Visual Basic Reference

Visual Studio 6.0

Icon Property

See Also Example Applies To

Returns the icon displayed when a form is minimized at run time.

Syntax

object.**Icon**

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

Remarks

Use this property to specify an icon for any form that the user can minimize at run time.

For example, you can assign a unique icon to a form to indicate the form's function. Specify the icon by loading it using the Properties window at design time. The file you load must have the .ico filename extension and format. If you don't specify an icon, the Visual Basic default icon for forms is used.

You can use the Visual Basic Icon Library (in the Icons subdirectory) as a source for icons. When you create an executable file, you can assign an icon to the application by using the **Icon** property of any form in that application.

Note You can see a form's icon in Windows 95/98 in the upper left corner of the form, or when the form is minimized in Windows 95/98 and Windows NT. If the form is minimized, the **BorderStyle** property must be set to either 1 (Fixed Single) or 2 (Sizable) and the **MinButton** property must be set to **True** for the icon to be visible.

At run time, you can assign an object's **Icon** property to another object's **DragIcon** or **Icon** property. You can also assign an icon returned by the **LoadPicture** function. Using **LoadPicture** without an argument assigns an empty (null) icon to the form, which enables you to draw on the icon at run time.

Visual Basic Reference

Icon Property Example

This example creates a blank icon for a form and draws colored dots on the icon as long as the form is minimized. To try this example, paste the code into the Declarations section of a form, and then press F5 and minimize the form.

Note This example works only with Windows NT.

```
Private Sub Form_Resize ()
   Dim X, Y   ' Declare variables.
   If Form1.WindowState = vbMinimized Then
      Form1.Icon = LoadPicture()   ' Load a blank icon.
      Do While Form1.WindowState = vbMinimized
  ' While form is minimized,
         Form1.DrawWidth = 10   ' set size of dot.
         ' Choose random color for dot.
         Form1.ForeColor = QBColor(Int(Rnd * 15))
         ' Set random location on icon.
         X = Form1.Width * Rnd
         Y = Form1.Height * Rnd
         PSet (X, Y)   ' Draw dot on icon.
         DoEvents   ' Allow other events.
      Loop
   End If
End Sub
```

This is the same example, except that it uses the **LoadPicture** method to set the **Icon** property. This example works with all versions of Windows:

```
Private Sub Form_Resize ()
   Dim X, Y   ' Declare variables.
   If Form1.WindowState = vbMinimized Then
      Form1.Icon = LoadPicture("c:\myicon.ico")
  ' An icon named "myicon.ico" must be in the
  ' c:\ directory for this example to work
  ' correctly.
   End If
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

Icon Property (Windows Common Controls)

See Also Example Applies To

Returns or sets an icon to be displayed by the object.

Syntax

object.**Icon** [= *icon*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
icon	The **Key** or **Index** property value of a ListImage object.

Remarks

To set an icon for an object, an **ImageList** control must be populated with **ListImage** objects (each representing an image), and an appropriate property must be set to the **ImageList** control. The **Icon** property can then be set to the **Key** or **Index** property of a **ListImage** object.

Visual Basic: Windows Controls

ColumnHeaderIcons, Icon Property Example

The example sets the **ColumnHeaderIcons** property an **ImageList** control, then sets the **Icon** property to the **Key** of a **ListImage** object.

```
Option Explicit
Private Sub Form_Load()
   ' Assumes an ImageList control populated with at least one image.
   Dim c As ColumnHeader
   Dim i As Integer

   For i = 1 To 4 ' Create four ColumnHeader objects.
      ListView1.ColumnHeaders.Add , , "Col " & i
   Next I

   ListView1.View = lvwReport

   ImageList1.ListImages(1).Key = "Key1" ' Set Key property of ListImage.
   ListView1.ColumnHeaderIcons = ImageList1
   For Each c In ListView1.ColumnHeaders
      c.Icon = "Key1"
   Next
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

Icons, SmallIcons Properties

See Also Example Applies To

Returns or sets the **ImageList** controls associated with the Icon and SmallIcon views in a **ListView** control.

Syntax

object.**Icons** [= *imagelist*]

object.**SmallIcons** [= *imagelist*]

The **Icons**, **SmallIcons** properties syntax has the following parts:

Part	Description
object	An object expression that evaluates to the **ListView** control.
imagelist	An object expression that evaluates to an **ImageList** control.

Remarks

To associate an **ImageList** control with a **ListView** control at run time, set these properties to the desired **ImageList** control.

Each **ListItem** object in the **ListView** control also has **Icon** and **SmallIcon** properties, which index the **ListImage** objects and determine which image is displayed.

Once you associate an **ImageList** with the **ListView** control, you can use the value of either the **Index** or **Key** property to refer to a **ListImage** object in a procedure.

Visual Basic: Windows Controls

Icon, SmallIcon, Icons, SmallIcons, View Properties Example

This example populates a **ListView** control with the contents of the Publishers table in the Biblio.mdb database. Four **OptionButton** controls are labeled with **View** property choices. You must place two **ImageList** controls on the form, one to contain images for the **Icon** property, and a second to contain images for the **SmallIcon** property of each **ListItem** object. To try the example, place a **ListView**, a control array of four **OptionButton** controls, and two **ImageList** controls on a form and paste the code into the form's Declarations section.

Note The example will not run unless you add a reference to the Microsoft DAO 3.51 Object Library by using the References command on the Tools menu. Run the example and click on the **ComboBox** control to switch views.

```
Private Sub Option1_Click(Index as Integer)
   ' Set the ListView control's View property to the
   ' Index of Option1
   ListView1.View = Index
End Sub

Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders. The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Company", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Address", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Phone", ListView1.Width / 3)

   ListView1.BorderStyle = ccFixedSingle ' Set BorderStyle property.
   ListView1.View = lvwReport ' Set View property to Report.

   ' Add one image to ImageList1--the Icons ImageList.
   Dim imgX As ListImage
   Set imgX = ImageList1.ListImages. _
   Add(, , LoadPicture("icons\mail\mail01a.ico"))
   ' Add an image to ImageList2--the SmallIcons ImageList.
   Set imgX = ImageList2.ListImages. _
   Add(, , LoadPicture("bitmaps\assorted\w.bmp"))

   ' To use ImageList controls with the ListView control, you must
   ' associate a particular ImageList control with the Icons and
   ' SmallIcons properties.
   ListView1.Icons = ImageList1
   ListView1.SmallIcons = ImageList2
   ' Label OptionButton controls with View options.
      Option1(0).Caption = "Icon"
      Option1(1).Caption = "SmallIcon"
      Option1(2).Caption = "List"
      Option1(3).Caption = "Report"
```

```
      ListView1.View = lvwIcon ' Set to Icon view

   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
   ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the Publishers table.
   Set myRs = myDb.OpenRecordset("Publishers", dbOpenDynaset)

   ' Create a variable to add ListItem objects.
   Dim itmX As ListItem

   ' While the record is not the last record, add a ListItem object.
   ' Use the Name field for the ListItem object's text.
   ' Use the Address field for the ListItem object's SubItem(1)
   ' Use the Phone field for the ListItem object's SubItem(2)

   While Not myRs.EOF

      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))
      itmX.Icon = 1   ' Set an icon from ImageList1.
      itmX.SmallIcon = 1   ' Set an icon from ImageList2.

      ' If the Address field is not Null, set SubItem 1 to the field.
      If Not IsNull(myRs!Address) Then
         itmX.SubItems(1) = CStr(myRs!Address) ' Address field.
      End If

      ' If the Phone field is not Null, set SubItem 2 to the field.
      If Not IsNull(myRs!Telephone) Then
         itmX.SubItems(2) = myRs!Telephone  ' Phone field.
      End If

      myRs.MoveNext  ' Move to next record.
   Wend
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

Icon, SmallIcon Properties (ListItem Object)

See Also Example Applies To

Returns or sets the index or key value of an icon or small icon associated with a **ListItem** object in an **ImageList** control.

Syntax

object.**Icon** [= *index*]

object.**SmallIcon** [= *index*]

The **Icon**, **SmallIcon** properties syntax has the following parts:

Part	Description
object	An object expression that evaluates to a **ListItem** object.
index	An integer or unique string that identifies an icon or small icon in an associated **ImageList** control. The integer is the value of the **ListItem** object's **Index** property; the string is the value of the **Key** property.

Remarks

Before you can use an icon in a **ListItem** object, you must associate an **ImageList** control with the **ListView** control containing the object. See the **Icons**, **SmallIcons** Properties (**ListView** Control) for more information. The example below shows the proper syntax:

```
ListView1.ListItems(1).SmallIcons=1
```

The images will appear when the **ListView** control is in SmallIcons view.

Visual Basic: Windows Controls

Icon, SmallIcon, Icons, SmallIcons, View Properties Example

This example populates a **ListView** control with the contents of the Publishers table in the Biblio.mdb database. Four **OptionButton** controls are labeled with **View** property choices. You must place two **ImageList** controls on the form, one to contain images for the **Icon** property, and a second to contain images for the **SmallIcon** property of each **ListItem** object. To try the example, place a **ListView**, a control array of four **OptionButton** controls, and two **ImageList** controls on a form and paste the code into the form's Declarations section.

Note The example will not run unless you add a reference to the Microsoft DAO 3.51 Object Library by using the References command on the Tools menu. Run the example and click on the **ComboBox** control to switch views.

```
Private Sub Option1_Click(Index as Integer)
   ' Set the ListView control's View property to the
   ' Index of Option1
   ListView1.View = Index
End Sub

Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders. The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Company", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Address", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Phone", ListView1.Width / 3)

   ListView1.BorderStyle = ccFixedSingle ' Set BorderStyle property.
   ListView1.View = lvwReport ' Set View property to Report.

   ' Add one image to ImageList1--the Icons ImageList.
   Dim imgX As ListImage
   Set imgX = ImageList1.ListImages. _
   Add(, , LoadPicture("icons\mail\mail01a.ico"))
   ' Add an image to ImageList2--the SmallIcons ImageList.
   Set imgX = ImageList2.ListImages. _
   Add(, , LoadPicture("bitmaps\assorted\w.bmp"))

   ' To use ImageList controls with the ListView control, you must
   ' associate a particular ImageList control with the Icons and
   ' SmallIcons properties.
   ListView1.Icons = ImageList1
   ListView1.SmallIcons = ImageList2
   ' Label OptionButton controls with View options.
      Option1(0).Caption = "Icon"
      Option1(1).Caption = "SmallIcon"
      Option1(2).Caption = "List"
      Option1(3).Caption = "Report"
```

```
      ListView1.View = lvwIcon ' Set to Icon view

   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
   ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the Publishers table.
   Set myRs = myDb.OpenRecordset("Publishers", dbOpenDynaset)

   ' Create a variable to add ListItem objects.
   Dim itmX As ListItem

   ' While the record is not the last record, add a ListItem object.
   ' Use the Name field for the ListItem object's text.
   ' Use the Address field for the ListItem object's SubItem(1)
   ' Use the Phone field for the ListItem object's SubItem(2)

   While Not myRs.EOF

      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))
      itmX.Icon = 1   ' Set an icon from ImageList1.
      itmX.SmallIcon = 1   ' Set an icon from ImageList2.

      ' If the Address field is not Null, set SubItem 1 to the field.
      If Not IsNull(myRs!Address) Then
         itmX.SubItems(1) = CStr(myRs!Address) ' Address field.
      End If

      ' If the Phone field is not Null, set SubItem 2 to the field.
      If Not IsNull(myRs!Telephone) Then
         itmX.SubItems(2) = myRs!Telephone  ' Phone field.
      End If

      myRs.MoveNext  ' Move to next record.
   Wend
End Sub
```

Visual Basic Extensibility Reference

Visual Studio 6.0

IconState Property

See Also Example Applies To

Returns or sets the source code control icon (or "glyph") for the project in the project window, indicating its status.

Syntax

object.**IconState** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A long integer or constant that determines the file status, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
vbextSCCStatusNotControlled	0	File is not under source code control.
vbextSCCStatusControlled	1	File is under source code control.
vbextSCCStatusCheckedOut	2	File is checked out to current user.
vbextSCCStatusOutOther	4	File is checked out to another user.
vbextSCCStatusOutOfDate	32	The file is not the most recent.
vbextSCCStatusShared	512	File is shared between projects.

Remarks

The **IconState** property can be logically **OR**'ed together to form combined states.

Visual Basic: Page Designer

Visual Studio 6.0

ID Property

See Also Example Applies To

Sets the name of an element on an HTML page, or of the **DHTMLPageDesigner** object. Not available at run time.

Remarks

This property allows you to assign an identifier to each **DHTMLPageDesigner** object or to the selected element. You must assign an identifier to an HTML element in order to put code behind it.

Until an element has an identifier, it will not appear in the Object dropdown in the Code window. The treeview in the designer window displays in bold the elements that have an identifier.

Visual Basic Reference

Visual Studio 6.0

Image Property

See Also Example Applies To

Returns a handle to a persistent graphic; the handle is provided by the Microsoft Windows operating environment.

Syntax

object.**Image**

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

Remarks

An object's **AutoRedraw** property determines whether the repainting of an object occurs with a persistent graphics or through Paint events. The Windows operating environment identifies an object's persistent graphic by assigning a handle to it; you can use the **Image** property to get this handle.

An **Image** value exists regardless of the setting for the **AutoRedraw** property. If **AutoRedraw** is **True** and nothing has been drawn, the image displays only the color set by the **BackColor** property and the picture.

You can assign the value of **Image** to the **Picture** property. The **Image** property also provides a value to pass to Windows API calls.

The **Image**, **DragIcon**, and **Picture** properties are normally used when assigning values to other properties, when saving with the **SavePicture** statement, or when placing something on the Clipboard. You can't assign these to a temporary variable, other than the Picture data type.

The **AutoRedraw** property can cause **Image**, which is a handle to a bitmap, to change. When **AutoRedraw** is **True**, an object's **hDC** property becomes a handle to a device context that contains the bitmap returned by **Image**.

Visual Basic Reference

Image Property Example

This example draws a circle in the first **PictureBox** control each time you click it. When you click the second **PictureBox**, the graphic from the first **PictureBox** is copied into it. To try this example, paste the code into the Declarations section of a form that has two large, equal-sized **PictureBox** controls. Press F5 to run the program, and then click the **PictureBox** controls.

```
Private Sub Form_Load ()
   ' Set AutoRedraw to True.
   Picture1.AutoReDraw = True
End Sub

Private Sub Picture1_Click ()
 ' Declare variables.
   Dim PW, PH
 ' Set FillStyle to Solid.
   Picture1.FillStyle = vbFSSolid
 ' Choose random color.
   Picture1.FillColor = QBColor(Int(Rnd * 15))
   PW = Picture1.ScaleWidth   ' Set ScaleWidth.
   PH = Picture1.ScaleHeight   ' Set ScaleHeight.
   ' Draw a circle in random location.
   Picture1.Circle (Int(Rnd * PW), Int(Rnd * PH)), 250
End Sub

Private Sub Picture2_Click ()
 ' Copy Image to Picture2.
   Picture2.Picture = Picture1.Image
End Sub
```

Visual Basic Reference

Visual Studio 6.0

Image Property (ActiveX Controls)

See Also Example Applies To

Returns or sets a value that specifies which **ListImage** object in an **ImageList** control to use with another object.

Syntax

object.**Image** [= *index*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer or unique string specifying the **ListImage** object to use with *object*. The integer is the value of the **Index** property; the string is the value of the **Key** property.

Remarks

Before setting the **Image** property, you must associate an **ImageList** control with a **Toolbar**, **TreeView**, or **TabStrip** control by setting each control's **ImageList** property to an **ImageList** control.

At design time, put an **ImageList** control on the form and load it with images, each of which is a **ListImage** object assigned an index number in a **ListImages** collection. On the General tab in the control's Property Pages dialog box, select the **ImageList** you want from the **ImageList** list box, such as `ImageList1`. For **Tab** and **Button** objects, you can also specify the image you want to associate with these objects by typing the index number of the specific **ListImage** object in the Image field on the Tabs or Buttons tab.

At run time, use code like the following to associate an **ImageList** to a control and then a **ListImage** to a specific object:

```
Set TabStrip1.ImageList=ImageList1
TabStrip1.Tabs(1).Image=2
```

Use the **Key** property to specify an **ImageList** control's **ListImage** object when you wish your code to be self-documenting, as follows:

```
' Assuming there is a ListImage object with the Key property value =
' "close," use that image for a Toolbar button.
Toolbar1.Buttons(1).Image = "close"

' This is easier to read than just specifying an Index value, as below:
```

```
Toolbar1.Buttons(1).Image = 4 ' Requires that the ListImage object
' with Index property = 4 is the "close" image.
```

The value of the **Index** property of an object can change when objects in the collection are reordered, such as when you set the **Sorted** property to **True**. If you expect the **Index** property to change dynamically, it may be more useful to refer to objects in a collection by using the **Key** property.

If there are no images for a **Tabs** collection, the value of *index* is -1.

Visual Basic: Windows Controls

Visual Studio 6.0

Image Property (Band Object)

See Also Example Applies To

Returns or sets a value that specifies which **ListImage** object in an **ImageList** control will appear on a **Band** object of a **CoolBar** control.

Syntax

object.**Image** [= *index*]

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

Part	Description
object	An object expression that evaluates to a **Band** object.
index	An integer or unique string specifying the **ListImage** object to use with *object*. The integer is the value of the **Index** property, the string is the value of the **Key** property.

Remarks

Before setting the **Image** property, you must associate an **ImageList** control with the **CoolBar** control by setting the CoolBars **ImageList** property to an **ImageList** control.

At run time, the image contained in the **ListImage** object specified in the **Image** property will be displayed on the **Band** object between the move handle and the caption. The image is not displayed at design time.

Visual Basic: Windows Controls

Visual Studio 6.0

ImageHeight, ImageWidth Properties

See Also Example Applies To

- The **ImageHeight** property returns or sets the height of **ListImage** objects in an **ImageList** control.
- The **ImageWidth** property returns or sets the width of **ListImage** objects in an **ImageList** control.

Syntax

object.**ImageHeight**

object.**ImageWidth**

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

Remarks

Both height and width are measured in pixels.

Before any **ListImage** objects have been added to the control, you can set both **ImageHeight** and **ImageWidth** properties. However, once a **ListImage** object has been added, the properties cannot be changed.

Important When the ImageList control is bound to another Windows Common Control, all images in the **ListImages** collection no matter what their size will be displayed in the second (bound control) at the size specified by the **ImageHeight** and **ImageWidth** properties.

Note You can the use the **ImageList** control with any control by setting the **Picture** property of the second control to the **Picture** object of any image contained by the **ImageList** control. However, the size of the displayed image will not be affected by the **ImageHeight** and **ImageWidth** properties. In other words, the second control will display the image at its original size.

Visual Basic: Windows Controls

ImageHeight, ImageWidth Properties Example

This example loads an icon into an **ImageList** control, and uses the image in a **ListView** control. When the user clicks the form, the code uses the **ImageHeight** property to adjust the height of the **ListView** control to accommodate the **ListImage** object. To try the example, place **ImageList** and **ListView** controls on a form and paste the code into the form's Declarations section. Run the example and click the form.

```
Private Sub Form_Load()
   ' Create variables for ImageList and ListView objects.
   Dim imgX As ListImage
   Dim itmX As ListItem

   Form1.ScaleMode = vbPixels ' Make sure ScaleMode is set to pixels.

   ListView1.BorderStyle = FixedSingle ' Show border.
   ' Shorten ListView control so later contrast is more obvious.
   ListView1.Height = 50

   ' Put a large bitmap into the ImageList.
   Set imgX = ImageList1.ListImages. _
   Add(,, LoadPicture("bitmaps\gauge\vert.bmp"))

   ListView1.Icons = ImageList1 ' Set Icons property.

   ' Add an item to the ListView control.
   Set itmX = ListView1.ListItems.Add()
   itmX.Icon = 1   ' Set Icon property to ListImage 1 of ImageList.
   itmX.Text = "Thermometer"   ' Set text of ListView ListItem object.
End Sub

Private Sub Form_Click()
   Dim strHW As String

   strHW = "Height: " & ImageList1.ImageHeight & _
   "   Width: " & ImageList1.ImageWidth
   caption = strHW   ' Show dimensions.
   ' Enlarge ListView to accommodate the tallest image.
   ListView1.Height = ImageList1.ImageHeight + 50
End Sub
```

Visual Basic Reference

Visual Studio 6.0

ImageList Property (ActiveX Controls)

See Also Example Applies To

Returns or sets the **ImageList** control, if any, that is associated with another control.

Syntax

object.**ImageList** [= *imagelist*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
imagelist	An object reference that specifies which **ImageList** control to use.

Remarks

For a control to use the **ImageList** property, you must place an **ImageList** control on the form. Then, at design time, you can set the **ImageList** property in the associated control's Property Pages dialog box. To associate an **ImageList** with a control at run time, set the control's **ImageList** property to the **ImageList** control you want to use, as in this example:

```
Set TabStrip1.ImageList = ImageList1
```

Visual Basic Reference

Visual Studio 6.0

IMEMode Property

See Also Example Applies To

Returns or sets a value that determines the IME (Input Method Editor) status of the object when that object is selected.

Note This property applies only to East Asian versions of Visual Basic.

object.**IMEMode** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An Integer that specifies the IME mode to be used by the object.

Settings

The settings for *value* are:

Setting	Description
0	None (Default). This value indicates "No control to IME". When the **IMEMode** property is set to 0, you can use the **IMEStatus** function to determine the current IME status.
1	IME on. This value indicates that the IME is on and characters specific to Chinese or Japanese can be entered. This setting is valid for Japanese, Simplified Chinese, and Traditional Chinese IME only.
2	IME off. This mode indicates that the IME is off, meaning that the object behaves the same as English entry mode. This setting is valid for Japanese, Simplified Chinese, and Traditional Chinese IME only.
3	IME disabled. This mode is similar to IMEMode = 2, except the value 3 disables IME. With this setting, the users cannot turn the IME on from the keyboard, and the IME floating window is hidden. This setting is valid for Japanese IME only.
4	Hiragana double-byte characters (DBC). This setting is valid for Japanese IME only.
5	Katakana DBC. This setting is valid for Japanese IME only.

6	Katakana single-byte characters (SBC). This setting is valid for Japanese IME only.
7	Alphanumeric DBC. This setting is valid for Japanese IME only.
8	Alphanumeric SBC. This setting is valid for Japanese IME only.
9	Hangeul DBC. This setting is valid for Korean IME only.
10	Hangeul SBC. This setting is valid for Korean IME only.

Remarks

At design time, you can set the **IMEMode** property of an object using the Properties window. At run time, you can return or set the **IMEMode** property through code. If you change the setting of the **IMEMode** property while the object has the focus, the IME status of the object changes accordingly. If you set the **IMEMode** property to 0 (None), it always returns 0 regardless of the object's current IME status. If you set this property to any valid setting other than 0, the **IMEMode** property returns the current IME status.

With Japanese IME, you can use the settings 0 to 8 only. The settings 9 and 10 are invalid on Japanese systems.

With Korean IME, you can use the settings 0 and 7 to 10 only. Settings 1 to 6 are invalid on Korean systems.

With Simplified Chinese and Traditional Chinese IME, you can use settings 0 to 2 only. Settings 3 to 10 are invalid on Chinese systems.

Visual Basic: MSComm Control

Visual Studio 6.0

InBufferCount Property

See Also Example Applies To

Returns the number of characters waiting in the receive buffer. This property is not available at design time.

Syntax

object.**InBufferCount**[= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the number of characters waiting in the receive buffer.

Remarks

InBufferCount refers to the number of characters that have been received by the modem and are waiting in the receive buffer for you to take them out. You can clear the receive buffer by setting the **InBufferCount** property to 0.

Note Do not confuse this property with the **InBufferSize** property. The **InBufferSize** property reflects the total size of the receive buffer.

Data Type

Integer

Visual Basic: MSComm Control

Visual Studio 6.0

InBufferSize Property

See Also Example Applies To

Sets and returns the size of the receive buffer in bytes.

Syntax

object.**InBufferSize**[= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the size of the receive buffer in bytes.

Remarks

InBufferSize refers to the total size of the receive buffer. The default size is 1024 bytes. Do not confuse this property with the **InBufferCount** property which reflects the number of characters currently waiting in the receive buffer.

Note Note that the larger you make the receive buffer, the less memory you have available to your application. However, if your buffer is too small, it runs the risk of overflowing unless handshaking is used. As a general rule, start with a buffer size of 1024 bytes. If an overflow error occurs, increase the buffer size to handle your application's transmission rate.

Data Type

Integer

Visual Basic: Windows Controls

Visual Studio 6.0

Increment Property

See Also Example Applies To

Sets or returns a value that determines the amount by which the **Value** property changes when the **UpDown** control's buttons are clicked.

Syntax

object.**Increment** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A long integer that determines the amount by which the **Value** property changes. The *value* cannot be a negative number. The default for *value* is 1.

Remarks

The **Increment** property determines the amount the **Value** property changes when you click the arrow buttons on the **UpDown** control. Clicking the up or right arrow, causes **Value** to approach the **Max** property by the amount specified by the **Increment** property. Clicking the down or left arrow, causes **Value** to approach the **Min** property by the amount specified by the **Increment** property.

Visual Basic: Windows Controls

Visual Studio 6.0

Indentation Property

See Also Example Applies To

Returns or sets the width of the indentation of objects in a control.

Syntax

object.**Indentation**[= *number*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
number	An numeric expression specifying the width that each object is indented.

Remarks

For the **ImageCombo** control and **ComboItem** object, each indent is equal to 10 pixels.

If you change the **Indentation** property at run time, the **TreeView** is redrawn to reflect the new width. The property value cannot be negative.

Visual Basic: Windows Controls

Indentation Property Example

This example adds several **Node** objects to a **TreeView** control, while the **Indentation** property is shown in the form's caption. An **OptionButton** control array provides alternate values for the **Indentation** width. To try the example, place a **TreeView** control and a control array of three **OptionButton** controls on a form, and paste the code into the form's Declarations section. Run the example, and click an **OptionButton** to change the **Indentation** property.

```
Private Sub Form_Load()
   ' Label OptionButton controls with Indentation choices.
   Option1(0).Caption = "250"
   Option1(1).Caption = "500"
   Option1(2).Caption = "1000"

   ' Select the first option, and set the indentation to 250 initially
   Option1(0).Value = True
   Treeview1.Indentation = 250

   Dim nodX As Node
   Dim i As Integer

   Set nodX = TreeView1.Nodes.Add(,,,CStr(1)) ' Add first node.

   For i = 1 To 6    ' Add 6 nodes.
      Set nodX = TreeView1.Nodes.Add(i,tvwChild,,CStr(i + 1))
   Next i

   nodX.EnsureVisible    ' Makes sure all nodes are visible.
   Form1.Caption = "Indentation = " & TreeView1.Indentation
End Sub

Private Sub Option1_Click(Index as Integer)    ' Change Indentation with OptionButton value.
   TreeView1.Indentation = Val(Option1(Index).Caption)
   Form1.Caption = "Indentation = " & TreeView1.Indentation
End Sub
```

Visual Basic Reference

Visual Studio 6.0

Index Property (ActiveX Controls)

See Also Example Applies To

Returns or sets the number that uniquely identifies an object in a collection.

Syntax

object.**Index**

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

Remarks

The **Index** property is set by default to the order of the creation of objects in a collection. The index for the first object in a collection will always be one (1).

The value of the **Index** property of an object can change when objects in the collection are reordered, such as when you set the **Sorted** property to **True**. If you expect the **Index** property to change dynamically, it may be more useful to refer to objects in a collection by using the **Key** property.

Visual Studio 6.0

Visual Basic: MSChart Control

Index Property (Brush)

See Also Example Applies To

Returns or sets the pattern or hatch used in the brush if its **Style** property is set to **VtBrushStylePattern** or **VtBrushStyleHatch**.

Syntax

object.**Index** [= *num*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
num	A VtBrushPattern constant or VtBrushHatch constant describing the brush pattern.

Visual Basic Reference

Visual Studio 6.0

Index Property (Control Array)

See Also Example Applies To

Returns or sets the number that uniquely identifies a control in a control array. Available only if the control is part of a control array.

Syntax

object[**(***number***)**].**Index**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
number	A numeric expression that evaluates to an integer that identifies an individual control within a control array.

Settings

The settings for *number* are:

Setting	Description
No value	(Default) Not part of a control array.
0 to 32,767	Part of an array. Specifies an integer greater than or equal to 0 that identifies a control within a control array. All controls in a control array have the same **Name** property. Visual Basic automatically assigns the next integer available within the control array.

Remarks

Because control array elements share the same **Name** property setting, you must use the **Index** property in code to specify a particular control in the array. **Index** must appear as an integer (or a numeric expression evaluating to an integer) in parentheses next to the control array name for example, `MyButtons(3)`. You can also use the **Tag** property setting to distinguish one control from another within a control array.

When a control in the array recognizes that an event has occurred, Visual Basic calls the control array's event procedure and passes the applicable **Index** setting as an additional argument. This property is also used when you create controls dynamically at run time with the **Load** statement or remove them with the **Unload** statement.

Although Visual Basic assigns, by default, the next integer available as the value of **Index** for a new control in a control array, you can override this assigned value and skip integers. You can also set **Index** to an integer other than 0 for the first control in the array. If you reference an **Index** value in code that doesn't identify one of the controls in a control array, a Visual Basic run-time error occurs.

Note To remove a control from a control array, change the control's **Name** property setting, and delete the control's **Index** property setting.

Visual Basic Reference

Index Property Example

This example starts with two **OptionButton** controls and adds a new **OptionButton** to the form each time you click a **CommandButton** control. When you click an **OptionButton**, the **FillStyle** property is set and a new circle is drawn. To try this example, paste the code into the Declarations section of a form that has two **OptionButton** controls, a **CommandButton**, and a large **PictureBox** control. Set the **Name** property of both **OptionButton** controls to **optButton** to create a control array.

```
Private Sub OptButton_Click (Index As Integer)
   Dim H, W   ' Declare variables.
   Picture1.Cls   ' Clear picture.
   Picture1.FillStyle = Index   ' Set FillStyle.
   W = Picture1.ScaleWidth / 2   ' Get size of circle.
   H = Picture1.ScaleHeight / 2
   Picture1.Circle (W, H), W / 2   ' Draw circle.
End Sub

Private Sub Command1_Click ()
   Static MaxIdx   ' Largest index in array.
   If MaxIdx = 0 Then MaxIdx = 1   ' Preset MaxIdx.
   MaxIdx = MaxIdx + 1   ' Increment index.
   If MaxIdx > 7 Then Exit Sub   ' Put eight buttons on form.
   Load OptButton(MaxIdx)   ' Create new item in array.
   ' Set location of new option button under previous button.
   OptButton(MaxIdx).Top = OptButton(MaxIdx - 1).Top + 360
   OptButton(MaxIdx).Visible = True   ' Make new button visible.
End Sub
```

Visual Studio 6.0

Visual Basic: MSChart Control

Index Property (Intersection)

See Also Example Applies To

Returns which axis intersects another axis when there is more than one axis with the same index.

Syntax

object.**Index**

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

Return Value

The return value is an integer that specifies the index of the intersecting axis. Currently, 1 is the only valid value for this argument.

Visual Basic: DataGrid Control

Visual Studio 6.0

Index Property (Split Object)

See Also Example Applies To

Returns the index of the selected split.

Syntax

object.**Index**

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

Remarks

This property is useful if you need to identify a given **Split** object.

Visual Basic Extensibility Reference

Visual Studio 6.0

IndexedValue Property

See Also Example Applies To

Returns or sets a value in an indexed list or an array.

Syntax

object.**IndexedValue** [*index1*, *index2*,][*index3*,] [*index4*]][= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
indexn	A numeric expression specifying index position. **IndexedValue** accepts up to four indices. The number of indices accepted by **IndexedValue** is the value for the same property returned by the **NumIndices** property.
value	An expression that evaluates to a type acceptable to the control.

Remarks

If the property is a list (as indicated by **NumIndices**), then **IndexedValue** returns the element of the list specified by the index parameters.

IndexedValue is used only if the value of the **NumIndices** property is greater than zero. Values in indexed lists, as in the **List** property of a **ListBox** control, are set or returned with a single index.

Visual Basic Extensibility Reference

Visual Studio 6.0

IndexedValue Property (VBA Add-In Object Model)

See Also Example Applies To Specifics

Returns or sets a value for a member of a property that is an indexed list or an array.

Remarks

The value returned or set by the **IndexedValue** property is an expression that evaluates to a type that is accepted by the object. For a property that is an indexed list or array, you must use the **IndexedValue** property instead of the **Value** property. An indexed list is a numeric expression specifying index position.

IndexedValue accepts up to 4 indices. The number of indices accepted by **IndexedValue** is the value returned by the **NumIndices** property.

The **IndexedValue** property is used only if the value of the **NumIndices** property is greater than zero. Values in indexed lists are set or returned with a single index.

Visual Basic: CommonDialog Control

Visual Studio 6.0

InitDir Property

See Also Example Applies To

Returns or sets the initial file directory.

Syntax

object.**InitDir** [= *string*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
string	A string expression specifying the initial file directory.

Remarks

This property is used to specify the initial directory for an Open or Save As dialog. If this property isn't specified, the current directory is used.

Data Type

String

Visual Basic: MSComm Control

Visual Studio 6.0

Input Property

See Also Example Applies To

Returns and removes a stream of data from the receive buffer. This property is not available at design time and is read-only at run time.

Syntax

object.**Input**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Remarks

The **InputLen** property determines the number of characters that are read by the **Input** property. Setting **InputLen** to 0 causes the **Input** property to read the entire contents of the receive buffer.

The **InputMode** property determines the type of data that is retrieved with the **Input** property. If **InputMode** is set to **comInputModeText** then the **Input** property returns text data in a **Variant**. If **InputMode** is **comInputModeBinary** then the **Input** property returns binary data in an array of bytes in a **Variant**.

Data Type

Variant

Visual Basic: MSComm Control

Input Property Example

This example shows how to retrieve data from the receive buffer:

```
Private Sub Command1_Click()
Dim InString as String
' Retrieve all available data.
MSComm1.InputLen = 0

' Check for data.
If MSComm1.InBufferCount Then
   ' Read data.
   InString = MSComm1.Input
End If
End Sub
```

Visual Basic: MSComm Control

Visual Studio 6.0

InputLen Property

See Also Example Applies To

Sets and returns the number of characters the **Input** property reads from the receive buffer.

Syntax

object.**InputLen** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the number of characters the **Input** property reads from the receive buffer.

Remarks

The default value for the **InputLen** property is 0. Setting **InputLen** to 0 causes the **MSComm** control to read the entire contents of the receive buffer when **Input** is used.

If **InputLen** characters are not available in the receive buffer, the **Input** property returns a zero-length string (""). The user can optionally check the **InBufferCount** property to determine if the required number of characters are present before using **Input**.

This property is useful when reading data from a machine whose output is formatted in fixed-length blocks of data.

Data Type

Integer

Visual Basic: MSComm Control

InputLen Property Example

This example shows how to read 10 characters of data:

```
Private Command1_Click()
Dim CommData as String
' Specify a 10 character block of data.
MSComm1.InputLen = 10
' Read data.
CommData = MSComm1.Input
End Sub
```

Visual Basic: MSComm Control

Visual Studio 6.0

InputMode Property

See Also Example Applies To

Sets or returns the type of data retrieved by the **Input** property.

Syntax

object.**InputMode** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A value or constant that specifies the input mode, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
comInputModeText	0	(Default) Data is retrieved through the **Input** property as text.
comInputModeBinary	1	Data is retrieved through the **Input** property as binary data.

Remarks

The **InputMode** property determines how data will be retrieved through the **Input** property. The data will either be retrieved as string or as binary data in a byte array.

Use **comInputModeText** for data that uses the ANSI character set. Use **comInputModeBinary** for all other data such as data that has embedded control characters, Nulls, etc.

Visual Basic: MSComm Control

InputMode Property Example

This example reads 10 bytes of binary data from the communications port and assigns it to a byte array.

```
Private Sub Command1_Click()
Dim Buffer as Variant
Dim Arr() as Byte

' Set and open port
MSComm1.CommPort = 1
MSComm1.PortOpen = True

' Set InputMode to read binary data
MSComm1.InputMode = comInputModeBinary

' Wait until 10 bytes are in the input buffer
Do Until MSComm1.InBufferCount < 10
  DoEvents
Loop

' Assign to byte array for processing
Arr = MSComm1.Input

End Sub
```

Visual Basic Extensibility Reference

Visual Studio 6.0

InSelection Property

See Also Example Applies To

Returns or assigns a control's selection state.

Syntax

object.**InSelection**

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

Remarks

If the **InSelection** property of a control contained within another control is set to **True**, then any controls not within that control (or any controls within other controls) will be unselected.

Visual Basic Reference

Visual Studio 6.0

Instancing Property

See Also Example Applies To

Sets a value that specifies whether you can create instances of a public class outside a project, and if so, how it will behave. Not available at run time.

Settings

The **Instancing** property has these settings:

Setting	Description
1	(Default) Private. Other applications arent allowed access to type library information about the class, and cannot create instances of it. Private objects are only for use within your component. The Instancing property default varies depending on the project type. Private is the default only for class modules in Standard Exe projects. When you insert a new class module into an ActiveX Exe project or an ActiveX DLL project, the default value of the Instancing property is MultiUse. When you insert a new class module into an ActiveX Control project, the default value of the Instancing property is PublicNotCreatable.
2	PublicNotCreatable. Other applications can use objects of this class only if your component creates the objects first. Other applications cannot use the **CreateObject** function or the **New** operator to create objects from the class.
3	SingleUse. Allows other applications to create objects from the class, but every object of this class that a client creates starts a new instance of your component. Not allowed in ActiveX DLL projects.
4	GlobalSingleUse. Similar to SingleUse, except that properties and methods of the class can be invoked as if they were simply global functions. Not allowed in ActiveX DLL projects.
5	MultiUse. Allows other applications to create objects from the class. One instance of your component can provide any number of objects created in this fashion.
6	GlobalMultiUse. Similar to MultiUse, with one addition: properties and methods of the class can be invoked as if they were simply global functions. Its not necessary to explicitly create an instance of the class first, because one will automatically be created.

Setting	Applies to Project Type			

	ActiveX Exe	ActiveX DLL	ActiveX Control	Std. Exe
Private	X	X	X	X
PublicNotCreatable	X	X	X	
SingleUse	X			
GlobalSingleUse	X			
MultiUse	X	X		
GlobalMultiUse	X	X		

Remarks

The **Instancing** property applies to Class modules and was expanded in Visual Basic 5.0 to incorporate the functionality of the Visual Basic 4.0 **Public** property.

When a class is creatable, you can use any of the following techniques to create instances of the class from other applications:

- Use the **CreateObject** function, as in:

  ```
  Set MyInstance = CreateObject("MyProject.MyClass")
  ```

- Use the **Dim** statement within the same project (or outside the project if the **Public** property is also set to **True**), as in:

  ```
  Dim MyInstance As New MyClass
  ```

  The **New** keyword indicates that MyInstance is to be declared as a new instance of MyClass.

If the **Public** property is **False**, the setting of the **Instancing** property is ignored. You can always create instances of the class within the project that defines the class. If the **Public** property is **True**, the class is visible and therefore can be controlled by other applications once an instance of the class exists.

Note The properties and methods of a GlobalMultiUse object are not part of the global name space of the component that provides the object. For example, within the project that contains the GlobalUtility class module, you must explicitly create an instance of GlobalUtility in order to use the object's properties and methods. Other limitations of global objects are listed in "Global Objects and Code Libraries," in "Building Code Components" in the *Component Tools Guide*.

Visual Basic Reference

Visual Studio 6.0

IntegralHeight Property

See Also Example Applies To

Returns or sets a value indicating if the control displays partial items. Read-only at run time.

Syntax

object.**IntegralHeight** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A Boolean expression that determines whether the list is resized, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	(Default) The list resizes itself to display only complete items.
False	The list doesn't resize itself even if the item is too tall to display completely.

Remarks

If the number of items in a list exceeds what can be displayed, a scroll bar is automatically added to the control. You can prevent partial rows from being displayed by setting the IntegralHeight property to True.

Data Type

Boolean

Visual Basic: DataRepeater Control

Visual Studio 6.0

IntegralHeight Property (DataRepeater Control)

See Also Example Applies To

Returns or sets a value that determines if the control displays partial rows.

Syntax

object.**IntegralHeight** [=*boolean*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression specifying if the control shows partial rows, as shown in Settings.

Settings

The settings for *boolean* are:

Constant	Description
False	Partial rows are shown.
True	(Default) The control shows only complete rows.

Visual Basic: MSChart Control

Intensity Property

See Also Example Applies To

Returns or sets the strength of the light coming from the light source.

Syntax

object.**Intensity** [= *strength*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
strength	Single. The light strength. If the intensity is set to 100 percent (1), chart surfaces facing the light source are fully illuminated. If the light is set at 50 percent (.5), these surfaces receive 50 percent illumination from this light. Valid range is 0 to 1.

Remarks

Intensity is the default property of the **LightSource** object.

Visual Studio 6.0

Visual Basic: MSChart Control

Intersection Property

See Also Example Applies To

Returns a reference to an **Intersection** object that describes the point at which an axis intersects another axis on a chart.

Syntax

object.**Intersection**

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

Visual Basic Reference

Visual Studio 6.0

Interval Property

See Also Example Applies To

Returns or sets the number of milliseconds between calls to a **Timer** control's Timer event.

Syntax

object.**Interval** [= *milliseconds*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
milliseconds	A numeric expression specifying the number of milliseconds, as described in Settings.

Settings

The settings for *milliseconds* are:

Setting	Description
0	(Default) Disables a **Timer** control.
1 to 65,535	Sets an interval (in milliseconds) that takes effect when a **Timer** control's **Enabled** property is set to **True**. For example, a value of 10,000 milliseconds equals 10 seconds. The maximum, 65,535 milliseconds, is equivalent to just over 1 minute.

Remarks

You can set a **Timer** control's **Interval** property at design time or run time. When using the **Interval** property, remember:

- The **Timer** control's **Enabled** property determines whether the control responds to the passage of time. Set **Enabled** to **False** to turn a **Timer** control off, and to **True** to turn it on. When a **Timer** control is enabled, its countdown always starts from the value of its **Interval** property setting.
- Create a Timer event procedure to tell Visual Basic what to do each time the **Interval** has passed.

Visual Basic Reference

Interval Property Example

This example enables you to adjust the speed at which a form switches colors. To try this example, paste the code into the Declarations section of a form that contains a **Timer** control, an **HScrollBar** control (horizontal scroll bar), and a **PictureBox** control, and then press F5 and click the scroll bar.

```
Private Sub Form_Load ()
   Timer1.Interval = 900   ' Set interval.
   HScroll1.Min = 100   ' Set minimum.
   HScroll1.Max = 900   ' Set maximum.
End Sub
Private Sub HScroll1_Change ()
   ' Set interval according to scroll bar value.
   Timer1.Interval = 1000 - HScroll1.Value
End Sub
Private Sub Timer1_Timer ()
   ' Switch BackColor between red and blue.
   If Picture1.BackColor = RGB(255, 0, 0) Then
      Picture1.BackColor = RGB(0, 0, 255)
   Else
      Picture1.BackColor = RGB(255, 0, 0)
   End If
End Sub
```

Visual Basic Reference

Visual Studio 6.0

InvisibleAtRuntime Property

See Also Example Applies To

Returns or sets a value determining if a control should not have a visible window at run time. The **InvisibleAtRuntime** property is read/write at the controls authoring time, and not available at the controls run time.

Settings

The settings for **InvisibleAtRuntime** are:

Setting	Description
True	Allows the control to not have a visible window at run time. The container of the control may keep the control invisible during run time, like the **Timer** control. The control is still active, and therefore the developer who uses the control can still write programs that can interact with the control. There will be no **Visible** property in the extender object.
False	(Default) The control acts as a normal control at run time, where the state of the **Visible** extender property determines the visibility of the control.

Remarks

Important Dont use the **Visible** extender property to make the control invisible at run time. If you do, the control will still have all the overhead of a visible control at run time. Furthermore, the extender properties are available to the developer and end user, who may make the control visible.

Some containers may not support the **InvisibleAtRuntime** property; in this case the control will be visible at run time.

Before creating a control that is invisible at run time, consider creating an ordinary object provided by an in-process code component (ActiveX DLL) instead. Objects provided by in-process code components require fewer resources than controls, even invisible controls. The only reason to implement an invisible control is to take advantage of a feature that is only available to ActiveX controls.

Visual Basic Reference

Visual Studio 6.0

IsBindable Property

See Also Example Applies To

Returns a boolean value indicating whether the property is bindable. This property is read-only.

Syntax

object.**IsBindable**

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

Remarks

Use this property to determine if the property is bindable.

Note This property is usually used in a Wizard to check whether a property is bindable.

Visual Basic Extensibility Reference

Visual Studio 6.0

IsBroken Property

See Also Example Applies To Specifics

Returns a Boolean value indicating whether or not the **Reference** object points to a valid reference in the registry. Read-only.

Return Values

The **IsBroken** property returns these values:

Value	Description
True	The **Reference** object no longer points to a valid reference in the registry.
False	The **Reference** object points to a valid reference in the registry.

Visual Basic Extensibility Reference

IsBroken Property Example

The following example uses the **IsBroken** property to return a value indicating whether or not the specified **Reference** object in a particular project is broken.

```
Debug.Print Application.VBE.vbprojects(1).References(1).IsBroken
```

Visual Basic Reference

Visual Studio 6.0

IsDataSource Property

See Also Example Applies To

Returns a boolean value indicating whether the property is a data source. This property is read-only.

Syntax

object.**IsDataSource**

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

Remarks

Use this property to determine if the property is a data source and can be attached to a data control.

Note This property is usually used in a Wizard to check whether a property is a data source.

Visual Basic Reference

Visual Studio 6.0

IsDirty Property

See Also Example Applies To

Returns a value indicating whether this component was edited since the last time it was saved.

Syntax

object.**IsDirty**

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

Visual Basic for Applications Reference

Visual Studio 6.0

IsReady Property

See Also Example Applies To Specifics

Description

Returns **True** if the specified drive is ready; **False** if it is not.

Syntax

object.**IsReady**

The object is always a **Drive** object.

Remarks

For removable-media drives and CD-ROM drives, **IsReady** returns **True** only when the appropriate media is inserted and ready for access.

The following code illustrates the use of the **IsReady** property:

```
Sub ShowDriveInfo(drvpath)
    Dim fs, d, s, t
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set d = fs.GetDrive(drvpath)
    Select Case d.DriveType
        Case 0: t = "Unknown"
        Case 1: t = "Removable"
        Case 2: t = "Fixed"
        Case 3: t = "Network"
        Case 4: t = "CD-ROM"
        Case 5: t = "RAM Disk"
    End Select
    s = "Drive " & d.DriveLetter & ": - " & t
    If d.IsReady Then
        s = s & vbCrLf & "Drive is Ready."
    Else
        s = s & vbCrLf & "Drive is not Ready."
    End If
    MsgBox s
End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

IsRootFolder Property

See Also Example Applies To Specifics

Description

Returns **True** if the specified folder is the root folder; **False** if it is not.

Syntax

object.**IsRootFolder**

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

Remarks

The following code illustrates the use of the **IsRootFolder** property:

```
Dim fs
Set fs = CreateObject("Scripting.FileSystemObject")
Sub DisplayLevelDepth(pathspec)
    Dim f, n
    Set f = fs.GetFolder(pathspec)
    If f.IsRootFolder Then
        MsgBox "The specified folder is the root folder."
    Else
        Do Until f.IsRootFolder
            Set f = f.ParentFolder
            n = n + 1
        Loop
        MsgBox "The specified folder is nested " & n & " levels deep."
    End If
End Sub
```

Visual Basic Reference

Visual Studio 6.0

Italic Property

See Also Example Applies To

Returns or sets the font style of the **Font** object to either italic or nonitalic.

Syntax

object.**Italic** [= *boolean*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression specifying the font style as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	Turns on italic formatting.
False	(Default) Turns off italic formatting.

Remarks

The **Font** object isn't directly available at design time. Instead you set the **Italic** property by selecting a control's **Font** property in the Properties window and clicking the Properties button. In the Font Style box of the Font dialog box, select either Italic or Bold Italic. At run time, however, you set **Italic** directly by specifying its setting for the **Font** object.

Visual Basic Reference

Bold, Italic, Size, StrikeThrough, Underline, Weight Properties Example

This example prints text on a form with each mouse click. To try this example, paste the code into the Declarations section of a form, and then press F5 and click the form twice.

```
Private Sub Form_Click ()
   Font.Bold = Not Font.Bold   ' Toggle bold.
   Font.StrikeThrough = Not Font.StrikeThrough ' Toggle strikethrough.
   Font.Italic = Not Font.Italic   ' Toggle italic.
   Font.Underline = Not Font.Underline   ' Toggle underline.
   Font.Size = 16   ' Set Size property.
   If Font.Bold Then
      Print "Font weight is " & Font.Weight & " (bold)."
   Else
      Print "Font weight is " & Font.Weight & " (not bold)."
   End If
End Sub
```

Visual Basic Reference

Visual Studio 6.0

Item Property

See Also Example Applies To

Returns a specific member of a **Collection** object either by position or by key.

Syntax

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

The **Item** property syntax has the following object qualifier and part:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
index	Required. An expression that specifies the position of a member of the collection. If a numeric expression, index must be a number from 1 to the value of the collection's Count property. If a string expression, index must correspond to the key argument specified when the member referred to was added to the collection.

Remarks

If the value provided as index doesnt match any existing member of the collection, an error occurs.

Item is the default property for a collection. Therefore, the following lines of code are equivalent:

```
Print MyCollection(1)
Print MyCollection.Item(1)
```

Visual Basic Reference

Visual Studio 6.0

Item Property (ActiveX Controls)

See Also Example Applies To

Returns a specific member of a **Collection** object either by position or by key.

Syntax

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

The **Item** property syntax has the following object qualifier and part:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
index	Required. An expression that specifies the position of a member of the collection. If a numeric expression, index must be a number from 1 to the value of the collection's Count property. If a string expression, index must correspond to the key argument specified when the member referred to was added to the collection.

Remarks

If the value provided as index doesnt match any existing member of the collection, an error occurs.

Item is the default property for a collection. Therefore, the following lines of code are equivalent:

```
Print MyCollection(1)
Print MyCollection.Item(1)
```

Visual Basic Reference

Visual Studio 6.0

Item Property (DEDesigner Extensibility)

See Also Example Applies To

Returns a specific member of a collection object either by position or by key.

Syntax

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

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

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
index	Required. An expression that specifies the ordinal position, key, or the name of the returning collection object. For more information, see Visual Basic's Item property.

Remarks

If the value provided as index doesnt match any existing member of the collection, an error occurs.

Item is the default property for a collection. Therefore, the following lines of code are equivalent.

```
Print MyCollection(1)
Print MyCollection.Item(1)
```

Visual Basic for Applications Reference

Visual Studio 6.0

Item Property

See Also Example Applies To Specifics

Description

Sets or returns an *item* for a specified *key* in a **Dictionary** object. For collections, returns an *item* based on the specified *key*. Read/write.

Syntax

object.**Item(***key***)** [= *newitem*]

The **Item** property has the following parts:

Part	Description
object	Required. Always the name of a collection or **Dictionary** object.
key	Required. *Key* associated with the item being retrieved or added.
newitem	Optional. Used for **Dictionary** object only; no application for collections. If provided, *newitem* is the new value associated with the specified *key*.

Remarks

If *key* is not found when changing an *item*, a new *key* is created with the specified *newitem*. If *key* is not found when attempting to return an existing item, a new *key* is created and its corresponding item is left empty.

Visual Studio 6.0

Visual Basic: MSChart Control

Item Property (MSChart)

See Also Example Applies To

Returns a reference to an object within a collection that describes a chart element.

Syntax

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

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An unique integer that identifies the member of the collection.

Visual Basic: RDO Data Control

Visual Studio 6.0

Item Property (RDO)

See Also Example Applies To

Returns a specific member of an Remote Data Objects (RDO) collection object either by position or by key.

Syntax

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

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	Required. An expression that specifies the position of a member of the collection. If a numeric expression, index must be a number from 1 to the value of the collection's **Count** property. If a String expression, index must correspond to the key argument specified when the member referred to was added to the collection.

Remarks

Basically, the **Item** property is used to choose a member of the collection either by ordinal number or by a key value.

If the value provided as *index* doesnt match any existing member of the collection, an error occurs.

The **Item** property is the default method for a collection and is rarely used to reference collection members. Therefore, the following lines of code are equivalent:

```
Print MyCollection(1)
Print MyCollection.Item(1)
```

Visual Basic Reference

Visual Studio 6.0

ItemData Property

See Also Example Applies To

Returns or sets a specific number for each item in a **ComboBox** or **ListBox** control.

Syntax

object.**ItemData(***index***)** [= *number*]

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

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Index	The number of a specific item in the object.
Number	The number to be associated with the specified item.

Remarks

The **ItemData** property is an array of long integer values with the same number of items as a control's **List** property. You can use the numbers associated with each item to identify the items. For example, you can use an employee's identification number to identify each employee name in a **ListBox** control. When you fill the **ListBox**, also fill the corresponding elements in the **ItemData** array with the employee numbers.

The **ItemData** property is often used as an index for an array of data structures associated with items in a **ListBox** control.

Note When you insert an item into a list with the **AddItem** method, an item is automatically inserted in the **ItemData** array as well. However, the value isn't reinitialized to zero; it retains the value that was in that position before you added the item to the list. When you use the **ItemData** property, be sure to set its value when adding new items to a list.

Visual Basic Reference

ItemData Property Example

This example fills a **ListBox** control with employee names and fills the **ItemData** property array with employee numbers using the **NewIndex** property to keep the numbers synchronized with the sorted list. A **Label** control displays the name and number of an item when the user makes a selection. To try this example, paste the code into the Declarations section of a form that contains a **ListBox** and a **Label**. Set the **Sorted** property for the **ListBox** to **True**, and then press F5 and click the **ListBox**.

```
Private Sub Form_Load ()
   ' Fill List1 and ItemData array with
   ' corresponding items in sorted order.
   List1.AddItem "Judy Phelps"
   List1.ItemData(List1.NewIndex) = 42310
   List1.AddItem "Chien Lieu"
   List1.ItemData(List1.NewIndex) = 52855
   List1.AddItem "Mauro Sorrento"
   List1.ItemData(List1.NewIndex) = 64932
   List1.AddItem "Cynthia Bennet"
   List1.ItemData(List1.NewIndex) = 39227
End Sub

Private Sub List1_Click ()
   ' Append the employee number and the employee name.
   Msg = List1.ItemData(List1.ListIndex) & " "
   Msg = Msg & List1.List(List1.ListIndex)
   Label1.Caption = Msg
End Sub
```