

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Page Property

[See Also](#) [Example](#) [Applies To](#)

Returns the current page number.

### Syntax

*object*.**Page**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

### Remarks

Visual Basic keeps a count of pages that have been printed since your application started or since the last time the **EndDoc** statement was used on the **Printer** object. This count starts at one and increases by one if:

- You use the **NewPage** method.
- You use the **Print** method and the text you want to print doesn't fit on the current page.

**Note** Graphics methods output that doesn't fit on the page doesn't generate a new page. The output is clipped to fit the page's printable area.

© 2018 Microsoft

# Visual Basic Reference

## Page Property Example

This example prints three pages of text with the current page number at the top of each page. To try this example, paste the code into the Declarations section of a form, and then press F5 and click the form.

```
Private Sub Form_Click ()
    Dim Header, I, Y    ' Declare variables.
    Print "Now printing..."    ' Put notice on form.
    Header = "Printing Demo - Page "    ' Set header string.
    For I = 1 To 3
        Printer.Print Header;    ' Print header.
        Printer.Print Printer.Page    ' Print page number.
        Y = Printer.CurrentY + 10    ' Set position for line.
        ' Draw a line across page.
        Printer.Line (0, Y) - (Printer.ScaleWidth, Y)    ' Draw line.
        For K = 1 To 50
            Printer.Print String(K, " ");    ' Print string of spaces.
            Printer.Print "Visual Basic ";    ' Print text.
            Printer.Print Printer.Page    ' Print page number.
        Next
        Printer.NewPage
    Next I
    Printer.EndDoc
End
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Palette Property

See Also   Example   [Applies To](#)

Returns or sets an image that contains the palette to use for the control.

### Syntax

*object*.**Palette** = *path*

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>path</i>	The path of the bitmap image containing the palette to be used.

### Remarks

You can use a .dib, .gif, or .bmp file to set the palette.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PaletteMode Property

See Also   Example   [Applies To](#)

Returns or sets a value that determines which palette to use for the controls on a object.

### Syntax

*object*.**PaletteMode** = *integer*

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>integer</i>	Determines the palette mode to be used, as described in Settings, below.

### Settings

The settings for *integer* are:

Constant	Value	Description
<b>vbPaletteModeHalfTone</b>	0	(Default) Use the Halftone palette.
<b>vbPaletteModeUseZOrder</b>	1	Use the palette from the topmost control that has a palette.
<b>vbPaletteModeCustom</b>	2	Use the palette specified in the <b>Palette</b> property.
<b>vbPaletteModeContainer</b>	3	Use the container's palette for containers that support ambient <b>Palette</b> property. Applies to UserControls only.
<b>vbPaletteModeNone</b>	4	Do not use any palette. Applies to UserControls only.
<b>vbPaletteModeObject</b>	5	Use the ActiveX designers palette. (Applies only to ActiveX designers which contain a palette.)

### Remarks

If no palette is available, the halftone palette becomes the default palette.

**Note** For previous versions of Visual Basic, PaletteMode corresponded to UseZOrder.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## Panels Property

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to a collection of **Panel** objects.

### Syntax

*object*.**Panels**

The *object* placeholder is an object expression that evaluates to a **StatusBar** control.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PaperBin Property

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets a value indicating the default paper bin on the printer from which paper is fed when printing. Not available at design time.

### Syntax

*object*.**PaperBin** [= *value*]

The **PaperBin** property syntax has these parts:

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Value</i>	A value or constant specifying the default paper bin, as described in Settings.

### Settings

The settings for *value* are:

Constant	Value	Description
<b>vbPRBNUpper</b>	1	Use paper from the upper bin.
<b>vbPRBNLower</b>	2	Use paper from the lower bin.
<b>vbPRBNMiddle</b>	3	Use paper from the middle bin.
<b>vbPRBNManual</b>	4	Wait for manual insertion of each sheet of paper.
<b>vbPRBNEvelope</b>	5	Use envelopes from the envelope feeder.
<b>vbPRBNEnvManual</b>	6	Use envelopes from the envelope feeder, but wait for manual insertion.
<b>vbPRBNAuto</b>	7	(Default) Use paper from the current default bin.
<b>vbPRBNTractor</b>	8	Use paper fed from the tractor feeder.

<b>vbPRBNSmallFmt</b>	9	Use paper from the small paper feeder.
<b>vbPRBNLargeFmt</b>	10	Use paper from the large paper bin.
<b>vbPRBNLargeCapacity</b>	11	Use paper from the large capacity feeder.
<b>vbPRBNCassette</b>	14	Use paper from the attached cassette cartridge.

## Remarks

These constants are listed in the Visual Basic (VB) [object library](#) in the [Object Browser](#).

Not all of the bin options are available on every printer. Check the printer documentation for more specific descriptions of these options.

**Note** The effect of the properties of the **Printer** object depends on the driver supplied by the printer manufacturer. Some property settings may have no effect, or several different property settings may all have the same effect. Settings outside the accepted range may or may not produce an error. For more information, see the manufacturer's documentation for the specific driver.

© 2018 Microsoft



This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PaperSize Property

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets a value indicating the paper size for the current printer. Not available at design time.

### Syntax

*object*.**PaperSize** [= *value*]

The **PaperSize** property syntax has these parts:

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Value</i>	A value or constant specifying the paper size, as described in Settings.

### Settings

The settings for *value* are:

Constant	Value	Description
<b>vbPRPSLetter</b>	1	Letter, 8 1/2 x 11 in.
<b>vbPRPSLetterSmall</b>	2	Letter Small, 8 1/2 x 11 in.
<b>vbPRPSTabloid</b>	3	Tabloid, 11 x 17 in.
<b>vbPRPSLedger</b>	4	Ledger, 17 x 11 in.
<b>vbPRPSLegal</b>	5	Legal, 8 1/2 x 14 in.
<b>vbPRPSStatement</b>	6	Statement, 5 1/2 x 8 1/2 in.
<b>vbPRPSExecutive</b>	7	Executive, 7 1/2 x 10 1/2 in.
<b>vbPRPSA3</b>	8	A3, 297 x 420 mm
<b>vbPRPSA4</b>	9	A4, 210 x 297 mm

<b>vbPRPSA4Small</b>	10	A4 Small, 210 x 297 mm
<b>vbPRPSA5</b>	11	A5, 148 x 210 mm
<b>vbPRPSB4</b>	12	B4, 250 x 354 mm
<b>vbPRPSB5</b>	13	B5, 182 x 257 mm
<b>vbPRPSFolio</b>	14	Folio, 8 1/2 x 13 in.
<b>vbPRPSQuarto</b>	15	Quarto, 215 x 275 mm
<b>vbPRPS10x14</b>	16	10 x 14 in.
<b>vbPRPS11x17</b>	17	11 x 17 in.
<b>vbPRPSNote</b>	18	Note, 8 1/2 x 11 in.
<b>vbPRPSEnv9</b>	19	Envelope #9, 3 7/8 x 8 7/8 in.
<b>vbPRPSEnv10</b>	20	Envelope #10, 4 1/8 x 9 1/2 in.
<b>vbPRPSEnv11</b>	21	Envelope #11, 4 1/2 x 10 3/8 in.
<b>vbPRPSEnv12</b>	22	Envelope #12, 4 1/2 x 11 in.
<b>vbPRPSEnv14</b>	23	Envelope #14, 5 x 11 1/2 in.
<b>vbPRPSCSheet</b>	24	C size sheet
<b>vbPRPSDSheet</b>	25	D size sheet
<b>vbPRPSESheet</b>	26	E size sheet
<b>vbPRPSEnvDL</b>	27	Envelope DL, 110 x 220 mm
<b>vbPRPSEnvC3</b>	29	Envelope C3, 324 x 458 mm
<b>vbPRPSEnvC4</b>	30	Envelope C4, 229 x 324 mm
<b>vbPRPSEnvC5</b>	28	Envelope C5, 162 x 229 mm
<b>vbPRPSEnvC6</b>	31	Envelope C6, 114 x 162 mm
<b>vbPRPSEnvC65</b>	32	Envelope C65, 114 x 229 mm
<b>vbPRPSEnvB4</b>	33	Envelope B4, 250 x 353 mm
<b>vbPRPSEnvB5</b>	34	Envelope B5, 176 x 250 mm
<b>vbPRPSEnvB6</b>	35	Envelope B6, 176 x 125 mm
<b>vbPRPSEnvItaly</b>	36	Envelope, 110 x 230 mm

<b>vbPRPSEnvMonarch</b>	37	Envelope Monarch, 3 7/8 x 7 1/2 in.
<b>vbPRPSEnvPersonal</b>	38	Envelope, 3 5/8 x 6 1/2 in.
<b>vbPRPSFanfoldUS</b>	39	U.S. Standard Fanfold, 14 7/8 x 11 in.
<b>vbPRPSFanfoldStdGerman</b>	40	German Standard Fanfold, 8 1/2 x 12 in.
<b>vbPRPSFanfoldLglGerman</b>	41	German Legal Fanfold, 8 1/2 x 13 in.
<b>vbPRPSUser</b>	256	User-defined

## Remarks

These constants are listed in the Visual Basic (VB) [object library](#) in the [Object Browser](#).

Setting a printer's **Height** or **Width** property automatically sets **PaperSize** to **vbPRPSUser**.

**Note** The effect of the properties of the **Printer** object depends on the driver supplied by the printer manufacturer. Some property settings may have no effect, or several different property settings may all have the same effect. Settings outside the accepted range may or may not produce an error. For more information, see the manufacturer's documentation for the specific driver.

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Parent Property

[See Also](#) [Example](#) [Applies To](#)

Returns the form, object, or collection that contains a control or another object or collection.

### Syntax

*object*.**Parent**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

### Remarks

Use the **Parent** property to access the properties, methods, or controls of an object's parent. For example:

```
MyButton.Parent.MousePointer = 4
```

The **Parent** property is useful in an application in which you pass objects as arguments. For example, you could pass a control variable to a general procedure in a module, and use the **Parent** property to access its parent form.

There is no relationship between the **Parent** property and the **MDIChild** property. There is, however, a parent-child relationship between an **MDIForm** object and any **Form** object that has its **MDIChild** property set to **True**.

© 2018 Microsoft

# Visual Basic Reference

## Parent Property Example

This example passes a control from a form that doesn't have the focus to a procedure in a module, and then displays the state of the control on the parent form. To try this example, create three forms: Form1, containing a **CommandButton** control, and Form2 and Form3, each containing a **CheckBox** control. You must also create a new module (click Add Module in the Project menu). Paste the code into the Declarations sections of the respective forms or module, and then press F5 to run the program.

' Enter this code into Form1.

```
Private Sub Form_Load ()  
    Form2.Show    ' Display all forms.  
    Form3.Show  
    Form2.AutoRedraw = True  
    Form3.AutoRedraw = True  
End Sub
```

```
Private Sub Command1_Click ()  
    ReadCheckBox Form2.Check1    ' Call procedure in other module  
    ReadCheckBox Form3.Check1    ' and send control as argument.  
End Sub
```

' Enter this code into Module1.

```
Sub ReadCheckBox (Source As Control)  
    If Source.Value Then  
        Source.Parent.Cls    ' Clear parent form.  
        Source.Parent.Print "CheckBox is ON."    ' Display on parent form.  
    Else  
        Source.Parent.Cls    ' Clear parent form.  
        Source.Parent.Print "CheckBox is OFF."    ' Display on parent form.  
    End If  
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## Parent Property (ButtonMenu Object)

See Also   Example   [Applies To](#)

Returns or sets a reference to the parent of a **ButtonMenu** object.

### Syntax

*object*.**Parent** [= *button*]

The **Parent** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>button</i>	The parent <b>Button</b> object of the <b>ButtonMenu</b> object.

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## Parent Property (Node Object)

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets the parent object of a **Node** object. Available only at run time.

### Syntax

```
object.Parent[ = node]
```

The **Parent** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>node</i>	A <b>Node</b> object that becomes the parent of the object.

### Remarks

At run time, an error occurs if you set this property to an object that creates a loop. For example, you cannot set any **Node** to become a child **Node** of its own descendants.

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore, you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).Parent
    .Text = "New text"
    .Key = "New key"
    .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodParent As Node
' Get a reference to the parent of Node x.
Set NodParent = TreeView1.Nodes(x).Parent
' Use this reference to perform operations on the Parent Node.
With nodParent
    .Text = "New text"     ' Change the text.
    .Key = "New key"     ' Change key.
    .SelectedImage = 3     ' Change SelectedImage.
End With
```

# Visual Basic: Windows Controls

## Parent Property Example (Node Object)

This example adds several **Node** objects to a **TreeView** control. After you select a **Node** object, you can then click and drag it to any other **Node** to make it a child of the target **Node**. To try the example, place **TreeView** and **ImageList** controls on a form and paste the code into the form's Declaration section. Run the example and drag **Node** objects onto other **Node** objects to see the result.

**Note** The graphics files in the code below can be found on Disk 1 of the Visual Basic or Visual Studio CDs, in the Common\Graphics directory. Change the path in the code, or copy the graphics files to your hard disk before running the code.

```
' Declare global variables.
Dim indrag As Boolean ' Flag that signals a Drag Drop operation.
Dim nodX As Object ' Item that is being dragged.

Private Sub Form_Load()
    ' Load a bitmap into an ImageList control.
    Dim imgX As ListImage
    Dim BitmapPath As String
    BitmapPath = "icons\mail\mail01a.ico" ' Change to a valid path.
    Set imgX = ImageList1.ListImages.Add(, , LoadPicture(BitmapPath))

    ' Initialize TreeView control and create several nodes.
    TreeView1.ImageList = ImageList1
    Dim nodX As Node ' Create a tree.
    Set nodX = TreeView1.Nodes.Add(, , "Parent1", 1)
    Set nodX = TreeView1.Nodes.Add(, , "Parent2", 1)
    Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 1", 1)
    Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 2", 1)
    Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 3", 1)
    Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 4", 1)
    Set nodX = TreeView1.Nodes.Add(3, tvwChild, , "Child 5", 1)
    nodX.EnsureVisible ' Expand tree to show all nodes.
End Sub

Private Sub TreeView1_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
    Set nodX = TreeView1.SelectedItem ' Set the item being dragged.
    Set TreeView1.DropHighlight = Nothing
End Sub

Private Sub TreeView1_MouseMove _
(Button As Integer, Shift As Integer, x As Single, y As Single)
    If Button = vbLeftButton Then ' Signal a Drag operation.
        indrag = True ' Set the flag to true.
        ' Set the drag icon with the CreateDragImage method.
        TreeView1.DragIcon = TreeView1.SelectedItem.CreateDragImage
        TreeView1.Drag vbBeginDrag ' Drag operation.
    End If
End Sub

Private Sub TreeView1_DragDrop(Source As Control, x As Single, y As Single)
    ' If user didn't move mouse or released it over an invalid area.
    If TreeView1.DropHighlight Is Nothing Then
```



```

    indrag = False
    Exit Sub
Else
    ' Set dragged node's parent property to the target node.
    On Error GoTo checkerror ' To prevent circular errors.
    Set nodX.Parent = TreeView1.DropHighlight
    Cls
    Print TreeView1.DropHighlight.Text & _
    " is parent of " & nodX.Text
    ' Release the DropHighlight reference.
    Set TreeView1.DropHighlight = Nothing
    indrag = False
    Exit Sub ' Exit if no errors occurred.
End If

checkerror:
    ' Define constants to represent Visual Basic errors code.
    Const CircularError = 35614
    If Err.Number = CircularError Then
        Dim msg As String
        msg = "A node can't be made a child of its own children."
        ' Display the message box with an exclamation mark icon
        ' and with OK and Cancel buttons.
        If MsgBox(msg, vbExclamation & vbOKCancel) = vbOK Then
            ' Release the DropHighlight reference.
            indrag = False
            Set TreeView1.DropHighlight = Nothing
            Exit Sub
        End If
    End If
End Sub

Private Sub TreeView1_DragOver(Source As Control, x As Single, y As Single, State As Integer)
    Set TreeView1.DropHighlight = TreeView1.HitTest(x, y)
End Sub

```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Parent Property (UserControl Object)

See Also   Example   Applies To

Returns a reference to the container object on which the control is sited. Not available at design time and read-only at [run time](#).

### Syntax

*object*.**Parent**

The **Parent** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.

### Remarks

The **Parent** property returns a reference to the container object even when the **UserControl**'s **AmbientProperties** object does not provide a **Parent** property. You can use the **Parent** property of the **UserControl** object to access the container's object model.

By testing `TypeName(Parent)`, you can determine what container your control is sited on.

- **Excel** returns the workbook.
- **Word** returns the document.
- **Powerpoint** returns the presentation.
- **Visual Basic** returns the form.
- **Internet Explorer** returns an object whose `Script` property returns the `IOmWindow` object.

For example, in Internet Explorer, the following code will change the background color of the HTML page on which your control is located:

```
Parent.Script.get_document.bgColor = "Blue"
```

More information on the Internet Explorer Scripting Object Model can be found on Microsoft's Web site.

**Important** Always use *late binding* for calls to the Internet Explorer Scripting Object Model. Using early binding will almost certainly cause compatibility problems in the future, while late binding will always work. In other containers, you can use

early binding.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## ParentCommandName Property

See Also   Example   [Applies To](#)

Returns or sets the name of an existing parent **DECommand** object to which you are linking the child **DECommand** object. This property is only used for relation hierarchies.

### Syntax

*object*.**ParentCommandName** [=*string*]

The **ParentCommandName** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an item in the Applies To list.
<i>string</i>	A <a href="#">string expression</a> that specifies the name of an existing DECommand object to which you are linking the child DECommand object.

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## ParentControls Property

[See Also](#)   [Example](#)   [Applies To](#)

Returns a collection of the other controls in the controls container. The **ParentControls** property is not available at the controls authoring time, and read-only at the controls run time.

### Syntax

*object*.**ParentControls**

The **ParentControls** property syntax has this part:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.

### Remarks

In most cases, the container of the control will be a form; this collection functions in a similar manner to the Controls collection on the form, but will also contain the form itself.

This collection is useful if the control wants to perform some action on the controls on the form; the control can iterate through the collection.

Controls cannot be added or removed by the developer who uses the control through this collection; the controls must be changed in whatever manner the container allows.

The contents of this collection is determined entirely by the container.

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## ParentControlsType Property

See Also   Example   [Applies To](#)

Returns or sets a value that determines whether the **ParentControls** collection contains references to controls incorporating the container's Extender object, or to controls without the Extender.

### Syntax

*object*.**ParentControlsType** [= *type*]

The **ParentControlsType** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>type</i>	An integer or named constant that indicates what the ParentControls collection will return.

### Settings

The settings for *type* are as follows:

Constant	Value	Description
<b>vbExtender</b>	1	(Default) The ParentControls collection will return the control and extender.
<b>vbNoExtender</b>	0	The ParentControls collection will return the control itself, without the extender.

### Remarks

The **ParentControls** collection allows you to access the other controls on a container where your control has been sited. The default is for the references to these controls to include the Extender object properties and methods provided by the container.

Some containers, such as Internet Explorer, provide an extender object that cannot be used by Visual Basic. In such a container, Visual Basic will raise an error when you attempt to access the objects in **ParentControls** using the default settings.

You can access the controls on an HTML page in Internet Explorer by setting **ParentControlsType** to **vbNoExtender**, so that **ParentControls** will contain references to the controls themselves, without the extender properties and methods.

Because the property can be set at run time, you can switch back and forth between **vbExtender** and **vbNoExtender** depending on what container your control is sited on. If necessary, you can alternate between the settings in containers that support both.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## ParentField Property

See Also   Example   [Applies To](#)

Returns or sets the name of the Field object that the parent **DECommand** object uses in its hierarchical comparison condition.

**Syntax**

*object*.**ParentField** [=string]

The **ParentField** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an item in the Applies To list.
<i>string</i>	A <a href="#">string expression</a> that specifies the name of the Field object that the parent DECommand object uses in its hierarchical comparison condition.



This documentation is archived and is not being maintained.

# Visual Basic for Applications Reference

Visual Studio 6.0

## ParentFolder Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

### Description

Returns the folder object for the parent of the specified file or folder. Read-only.

### Syntax

*object*.**ParentFolder**

The *object* is always a **File** or **Folder** object.

### Remarks

The following code illustrates the use of the **ParentFolder** property with a file:

```
Sub ShowFileAccessInfo(filespec)
    Dim fs, f, s
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.GetFile(filespec)
    s = UCase(f.Name) & " in " & UCase(f.ParentFolder) & vbCrLf
    s = s & "Created: " & f.DateCreated & vbCrLf
    s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
    s = s & "Last Modified: " & f.DateLastModified
    MsgBox s, 0, "File Access Info"
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: MSComm Control

Visual Studio 6.0

## ParityReplace Property

[See Also](#)   [Example](#)   [Applies To](#)

Sets and returns the character that replaces an invalid character in the data stream when a parity error occurs.

### Syntax

*object*.**ParityReplace** [ = *value* ]

The **ParityReplace** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A <a href="#">string expression</a> representing a character, as described in Remarks.

### Remarks

The *parity bit* refers to a bit that is transmitted along with a specified number of data bits to provide a small amount of error checking. When you use a parity bit, the **MSComm** control adds up all the bits that are set (having a value of 1) in the data and tests the sum as being odd or even (according to the parity setting used when the port was opened).

By default, the control uses a question mark (?) character for replacing invalid characters. Setting **ParityReplace** to an empty string ("") disables replacement of the character where the parity error occurs. The **OnComm** event is still fired and the **CommEvent** property is set to **comEventRXParity**.

The **ParityReplace** character is used in a byte-oriented operation, and must be a single-byte character. You can specify any ANSI character code with a value from 0 to 255.

### Data Type

String

This documentation is archived and is not being maintained.

# Visual Basic: ADO Data Control

Visual Studio 6.0

## Password Property (ADO Data Control)

See Also   Example   Applies To

Sets the password used during creation of an **ADO Recordset** object.

### Syntax

*object*.**Password** [= *string*]

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>string</i>	Sets the password for the user named in the <b>UserName</b> property.

### Remarks

This property setting is write-only it may only be provided in code, it cannot be read back from the **Password** property.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Internet Control

Visual Studio 6.0

## Password Property (Internet Transfer Control)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the password that will be sent with the request to log on to remote computers. If this property is left blank, the control will send a default password.

### Syntax

*object*.**Password** = *string*

The **Password** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>string</i>	The password to be sent when logging on to a remote computer.

### Remarks

The default password the control sends will depend on the exact scenario, as shown in the table below:

UserName Property	Password Property	UserName sent to FTP Server	Password sent to FTP server
Null or ""	Null or ""	"anonymous"	User's email name
Non-null string	Null or ""	UserName property	""
Null	Non-null string	Error	Error
Non-null string	Non-null string	UserName property	Password property

This documentation is archived and is not being maintained.

# Visual Basic: MAPI Controls

Visual Studio 6.0

## Password Property (MAPISession Control)

[See Also](#)   [Example](#)   [Applies To](#)

Specifies the account password associated with the **UserName** property.

### Syntax

`object.Password[ = value]`

The **Password** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A <a href="#">string expression</a> specifying the account password

### Remarks

On computers that have Microsoft Exchange or Microsoft Outlook installed, the **UserName** property sets the Profile to be used when establishing a session. Profiles include the actual user name and password to be used, making the Password property unused. New profiles are set up by clicking on the **Mail and Fax** icon in the Windows Control Panel.

On computers that have Microsoft Mail installed, an empty string in this property indicates that a sign-on dialog box with an empty password field should be generated. The default is an empty string.

### Data Type

String

This documentation is archived and is not being maintained.

# Visual Basic: RDO Data Control

Visual Studio 6.0

## Password Property (Remote Data)

[See Also](#) [Example](#) [Applies To](#)

Represents the password used during creation of an **rdoEnvironment** object.

### Syntax

*object*.**Password**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

### Remarks

The **rdoDefaultPassword** property of the **rdoEngine** object is used as a default if no password is provided. The initial default password is "".

This property setting is write-only it may only be provided in code, it cannot be read back from the **Password** property.

The password is set:

- When the **rdoEnvironment** is created automatically by the **RemoteData** control.
- By the first reference to a **RemoteData** object.
- When the **rdoCreateEnvironment** method is executed.
- In the connect string via the **Connect** property or the **Connect** argument of the **OpenConnection** method.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PasswordChar Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value indicating whether the characters typed by a user or placeholder characters are displayed in a **TextBox** control; returns or sets the character used as a placeholder.

### Syntax

*object*.**PasswordChar** [= *value*]

The **PasswordChar** property syntax has these parts:

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Value</i>	A <a href="#">string expression</a> specifying the placeholder character.

### Remarks

Use this property to create a password field in a dialog box. Although you can use any character, most Windows-based applications use the asterisk (\*) (**Chr**(42)).

This property doesn't affect the **Text** property; **Text** contains exactly what the user types or what was set from code. Set **PasswordChar** to a zero-length string (""), which is the default, to display the actual text.

You can assign any string to this property, but only the first character is significant; all others are ignored.

**Note** If the **MultiLine** Property is set to **True**, setting the **PasswordChar** property will have no effect.

# Visual Basic Reference

## PasswordChar Property Example

This example illustrates how the **PasswordChar** property affects the way a **TextBox** control displays text. To try this example, paste the code into the Declarations section of a form that contains a **TextBox**, and then press F5 and click the form. Each time you click the form, the text toggles between an asterisk (\*) password character and plain text.

```
Private Sub Form_Click ()  
    If Text1.PasswordChar = "" Then  
        Text1.PasswordChar = "*"   
    Else  
        Text1.PasswordChar = ""  
    End If  
End Sub
```

© 2018 Microsoft



This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PasteOK Property

[See Also](#) [Example](#) [Applies To](#)

Returns a value that determines whether the contents of the system Clipboard can be pasted into the **OLE** container control.

### Syntax

*object*.**PasteOK**

The *object* is an object expression that evaluates to an object in the Applies To list.

### Remarks

When this property setting is **True**, you can paste the contents of the system Clipboard into the **OLE** container control.

Use the **OLETypeAllowed** property to specify the type of object (linked or embedded) you want to paste into the **OLE** container control. Once you successfully paste an object into the **OLE** container control, you can check the **OLEType** property setting to determine the type of object that was created.

You can use this property if your application supports a Paste command on an Edit menu. If **PasteOK** is **False**, disable the menu command; otherwise, it can be enabled. Enable and disable menu commands by setting their **Enabled** property to **True** or **False**, respectively.

You paste an object into the **OLE** container control with the **Paste** method.

To provide more flexibility to the user, display a Paste Special dialog box when the user chooses the Edit Paste command. (Set **OLETypeAllowed** = 2, and then use the **PasteSpecialDlg** method.) When this dialog box is displayed, an object is pasted onto the system Clipboard based on the user's selections in the dialog box.

© 2018 Microsoft

# Visual Basic Reference

## PasteOK Property Example

This example pastes an object in the **OLE** container control if the **PasteOK** property setting is **True**. Otherwise, the example displays a message box.

```
Private Sub mnuEditPaste_Click ()  
    ' Check value of PasteOK.  
    If Ole1.PasteOK Then  
        Ole1.Paste      ' Enable Paste command if True.  
    Else                ' Otherwise, disable Paste  
        mnuEditPaste.Enabled = False    ' menu command and give  
        MsgBox "Can't paste."           ' appropriate message.  
    End If  
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic for Applications Reference

Visual Studio 6.0

## Path Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

### Description

Returns the path for a specified file, folder, or drive.

### Syntax

*object*.**Path**

The *object* is always a **File**, **Folder**, or **Drive** object.

### Remarks

For drive letters, the root drive is not included. For example, the path for the C drive is C:, not C:\.

The following code illustrates the use of the **Path** property with a **File** object:

```
Sub ShowFileAccessInfo(filespec)
    Dim fs, d, f, s
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.GetFile(filespec)
    s = UCase(f.Path) & vbCrLf
    s = s & "Created: " & f.DateCreated & vbCrLf
    s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
    s = s & "Last Modified: " & f.DateLastModified
    MsgBox s, 0, "File Access Info"
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Path Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the current path. Not available at design time. For the **App** object, read-only at [run time](#).

### Syntax

*object*.**Path** [= *pathname*]

The **Path** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>pathname</i>	A <a href="#">string expression</a> that evaluates to the path name.

### Remarks

The value of the **Path** property is a string indicating a path, such as C:\Ob or C:\Windows\System. For a **DirListBox** or **FileListBox** control, the default is the current path when the control is created at run time. For the **App** object, **Path** specifies the path of the project .VBP file when running the application from the development environment or the path of the .exe file when running the application as an executable file.

Use this property when building an application's file-browsing and manipulation capabilities. Setting the **Path** property has effects on a control similar to the MS-DOS **chdir** command relative paths are allowed with or without a drive specification. Specifying only a drive with a colon (:) selects the current directory on that drive.

The **Path** property can also be set to a qualified network path without a drive connection using the following syntax:

\\*servername*\sharename\path

The preceding syntax changes the **Drive** property to a zero-length string ("").

Changing the value of **Path** has these effects:

- For a **DirListBox** control, generates a Change event.
- For a **FileListBox** control, generates a PathChange event.

**Note** For **DirListBox**, the return value of **Path** is different from that of `List(ListIndex)`, which returns only the selection.

# Visual Basic Reference

## Path Property Example

This example displays a list of files for the selected drive and directory. To try this example, paste the code into the Declarations section of a form that contains **DriveListBox**, **DirListBox**, and **FileListBox** controls. Press F5. Use the mouse to change the drive or directory.

```
Private Sub Drive1_Change ()  
    Dir1.Path = Drive1.Drive    ' Set directory path.  
End Sub
```

```
Private Sub Dir1_Change ()  
    File1.Path = Dir1.Path    ' Set file path.  
End Sub
```

© 2018 Microsoft

# Visual Basic: Windows Controls

## PathSeparator Property Example

This example adds several **Node** objects to a **TreeView** control, and uses an **OptionButton** control array to change the **PathSeparator** property. To try the example, place a **TreeView** control and an **OptionButton** control array on a form, and paste the code into the form's Declarations section. Run the example, select a **Node**, and click the form. Change the **PathSeparator** property value using the **OptionButtons**.

```
Private Sub Form_Load
    TreeView1.BorderStyle = vbFixedSingle ' Show border.
    ' Label OptionButton controls with Style choices.
    Option1(0).Caption = "/"
    Option1(1).Caption = "-"
    Option1(2).Caption = ":"

    ' Select the last option, and set the initial Style
    Option2(1).Value = True
    Treeview1.PathSeparator = Option1(1).Caption

    Dim nodX As Node
    Dim i As Integer
    Set nodX = TreeView1.Nodes.Add(,,CStr(1)) ' Add first node.

    For i = 1 to 5 ' Add other nodes.
        Set nodX = TreeView1.Nodes.Add(i,tvwChild,,CStr(i + 1))
    Next i

    nodX.EnsureVisible ' Ensure all are visible.
End Sub

Private Sub Option1_Click(Index as Integer)
    ' Change the delimiter character.
    TreeView1.PathSeparator = Option1(Index).Caption
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
    ' Show path in form's caption.
    Me.Caption = Node.FullPath
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## PathSeparator Property (TreeView Control)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the delimiter character used for the path returned by the **FullPath** property.

### Syntax

```
object.PathSeparator [ = string]
```

The **PathSeparator** syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>string</i>	A string that determines the <b>PathSeparator</b> , usually a single character.

### Remarks

The default character is "\".

# Visual Basic: Windows Controls

## PathSeparator Property Example

This example adds several **Node** objects to a **TreeView** control, and uses an **OptionButton** control array to change the **PathSeparator** property. To try the example, place a **TreeView** control and an **OptionButton** control array on a form, and paste the code into the form's Declarations section. Run the example, select a **Node**, and click the form. Change the **PathSeparator** property value using the **OptionButtons**.

```
Private Sub Form_Load
    TreeView1.BorderStyle = vbFixedSingle ' Show border.
    ' Label OptionButton controls with Style choices.
    Option1(0).Caption = "/"
    Option1(1).Caption = "-"
    Option1(2).Caption = ":"

    ' Select the last option, and set the initial Style
    Option2(1).Value = True
    Treeview1.PathSeparator = Option1(1).Caption

    Dim nodX As Node
    Dim i As Integer
    Set nodX = TreeView1.Nodes.Add(,,,CStr(1)) ' Add first node.

    For i = 1 to 5 ' Add other nodes.
        Set nodX = TreeView1.Nodes.Add(i,tvwChild,,CStr(i + 1))
    Next i

    nodX.EnsureVisible ' Ensure all are visible.
End Sub

Private Sub Option1_Click(Index as Integer)
    ' Change the delimiter character.
    TreeView1.PathSeparator = Option1(Index).Caption
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
    ' Show path in form's caption.
    Me.Caption = Node.FullPath
End Sub
```

© 2018 Microsoft



This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Pattern Property

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets a value indicating the filenames displayed in a **FileListBox** control at [run time](#).

### Syntax

*object*.**Pattern** [= *value*]

The **Pattern** property syntax has these parts:

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Value</i>	A <a href="#">string expression</a> indicating a file specification, such as "*.*" or "*.FRM". The default is "*.*", which returns a list of all files. In addition to using wildcard characters, you can also use multiple patterns separated by semicolons (;). For example, "*.exe; *.bat" would return a list of all executable files and all MS-DOS batch files.

### Remarks

The **Pattern** property plays a key role in designing an application's file-browsing and manipulation capabilities. Use **Pattern** in combination with other file-control properties to provide the user with ways to explore files or groups of similar files. For example, in an application dedicated to launching other programs, you could designate that only .exe files be displayed in the file list box (\*.exe). Other key file-control properties include **Drive**, **FileName**, and **Path**.

Changing the value of the **Pattern** property generates a PatternChange event.

# Visual Basic Reference

## Pattern Property Example

This example updates a **TextBox** control with the new pattern selected in a **FileListBox** control. The controls are set up so that when the user enters a pattern in the **TextBox**, such as \*.txt, it's reflected in the **FileListBox**, much like the interaction you see in a typical File Open dialog box in a Windows-based application. If a full path such as C:\Bin\\*.exe is entered into the **TextBox** control, the text is automatically parsed into path and pattern components by the **FileListBox** control. To try this example, paste the code into the Declarations section of a form that contains the following controls: a **DirListBox**, a **FileListBox**, a **TextBox**, and a **CommandButton**. Press F5 and type a valid file pattern into the **TextBox**.

```
Private Sub Form_Load ()
    Command1.Default = True    ' Set Default property.
End Sub

Private Sub Command1_Click ()
    ' Text is parsed into path and pattern components.
    File1.FileName = Text1.Text
    Dir1.Path = File1.Path    ' Set directory path.
End Sub

Private Sub File1_PatternChange ()
    Text1.Text = File1.Pattern    ' Set text to new pattern.
End Sub

Private Sub Dir1_Change
    File1.Path = Dir1.Path    ' Set file list box path.
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

## Visual Studio 6.0

*Visual Basic: MSChart Control*

# PatternColor Property

See Also   Example   [Applies To](#)


Returns a reference to a **VtColor** object that describes the pattern color used to fill a chart element.

## Syntax

*object*.**PatternColor**

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

© 2018 Microsoft



This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

# Pen Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a reference to a **Pen** object that describes the color and pattern of lines or edges on chart elements.

## Syntax

*object*.**Pen**

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

# PercentBasis Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the type of percentage used to plot chart values on a percent axis.

## Syntax

*object*.**PercentBasis** [ = *type* ]

The **PercentBasis** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>type</i>	A <a href="#">VtChPercentAxisBasis</a> constant used to describe the percentage used to plot percent axis values.

## Remarks

This property has an effect only when the **Type** property is set to **VtChScaleTypePercent**.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

# PercentFormat Property

See Also   Example   [Applies To](#)

Returns or sets a string that describes the format used to display the label as a percent.

**Syntax**

*object*.**PercentFormat** [ = *format*]

The **PercentFormat** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>format</i>	String. Describes the format used to display a label as a percent.

**Remarks**

Use the **DataPointLabel** object's **Component** property to change the label type.

The following table lists several examples of percentage format strings. The values listed at left are the valid formats.

	3	-3	.3
0%	300%	-300%	30%
0.0%	300.0%	-300.0%	30.0%
0.00%	300.00%	-300.00%	30.00%

This documentation is archived and is not being maintained.

# Visual Basic: RDO Data Control

Visual Studio 6.0

## PercentPosition Property (Remote Data)

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets a value that indicates or changes the approximate location of the [current row](#) in the **rdoResultset** object based on a percentage of the rows in the **rdoResultset**.

### Syntax

*object*.**PercentPosition** [= *value*]

The **PercentPosition** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A number between 0.0 and 100.00. (Data type is <a href="#">Single</a> )

### Remarks

To indicate or change the approximate position of the current row in an **rdoResultset**, you can check or set the **PercentPosition** property. Before you set or check the **PercentPosition** property, populate the **rdoResultset** by moving to the last row. If you use the **PercentPosition** property *before* fully populating the **rdoResultset**, the amount of movement is relative to the number of rows accessed as indicated by the **RowCount** property. You can move to the last row and populate the **rdoResultset** using the **MoveLast** method.

**Note** Using the **PercentPosition** property to move the current row to a specific row in an **rdoResultset** isn't recommended the **Bookmark** property or **AbsolutePosition** property is better suited for this task.

Once you set the **PercentPosition** property to a value, the row at the approximate position corresponding to that value becomes current, and the **PercentPosition** property is reset to a value that reflects the approximate position of the current row. For example, if your **rdoResultset** contains only five rows, and you set its **PercentPosition** value to 77, the value returned from the **PercentPosition** property might be 80, not 77.

You can use the **PercentPosition** property with a scroll bar on a **Form** or **TextBox** to indicate the location of the current row in an **rdoResultset**.

The **PercentPosition** property is not supported by all [cursor](#) types and driver combinations. For example, this property applies only to [keyset-type](#) and [dynamic-type](#) **rdoResultset** objects. If the setting is not supported, the **PercentPosition** property returns 50. If the position cannot be set, no movement occurs.

# Visual Basic: RDO Data Control

## PercentPosition Property Example

This example illustrates use of the **PercentPosition** property. In this example a list of publishers is generated and when one of these is chosen, a list of associated titles is displayed in a **DataGrid** control. When the scroll bar associated with the grid is manipulated, the relative location of the selected row is determined by examining the **PercentPosition** property and displayed. See the **AbsolutePosition** property example for further details on this example.

```
Dim rs As rdoResultset

Private Sub Form_Load()
Dim Li As Integer
'
'   Fill Sections list combo box.
'
Set en = rdoEnvironments(0)
Set cn = en.OpenConnection(dsName:="", _
    Prompt:=rdDriverNoPrompt, _
    Connect:"uid=;pwd=;driver={SQL Server};" _
        & "server=BETAV486;database=pubs;")

MsRdc1.Connect = cn.Connect

Set rs = cn.OpenResultset _
    ("Select distinct Pub_Name, Pub_ID from Publishers", rdOpenStatic, rdConcurReadOnly)
Do Until rs.EOF
    If rs(0) = Null Then
    Else
        PubList.AddItem " " & rs!Pub_ID & ":" & rs!Pub_Name
    End If
    rs.MoveNext
Loop
PubList.ListIndex = 1
rs.Close

Publist_Click

End Sub

Private Sub MoveCRow_Change()
MoveCRow_Scroll
End Sub

Private Sub MoveCRow_Scroll()
PercentPoint = MoveCRow.Value & "%"
MsRdc1.Resultset.PercentPosition = MoveCRow.Value
End Sub

Private Sub Publist_Click()
SetSQL
If MsRdc1.Resultset.EOF Then
    MoveCRow.Enabled = False
Else
```



```
        MoveCRow.Enabled = True
        MsRdc1.Resultset.MoveFirst
    End If
End Sub

Sub SetSQL()
    Dim PubWanted As String
    PubWanted = Trim(Left(PubList, InStr(PubList, ":") - 1))
    Screen.MousePointer = vbHourglass

    MsRdc1.SQL = "select * from Titles" _
        & " where Pub_ID = '" _
        & PubWanted & "'" _
        & " order by Title"
    MsRdc1.Refresh
    Screen.MousePointer = vbDefault
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Persistable Property

See Also   Example   [Applies To](#)

Sets a value that determines if an object can save and restore data across instances. May only be set at design time.

### Syntax

```
object.Persistable [= number]
```

The **Persistable** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>number</i>	An integer that specifies persistence behavior, as described in Settings.

### Settings

The settings for *number* are:

Constant	Setting	Description
<b>vbNotPersistable</b>	0	(Default) The object can't be persisted.
<b>vbPersistable</b>	1	The object can be persisted.

### Remarks

The **Persistable** property is only available for classes that are public and creatable. When **Persistable** is set to **vbPersistable**, the following events are added to the class: **InitProperties**, **ReadProperties**, and **WriteProperties**. The **PropertyChanged** method is added to the class as well.

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Picture Property (ActiveX Controls)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a graphic to be displayed in a control. For the **OLE** container control, not available at design time and read-only at [run time](#).

### Syntax

`object.Picture [= picture]`

The **Picture** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>picture</i>	A <a href="#">string expression</a> specifying a file containing a graphic, as described in Settings.

### Settings

The settings for *picture* are:

Setting	Description
(None)	(Default) No picture.
(Bitmap, icon, metafile)	Specifies a graphic. You can load the graphic from the Properties window at design time. At run time, you can also set this property using the <b>LoadPicture</b> function on a bitmap, icon, or metafile.

### Remarks

At design time, you can transfer a graphic with the Clipboard using the Copy, Cut, and Paste commands on the Edit menu. At run time, you can use Clipboard methods such as **GetData**, **SetData**, and **GetFormat** with the nontext Clipboard [constants](#) **vbCFBitmap**, **vbCFMetafile**, and **vbCFDIB**, which are listed in the Visual Basic (VB) [object library](#) in the [Object Browser](#).

When setting the **Picture** property at design time, the graphic is saved and loaded with the form. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic isn't saved with the application. Use the **SavePicture** statement to save a graphic from a form or picture box into a file.

**Note** At run time, the **Picture** property can be set to any other object's **DragIcon**, **Icon**, **Image**, or **Picture** property, or you can assign it the graphic returned by the **LoadPicture** function. The exception to this is the **Picture** property of the **ListImages** object, which is a read-only property.

© 2018 Microsoft

# Visual Basic Reference

## Picture Property Example

This example loads icons from the Visual Basic icon library into two of three **PictureBox** controls. When you click the form, the third **PictureBox** is used to switch the icons. You can use any two icons. Paste the code into the Declarations section of a form that has three small **PictureBox** controls (for Picture3, set **Visible** = **False**). Press F5 to run the program, and then click the form.

```
Private Sub Form_Load ()
    ' Load the icons.
    Picture1.Picture = LoadPicture("ICONS\COMPUTER\TRASH02A.ICO")
    Picture2.Picture = LoadPicture("ICONS\COMPUTER\TRASH02B.ICO")
End Sub

Private Sub Form_Click ()
    ' Switch the icons.
    Picture3.Picture = Picture1.Picture
    Picture1.Picture = Picture2.Picture
    Picture2.Picture = Picture3.Picture
    ' Clear the third picture (not necessary if not visible).
    Picture3.Picture = LoadPicture()
End Sub
```

This example pastes a bitmap from the Clipboard into a **PictureBox** control. To find the value of Clipboard format constants (starting with **vbCF**), see the Visual Basic (VB) object library in the Object Browser. To try this example, paste the code into the Declarations section of a form that has a **PictureBox** control. Press F5, and then in another application, copy an icon onto the Clipboard, switch to Visual Basic, and click the form.

```
Private Sub Form_Click ()
    Picture1.Picture = Clipboard.GetData(vbCFDIB)
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## Picture Property (Coolbar Control)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a graphic to be displayed in a control.

### Syntax

*object*.**Picture** = **LoadPicture**(*pathname*)

*object*.**Picture** [= *picture*]

The **Picture** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>pathname</i>	A <a href="#">string expression</a> specifying the path and filename of the file containing a graphic, as described in Settings.
<i>picture</i>	The <b>Picture</b> property of a <b>Form</b> object, <b>PictureBox</b> control, or <b>Image</b> control.

### Settings

The settings for *picture* are:

Setting	Description
<b>(None)</b>	(Default) No picture.
<b>(Bitmap, GIF, JPEG)</b>	Specifies a graphic. You can load a graphic from the Property Page at design time. At run time, you can also set this property using the <b>LoadPicture</b> function.

### Remarks

The **Picture** property of the **CoolBar** control displays a background graphic across all bands on the **CoolBar**, behind any child controls. Each **Band** object on a **CoolBar** control also has a **Picture** property and a **UseCoolBarPicture** property which can be used to override the **Picture** property of the control.

This documentation is archived and is not being maintained.

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

## Picture Property (MSHFlexGrid)

[See Also](#)   [Example](#)   [Applies To](#)

Returns a picture of your **MSHFlexGrid**. This picture is suitable for printing, saving to disk, copying to the clipboard, or assigning to a different control.

**Syntax**

*object*.**Picture** [=*picture*]

The **Picture** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>picture</i>	A bitmap showing your <b>MSHFlexGrid</b> .

**Remarks**

The bitmap picture is a snapshot of your entire **MSHFlexGrid** on the **Project Form** window. It can, therefore, be very large. There are two ways to reduce the size of a bitmap picture. One option is to create a picture of a section of your **MSHFlexGrid**. To do this, write a routine to hide all elements you dont want to show, acquire the picture, and then restore the **MSHFlexGrid**.

Alternatively, you can set the **PictureType** property to 1 (Monochrome). However, this decreases not only the amount of memory used, but also the picture resolution.

© 2018 Microsoft

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

## ColData, RowData, Picture Properties Example

The following code shows how you can trap out-of-memory errors and automatically switch to monochrome mode using the **PictureType** property. In addition, it shows how to create a picture within the **MSHFlexGrid** that includes only the current selection.

**Note** If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub CopySelectedPictureToClipboard (myFlex As _
MSHFlexGrid)
    Dim i As Integer, tr As Long, lc As Long, _
    hl As Integer
    ' Get ready to operate.
    MyFlex.Redraw = False ' To eliminate flicker.
    hl = MyFlex.HighLight ' Save current settings.
    tr = MyFlex.TopRow
    lc = MyFlex.LeftCol
    MyFlex.HighLight = 0 ' No highlight on picture.
    ' Hide nonselected rows and columns.
    ' (Save original sizes in RowData/ColData
    ' properties.)
    For i = MyFlex.FixedRows To MyFlex.Rows - 1
        If i < MyFlex.Row Or i > MyFlex.RowSel Then
            MyFlex.RowData(i) = MyFlex.RowHeight(i)
            MyFlex.RowHeight(i) = 0
        End If
    Next
    For i = MyFlex.FixedCols To MyFlex.Cols - 1
        If i < MyFlex.Col Or i > MyFlex.ColSel Then
            MyFlex.ColData(i) = MyFlex.ColWidth(i)
            MyFlex.ColWidth(i) = 0
        End If
    Next
    ' Scroll to top left corner.
    MyFlex.TopRow = MyFlex.FixedRows
    MyFlex.LeftCol = MyFlex.FixedCols
    ' Copy picture.
    clipboard.Clear
    On Error Resume Next
    MyFlex.PictureType = 0 ' Color.
    clipboard.SetData MyFlex.Picture
    If Error <> 0 Then
        MyFlex.PictureType = 1 ' Monochrome.
        clipboard.SetData MyFlex.Picture
    Endif
    ' Restore control.
    For i = MyFlex.FixedRows To MyFlex.Rows - 1
        If i < MyFlex.Row Or i > MyFlex.RowSel Then
```



```
        MyFlex.RowHeight(i) =MyFlex.RowData(i)
    End If
Next
For i =MyFlex.FixedCols To MyFlex.Cols - 1
    If i < MyFlex.Col Or i > MyFlex.ColSel Then
        MyFlex.ColWidth(i) =MyFlex.ColData(i)
    End If
Next
MyFlex.TopRow =tr
MyFlex.LeftCol =lc
MyFlex.HighLight =hl
MyFlex.Redraw =True
End Sub
```

The following example shows how to set the **MSHFlexGrid's Picture** property to a **PictureBox** control:

**Note** If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Private Sub Form_Click ()
    Set Picture1.Picture =MSHFlexGrid1.Picture
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: PictureBox Control

Visual Studio 6.0

## Picture Property (PictureBox Control)

[See Also](#) [Example](#) [Applies To](#)

This property is the same as the standard Visual Basic **Picture** property except that it supports only bitmap (.bmp) files.

© 2018 Microsoft

# 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
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: MSTab Control

Visual Studio 6.0

## Picture Property (SSTab Control)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a graphic to be displayed in the current tab of an **SSTab** control.

### Syntax

*object*.**Picture** [ = *picture*]

The **Picture** property syntax has these parts:

Part	Description
<i>object</i>	An <a href="#">object expression</a> that evaluates to an <b>SSTab</b> control.
<i>picture</i>	A <a href="#">string expression</a> that designates a bitmap or icon to display on the current tab, as described in Settings.

### Settings

The settings for *picture* are:

Setting	Description
(None)	An object expression that evaluates to an <b>SSTab</b> control.
(Bitmap, icon, metafile)	A <a href="#">string expression</a> that designates a bitmap or icon to display on the current tab.

### Remarks

At design time, you set the **Picture** property for a tab by clicking that tab and then setting the property in the Properties window. At run time, you can set the **Picture** property using the **LoadPicture** function or the **Picture** property of another control or of a **Form** object. You can make any tab the current tab by setting the **Tab** property.

When setting the **Picture** property at design time, the graphic is saved and loaded with the **Form** object containing the **SSTab** control. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic isn't saved with the application.

Setting the **Picture** property affects the value of the **TabPicture** property for the current tab as well as displays the picture in the active tab.

# Visual Basic: MSTab Control

## Picture Property (SSTab Control) Example

This example loads a bitmap from a file and places that bitmap on the active tab. To try this example, put the **SSTab** and **CommandButton** controls on the **Form**. Then run the example.

```
Private Sub Command1_Click()  
    SSTab1.Picture = LoadPicture("c:\windows\cars.bmp")  
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Picture Property

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets a graphic to be displayed in a control. For the **OLE** container control, not available at design time and read-only at [run time](#).

### Syntax

`object.Picture [= picture]`

The **Picture** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>picture</i>	A <a href="#">string expression</a> specifying a file containing a graphic, as described in Settings.

### Settings

The settings for *picture* are:

Setting	Description
(None)	(Default) No picture.
(Bitmap, icon, metafile, GIF, JPEG)	Specifies a graphic. You can load the graphic from the Properties window at design time. At run time, you can also set this property using the <b>LoadPicture</b> function on a bitmap, icon, or metafile.

### Remarks

At design time, you can transfer a graphic with the Clipboard using the Copy, Cut, and Paste commands on the Edit menu. At run time, you can use Clipboard methods such as **GetData**, **SetData**, and **GetFormat** with the nontext Clipboard [constants](#) **vbCFBitmap**, **vbCFMetafile**, and **vbCFDIB**, which are listed in the Visual Basic (VB) [object library](#) in the [Object Browser](#).

When setting the **Picture** property at design time, the graphic is saved and loaded with the form. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic isn't saved with the application. Use the **SavePicture** statement to save a graphic from a form or picture box into a file.

**Note** At run time, the **Picture** property can be set to any other object's **DragIcon**, **Icon**, **Image**, or **Picture** property, or you can assign it the graphic returned by the **LoadPicture** function.

**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 either of these controls (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.

© 2018 Microsoft

# Visual Basic Reference

## Picture Property Example

This example loads icons from the Visual Basic icon library into two of three **PictureBox** controls. When you click the form, the third **PictureBox** is used to switch the icons. You can use any two icons. Paste the code into the Declarations section of a form that has three small **PictureBox** controls (for Picture3, set **Visible** = **False**). Press F5 to run the program, and then click the form.

```
Private Sub Form_Load ()  
    ' Load the icons.  
    Picture1.Picture = LoadPicture("ICONS\COMPUTER\TRASH02A.ICO")  
    Picture2.Picture = LoadPicture("ICONS\COMPUTER\TRASH02B.ICO")  
End Sub  
  
Private Sub Form_Click ()  
    ' Switch the icons.  
    Picture3.Picture = Picture1.Picture  
    Picture1.Picture = Picture2.Picture  
    Picture2.Picture = Picture3.Picture  
    ' Clear the third picture (not necessary if not visible).  
    Picture3.Picture = LoadPicture()  
End Sub
```

This example pastes a bitmap from the Clipboard into a **PictureBox** control. To find the value of Clipboard format constants (starting with **vbCF**), see the Visual Basic (VB) object library in the Object Browser. To try this example, paste the code into the Declarations section of a form that has a **PictureBox** control. Press F5, and then in another application, copy an icon onto the Clipboard, switch to Visual Basic, and click the form.

```
Private Sub Form_Click ()  
    Picture1.Picture = Clipboard.GetData(vbCFDIB)  
End Sub
```

© 2018 Microsoft



This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PictureAlignment Property

See Also   Example   [Applies To](#)

Returns or sets a value that determines where the picture will appear in a Data Report designer's Image control.

### Syntax

*object*.**PictureAlignment** [=*integer*]

The **PictureAlignment** property syntax has these parts:

Part	Description
<i>object</i>	Required. An object expression that evaluates to an object in the Applies To list.
<i>integer</i>	Optional. A numeric expression that specifies the position of the picture, as shown in Settings.

### Settings

The settings for *integer* are:

Constant	Value	Description
<b>rptPATopLeft</b>	0	The picture appears at the top left.
<b>rptPATop</b>	1	The picture appears at the top.
<b>rptPATopRight</b>	2	The picture appears at the top right.
<b>rptPARight</b>	3	The picture appears at the right.
<b>rptPABottomRight</b>	4	The picture appears at the bottom right.
<b>rptPABottom</b>	5	The picture appears at the bottom.
<b>rptPABottomLeft</b>	6	The picture appears at the bottom left.
<b>rptPALeft</b>	7	The picture appears at the left.
<b>rptPACenter</b>	8	The picture appears centered.

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## PictureAlignment Property (ListView Control)

See Also   Example   Applies To

Returns or sets a value that determines the picture alignment of an object.

### Syntax

*object*.**PictureAlignment** [= *integer*]

The **PictureAlignment** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>integer</i>	A <a href="#">numeric expression</a> that determines the alignment of pictures as shown in Settings.

### Settings

The settings for *integer* are:

Constant	Value	Description
<b>lvwTopLeft</b>	0	Top left.
<b>lvwTopRight</b>	1	Top right.
<b>lvwBottomLeft</b>	2	Bottom left.
<b>lvwBottomRight</b>	3	Bottom right.
<b>lvwCenter</b>	4	Centered.
<b>lvwTile</b>	5	(Default) Tiled.

This documentation is archived and is not being maintained.

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

## PictureType Property

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets the type of picture to be generated by the **Picture** property.

### Syntax

*object*.**PictureType** [=*type*]

The **PictureType** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>type</i>	An integer or <a href="#">constant</a> that specifies the type of picture that should be generated, as described in Settings.

### Settings

The settings for *type* are:

Constant	Value	Description
<b>flexPictureColor</b>	0	This produces a high-quality full-color image.
<b>flexPictureMonochrome</b>	1	This produces a lower-quality, monochrome, image that consumes less memory.

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## Placement Property

See Also   Example   [Applies To](#)

Returns or sets a value that specifies the placement of tabstop, bottom, left, or right.

### Syntax

*object*.**Placement** [= *integer*]

The **Placement** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>integer</i>	A <a href="#">numeric expression</a> specifying the tabs' location, as described in Settings.

### Settings

The settings for *integer* are:

Constant	Value	Description
<b>tabPlacementTop</b>	0	(Default) The tabs appear at the top of the control.
<b>tabPlacementBottom</b>	1	The tabs appears at the bottom of the control.
<b>tabPlacementLeft</b>	2	The tabs appears at the control's left.
<b>tabPlacementRight</b>	3	The tabs appears at the control's right.

### Remarks

The **Placement** property also has an effect on the behavior of the **TabStyle** property. For example, if **Placement** is set to **tabPlacementLeft**, and **TabStyle** is set to **tabTabOpposite**, when the user clicks a tab, the remaining tabs will be repositioned at the right side of the control.

This documentation is archived and is not being maintained.

## Visual Studio 6.0

*Visual Basic: MSChart Control*

# Plot Property

See Also   Example   [Applies To](#)

Returns a reference to a **Plot** object that describes the area upon which a chart is displayed.

## Syntax

*object*.**Plot**

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

© 2018 Microsoft

This documentation is archived and is not being maintained.

## Visual Studio 6.0

*Visual Basic: MSChart Control*

# PlotBase Property

See Also   Example   [Applies To](#)

Returns a reference to a **PlotBase** object that describes the appearance of the area beneath a chart.

## Syntax

*object*.**PlotBase**

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

# Point Property

See Also   Example   [Applies To](#)

Returns or sets the point where the current axis intersects with another axis.

## Syntax

*object*.**Point** [ = *point*]

The **Point** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>point</i>	Double. The point on an axis where the current axis intersects.

## Remarks

If this property is set, then the Intersection object's **Auto** property is automatically set to **False**.

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Port Property

[See Also](#) [Example](#) [Applies To](#)

Returns the name of the port through which a document is sent to a printer.

### Syntax

*object*.**Port**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

### Remarks

The operating system determines the name of the port, such as LPT1: or LPT2:.

**Note** The effect of the properties of the **Printer** object depends on the driver supplied by the printer manufacturer. Some property settings may have no effect, or several different property settings may all have the same effect. Settings outside the accepted range may or may not produce an error. For more information, see the manufacturer's documentation for the specific driver.

© 2018 Microsoft



# Visual Basic Reference

## Port Property Example

This example examines each **Printer** object in the **Printers** collection to find one connected to a specific port and makes it the default printer.

```
Dim P As Object
For Each P In Printers
    If P.Port = "LPT2:" Or P.DeviceName Like "*LaserJet*" Then
        Set Printer = P
        Exit For
    End If
Next P
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: MSComm Control

Visual Studio 6.0

## PortOpen Property

[See Also](#) [Example](#) [Applies To](#)

Sets and returns the state of the communications port (open or closed). Not available at design time.

### Syntax

*object*.**PortOpen** [ = *value* ]

The **PortOpen** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A <a href="#">boolean expression</a> specifying the state of the communications port.

### Settings

The settings for *value* are:

Setting	Description
<b>True</b>	Port is opened
<b>False</b>	Port is closed

### Remarks

Setting the **PortOpen** property to **True** opens the port. Setting it to **False** closes the port and clears the receive and transmit buffers. The **MSComm** control automatically closes the serial port when your application is terminated.

Make sure the **CommPort** property is set to a valid port number before opening the port. If the **CommPort** property is set to an invalid port number when you try to open the port, the **MSComm** control generates error 68 (Device unavailable).

In addition, your serial port device must support the current values in the **Settings** property. If the **Settings** property contains communications settings that your hardware does not support, your hardware may not work correctly.

If either the **DTRenable** or the **RTSenable** properties is set to **True** before the port is opened, the properties are set to **False** when the port is closed. Otherwise, the DTR and RTS lines remain in their previous state.

## Data Type

Boolean

© 2018 Microsoft

# Visual Basic: MSComm Control

## PortOpen Property Example

The following example opens communications port number 1 at 9600 baud with no parity checking, 8 data bits, and 1 stop bit:

```
MSComm1.Settings = "9600,n,8,1"  
MSComm1.CommPort = 1  
MSComm1.PortOpen = True
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## Position Property (Band Object)

See Also   Example   [Applies To](#)

Returns the position of a **Band** object within a **CoolBar** control.

### Syntax

*object*.**Position** [= *integer*]

The **Position** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to a <b>Band</b> object.
<i>integer</i>	An integer indicating the current position.

### Remarks

The **Position** property can be used to determine the current location of a **Band** within a **CoolBar** control. While the **Index** of a **Band** remains constant, the **Position** changes as the bands are rearranged.

The **Position** property begins with a value of 1 for the **Band** in the upper left position and increments from left to right, top to bottom when the **CoolBar** orientation is horizontal. When the orientation is vertical, **Position** increments top to bottom, left to right.

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## Position Property (ColumnHeader Object)

See Also   Example   [Applies To](#)

Returns or sets the position of the object.

### Syntax

*object*.**Position** [= *integer*]

The **Position** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>integer</i>	A <a href="#">numeric expression</a> that specifies the position of the <b>ColumnHeader</b> object. The integer can be any number from 1 to <i>n</i> , where <i>n</i> is the number of <b>ColumnHeader</b> objects.

### Remarks

Use the property to rearrange the order of columns.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

# Position Property (MSChart)

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to a **SeriesPosition** object that describes the location of one series in relation to other chart series.

## Syntax

*object*.**Position**

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Multimedia MCI Control

Visual Studio 6.0

## Position Property (Multimedia MCI Control)

[See Also](#) [Example](#) [Applies To](#)

Specifies, as defined by the **Multimedia MCI** control **TimeFormat** property, the current position of an open MCI device. This property is not available at design time and is read-only at run time.

### Syntax

`[form.]MMControl.Position`

### Data Type

Long

© 2018 Microsoft



This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Precision Property

See Also   Example   [Applies To](#)

Returns or sets the maximum total number of digits, the precision, used by the data type of the **DEField** or **DEParameter** object. This property is read-only for the **DEField** object, is read-write for the **DEParameter** object, and applies only to numeric **DEField** and **DEParameter** objects.

### Syntax

*object*.**Precision** [=value]

The **Precision** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an item in the Applies To list.
<i>value</i>	Integer. A <a href="#">numeric expression</a> that specifies the maximum total number of digits used by the <b>DEField</b> or <b>DEParameter</b> object.

### Remarks

This property corresponds to the ADO Field or Parameter Precision properties.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Prepared Property (DEDesigner Extensibility)

See Also   Example   [Applies To](#)

Specifies whether the source of the **DECommand** object is prepared before the **DECommand** object is executed.

### Syntax

*object*.**Prepared** [=Boolean]

The **Prepared** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an item in the Applies To list.
<i>Boolean</i>	A <a href="#">Boolean expression</a> that specifies whether the source of the DECommand object is prepared before the DECommand object is executed. The default is <b>False</b> .

### Remarks

The provider produces a compiled representation of the **DECommand** object during the first invocation. When the representation is maintained, preparing implies that successive invocations of the **DECommand** object are accelerated.

If the provider cannot prepare a **DECommand** object, an exception indicates that **Prepared** DECommand objects are not supported. This occurs when you attempt to set this property to **True**.

This property corresponds to the ADO Command Prepared property.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: RDO Data Control

Visual Studio 6.0

## Prepared Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that determines if the query should be prepared using the **SQLPrepare** or **SQLExecDirect** ODBC API function.

### Syntax

*object*.**Prepared** [= *value*]

The **Prepared** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A <b>Boolean</b> expression as described in Settings.

### Settings

The **Prepared** property has these settings:

Setting	Description
<b>True</b>	The statement is prepared. (Default)
<b>False</b>	The statement is not prepared.

### Remarks

By default the **Prepared** property is **True**. However, you can set this property to **False** to prohibit "preparation" of the query. In this case, the query is executed using the **SQLExecDirect** API.

When the ODBC interface submits a query to the remote server, it either submits the query directly to the server, or creates a stored procedure to perform the operation. Creating a stored procedure can slow down the initial operation, but increases performance of all subsequent references to the query. However, some queries cannot be executed in the form of stored procedures. In these cases, you must set the **Prepare** property to **False**.

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PrevInstance Property

See Also   Example   [Applies To](#)

Returns a value indicating whether a previous instance of an application is already running.

### Syntax

*object*.**PrevInstance**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

### Remarks

You can use this property in a Load event procedure to specify whether a user is already running an instance of an application. Depending on the application, you might want only one instance running in the Microsoft Windows operating environment at a time.

**Note** Since a computer running Windows NT can support multiple desktops, if you use a component designed to work with distributed COM, it can result in the following scenario:

- A client program in a user desktop requests one of the objects the component provides. Because the component is physically located on the same machine, the component is started in the user desktop.
- Subsequently, a client program on another computer uses distributed COM to request one of the objects the component provides. A second instance of the component is started, in a system desktop.

There are now two instances of the component running on the same NT computer, in different desktops.

This scenario is not a problem unless the author of the component has placed a test for **App.PrevInstance** in the startup code for the component to prevent multiple copies of the component from running on the same computer. In this case, the remote component creation will fail.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Windows Controls

Visual Studio 6.0

## Previous Property (Node Object)

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to the previous sibling of a **Node** object.

### Syntax

*object*. **Previous**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

### Remarks

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).Previous
    .Text = "New text"
    .Key = "New key"
    .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodPrevious As Node
' Get a reference to the node previous to Node x.
Set NodChild = TreeView1.Nodes(x).Previous
' Use this reference to perform operations on the previous Node.
With nodPrevious
    .Text = "New text"      ' Change the text.
    .Key = "New key"       ' Change key.
    .SelectedImage = 3     ' Change SelectedImage.
End With
```

© 2018 Microsoft

# Visual Basic: Windows Controls

## Previous Property Example

This example adds several nodes to a **TreeView** control. The **Previous** property, in conjunction with the **LastSibling** property and the **FirstSibling** property, is used to navigate through a clicked **Node** object's hierarchy level. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, and click the various nodes to see what is returned.

```
Private Sub Form_Load()  
    Dim nodX As Node  
    Set nodX = TreeView1.Nodes.Add(, , "r", "Root")  
    Set nodX = TreeView1.Nodes.Add(, , "p", "parent")  
  
    Set nodX = TreeView1.Nodes.Add("r", tvwChild, , "Child 1")  
    Set nodX = TreeView1.Nodes.Add("r", tvwChild, , "Child 2")  
    Set nodX = TreeView1.Nodes.Add("r", tvwChild, , "Child 3")  
    nodX.EnsureVisible ' Show all nodes.  
  
    Set nodX = TreeView1.Nodes.Add("p", tvwChild, , "Child 4")  
    Set nodX = TreeView1.Nodes.Add("p", tvwChild, , "Child 5")  
    Set nodX = TreeView1.Nodes.Add("p", tvwChild, , "Child 6")  
    nodX.EnsureVisible ' Show all nodes.  
End Sub  
  
Private Sub TreeView1_NodeClick(ByVal Node As Node)  
    Dim strText As String  
    Dim n As Integer  
    ' Set n to LastSibling's index.  
    n = Node.LastSibling.Index  
    ' Place LastSibling's text & linefeed in string variable.  
    strText = Node.LastSibling.Text & vbCrLf  
    While n <> Node.FirstSibling.Index  
        ' While n is not the index of the FirstSibling, go to the  
        ' previous sibling and place its text into the string variable.  
        strText = strText & TreeView1.Nodes(n).Previous.Text & vbCrLf  
        ' Set n to the previous node's index.  
        n = TreeView1.Nodes(n).Previous.Index  
    Wend  
    MsgBox strText ' Display results.  
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Printer Property

[See Also](#) [Example](#) [Applies To](#)

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

### Syntax

### Printer

### 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.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: CommonDialog Control

Visual Studio 6.0

## PrinterDefault Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets an option that determines if the user's selections in the Print dialog box are used to change the system's default printer settings.

### Syntax

*object*.**PrinterDefault** [= *value*]

The **PrinterDefault** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A <a href="#">Boolean expression</a> specifying whether the user's selections are used to change the system's default printer settings, as described in Settings.

### Settings

The settings for *value* are:

Setting	Description
<b>True</b>	Any selections the user makes in the Setup portion of the Print dialog box (printer selection, orientation, and so on) are used to change the printer settings in the system's registry.
<b>False</b>	User's selections can't be used to change the system's default printer settings.

### Remarks

When **PrinterDefault** is **True**, you can write code to print directly to the Visual Basic **Printer** object. Otherwise, you must use the graphic device interface (GDI) calls to print to the printer specified by the control's **hDC** property.

**Note** If you've previously printed to the **Printer** object, make sure you've ended that print job using `Printer.EndDoc`. This releases the hDC associated with that printer. You will get a new hDC for the default printer the next time you print to the **Printer** object. If you don't do this, it's possible for the user to select a new printer while the **Printer** object contains a [handle](#) to the old printer.



## Data Type

Boolean

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Printers Property

[See Also](#) [Example](#) [Applies To](#)

Returns a **Printers** collection, which enables you to gather information about all the available printers on the system.

### Syntax

**Printers**(*index*)

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

### Remarks

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.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PrintQuality Property

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets a value indicating the printer resolution. Not available at design time.

### Syntax

*object*.**PrintQuality** [= *value*]

The **PrintQuality** property syntax has these parts:

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Value</i>	A value or constant specifying printer resolution, as described in Settings.

### Settings

The settings for *value* are:

Constant	Value	Description
<b>vbPRPQDraft</b>	-1	Draft resolution
<b>vbPRPQLow</b>	-2	Low resolution
<b>vbPRPQMedium</b>	-3	Medium resolution
<b>vbPRPQHigh</b>	-4	High resolution

In addition to the predefined negative values, you can also set *value* to a positive dots per inch (dpi) value, such as 300.

### Remarks

These constants are listed in the Visual Basic (VB) [object library](#) in the [Object Browser](#).

The default value depends on the printer driver and the current settings of the printer. The effect of these settings varies among printers and printer drivers. On some printers, some or all of the settings may produce the same result.

**Note** The effect of the properties of the **Printer** object depends on the driver supplied by the printer manufacturer. Some property settings may have no effect, or several different property settings may all have the same effect. Settings outside the accepted range may or may not produce an error. For more information, see the manufacturer's documentation for the specific driver.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Extensibility Reference

Visual Studio 6.0

## ProcBodyLine Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

Returns the first line of a procedure.

### Syntax

*object*.**ProcBodyLine**(*procname*, *prockind*) **As Long**

The **ProcBodyLine** syntax has these parts:

Part	Description
<i>object</i>	Required. An object expression that evaluates to an object in the Applies To list.
<i>procname</i>	Required. A String containing the name of the procedure.
<i>prockind</i>	Required. Specifies the kind of procedure to locate. Because property procedures can have multiple representations in the module, you must specify the kind of procedure you want to locate. All procedures other than property procedures (that is, <b>Sub</b> and <b>Function</b> procedures) use <b>vbext_pk_Proc</b> .

You can use one of the following constants for the *prockind* argument:

Constant	Description
<b>vbext_pk_Get</b>	Specifies a procedure that returns the value of a property.
<b>vbext_pk_Let</b>	Specifies a procedure that assigns a value to a property.
<b>vbext_pk_Set</b>	Specifies a procedure that sets a reference to an object.
<b>vbext_pk_Proc</b>	Specifies all procedures other than property procedures.

### Remarks

The first line of a procedure is the line on which the **Sub**, **Function**, or **Property** statement appears.

# Visual Basic Extensibility Reference

## ProcBodyLine Property Example

The following example uses the **ProcBodyLine** property to return the line number of the first line of code in the specified procedure, SetupTabs, in a particular code pane.

```
Debug.Print Application.VBE.CodePanels(3).CodeModule.ProcBodyLine ("SetupTabs", vbext_pk_Proc)
```

This documentation is archived and is not being maintained.

# Visual Basic Extensibility Reference

Visual Studio 6.0

## ProcCountLines Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

Returns the number of lines in the specified procedure.

### Syntax

*object*.**ProcCountLines**(*procname*, *prockind*) **As Long**

The **ProcCountLines** syntax has these parts:

Part	Description
<i>object</i>	Required. An object expression that evaluates to an object in the Applies To list.
<i>procname</i>	Required. A String containing the name of the procedure.
<i>prockind</i>	Required. Specifies the kind of procedure to locate. Because property procedures can have multiple representations in the module, you must specify the kind of procedure you want to locate. All procedures other than property procedures (that is, <b>Sub</b> and <b>Function</b> procedures) use <b>vbext_pk_Proc</b> .

You can use one of the following constants for the *prockind* argument:

Constant	Description
<b>vbext_pk_Get</b>	Specifies a procedure that returns the value of a property.
<b>vbext_pk_Let</b>	Specifies a procedure that assigns a value to a property.
<b>vbext_pk_Set</b>	Specifies a procedure that sets a reference to an object.
<b>vbext_pk_Proc</b>	Specifies all procedures other than property procedures.

### Remarks

The **ProcCountLines** property returns the count of all blank or comment lines preceding the procedure declaration and, if the procedure is the last procedure in a code module, any blank lines following the procedure.

# Visual Basic Extensibility Reference

## ProcCountLines Property Example

The following example uses the **ProcCountLines** property to return the number of lines of code in the specified procedure, SetupTabs, in a particular code pane.

```
Debug.Print Application.VBE.CodePanels(3).CodeModule.ProcCountLines ("SetupTabs", vbext_pk_Proc)
```

© 2018 Microsoft



This documentation is archived and is not being maintained.

# Visual Basic Extensibility Reference

Visual Studio 6.0

## ProcOfLine Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

Returns the name of the procedure that the specified line is in.

### Syntax

*object*.**ProcOfLine**(*line*, *prockind*) **As String**

The **ProcOfLine** syntax has these parts:

Part	Description
<i>object</i>	Required. An object expression that evaluates to an object in the Applies To list.
<i>line</i>	Required. A Long specifying the line to check.
<i>prockind</i>	Required. Specifies the kind of procedure to locate. Because property procedures can have multiple representations in the module, you must specify the kind of procedure you want to locate. All procedures other than property procedures (that is, <b>Sub</b> and <b>Function</b> procedures) use <b>vbext_pk_Proc</b> .

You can use one of the following constants for the *prockind* argument:

Constant	Description
<b>vbext_pk_Get</b>	Specifies a procedure that returns the value of a property.
<b>vbext_pk_Let</b>	Specifies a procedure that assigns a value to a property.
<b>vbext_pk_Set</b>	Specifies a procedure that sets a reference to an object.
<b>vbext_pk_Proc</b>	Specifies all procedures other than property procedures.

### Remarks

A line is within a procedure if it's a blank line or comment line preceding the procedure declaration and, if the procedure is the last procedure in a code module, a blank line or lines following the procedure.



# Visual Basic Extensibility Reference

## ProcOfLine Property Example

The following example uses the **ProcOfLine** property to return the name of the procedure containing the specified line number in a particular code pane.

```
Debug.Print Application.VBE.CodePanels(3).CodeModule.ProcOfLine (1270, vbext_pk_Proc)
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Extensibility Reference

Visual Studio 6.0

## ProcStartLine Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

Returns the line at which the specified procedure begins.

### Syntax

*object*.**ProcStartLine**(*procname*, *prockind*) **As Long**

The **ProcStartLine** syntax has these parts:

Part	Description
<i>object</i>	Required. An object expression that evaluates to an object in the Applies To list.
<i>procname</i>	Required. A String containing the name of the procedure.
<i>prockind</i>	Required. Specifies the kind of procedure to locate. Because property procedures can have multiple representations in the module, you must specify the kind of procedure you want to locate. All procedures other than property procedures (that is, <b>Sub</b> and <b>Function</b> procedures) use <b>vbext_pk_Proc</b> .

You can use one of the following constants for the *prockind* argument:

Constant	Description
<b>vbext_pk_Get</b>	Specifies a procedure that returns the value of a property.
<b>vbext_pk_Let</b>	Specifies a procedure that assigns a value to a property.
<b>vbext_pk_Set</b>	Specifies a procedure that sets a reference to an object.
<b>vbext_pk_Proc</b>	Specifies all procedures other than property procedures.

### Remarks

A procedure starts at the first line below the **End Sub** statement of the preceding procedure. If the procedure is the first procedure, it starts at the end of the general Declarations section.

# Visual Basic Extensibility Reference

## ProcStartLine Property Example

The following example uses the **ProcStartLine** property to return the line at which the specified procedure begins in a particular code pane.

```
Debug.Print Application.VBE.CodePanels(3).CodeModule.ProcStartLine ("SetupTabs", vbext_pk_Proc)
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## ProductName Property

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets a string value containing the product name of the running application. Read only at [run time](#).

### Syntax

*object*.**ProductName**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

### Remarks

You can set this property at design time in the Type box in the Make tab of the Project Properties dialog box.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Extensibility Reference

Visual Studio 6.0

## ProgID Property

See Also   Example   [Applies To](#)

Returns the ProgID (programmatic ID) for the control represented by the **VBControl** object.

### Syntax

*object*.**ProgID**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

# Projection Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the type of projection used to display the chart.

## Syntax

*object*.**Projection** [ = *type*]

The **Projection** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>type</i>	Integer. A <a href="#">VtProjectionType constant</a> used to describe the type of chart projection.

## Remarks

The **Projection** property determines the appearance of 3 dimensional charts.

© 2018 Microsoft



# Projection Property Example

The example allows you to see the different effects obtainable by setting the **Projection** property. To try the example, draw an **MSChart** and a **ComboBox** control on a form. Paste the code into the Declarations section, and press F5. Click the **ComboBox** to see the different settings.

Option Explicit

```
Private Sub Combo1_Click()  
    ' Change the projection when clicked.  
    MSChart1.Plot.Projection = Combo1.ListIndex  
End Sub  
  
Private Sub Form_Load()  
    ' Set the chart to a 3D type.  
    MSChart1.chartType = VtChChartType3dBar  
    ' Configure the ComboBox by adding the valid settings.  
    With Combo1  
        .AddItem "VtProjectionTypePerspective" ' 0  
        .AddItem "VtProjectionTypeOblique"      ' 1  
        .AddItem "VtProjectionTypeOrthogonal"   ' 2  
        .AddItem "VtProjectionTypeFrontal"      ' 3  
        .AddItem "VtProjectionTypeOverhead"     ' 4  
        .ListIndex = 0  
    End With  
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: RDO Data Control

Visual Studio 6.0

## Prompt Property (Remote Data)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that specifies if the [ODBC driver manager](#) should prompt for missing connect string arguments.

### Syntax

*object*.**Prompt** [= *value*]

The **Prompt** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A constant or <b>Integer</b> as described in Settings.

### Settings

The settings for the **Prompt** property are:

Constant	Value	Description
<b>rdDriverPrompt</b>	0	The driver manager displays the ODBC Data Sources dialog box. The connection string used to establish the connection is constructed from the data source name (DSN) selected and completed by the user via the dialog boxes. Or, if no DSN is chosen and the <b>DataSourceName</b> property is empty, the default DSN is used.
<b>rdDriverNoPrompt</b>	1	The driver manager uses the connection string provided in <i>connect</i> . If sufficient information is not provided, the <b>OpenConnection</b> method returns a trappable error.
<b>rdDriverComplete</b>	2	If the connection string provided includes the DSN keyword, the driver manager uses the string as provided in <i>connect</i> , otherwise it behaves as it does when <b>rdDriverPrompt</b> is specified.
<b>rdDriverCompleteRequired</b>	3	(Default) Behaves like <b>rdDriverComplete</b> except the driver disables the controls for any information not required to complete the connection.

**Remarks**

When RDO opens a connection based on the parameters of the **RemoteData** control, the **Connect** property is expected to contain sufficient information to establish the connection. If information like the data source name, user name, or password are not provided, the ODBC driver manager exposes one or more dialog boxes to gather this information from the user. If you do not want these dialog boxes to appear, set the **Prompt** property accordingly to disable this feature.

The constants shown above are also used to set the ODBC prompt behavior for the **EstablishConnection** method.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PromptBeforeExecuting Property

See Also   Example   [Applies To](#)

Specifies whether a confirmation displays before an object is executed at design time.

### Syntax

*object*.**PromptBeforeExecuting** [=Boolean]

The **PromptBeforeExecuting** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an item in the Applies To list.
<i>Boolean</i>	A <a href="#">Boolean expression</a> that specifies whether a confirmation dialog box displays before an object is executed.

### Remarks

This property corresponds to the **Prompt before executing command** check box in the **Options** dialog box and is specific to your system. Therefore, changing it for one DataEnvironment object automatically changes it for all DataEnvironment objects on your system.

When setting the source of a **DECommand** object using the Data Environment Extensibility Object Model, first set the **PromptBeforeExecuting** property to **False** to prevent the confirmation dialog box from appearing.

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PromptDelete Property

See Also   Example   [Applies To](#)

Specifies whether a confirmation displays when an object is deleted.

### Syntax

*object*.**PromptDelete** [=*Boolean*]

The **PromptDelete** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an item in the Applies To list.
<i>Boolean</i>	A <a href="#">Boolean expression</a> that specifies whether a confirmation dialog box displays when an object within a DataEnvironment object is deleted.

### Remarks

This property corresponds to the **Confirm Delete** check box in the **Options** dialog box and is specific to your system. Therefore, changing it for one DataEnvironment object automatically changes it for all DataEnvironment objects on your system.

When removing objects using the Data Environment Extensibility Object Model, first set the **PromptDelete** property to **False** to prevent the confirmation dialog box from appearing.

This documentation is archived and is not being maintained.

# Visual Basic: MaskedEdit Control

Visual Studio 6.0

## PromptChar Property

[See Also](#) [Example](#) [Applies To](#)

Sets or returns the character used to prompt a user for input.

### Syntax

```
[form.]MaskedEdit.PromptChar [ = char$]
```

### Remarks

The underscore character "\_" is the default character value for the property. The **PromptChar** property can only be set to exactly one character.

Use the **PromptInclude** property to specify whether prompt characters are contained in the **Text** property.

Use the **AllowPrompt** property to test whether a prompt character is entered by a user.

### Data Type

String

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: MaskedEdit Control

Visual Studio 6.0

## PromptInclude Property

[See Also](#) [Example](#) [Applies To](#)

Specifies whether prompt characters are contained in the **Text** property value. Use the **PromptChar** property to change the value of the prompt character.

### Syntax

[form.]**MaskedEdit.PromptInclude** [ = { **True** | **False** } ]

### Remarks

The following table lists the **PromptInclude** property settings for the **Masked Edit** control.

Setting	Description
<b>False</b>	The value of the <b>Text</b> property does not contain any prompt character.
<b>True</b>	(Default) The value of the <b>Text</b> property contains prompt characters, if any.

If the **Masked Edit** control is bound to a data control, the **PromptInclude** property affects how the data control reads the bound **Text** property. If **PromptInclude** is **False**, the data control ignores any literals or prompt characters in the **Text** property. In this mode, the value that the data control retrieves from the **Masked Edit** control is equivalent to the value of the **ClipText** property.

If **PromptInclude** is **True**, the data control uses the value of the **Text** property as the data value to store.

### Data Type

Integer (Boolean)

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Extensibility Reference

Visual Studio 6.0

## Properties Property

See Also   Example   [Applies To](#)   Specifics

Returns the properties of an object. Read-only.

### Remarks

The **Properties** property is an accessor property (that is, a property that returns an object of the same type as the property name).

© 2018 Microsoft



This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Properties Property (WebItem Object)

[See Also](#) [Example](#) [Applies To](#)

Returns a **WebItemProperties** object that is a collection of user-defined properties for a **WebItem**.

### Syntax

*object*.**Properties**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

### Remarks

Allows you to create and remove properties on a **WebItem**.

© 2018 Microsoft

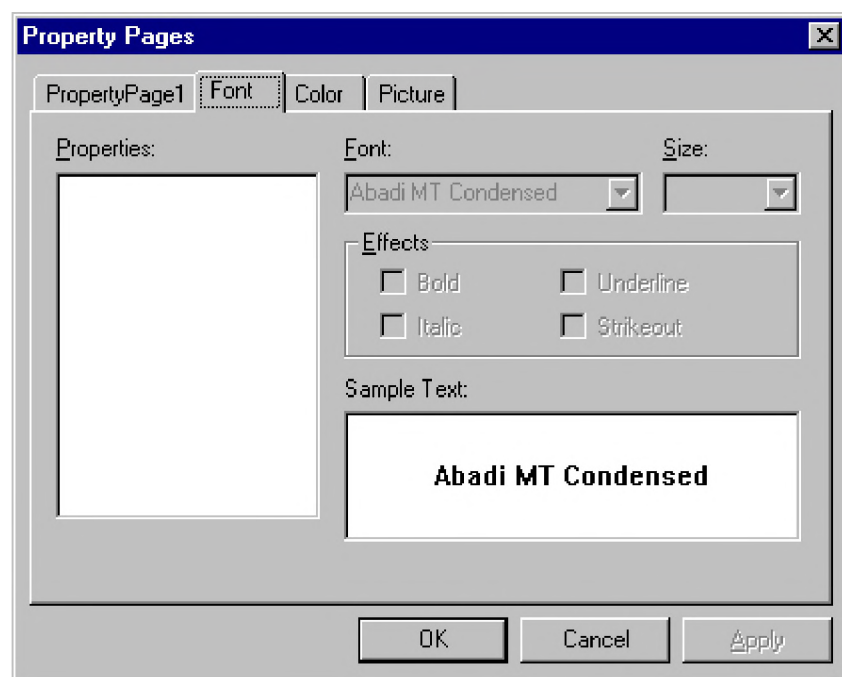
This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Property Pages Dialog Box (ActiveX Controls)

See Also



Allows you to change a control's properties at design time.

### Dialog Box Options

#### Tabs

Visual Basic creates a tabbed dialog box that acts like form by writing code to handle updating property values when a user changes values in the control.

You can add Property Pages to your project using the Add Property Pages command on the Project menu.

#### OK

Adds the Property Pages and closes the Property Pages dialog box.

#### Apply

Adds the Property Page without closing the dialog box.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PropertyChanged Method

[See Also](#)   [Example](#)   [Applies To](#)

Notifies the container that a property's value has been changed.

### Syntax

*object*.**PropertyChanged** *PropertyName*

The **PropertyChanged** method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>PropertyName</i>	A string expression that represents a name of the property that the control has changed the value of.

### Remarks

By notifying the container that a property's value has changed, the container can synchronize its Properties window with the new values of the object's properties. Also, the container would not know if an instance of the object needed to be saved (through raising a WriteProperties event) unless the container was notified that a property's value had changed.

This method needs to be called, for example, when a user changes a property value on a property page, or the object itself changes a property value. This method should also be called when a databound property is modified; otherwise the data source will not be updated.

Properties that are available only at run time do not need to call the **PropertyChanged** method, unless they can be data-bound.

As an example, the following code shows how the **PropertyChanged** method is used:

```
Public Property Let Address(ByVal cValue As String)
    m_Address = cValue
    PropertyChanged "Address"
End Property
```

This documentation is archived and is not being maintained.

# Visual Basic: DataRepeater Control

Visual Studio 6.0

## PropertyName Property (RepeaterBinding Object)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the name of a bindable property of the user control.

### Syntax

*object*.**PropertyName** [=*string*]

The **PropertyName** property syntas has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>string</i>	A bindable property of the <b>RepeatedControl</b> .

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PropertyName Property

[See Also](#) [Example](#) [Applies To](#)

The behavior of the **PropertyName** property depends upon the context in which it is being used.

- **AsyncRead** method Sets the name of the property that will be associated with the **AsyncProperty** objects **Value** property.
- AsyncReadComplete event Specifies the name of the property currently being read. This should correspond to a name assigned to the **AsyncProperty** object when invoking the **AsyncRead** method.
- **DataBinding Object** Read-only. Returns the name of the property that the **DataBinding** object refers to.

### Syntax

*object*.**PropertyName** = *string*

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>string</i>	The name of a property to be saved or retrieved.

# Visual Basic Reference

## PropertyName Property Example

The example assigns a value to the **PropertyName** property in the **AsyncRead** method. The same value will be used to assign the result of the method to a **PictureBox** control. To try the example, place a **PictureBox** control on a **UserDocument** object. Paste the code into the General section, and press F5 to run. Start Internet Explorer 3.0 (or later), and type the path and file name of the UserDocument.vbd file into the **Address** box.

```
Private Sub UserDocument_InitProperties()  
    Dim strPath As String  
    ' Set the variable to a valid path for a bitmap  
    ' on your computer.  
    strPath = "C:\Program Files\Microsoft Visual Studio\VB\" & _  
        "Samples\VCR\Bfly1.bmp"  
    AsyncRead strPath, vbAsyncTypeFile, _  
        PropertyName:= "butterfly"  
End Sub  
  
Private Sub UserDocument_AsyncReadComplete (AsyncProp _  
    As AsyncProperty)  
    ' Use the Select statement to determine which  
    ' Property is being returned.  
    Select Case AsyncProp.PropertyName  
        Case "butterfly"  
            Picture1.Picture = _  
                LoadPicture(AsyncProp.Value)  
    End Select  
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: DataRepeater Control

Visual Studio 6.0

## PropertyNames Property

[See Also](#) [Example](#) [Applies To](#)

Returns an array of property names of the repeated control.

### Syntax

*object*.**PropertyNames**(*index*)

The **PropertyNames** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>index</i>	The index of an element in the array. The array is 0-based.

### Remarks

Use the **Ubound** method to determine how many elements are in the array. Since the array is 0-based, use the following code:

```
Debug.Print Ubound(DataRepeater1.DataFields) + 1
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PropertyPage Property

See Also   Example   [Applies To](#)

Returns or sets the PropertyPage attribute of a **Member** object.

### Syntax

*object*.**PropertyPage**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

© 2018 Microsoft



This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## PropertyPages Property

[See Also](#)   [Example](#)   [Applies To](#)

Returns or sets a string that is the name of a property page that is associated with a control.

### Syntax

*object*.**PropertyPages**(*index*) [= *PropPageName*]

The **PropertyPages** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>index</i>	Index into the string array.
<i>PropPageName</i>	A string containing the name of a property page in the project.

### Remarks

**PropertyPages** property is a string array containing the names of the property pages in the project that are associated with this control. A property page may be added to the array by setting the last item in the array (which is always empty). A property page may be deleted from the array by setting that element in the array to an empty string.

The order of the names of property pages in the array determine the order in which pages appear in the property pages dialog box for the control.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic Extensibility Reference

Visual Studio 6.0

## Protection Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

Returns a value indicating the state of protection of a project. Read-only.

### Return Values

The **Protection** property return values are:

Constant	Description
<b>vbext_pp_locked</b>	The specified project is locked.
<b>vbext_pp_none</b>	The specified project isn't protected.

# Visual Basic Extensibility Reference

## Protection Property Example

The following example uses the **Protection** property to return a value indicating whether or not a project is protected. The value returned is a number that corresponds to a predefined constant representing the project's status.

```
Debug.Print Application.VBE.ActiveVBProject.Protection
```

This documentation is archived and is not being maintained.

# Visual Basic: Internet Control

Visual Studio 6.0

## Protocol Property (Internet Transfer Control)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that specifies the protocol that will be used with the **Execute** method.

### Syntax

*object*.**Protocol** = *integer*

The **Protocol** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>integer</i>	Integer. A <a href="#">numeric expression</a> that determines the protocol used, as described in Settings.

### Settings

Valid settings for **Protocol** are:

Constant	Value	Description
<b>icUnknown</b>	0	Unknown.
<b>icDefault</b>	1	Default protocol.
<b>icFTP</b>	2	FTP. File Transfer Protocol.
<b>icReserved</b>	3	Reserved for future use.
<b>icHTTP</b>	4	HTTP. HyperText Transfer Protocol.
<b>icHTTPS</b>	5	Secure HTTP.

### Remarks

When this property is specified, the **URL** property is updated to show the new value. Also, if the protocol portion of the URL is updated, the **Protocol** property is updated to reflect the new value. The **OpenURL** and **Execute** methods may both modify

the value of this property.

Changing the value of this property will have no effect until the next **Execute** or **OpenURL** method is called.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Winsock Control

Visual Studio 6.0

## Protocol Property (Winsock Control)

See Also   Example   [Applies To](#)

Returns or sets the protocol, either TCP or UDP, used by the **Winsock** control.

### Syntax

*object*.**Protocol** [=*protocol*]

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

### Settings

The settings for *protocol* are:

Constant	Value	Description
<b>sckTCPProtocol</b>	0	Default. TCP protocol.
<b>sckUDPProtocol</b>	1	UDP protocol.

### Return Value

Void

### Remarks

The control must be closed (using the **Close** method) before this property can be reset.

© 2018 Microsoft

This documentation is archived and is not being maintained.

# Visual Basic: Internet Control

Visual Studio 6.0

## Proxy Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the name of the proxy server used to communicate with the Internet. This property is only used when the **AccessType** property is set to **icNamedProxy** (3).

### Syntax

*object*.**Proxy** = *proxy*

The **Proxy** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>proxy</i>	The name of the proxy server to be used.

### Data Type

String

### Remarks

Changing the value of this property will have no effect until the next **Execute** or **OpenURL** method is called.

You can set individual proxies for each protocol. For example, if your network has a proxy devoted to the ftp protocol named "CorpFTP" and it uses port 123, you can set it as:

```
Inet1.Proxy = "ftp=CorpFTP:123"
```

You can specify multiple gateways by separating each with a space. For example, if your HTTP proxy is named "CorpHTTP", and uses port 131, set both protocols this way:

```
Inet1.Proxy = "ftp=CorpFTP:123 HTTP=CorpHTTP:131"
```

If you do not specify which protocol you are trying to set, the same proxy will be used for all protocols. For example, the proxy named below will be used even the protocol is http:

```
Inet1.Proxy = "CorpFTP:123"
```

This documentation is archived and is not being maintained.

# Visual Basic Reference

Visual Studio 6.0

## Public Property

See Also   Example   [Applies To](#)

Returns or sets a value determining if a control can be shared with other applications. The **Public** property is read/write at the controls authoring time, and not available at the controls run time.

### Settings

The settings for **Public** are:

Setting	Description
True	The control can be shared with other applications. This is the default for ActiveX Control project types.
False	The control cannot be shared with other applications. When the control is contained in an ActiveX Control project, the control cannot be seen outside of the ActiveX Control project. This means that other controls or other forms in the project can use the control, but outside applications cannot. This is the only valid value for project types other than ActiveX Control.