'Loop through the array.
   For Counter = LBound(Workarea) To UBound(Workarea)
      'Set initial values for each item in the array.
      Workarea(Counter) = "Initial value" & Counter
```

```
      ProgressBar1.Value = Counter
   Next Counter
   ProgressBar1.Visible = False
   ProgressBar1.Value = ProgressBar1.Min
End Sub

Private Sub Form_Load()
   ProgressBar1.Align = vbAlignBottom
   ProgressBar1.Visible = False
   Command1.Caption = "Initialize array"
End Sub
```

Distribution Note The **ProgressBar** control is part of a group of ActiveX controls that are found in the MSCOMCTL.OCX file. To use the **ProgressBar** control in your application, you must add the MSCOMCTL.OCX file to the project. When distributing your application, install the MSCOMCTL.OCX file in the user's Microsoft Windows System or System32 directory. For more information on how to add an ActiveX control to a project, see the *Programmer's Guide*.

Visual Basic Extensibility Reference

Visual Studio 6.0

Properties Collection

See Also Example Properties Methods Events

Returns the available properties of a control or component.

Syntax

Properties

Remarks

This object or collection includes all the properties that can normally be accessed at design time.

The default value for the **Properties** collection is determined by the **Item** method.

Use the **Properties** collection to access the properties displayed in the **Properties** window. For every property listed in the **Properties** window, there is an object in the **Properties** collection.

Visual Basic Extensibility Reference

Visual Studio 6.0

Properties Collection (VBA Add-In Object Model)

See Also Example Properties Methods Events Specifics

Represents the properties of an object.

Remarks

Use the **Properties** collection to access the properties displayed in the Properties window. For every property listed in the **Properties** window, there is an object in the **Properties** collection.

Visual Basic Extensibility Reference

Visual Studio 6.0

Property Object

See Also Example Properties Methods Events Specifics

Represents the properties of an object that are visible in the Properties window for any given component.

Remarks

Use **Value** property of the **Property** object to return or set the value of a property of a component.

At a minimum, all components have a **Name** property. Use the **Value** property of the **Property** object to return or set the value of a property. The **Value** property returns a Variant of the appropriate type. If the value returned is an object, the **Value** property returns the **Properties** collection that contains **Property** objects representing the individual properties of the object. You can access each of the **Property** objects by using the **Item** method on the returned **Properties** collection.

If the value returned by the **Property** object is an object, you can use the **Object** property to set the **Property** object to a new object.

Visual Basic Reference

Visual Studio 6.0

PropertyBag Object

See Also Example Properties Methods Events

A **PropertyBag** object holds information that is to be saved and restored across invocations of an object.

Remarks

A **PropertyBag** object is passed into an object through the ReadProperties event and the WriteProperties event in order to save and restore the state of the object. Using the methods of the **PropertyBag** object, the object can read or write properties of itself. The **ReadProperty** method of the **PropertyBag** object is used to read a value for a property, while the **WriteProperty** method of the **PropertyBag** object is used to write a value of a property. The value of a property can itself be an object; in that case the **PropertyBag** object will attempt to save it.

Visual Basic Reference

Visual Studio 6.0

PropertyPage Object

See Also Example Properties Methods Events

The base object used to create an ActiveX Property Page.

Remarks

Property pages provide an alternative to the Properties window for viewing properties. You can group several related properties on a page, or use a page to provide a dialog-like interface for a property thats too complex for the Properties window. A **PropertyPage** object represents one page, which is to say one tab in the **Property Pages** dialog box.