Visual Basic Reference

Visual Studio 6.0

Name Property

See Also Example Applies To

- Returns the name used in code to identify a form, control, or data access object. Read-only at run time.
- Returns or sets the name of a font object.

Syntax

object.**Name**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list. If *object* is omitted, the form associated with the active form module is assumed to be *object*.

Remarks

The default name for new objects is the kind of object plus a unique integer. For example, the first new **Form** object is Form1, a new **MDIForm** object is MDIForm1, and the third **TextBox** control you create on a form is Text3.

An object's **Name** property must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (_) characters but can't include punctuation or spaces. Forms can't have the same name as another public object such as **Clipboard**, **Screen**, or **App**. Although the **Name** property setting can be a keyword, property name, or the name of another object, this can create conflicts in your code.

You can use a form's **Name** property with the **Dim** statement at run time to create other instances of the form. You can't have two forms with the same name at design time.

You can create an array of controls of the same type by setting the **Name** property to the same value. For example, when you set the name of all option buttons in a group to MyOpt, Visual Basic assigns unique values to the **Index** property of each control to distinguish it from others in the array. Two controls of different types can't share the same name.

Note Although Visual Basic often uses the **Name** property setting as the default value for the **Caption**, **LinkTopic**, and **Text** properties, changing one of these properties doesn't affect the others.

Changing the case of the **Name** property value for a Form or other module without otherwise changing the name itself, however, can cause a Conflicting names error message the next time the project containing the form or module is loaded. For example, changing Form1 to form1 will cause the error; changing Form1 to formX will not.

The error is caused by the way module names are stored within the project file the procedure for changing names within the project file isnt case sensitive, while the procedure for reading names on project load is.

Visual Basic Reference

Visual Studio 6.0

Name Property (DEDesigner Extensibility)

See Also Example Applies To

Returns or sets a unique name for each object within the current Data Environment designer. This can be the name of a **DEAggregate**, **DECommand**, **DEConnection**, **DEField**, or **DEParameter** object. For **DEField** objects, this is a read-only property.

The **Name** property enables you to easily differentiate multiple objects within your DataEnvironment object. Each name must be unique within its domain, as duplicate names result in errors. That is, the names of the **DEConnection** and **DECommand** objects must be unique among each other, and the name of a DEAggregate object must be unique among the **DEField** and other **DEAggregate** objects.

Syntax

object.**Name** [=*string*]

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

Part	**Description**
object	An object expression that evaluates to an item in the Applies To list.
string	A string expression that evaluates the name of the DataEnvironment, DEAggregate, DECommand, DEConnection, DEField, or DEParameter object.

Remarks

When you create a DataEnvironment object, its default name is "DataEnvironment" and an integer, such as DataEnvironment1. This default name is similar for the **DECommand**, **DEConnection**, and **DEAggregate** objects. You should set the **Name** property and provide the object with a meaningful, unique name. For example, if you are a creating a **DEConnection** object based on the Northwind database, a logical name could be "Northwind." Likewise, if you are creating a **DECommand** object based on the Customers table, a logical name could be "Customers."

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Name Property

See Also Example Applies To Specifics

Description

Sets or returns the name of a specified file or folder. Read/write.

Syntax

object.**Name** [= *newname*]

The **Name** property has these parts:

Part	Description
object	Required. Always the name of a **File** or **Folder** object.
newname	Optional. If provided, *newname* is the new name of the specified *object*.

Remarks

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

```
Sub ShowFileAccessInfo(filespec)
    Dim fs, f, s
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.GetFile(filespec)
    s = f.Name & " on Drive " & UCase(f.Drive) & 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
```

Visual Basic: RDO Data Control

Visual Studio 6.0

Name Property (Remote Data)

See Also Example Applies To

Returns the name of a **RemoteData** object.

Syntax

object.**Name**

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

Return Values

The **Name** property returns a string expression that represents the name assigned to the object. The following table describes how each object is assigned its name.

Assigning the Name Property for Remote Data Objects

Remote Data Object	Name property is determined by
rdoEnvironments(0)	**rdoEngine** Set to "Default_Environment."
rdoEnvironments(1-*n*)	*name* argument of **rdoCreateEnvironment.**
rdoConnection	Data source name (DSN) used for connection.
rdoResultset	First 256 characters of the SQL query.
rdoQuery	*name* argument in **CreateQuery** method or set directly for stand-alone **rdoQuery** objects.
rdoTable	Database table name once the **rdoTables** collection is populated.
rdoParameter	"Param*n*" where "*n*" is the ordinal number.
rdoColumn	Database column name.
rdoError	Not applicable. **rdoErrors** collection members can only be referenced by their ordinal number.

Remarks

rdoTable and **rdoQuery** objects can't share the same name. In other words, you cannot create two **rdoQuery** objects that have the same name.

Use the **Name** property to reference members of a collection in code, but in most cases, it is easier to simply use the ordinal number. Generally, you can use the **Name** property to map database table and column names.

Visual Basic: RDO Data Control

Name Property Example (RDO)

The following example illustrates use of the Name property to expose the names of all tables associated with a chosen database, the names of each column for the selected table, and specific type information about a selected column. This application uses three **Listbox** controls and a **Command** button control.

```
Option Explicit
Dim en As rdoEnvironment
Dim cn As rdoConnection
Dim rs As rdoResultset
Dim tb As rdoTable
Dim cl As rdoColumn
Dim er As rdoError

Private Sub Command1_Click()
Set en = rdoEngine.rdoEnvironments(0)

Set cn = en.OpenConnection(dsName:="WorkDB", _
    prompt:=rdDriverNoPrompt, _
    Connect:="Uid=;pwd=;database=Pubs")

For Each tb In cn.rdoTables
    List1.AddItem tb.Name
Next
List1.ListIndex = 1
End Sub

Private Sub List1_Click()
List3.Enabled = False
List2.Clear
For Each cl In cn.rdoTables((List1)).rdoColumns
    List2.AddItem cl.Name
Next
List3.Enabled = True
End Sub

Private Sub List2_Click()
List3.Clear
With cn.rdoTables((List1)).rdoColumns((List2))
    List3.AddItem "Source Column:" & .SourceColumn
    List3.AddItem "Source Table:" & .SourceTable
    List3.AddItem "Type:" & .Type
    List3.AddItem "Size:" & .Size
    List3.AddItem "Ordinal Position:"  _
   & .OrdinalPosition
    List3.AddItem "Allow Zero Length ?" _
    & .AllowZeroLength
    List3.AddItem "Required:" & .Required
    List3.AddItem "Chunk Required?:" & .ChunkRequired
    List3.AddItem "Updatable?:" & .Updatable
End With
End Sub
```

Visual Basic: SysInfo Control

Visual Studio 6.0

Name Property

See Also Example Applies To

Returns the name used in code to identify a **SysInfo** control.

Syntax

object.**Name**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list. If *object* is omitted, the form associated with the active form module is assumed to be *object*.

Remarks

The default name for new objects is the kind of object plus a unique integer. For example, the first new form object is Form1, a new **SysInfo** control is SysInfo1, and the third **TextBox** control you create on a form is Text3.

An object's **Name** property must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (_) characters but can't include punctuation or spaces. Forms can't have the same name as another public object such as **Clipboard**, **Screen**, or **App**. Although the **Name** property setting can be a keyword, property name, or the name of another object, this can create conflicts in your code.

You can create an array of controls of the same type by setting the **Name** property to the same value. For example, when you set the name of all option buttons in a group to MyOpt, Visual Basic assigns unique values to the **Index** property of each control to distinguish it from others in the array. Two controls of different types can't share the same name.

Visual Basic Extensibility Reference

Visual Studio 6.0

Name Property (VBA Add-In Object Model)

See Also Example Applies To Specifics

Returns or sets a String containing the name used in code to identify an object. For the **VBProject** object and the **VBComponent** object, read/write; for the **Property** object and the **Reference** object, read-only.

Remarks

The following table describes how the **Name** property setting applies to different objects.

Object	**Result of Using Name Property Setting**
VBProject	Returns or sets the name of the active project.
VBComponent	Returns or sets the name of the component. An error occurs if you try to set the **Name** property to a name already being used or an invalid name.
Property	Returns the name of the property as it appears in the **Property Browser**. This is the value used to index the **Properties** collection. The name can't be set.
Reference	Returns the name of the reference in code. The name can't be set.

The default name for new objects is the type of object plus a unique integer. For example, the first new Form object is Form1, a new Form object is Form1, and the third TextBox control you create on a form is TextBox3.

An object's **Name** property must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (_) characters but can't include punctuation or spaces. Forms and modules can't have the same name as another public object such as **Clipboard**, **Screen**, or **App**. Although the **Name** property setting can be a keyword, property name, or the name of another object, this can create conflicts in your code.

Visual Basic Extensibility Reference

Name Property Example

The following example uses the **Name** property to return the name of the specified member of the **VBComponents** collection in a particular project.

```
Debug.Print Application.VBE.VBProjects(1).VBComponents(1).Name
```

Visual Studio 6.0

Visual Basic: MSChart Control

Name Property (VtFont)

See Also Example Applies To

Returns or sets the name of the font. This is the default property of the **VtFont** object.

Syntax

object.**Name** [= *text*]

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

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
text	Integer. The text containing the font name.

Visual Basic Reference

Visual Studio 6.0

Name Property (WebClass, WebItem)

See Also Example Applies To

Sets or returns the name used in code to identify a **WebClass** or **WebItem** object. Read-only at run time.

Syntax

object.**Name** = [*value*]

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
value	Alphanumeric value; the name must must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (_) characters but can't include punctuation or spaces.

Remarks

The default name for new objects is the type of object plus a unique integer. For example, the first new **WebClass** object is WebClass1.

Visual Basic Reference

Visual Studio 6.0

NameInURL Property

See Also Example Applies To

Sets the name of the .htm file for the **WebClass** object.

Syntax

object.**NameInURL** = [*value*]

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	Alphanumeric value; the value must be unique across the project and across other projects in the same directory and project group. By default, is set equal to the value of the Name property.

Remarks

The NameinURL property value is the name that will be used in URL references to this WebClass. For example, if you specify CustomerInquiry as the **NameinURL** property value, the .htm file generated for the webclass will be named CustomerInquiry.htm. In URLs, this would appear as:

```
http://www.mycompany-inc-10.com/CustomerSupport/CustomerInquiry.htm
```

If you change the value of the NameinURL property, the designer will automatically update the URLs in your template that are shown in the Webclass designer's Detail panel. However, if you entered manual notation for a URL in the template, you will have to change the name of the .htm file yourself after you change the value of this property. Any links to the webclass other than those shown in the Detail panel need to be changed manually. These include URLs in other webclasses, .htm files, HTML pages, and client-side script.

Visual Basic Reference

Visual Studio 6.0

Negotiate Property

See Also Example Applies To

Sets a value that determines whether a control that can be aligned is displayed when an active object on the form displays one or more toolbars. Not available at run time.

Settings

The **Negotiate** property has these settings:

Setting	Description
True	If the control is aligned within the form (the **Align** property is set to a nonzero value), the control remains visible when an active object on the form displays a toolbar.
False	(Default) The control isn't displayed when an active object on the form displays a toolbar. The toolbar of the active object is displayed in place of the control.

Remarks

The **Negotiate** property exists for all controls with an **Align** property. You use the **Align** property to align the control within a **Form** or **MDIForm** object; however, the toolbar negotiation occurs only on the **MDIForm**. The aligned control must be on the **MDIForm**.

If the **NegotiateToolbars** property is set to **False**, the setting of the **Negotiate** property has no effect.

Visual Basic Reference

Visual Studio 6.0

NegotiateMenus Property

See Also Example Applies To

Sets a value that determines whether or not a form incorporates the menus from an object on the form on the form's menu bar. Not available at run time.

Settings

The **NegotiateMenus** property has these settings:

Setting	Description
True	(Default) When an object on the form is active for editing, the menus of that object are displayed on the form's menu bar.
False	Menus of objects on the form aren't displayed on the form's menu bar.

Remarks

Using the **NegotiateMenus** property, you determine if the menu bar of a form will share (or negotiate) space with the menus of an active object on the form. If you don't want to include the menus of the active object on the menu bar of your form, set **NegotiateMenus** to **False**.

You can't negotiate menus between an **MDIForm** object and an object on the **MDIForm**.

If **NegotiateMenus** is set to **True**, the form must have a menu bar defined, even if the menu bar isn't visible. If the **MDIChild** property of the form is set to **True**, the menus of the active object are displayed on the menu bar of the MDI parent window (**MDIForm** object).

When **NegotiateMenus** is set to **True**, you can use the **NegotiatePosition** property of individual **Menu** controls to determine the menus that your form displays along with the menus of the active object.

Visual Basic Reference

Visual Studio 6.0

NegotiatePosition Property

See Also Example Applies To

Sets a value that determines whether or not top-level **Menu** controls are displayed on the menu bar while a linked object or embedded object object on a form is active and displaying its menus. Not available at run time.

Settings

The **NegotiatePosition** property has these settings:

Setting	Description
0	(Default) None. The menu isn't displayed on the menu bar when the object is active.
1	Left. The menu is displayed at the left end of the menu bar when the object is active.
2	Middle. The menu is displayed in the middle of the menu bar when the object is active.
3	Right. The menu is displayed at the right end of the menu bar when the object is active.

Remarks

Using the **NegotiatePosition** property, you determine the individual menus on the menu bar of your form that share (or negotiate) menu bar space with the menus of an active object on the form. Any menu with **NegotiatePosition** set to a nonzero value is displayed on the menu bar of the form along with menus from the active object.

If the **NegotiateMenus** property is set to **False**, the setting of the **NegotiatePosition** property has no effect.

Visual Basic Reference

Visual Studio 6.0

NegotiateToolbars Property

See Also Example Applies To

Sets a value that determines whether the toolbars of an object on an MDI child form are displayed on the **MDIForm** when the object on the MDI child form is active. Not available at run time.

Settings

The **NegotiateToolbars** property has these settings:

Setting	Description
True	(Default) The **MDIForm** object displays the toolbars of the active object on the top or bottom of the **MDIForm**. The active object determines whether the toolbars are displayed at the top or bottom of the **MDIForm**.
False	The toolbars of the active object either aren't displayed at all or are displayed as floating tool palettes, as determined by the active object.

Remarks

Use the **NegotiateToolbars** property when creating a multiple-document interface (MDI) application that includes objects on MDI child forms. With this property, you determine how the active object displays its toolbars. By setting this property to **True**, the **MDIForm** shares (or negotiates) space at the top or bottom of the form to display the toolbars of the active object.

If the **MDIForm** also contains a toolbar, use the **Negotiate** property to determine how the various toolbars share the available space.

Visual Basic Reference

Visual Studio 6.0

NewIndex Property

See Also Example Applies To

Returns the index of the item most recently added to a **ComboBox** or **ListBox** control. Read only at run time.

Syntax

object.**NewIndex**

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

Remarks

You can use this property with sorted lists when you need a list of values that correspond to each item in the **ItemData** property array. As you add an item in a sorted list, Visual Basic inserts the item in the list in alphabetic order. This property tells you where the item was inserted so that you can insert a corresponding value in the **ItemData** property at the same index.

The **NewIndex** property returns -1 if there are no items in the list or if an item has been deleted since the last item was added.

Visual Basic: Windows Controls

Visual Studio 6.0

NewRow Property

See Also Example Applies To

Sets or returns whether a **Band** object will be displayed in a new row on the **CoolBar** control.

Syntax

object.**NewRow** [= *boolean*]

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

Part	**Description**
object	An object expression that evaluates to a **Band** object.
boolean	A Boolean expression specifying whether the band will be displayed in a new row.

Settings

The settings for *boolean* are:

Setting	**Description**
False	(Default) Band will be added to the last existing row.
True	Band will start a new row.

Note The **NewRow** property doesnt affect the first **Band** object in a **Bands** collection. The first **Band** always starts a new row.

Visual Basic: MAPI Controls

Visual Studio 6.0

NewSession Property

See Also Example Applies To

Specifies whether a new mail session should be established, even if a valid session currently exists.

Syntax

object.**NewSession**[= *value*]

The **NewSession** 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 specifying whether a new mail session should be established, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	A new messaging session is established, regardless of whether a valid session already exists.
False	(Default) Use the existing session established by the user.

Data Type

Boolean

Visual Basic: Windows Controls

Visual Studio 6.0

Next Property

See Also Example Applies To

Returns a reference to the next sibling **Node** of a **TreeView** control's **Node** object.

Syntax

object.**Next**

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).Child
   .Text = "New text"
   .Key = "New key"
   .SelectedImage = 3
End With
```

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

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

Visual Basic: Windows Controls

Next Property Example

This example adds several **Node** objects to a **TreeView** control. The **LastSibling** property, in conjunction with the **Next** 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(,,"dad","Mike")
   Set nodX = TreeView1.Nodes.Add(,,"mom","Carol")
   ' Alice is the LastSibling.
   Set nodX = TreeView1.Nodes.Add(,,,"Alice")

   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Marsha")
   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Jan")
   ' Cindy is the LastSibling.
   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Cindy")
   nodX.EnsureVisible ' Show all nodes.

   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Greg")
   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Peter")
   ' Bobby is the LastSibling.
   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Bobby")
   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 FirstSibling's index.
   n = Node.FirstSibling.Index
   ' Place FirstSibling's text & linefeed in string variable.
   strText = Node.FirstSibling.Text & vbLF
   ' While n is not the index of the last sibling, go to the
   ' next sibling and place its text into the string variable.
   While n <> Node.LastSibling.Index
      strText = strText & TreeView1.Nodes(n).Next.Text & vbLF
   ' Set n to the next node's index.
      n = TreeView1.Nodes(n).Next.Index
   Wend
   MsgBox strText ' Display results.
End Sub
```

Visual Basic Reference

Visual Studio 6.0

NextItem Property

See Also Example Applies To

Returns another **WebItem** object within the current **WebClass** object. Used to shift processing from one WebItem to another during a single request.

Syntax

object.**NextItem**

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

Remarks

Prior to firing events within a **WebClass,** this property is set to **Nothing**. On completion of an event procedure, if the property is set to another **WebItem** in the **WebClass**, the Respond event for that **WebItem** is fired. This property can be used in the Start event of the WebClass to display the initial **WebItem**. The value of **NextItem** is ignored for the following events:

- **EndRequest**
- **ProcessTag**
- **FatalErrorResponse**

Visual Basic: Windows Controls

Visual Studio 6.0

Nodes Property

See Also Example Applies To

Returns a reference to a collection of **TreeView** control **Node** objects.

Syntax

object.**Nodes**

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

Remarks

You can manipulate **Node** objects using standard collection methods (for example, the **Add** and **Remove** methods). You can access each element in the collection by its index, or by a unique key that you store in the **Key** property.

Visual Basic: Windows Controls

Nodes Property Example

This example adds several **Node** objects to a **TreeView** control. When the form is clicked, a reference to each **Node** is used to display each **Node** object's text. 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 form.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"R","Root")
   Set nodX = TreeView1.Nodes.Add("R", tvwChild,"C1","Child 1")
   Set nodX = TreeView1.Nodes.Add("R", tvwChild,"C2","Child 2")
   Set nodX = TreeView1.Nodes.Add("R", tvwChild,"C3","Child 3")
   Set nodX = TreeView1.Nodes.Add("R", tvwChild,"C4","Child 4")
   nodX.EnsureVisible
   TreeView1.Style = tvwTreelinesText ' Style 4.
   TreeView1.BorderStyle = vbFixedSingle
End Sub

Private Sub Form_Click()
   Dim i As Integer
   Dim strNodes As String
   For i = 1 To TreeView1.Nodes.Count
   strNodes = strNodes & TreeView1.Nodes(i).Index & " " & _
   "Key: " & TreeView1.Nodes(i).Key & " " & _
   "Text: " & TreeView1.Nodes(i).Text & vbLF
   Next i
   MsgBox strNodes
End Sub
```

Visual Basic Reference

Visual Studio 6.0

NonModalAllowed Property

See Also Example Applies To

Returns a value which indicates if a form can be shown non-modally (modeless). Not available at design-time.

Syntax

object.**nonModalAllowed**

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

Return Type

Boolean

Visual Basic: Multimedia MCI Control

Visual Studio 6.0

Notify Property (Multimedia MCI Control)

See Also Example Applies To

Determines if the next **MCI** command uses MCI notification services. If set to **True**, the **Notify** property generates a callback event (Done), which occurs when the next **MCI** command is complete. This property is not available at design time.

Syntax

[*form*.]*MMControl*.**Notify**[= {**True** | **False**}]

Remarks

The following table lists the **Notify** property settings for the **Multimedia MCI** control.

Setting	Description
False	(Default) The next command does not generate the Done event.
True	The next command generates the Done event.

The value assigned to this property is used only with the next **MCI** command. Subsequent **MCI** commands ignore the **Notify** property until it is assigned another (different or identical) value.

Note A notification message is aborted when you send a new command that prevents the callback conditions set by a previous command, from being satisfied. For example, to restart a paused device that does not support the **MCI Resume** command, the **Multimedia MCI** control sends the **Play** command to the paused device. However, the **Play** command that restarts the device sets callback conditions, superseding callback conditions and pending notifications from earlier commands.

Data Type

Integer (Boolean)

Visual Basic: Multimedia MCI Control

Examples (Multimedia MCI Control)

Visual Basic Example

The following example illustrates the procedure used to open an MCI device with a compatible data file. By placing this code in the Form_Load procedure, your application can use the **Multimedia MCI** control "as is" to play, record, and rewind the file Gong.wav. To try this example, first create a form with a **Multimedia MCI** control.

```
Private Sub Form_Load ()
   ' Set properties needed by MCI to open.
   MMControl1.Notify = FALSE
   MMControl1.Wait = TRUE
   MMControl1.Shareable = FALSE
   MMControl1.DeviceType = "WaveAudio"
   MMControl1.FileName = "C:\WINDOWS\MMDATA\GONG.WAV"

   ' Open the MCI WaveAudio device.
   MMControl1.Command = "Open"
End Sub
```

To properly manage multimedia resources, you should close those MCI devices that are open before exiting your application. You can place the following statement in the Form_Unload procedure to close an open MCI device before exiting from the form containing the **Multimedia MCI** control.

```
Private Sub Form_Unload (Cancel As Integer)
   MMControl1.Command = "Close"
End Sub
```

Visual Basic: Multimedia MCI Control

Visual Studio 6.0

NotifyMessage Property (Multimedia MCI Control)

See Also Example Applies To

Describes the notify code returned in the Done event, triggered by the **Notify** Property. The **NotifyMessage** property is not available at design time and is read-only at run time.

Syntax

[*form*.]*MMControl*.**NotifyMessage**

Data Type

String

Visual Basic: Multimedia MCI Control

Visual Studio 6.0

NotifyValue Property (Multimedia MCI Control)

See Also Example Applies To

Specifies the result of the last **MCI** command that requested a notification. This property is not available at design time and is read-only at run time.

Syntax

[*form*.]*MMControl*.**NotifyValue**

Remarks

The following table lists the **NotifyValue** return values for the **Multimedia MCI** control.

Value	Setting/Device mode
1	**mciNotifySuccessful**
	Command completed successfully.
2	**mciNotifySuperseded**
	Command was superseded by another command.
4	**mciNotifyAborted**
	Command was aborted by the user.
8	**mciNotifyFailure**
	Command failed.

The program can check the **Notify** code returned in the Done event to determine this value for the most recent **MCI** command.

Data Type

Integer (Enumerated)

Visual Basic: MSComm Control

Visual Studio 6.0

NullDiscard Property

See Also Example Applies To

Determines whether null characters are transferred from the port to the receive buffer.

Syntax

object.**NullDiscard** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An boolean expression specifying whether null characters are transferred from the port to the receive buffer, as described in Settings

Settings

The settings for *value* are:

Setting	Description
True	Null characters are *not* transferred from the port to the receive buffer.
False	(Default) Null characters are transferred from the port to the receive buffer.

Remarks

A null character is defined as ASCII character 0, `Chr$(0)`.

Data Type

Boolean

Visual Basic Reference

Visual Studio 6.0

NullValue Property

See Also Example Applies To

Sets or returns a value used to format and unformat Null values. Read/write both at design time and run time.

Syntax

object.**NullValue** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	Optional Variant. When formatting occurs, if the data is a Null, *value* is returned. When data is returned to the database, if the data matches *value*, a Null is written.

Remarks

Ignored when the **Type** property is set to fmtGeneral. The **NullValue** property is read each time a null field is fetched.

Visual Basic: RDO Data Control

Visual Studio 6.0

Number Property (Remote Data)

See Also Example Applies To

Returns a numeric value specifying a native error.

Syntax

object.**Number**

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

Return Values

The return value is a Long integer representing an error number.

Remarks

Use the **Number** property to determine the nature of an error that occurred on the remote server or in the ODBC interface with the data source. The value of the property corresponds to a unique number that corresponds to an error condition generated by a stored procedure, a syntax or other procedural error, a permissions or rule violation or some other type of error. This native number can also be generated by a remote procedure executing a statement such as SQL Server's RAISERROR statement.

Note The SQL Server error severity level is *not* returned by the ODBC driver, and is therefore unavailable.

Visual Basic: RDO Data Control

Error Description and Number Properties Example

The following code opens a read-only ODBC cursor connection against the SQL Server "SEQUEL" and includes a simple error handler that displays the error description and number.

```
Sub MakeConnection()
Dim rdoCn As New rdoConnection
On Error GoTo CnEh
With rdoCn
    .Connect = "UID=;PWD=;Database=WorkDB;" _
        & "Server=SEQUEL;Driver={SQL Server}" _
        & "DSN='';"
    .LoginTimeout = 5
    .CursorDriver = rdUseODBC
    .EstablishConnection rdDriverNoPrompt, True
End With
AbandonCn:
Exit Sub

CnEh:
Dim er As rdoError
Dim msg as string
    Msg = "An error occured " _
    & "while opening the connection:" _
    & Err & " - " & Error & VbCr
    For Each er In rdoErrors
        Msg = Msg & er.Description  _
         & ":" & er.Number & VbCr
    Next er
    Resume AbandonCn
End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

Number Property

See Also Example Applies To Specifics

Returns or sets a numeric value specifying an error. **Number** is the **Err** object's default property. Read/write.

Remarks

When returning a user-defined error from an object, set **Err.Number** by adding the number you selected as an error code to the **vbObjectError** constant. For example, you use the following code to return the number 1051 as an error code:

```
Err.Raise Number := vbObjectError + 1051, Source:= "SomeClass"
```

Visual Basic for Applications Reference

Number Property Example

The first example illustrates a typical use of the **Number** property in an error-handling routine. The second example examines the **Number** property of the **Err** object to determine whether an error returned by an Automation object was defined by the object, or whether it was mapped to an error defined by Visual Basic. Note that the constant **vbObjectError** is a very large negative number that an object adds to its own error code to indicate that the error is defined by the server. Therefore, subtracting it from **Err.Number** strips it out of the result. If the error is object-defined, the base number is left in `MyError`, which is displayed in a message box along with the original source of the error. If **Err.Number** represents a Visual Basic error, then the Visual Basic error number is displayed in the message box.

```
' Typical use of Number property
Sub test()
   On Error GoTo out

   Dim x, y
   x = 1 / y   ' Create division by zero error
   Exit Sub
   out:
   MsgBox Err.Number
   MsgBox Err.Description
   ' Check for division by zero error
   If Err.Number = 11 Then
      y = y + 1
   End If
   Resume
End Sub
```

```
' Using Number property with an error from an
' Automation object
Dim MyError, Msg
' First, strip off the constant added by the object to indicate one
' of its own errors.
MyError = Err.Number - vbObjectError
' If you subtract the vbObjectError constant, and the number is still
' in the range 0-65,535, it is an object-defined error code.
If MyError > 0 And MyError < 65535 Then
   Msg = "The object you accessed assigned this number to the error: " _
          & MyError & ". The originator of the error was: " _
         & Err.Source & ". Press F1 to see originator's Help topic."
' Otherwise it is a Visual Basic error number.
Else
   Msg = "This error (# " & Err.Number & ") is a Visual Basic error" & _
         " number. Press Help button or F1 for the Visual Basic Help" _
         & " topic for this error."
End If
   MsgBox Msg, , "Object Error", Err.HelpFile, Err.HelpContext
```

Visual Basic: DataGrid Control

Visual Studio 6.0

NumberFormat Property

See Also Example Applies To

Returns or sets a value indicating the format string for the **Column** object of a **DataGrid** control.

Syntax

object.**NumberFormat** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A string expression that defines how the expression in the **Value** property is formatted. The default value is a zero-length string ("").

Remarks

The **Text** property of the **Column** object is derived by applying this format to the **Value** property of the **Column** object. If **NumberFormat** is set to an invalid string, data in the cells are displayed as #ERR# and the value set in the **Value** property remains unchanged. See the **Format** function for information about valid format strings.

Visual Basic: DataGrid Control

NumberFormat Property Example

This example formats the second column in a **DataGrid** control as a *long date*:

```
Private Sub Command1_Click ()
   DataGrid1.Columns(1).NumberFormat = "long date"
End Sub
```

Visual Basic Reference

Visual Studio 6.0

NumericScale Property

See Also Example Applies To

Returns or sets the maximum number of digits to the right of the decimal point for 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.**NumericScale** [=*number*]

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

Part	**Description**
object	An object expression that evaluates to an item in the Applies To list.
number	A Long expression that specifies the maximum number of digits to the right of the decimal point for the DEField or DEParameter object.

Remarks

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

Visual Basic Extensibility Reference

Visual Studio 6.0

NumIndices Property (VBA Add-In Object Model)

See Also Example Applies To Specifics

Returns the number of indices on the property returned by the **Property** object.

Remarks

The value of the **NumIndices** property can be an integer from 0 4. For most properties, **NumIndices** returns 0. Conventionally indexed properties return 1. Property arrays might return 2.

Visual Basic Extensibility Reference

NumIndices Property Example

The following example uses the **NumIndices** property to return the number of indexes belonging to the specified property of a particular **VBComponent** object

```
Debug.Print Application.VBE.VBProjects(1).VBComponents(1).Properties(40).NumIndices
```

Visual Basic Extensibility Reference

Visual Studio 6.0

NumIndices Property

See Also Example Applies To

Returns the number of indices on the property returned by the **Property** object, which is the number of indices required to access the value.

Syntax

object.**NumIndices**

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

Remarks

The value of **NumIndices** can have a value from 0 to 4. For normal properties, as in the **ForeColor** property, **NumIndices** returns 0. Conventionally indexed properties, such as the **List** property of a **ListBox** control, return 1. Property arrays might return a 2.