Visual Basic Extensibility Reference

Visual Studio 6.0

MakeCompiledFile Method

See Also Example Applies To

Causes the current project to be written as an .exe file, DLL, or control, depending on project type.

Syntax

object.**MakeCompiledFile**

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

Visual Basic: RDO Data Control

Visual Studio 6.0

MoreResults Method (Remote Data)

See Also Example Applies To

Clears the current result set of any pending rows and returns a Boolean value that indicates if one or more additional result sets are pending.

Syntax

variable = *object*.**MoreResults**

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

Part	Description
variable	A Boolean variable that indicates if additional result sets are found as described in Return Values.
object	An object expression that evaluates to an open **rdoResultset** object variable.

Return Values

The return values for *variable* are:

Value	Description
True	Additional result sets are ready to be processed.
False	All result sets in the **rdoResultset** have been processed.

Remarks

Calling this method will flush the current result set, call the ODBC API **SQLMoreResults** function to see if there is another result set on the same statement, and if there is, loads the new result set, positions the current row pointer at the first row and returns **True**. If there are no more result sets, this method will return **False**, and both the **EOF** and **BOF** properties will be **True**.

If the result set was created asynchronously (developer used **rdAsyncEnable** in the ***Options*** parameter), the **MoreResults** method will be executed asynchronously as well. You should use the **StillExecuting** property to determine when the next result set has been enabled. Asynchronous execution of the **MoreResults** method follows the same rules as asynchronously opening a result set.

When the query used to create an **rdoResultset** returns more than one result set, use the **MoreResults** method to end processing of the current result set and test for subsequent result sets. If there are no additional result sets to process, the **MoreResults** method returns **False** and both **BOF** and **EOF** are set to **True**. In any case, using the **MoreResults** method flushes the current **rdoResultset**.

You can also use the **Cancel** method to flush the contents of an **rdoResultset**. However, **Cancel** also flushes any additional result sets not yet processed.

Not all cursor libraries support multiple resultset queries. For example, the Server-side cursor library does not support this type of query unless you disable the cursor processor by requesting a forward-only, read-only cursor with a **RowsetSize** property of 1.

Visual Basic: RDO Data Control

MoreResults Method Example

The following example illustrates use of the **MoreResults** method. In this example, an SQL query containing three separate SELECT queries is executed. The first query simply returns the number of publishers in the *Publishers* table. The next two queries each return two columns resulting from more complex join operations. All of this information is displayed in a **ListBox** control.

```
Option Explicit
Dim Cn As New rdoConnection
Dim Rs As rdoResultset
Dim SQL As String

Private Sub Test_Click()
SQL = "Select Count(*) From Publishers" _
   & " Select Pub_Name, Title  " _
   & " From Publishers P, Titles T" _
   & " Where P.Pub_ID = T.Pub_ID" _
   & " Select Au_Lname, Title " _
   & " From Titles T, TitleAuthor Ta, Authors A" _
   & " Where T.title_ID = ta.Title_ID " _
   & " and Ta.Au_ID = A.Au_ID"
Set Rs = Cn.OpenResultset(SQL, rdOpenForwardOnly, _
   rdConcurReadOnly)
' From the first set of results
List1.AddItem "Publishers: " & Rs(0)
'
'   Loop through all of the remaining result sets
'
Do While Rs.MoreResults
   List1.AddItem Rs(0).Name & " - " & Rs(1).Name
   Do Until Rs.EOF
      List1.AddItem Rs(0) & "     -    " & Rs(1)
      Rs.MoveNext
   Loop
Loop
End Sub

Private Sub Form_Load()
On Error GoTo CnEh
With Cn
   .Connect = "UID=;PWD=;Database=Pubs;" _
   & "Server=SEQUEL;Driver={SQL Server}" _
   & "DSN='';"
   .LoginTimeout = 5
   .CursorDriver = rdUseOdbc
   .EstablishConnection rdDriverNoPrompt, True
End With
Exit Sub

CnEh:
Dim er As rdoError
Debug.Print Err, Error
For Each er In rdoErrors
```

```
    Debug.Print er.Description, er.Number
Next er
Resume Next
End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

Move Method

See Also Example Applies To Specifics

Description

Moves a specified file or folder from one location to another.

Syntax

object.**Move** *destination*

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

Part	Description
object	Required. Always the name of a **File** or **Folder** object.
destination	Required. Destination where the file or folder is to be moved. Wildcard characters are not allowed.

Remarks

The results of the **Move** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.MoveFile** or **FileSystemObject.MoveFolder**. You should note, however, that the alternative methods are capable of moving multiple files or folders.

Visual Basic: RDO Data Control

Visual Studio 6.0

Move Method (Remote Data)

See Also Example Applies To

Repositions the current row pointer in an **rdoResultset** object.

Syntax

object.**Move** *rows*[, *start*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
rows	A signed Long value that specifies the number of rows the position will move as described in Settings.
start	A Variant value that identifies a bookmark as described in Settings.

Settings

If ***rows*** is greater than 0, the position is moved forward (toward the end of the cursor). If ***rows*** is less than 0, the position is moved backward (toward the beginning of the cursor). If ***rows*** is equal to 0, any pending edits are discarded and the current row is refreshed from the data source. At a lower level, when you use 0 as the ***rows*** argument, RDO executes the ODBC **SQLExtendedFetch** function to re-fetch the current rowset (as determined by the **RowsetSize** property) from the database.

If ***start*** is specified, the move begins relative to this bookmark. If ***start*** is not specified, **Move** begins from the current row.

Remarks

If using **Move** repositions the current row to a position before the first row, the position is moved to the beginning-of-file (**BOF**) position. If the **rdoResultset** contains no rows and its **BOF** property is set to **True**, using this method to move backward triggers a trappable run-time error. If either the **BOF** or **EOF** property is **True** and you attempt to use the **Move** method without a valid bookmark, a trappable error is triggered.

If using **Move** repositions the current row to a position after the last row, the position is moved to the end-of-file (**EOF**) position. If the **rdoResultset** contains no rows and its **EOF** property is set to **True**, then using this method to move forward produces a trappable run-time error.

If you use **Move** on an **rdoResultset** object based on an SQL-specific query or rdoQuery, the query is forced to completion and the **rdoResultset** object is fully populated.

If you use any method that repositions the current row pointer after using the **Edit** or **AddNew** method but before using the **Update** method, any changes made to the copy buffer are lost.

To make the first, last, next, or previous row in an **rdoResultset** the current row, use the **MoveFirst**, **MoveLast**, **MoveNext**, or **MovePrevious** method. To position the current row pointer based on an absolute row number, use the **AbsolutePosition** property. To position the current row pointer based on a percentage of the accessed rows of a result set, use the **PercentPosition** property.

When you use the **Move** method or any other method to reposition the current row pointer, the RowCurrencyChange event is fired.

When using a forward-only **rdoResultset**, the you can reposition the current row *only* by using the **MoveNext** method. You cannot use the **MoveLast**, **MovePrevious**, **MoveFirst**, or **Move** method, or the **PercentPosition** or **AbsolutePosition** property, to reposition the current row pointer.

Visual Basic Reference

Visual Studio 6.0

Move Method

See Also Example Applies To

Moves an **MDIForm**, **Form**, or control. Doesn't support named arguments.

Syntax

object.**Move** *left*, *top*, *width*, *height*

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

Part	Description
object	Optional. An object expression that evaluates to an object in the Applies To list. If *object* is omitted, the form with the focus is assumed to be *object*.
left	Required. Single-precision value indicating the horizontal coordinate (x-axis) for the left edge of *object*.
top	Optional. Single-precision value indicating the vertical coordinate (y-axis) for the top edge of *object*.
width	Optional. Single-precision value indicating the new width of *object*.
height	Optional. Single-precision value indicating the new height of *object*.

Remarks

Only the *left* argument is required. However, to specify any other arguments, you must specify all arguments that appear in the syntax before the argument you want to specify. For example, you can't specify *width* without specifying *left* and *top*. Any trailing arguments that are unspecified remain unchanged.

For forms and controls in a **Frame** control, the coordinate system is always in twips. Moving a form on the screen or moving a control in a **Frame** is always relative to the origin (0,0), which is the upper-left corner. When moving a control on a **Form** object or in a **PictureBox** (or an MDI child form on an **MDIForm** object), the coordinate system of the container object is used. The coordinate system or unit of measure is set with the **ScaleMode** property at design time. You can change the coordinate system at run time with the **Scale** method.

Visual Basic Reference

Move Method Example

This example uses the **Move** method to move a form around on the screen. To try this example, paste the code into the Declarations section of a form, and then press F5 and click the form.

```
Private Sub Form_Click ()
   Dim Inch, Msg   ' Declare variables.
   Msg = "Choose OK to resize and move this form by "
   Msg = Msg & "changing the value of properties."
   MsgBox Msg   ' Display message.
   Inch = 1440   ' Set inch in twips.
   Width = 4 * Inch   ' Set width.
   Height = 2 * Inch   ' Set height.
   Left = 0   ' Set left to origin.
   Top = 0   ' Set top to origin.
   Msg = "Now choose OK to resize and move this form "
   Msg = Msg & "using the Move method."
   MsgBox Msg   ' Display message.
   Move Screen.Width - 2 * Inch, Screen.Height - Inch, 2 * Inch, Inch
End Sub
```

Visual Basic: MSChart Control

MoveData Method

See Also Example Applies To

Moves a range of data within a data grid associated with a chart.

Syntax

object.**MoveData** (*top*, *left*, *bottom*, *right*, *overOffset*, *downOffset*)

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
top	Integer. Identifies the first row in the range to move.
left	Integer. Identifies the first column in the range to move.
bottom	Integer. Identifies the last row in the range to move.
right	Integer. Identifies the last column in the range to move.
overOffset	Integer. Identifies the horizontal direction data should be moved. A positive value moves data to the right; a negative value moves data to the left.
downOffset	Integer. Identifies the vertical direction data should be moved. A positive value moves data down, a negative value moves data up.

Visual Basic for Applications Reference

Visual Studio 6.0

MoveFile Method

See Also Example Applies To Specifics

Description

Moves one or more files from one location to another.

Syntax

object.**MoveFile** *source*, *destination*

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

Part	Description
object	Required. Always the name of a **FileSystemObject**.
source	Required. The path to the file or files to be moved. The *source* argument string can contain wildcard characters in the last path component only.
destination	Required. The path where the file or files are to be moved. The *destination* argument can't contain wildcard characters.

Remarks

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination file to create. In either case, three things can happen when an individual file is moved:

- If *destination* does not exist, the file gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If destination is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any files. The **MoveFile** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

Important This method allows moving files between volumes only if supported by the operating system.

Visual Basic: RDO Data Control

Visual Studio 6.0

MoveFirst, MoveLast, MoveNext, MovePrevious Methods (Remote Data)

See Also Example Applies To

Repositions the current row pointer to the first, last, next, or previous row in a specified **rdoResultset** object and makes that row the current row.

The syntax for these methods have these parts:

object.{**MoveFirst** | **MoveNext** | **MovePrevious**}

object. **MoveLast** ([*Options* as Variant])

Part	Description
object	An object expression that evaluates to a **rdoResultset** object.
options	A Variant or constant that determines how the operation is carried out, as specified in Settings.

Settings

You can use the following constant for the ***options*** argument:

Constant	Value	Description
rdAsyncEnable	32	Execute operation asynchronously.

Remarks

Use the *Move* methods to reposition the current row pointer from row to row without applying a condition.

If you specify the **rdAsyncEnable** option with the **MoveLast** method, the move operation is executed asynchronously. That is, control is returned to your application immediately often before the operation has completed. This prevents your application from blocking until the operation is complete. To check for completion of the operation, you can either wait for the QueryComplete or RowCurrencyChange events, or periodically check the **StillExecuting** property which returns **False** when the move operation is complete.

Caution If you edit the current row, be sure to save the changes using the **Update** method before you move to another row. If you move to another row without updating, your changes are lost without warning.

When you open the result set named by *object*, the first row is current and the **BOF** property is set to **False**. If the result set contains no rows, the **BOF** property is set to **True**, and there is no current row.

If the first or last row is already current when you use **MoveFirst** or **MoveLast**, the current row doesnt change.

If you use **MovePrevious** when the first row is current, the **BOF** property is set to **True**, and there is no current row. If you use **MovePrevious** again, an error occurs; **BOF** remains **True**.

If you use **MoveNext** when the last row is current, the **EOF** property is set to **True**, and there is no current row. If you use **MoveNext** again, an error occurs; **EOF** remains **True**.

If you use **MoveLast** on an **rdoResultset** object based on an SQL-specific query or rdoQuery, the query is forced to completion and the **rdoResultset** object is fully populated.

If you use any method that repositions the current row pointer after using the **Edit** or **AddNew** method but before using the **Update** method, any changes made to the copy buffer are lost.

To move the position of the current row in an **rdoResultset** object a specific number of rows forward or backward, use the **Move** method.

To position the current row pointer based on an absolute row number, use the **AbsolutePosition** property. To position the current row pointer based on a percentage of the accessed rows of a result set, use the **PercentPosition** property.

When you use the **Move** method or any other method to reposition the current row pointer, the RowCurrencyChange event is fired.

When using a forward-only **rdoResultset**, the you can reposition the current row *only* by using the **MoveNext** method. You cannot use the **MoveLast**, **MovePrevious**, **MoveFirst**, or **Move** method, or the **PercentPosition** or **AbsolutePosition** property, to reposition the current row pointer. If you use one of the prohibited Move methods on a forward-only result set, your code will trip an ODBC "Fetch type out of range" error.

Visual Basic: RDO Data Control

Move Methods Example

This example illustrates use of the **rdAsyncEnable** option in conjunction with the **MoveLast** method. The *Phones* table is simply a table with over 15,000 rows which takes some time to process. While this is not a recommended technique, it provides a way to illustrate a query that takes a significant length of time to run and fully populate as is done when you execute the **MoveLast** method. The application uses a status bar to indicate the degree of completion of the operations.

```
Option Explicit
Dim rdoCn As New rdoConnection
Dim rdoRs As rdoResultset
Dim SQL As String
Dim TimeExpected As Single
Dim Ts As Single, Tn As Single

Private Sub Command1_Click()
TimeExpected = 5      ' We expect this to take about 5 seconds
SQL = "Select Email, Name From Phones"
Set rdoRs = rdoCn.OpenResultset(Name:=SQL, _
   Type:=rdOpenStatic, _
   LockType:=rdConcurReadOnly, _
   Option:=rdAsyncEnable)
ShowProgress "Query"
    '
'   Query Has completed... now move to the last row
    '
rdoRs.MoveLast rdAsyncEnable
' We expect this to take about 15 seconds
TimeExpected = 15
ShowProgress "MoveLast"
rdoCn.Close
End Sub

Sub ShowProgress(Operation As String)
Ts = Timer
'  time to execute  query
ProgressBar1.Max = TimeExpected
While rdoRs.StillExecuting
   Tn = Int(Timer - Ts)
   If Tn < TimeExpected Then
      ProgressBar1 = Tn
   Else
      ProgressBar1.Max = ProgressBar1.Max + 10
      TimeExpected = ProgressBar1.Max
   End If
   DoEvents
Wend
Status = Operation & "Done. Duration:"  _
   & Int(Timer - Ts)
End Sub
Private Sub Form_Load()
With rdoCn
   .Connect = "UID=;PWD=;Database=WorkDB;" _
   & "Server=BETAV486;Driver={SQL Server}" _
```

```
    & "DSN='';"
    .LoginTimeout = 5
    .EstablishConnection rdDriverNoPrompt, True
End With
Exit Sub
End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

MoveFolder Method

See Also Example Applies To Specifics

Description

Moves one or more folders from one location to another.

Syntax

object.**MoveFolder** *source*, *destination*

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

Part	Description
object	Required. Always the name of a **FileSystemObject**.
source	Required. The path to the folder or folders to be moved. The *source* argument string can contain wildcard characters in the last path component only.
destination	Required. The path where the folder or folders are to be moved. The *destination* argument can't contain wildcard characters.

Remarks

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination folder to create. In either case, three things can happen when an individual folder is moved:

- If *destination* does not exist, the folder gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any folders. The **MoveFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

Important This method allows moving folders between volumes only if supported by the operating system.

Visual Basic Reference

Visual Studio 6.0

NavigateTo Method

See Also Example Applies To

Execute a hyperlink jump to the specified target.

Syntax

object.**NavigateTo** *Target* [, *Location* [, *FrameName*]]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Target	A string expression specifying the location to jump to. This can be a document or a URL.
Location	A string expression specifying the location within the URL specified in *Target* to jump to. If *Location* is not specified, the default document will be jumped to.
FrameName	A string expression specifying the frame within the URL specified in *Target* to jump to. If *FrameName* is not specified, the default frame will be jumped to.

Remarks

If the object is in a container that supports OLE hyperlinking, then the container will jump to the specified location. If the object is in a container that does not support OLE hyperlinking, then an application that is registered as supporting hyperlinking is started to handle the request.

If *Target* does not specify a valid location, an error is raised.

Visual Basic Reference

Visual Studio 6.0

NewPage Method

See Also Example Applies To

Ends the current page and advances to the next page on the **Printer** object.

Syntax

object.**NewPage**

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

Remarks

NewPage advances to the next printer page and resets the print position to the upper-left corner of the new page. When invoked, **NewPage** increments the **Printer** object's **Page** property by 1.

Visual Basic Reference

NewPage Method Example

This example uses the **NewPage** method to begin a new printer page after printing a single, centered line of text on a page. To try this example, paste the code into the Declarations section of a form, and then press F5 and click the form.

```
Private Sub Form_Click ()
   Dim HWidth, HHeight, I, Msg   ' Declare variables.
   On Error GoTo ErrorHandler   ' Set up error handler.
   Msg = "This is printed on page"
   For I = 1 To 2   ' Set up two iterations.
      HWidth = Printer.TextWidth(Msg) / 2   ' Get one-half width.
      HHeight = Printer.TextHeight(Msg) /2   ' Get one-half height.
      Printer.CurrentX = Printer.ScaleWidth / 2 - HWidth
      Printer.CurrentY = Printer.ScaleHeight / 2 - HHeight
      Printer.Print Msg & Printer.Page & "."   ' Print.
      Printer.NewPage   ' Send new page.
   Next I
   Printer.EndDoc   ' Print done.
   Msg = "Two pages, each with a single, centered line of text, "
   Msg = Msg & "have been sent to your printer."
   MsgBox Msg   ' Display message.
   Exit Sub
ErrorHandler:
   MsgBox "There was a problem printing to your printer."
   Exit Sub
End Sub
```

Visual Basic Reference

Visual Studio 6.0

OLEDrag Method (ActiveX Controls)

See Also Example Applies To

Causes a component to initiate an OLE drag/drop operation.

Syntax

object.**OLEDrag**

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

Remarks

When the **OLEDrag** method is called, the components OLEStartDrag event occurs, allowing it to supply data to a target component.

Visual Basic Reference

Visual Studio 6.0

OLEDrag Method

See Also Example Applies To

Causes a component to initiate an OLE drag/drop operation.

Syntax

object.**OLEDrag**

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

Remarks

When the **OLEDrag** method is called, the components OLEStartDrag event occurs, allowing it to supply data to a target component.

Visual Basic Extensibility Reference

Visual Studio 6.0

OnAddinsUpdate Method

See Also Example Applies To

Occurs automatically when changes to the Vbaddin.Ini file are saved.

Syntax

object.**IDTExtensibility_OnAddinsUpdate (***custom()* **As Variant)**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
custom()	An array of variant expressions to hold user-defined data.

Remarks

This method is part of the IDTExtensibility interface, which you should implement in the class that provides your connection object.

Important Don't directly enter the syntax given above. Instead, use the **Implements** statement to generate the appropriate method template for the interface. To do this, in the Declarations section of the Class module that provides your add-in's connection object, enter:

```
Implements IDTExtensibility
```

After adding this line, you can select **IDTExtensibility** from the module's **Object** drop down box. Select each method from the **Procedure** drop down to get the procedure template shown above in Syntax. Notice that the necessary code is automatically added to the Class module.

An interface's methods are exposed through the **Implements** statement. When the above syntax is entered in the Declarations section of the Class module that handles an add-in's events, the interface's methods become available for your use through the module's **Procedure** and **Object** drop down boxes. To add the code to the module, select the method from the **Procedure** drop down box.

Note While the **OnAddinsUpdate** method is a method to the IDTExtensibility interface, to you as a Visual Basic programmer it acts and behaves like an event. In other words, when changes made to the Vbaddin.Ini file are saved, any code in the **OnAddinsUpdate** method automatically occurs, just as if it were an event procedure.

Important Since an interface is a contract between an object and Visual Basic, you must be sure to implement *all* of the methods in the interface. This means that all four **IDTExtensibility** interface methods are present in your Class module, each

containing at least one executable statement. This can consist of as little as a single remark statement, but they must each contain at least one executable statement to prevent the compiler from removing them as empty procedures.

Visual Basic Extensibility Reference

Visual Studio 6.0

OnConnection Method

See Also Example Applies To

Occurs when an add-in is connected to the Visual Basic IDE, either through the **Add-In Manager** dialog box or another add-in.

Syntax

object. **IDTExtensibility_OnConnection (***vbinst* **As Object**, *connectmode* **As vbext_ConnectMode**, *addininst* **As AddIn**, *custom()* **As Variant)**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
vbinst	An object representing the instance of the current Visual Basic session.
connectmode	An enumerated value of type **vbext_ConnectMode**, as specified in Settings.
addininst	An **AddIn** object representing the instance of the add-in.
custom()	An array of variant expressions to hold user-defined data.

Settings

The settings for *connectmode* (**vbext_ConnectMode**) are:

Constant	Value	Description
vbext_cm_AfterStartup	0	Add-in was started after the initial **Open Project** dialog box was shown.
vbext_cm_Startup	1	Add-in was started before the initial **Open Project** dialog box was shown.
vbext_cm_External	2	Add-in was started externally by another program or component.

Remarks

This method is part of the IDTExtensibility interface, which you should implement in the class that provides your connection object.

Important Don't directly enter the syntax given above. Instead, use the **Implements** statement to generate the appropriate method template for the interface. To do this, in the Declarations section of the Class module that provides your add-in's connection object, enter:

```
Implements IDTExtensibility
```

After adding this line, you can then select **IDTExtensibility** from the module's **Object** drop down box. Select each method from the **Procedure** drop down to get the procedure template shown above in Syntax. Notice that the necessary code is automatically added to the Class module.

An interface's methods are exposed through the **Implements** statement. When the above syntax is entered in the Declarations section of the Class module that handles an add-in's events, the interface's methods become available for your use through the module's **Procedure** and **Object** drop down boxes. To add the code to the module, select the method from the **Procedure** drop down box.

Note While the **OnConnection** method is a method to the IDTExtensibility interface, to you as a Visual Basic programmer it acts and behaves like an event. In other words, when an add-in is connected to the Visual Basic IDE, either through the **Add-In Manager** dialog box or another add-in, any code in the **OnConnection** method automatically occurs, just as if it were an event procedure.

Important Since an interface is a contract between an object and Visual Basic, you must be sure to implement *all* of the methods in the interface. This means that all four **IDTExtensibility** interface methods are present in your Class module, each containing at least one executable statement. This can consist of as little as a single remark statement, but they must each contain at least one executable statement to prevent the compiler from removing them as empty procedures.

Visual Basic Extensibility Reference

Visual Studio 6.0

OnDisconnection Method

See Also Example Applies To

Occurs when an add-in is disconnected from the Visual Basic IDE, either programmatically or through the **Add-In Manager** dialog box.

Syntax

object. **IDTExtensibility_OnDisconnection (***removemode* **As vbext_DisconnectMode**, *custom()***As Variant)**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
removemode	An enumerated value of type **vbext_DisconnectMode**, as specified in Settings.
custom()	An array of variant expressions to hold user-defined data.

Settings

The settings for *removemode* (**vbext_ConnectMode**) are:

Constant	Value	Description
vbext_dm_HostShutdown	0	Add-in was removed by the add-in's host being closed.
vbext_dm_UserClosed	1	Add-in was removed by the user closing it.

Remarks

This method is part of the IDTExtensibility interface, which you should implement in the class that provides your connection object.

Important Don't directly enter the syntax given above. Instead, use the **Implements** statement to generate the appropriate method template for the interface. To do this, in the Declarations section of the Class module that provides your add-in's connection object, enter:

```
Implements IDTExtensibility
```

After adding this line, you can then select **IDTExtensibility** from the module's **Object** drop down box. Select each method from the **Procedure** drop down to get the procedure template shown above in Syntax. Notice that the necessary code is automatically added to the Class module.

An interface's methods are exposed through the **Implements** statement. When the above syntax is entered in the Declarations section of the Class module that handles an add-in's events, the interface's methods become available for your use through the module's **Procedure** and **Object** drop down boxes. To add the code to the module, select the method from the **Procedure** drop down box.

Note While the **OnDisconnection** method is a method to the IDTExtensibility interface, to you as a Visual Basic programmer it acts and behaves like an event. In other words, when an add-in is disconnected from the Visual Basic IDE, either programmatically or through the **Add-In Manager** dialog box, any code in the **OnDisconnection** method automatically occurs, just as if it were an event procedure.

Important Since an interface is a contract between an object and Visual Basic, you must be sure to implement *all* of the methods in the interface. This means that all four **IDTExtensibility** interface methods are present in your Class module, each containing at least one executable statement. This can consist of as little as a single remark statement, but they must each contain at least one executable statement to prevent the compiler from removing them as empty procedures.

Visual Basic Extensibility Reference

Visual Studio 6.0

OnStartupComplete Method

See Also Example Applies To

Occurs when the startup of the Visual Basic IDE is complete.

Syntax

object. **IDTExtensibility_OnStartupComplete (***custom()* **As Variant)**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
custom()	An array of variant expressions to hold user-defined data.

Remarks

This method is part of the IDTExtensibility interface, which you should implement in the class that provides your connection object.

Important Don't directly enter the syntax given above. Instead, use the **Implements** statement to generate the appropriate method template for the interface. To do this, in the Declarations section of the Class module that provides your add-in's connection object, enter:

```
Implements IDTExtensibility
```

After adding this line, you can then select **IDTExtensibility** from the module's **Object** drop down box. Select each method from the **Procedure** drop down to get the procedure template shown above in Syntax. Notice that the necessary code is automatically added to the Class module.

An interface's methods are exposed through the **Implements** statement. When the above syntax is entered in the Declarations section of the Class module that handles an add-in's events, the interface's methods become available for your use through the module's **Procedure** and **Object** drop down boxes. To add the code to the module, select the method from the **Procedure** drop down box.

Note While the **OnStartupComplete** method is a method to the IDTExtensibility interface, to you as a Visual Basic programmer it acts and behaves like an event. In other words, when the startup of the Visual Basic IDE is complete, any code in the **OnStartupComplete** method automatically occurs, just as if it were an event procedure.

Important Since an interface is a contract between an object and Visual Basic, you must be sure to implement *all* of the methods in the interface. This means that all four **IDTExtensibility** interface methods are present in your Class module, each

containing at least one executable statement. This can consist of as little as a single remark statement, but they must each contain at least one executable statement to prevent the compiler from removing them as empty procedures.

Visual Basic: Windows Controls

Visual Studio 6.0

Open Method (Animation Control)

See Also Example Applies To

Opens an .avi file to play. If the **AutoPlay** property is set to **True**, then the clip will start playing as soon as it is loaded. It will continue to repeat until the .avi file is closed or the **Autoplay** property is set to **False**.

Syntax

object.**Open** *file*

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
file	Required. The name of the file to play.

Remarks

The **Animation** control cannot play .avi files that include sound data. In addition, the Animation control can display only uncompressed .avi files or .avi files that have been compressed using Run-Length Encoding (RLE). An error will be returned when the **Open** method is invoked with a file that includes sound data or that is in an unsupported compression format.

Visual Basic: Windows Controls

Open Method (Animation Control) Example

The following example opens an .avi file by using the Open Dialog, and begins playing it automatically. To try the example, place an **Animation** control and a **CommonDialog** control on a form, and paste the code into the form's Declarations section. Run the example, and choose an .avi file to open.

```
Private Sub Animation1_Click ()
   With CommonDialog1
      .Filter = "avi (*.avi)|*.avi"
      .ShowOpen
   End With
   With Animation1
      .Autoplay = True
      .Open CommonDialog1.Filename
   End With
End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

OpenAsTextStream Method

See Also Example Applies To Specifics

Description

Opens a specified file and returns a **TextStream** object that can be used to read from, write to, or append to the file.

Syntax

object.**OpenAsTextStream(**[*iomode*, [*format*]]**)**

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

Part	Description
object	Required. Always the name of a **File** object.
iomode	Optional. Indicates input/output mode. Can be one of three constants: **ForReading**, **ForWriting**, or **ForAppending**.
format	Optional. One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.

Settings

The *iomode* argument can have any of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.
ForAppending	8	Open a file and write to the end of the file.

The *format* argument can have any of the following settings:

Constant	Value	Description
TristateUseDefault	2	Opens the file using the system default.
TristateTrue	1	Opens the file as Unicode.
TristateFalse	0	Opens the file as ASCII.

Remarks

The **OpenAsTextStream** method provides the same functionality as the **OpenTextFile** method of the **FileSystemObject**. In addition, the **OpenAsTextStream** method can be used to write to a file.

The following code illustrates the use of the **OpenAsTextStream** method:

```
Sub TextStreamTest
    Const ForReading = 1, ForWriting = 2, ForAppending = 3
    Const TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0
    Dim fs, f, ts, s
    Set fs = CreateObject("Scripting.FileSystemObject")
    fs.CreateTextFile "test1.txt"            'Create a file
    Set f = fs.GetFile("test1.txt")
    Set ts = f.OpenAsTextStream(ForWriting, TristateUseDefault)
    ts.Write "Hello World"
    ts.Close
    Set ts = f.OpenAsTextStream(ForReading, TristateUseDefault)
    s = ts.ReadLine
    MsgBox s
    ts.Close
End Sub
```

Visual Basic: RDO Data Control

Visual Studio 6.0

OpenConnection Method (Remote Data)

See Also Example Applies To

Opens a connection to an ODBC data source and returns a reference to the **rdoConnection** object that represents a specific database.

Syntax

Set *connection* = *environment*.**OpenConnection(***dsName*[, *prompt*[, *readonly*[, *connect*[, *options*]]]]**)**

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

Part	Description
connection	An object expression that evaluates to an **rdoConnection** object that youre opening.
environment	An object expression that evaluates to an existing **rdoEnvironment** object. You must provide an **rdoEnvironment** object.
dsName	A string expression that is the name of a registered ODBC data source name or a zero-length string () as described in Settings.
prompt	A Variant or constant that determines how the operation is carried out, as specified in Settings.
readonly	A Boolean value that is **True** if the connection is to be opened for read-only access, and **False** if the connection is to be opened for read/write access. If you omit this argument, the connection is opened for read/write access.
connect	A string expression used to pass arguments to the ODBC driver manager for opening the database the *connect string* as described in Settings.
options	A Variant or constant that determines how the operation is carried out, as specified in Settings.

Settings

The ***connect*** argument constitutes the ODBC connect arguments, and is dependent on the ODBC driver. See the **Connect** property for syntax and typical settings. If the ***connect*** argument is an empty string (), the user name and password are taken from the **rdoEnvironment** objects **UserName** and **Password** properties, and a ***dsName*** argument *must* be provided.

If the provided ***dsName*** doesnt refer to a valid ODBC data source name, and the Data Source Name (DSN) parameter does not appear in the ***connect*** argument an error occurs if *prompt* is **rdDriverNoPrompt**; otherwise, the user is prompted to

select from a list of registered data source names. If ***dsName*** is a zero-length string, the connect string must indicate the driver and server names.

Based on the ***prompt*** value, the ODBC driver manager exposes a dialog which prompts the user for connection information such as DSN, user name, and password. Use one of the following constants that defines how the user should be prompted:

***prompt* Constant**	**Value**	**Description**
rdDriverPrompt	0	The driver manager displays the ODBC Data Sources dialog box. The connection string used to establish the connection is constructed from the DSN selected and completed by the user via the dialog boxes, or, if no DSN is chosen and the **DataSourceName** property is empty (in the case of the RemoteData control), the default DSN is used.
rdDriverNoPrompt	1	The driver manager uses the connection string provided in *dsName* and *connect*. If sufficient information is not provided, the **OpenConnection** method returns a trappable error.
rdDriverComplete	2	(Default) If the connection string provided includes the DSN keyword, the driver manager uses the string as provided in *connect*. Otherwise it behaves as it does when **rdDriverPrompt** is specified.
rdDriverCompleteRequired	3	Behaves like **rdDriverComplete** except the driver disables the controls for any information not required to complete the connection. If the controls are disabled, users cannot select or specify missing arguments.

You can use the following constant for the ***options*** argument:

***options* Constant**	**Value**	**Description**
rdAsyncEnable	32	Execute operation asynchronously.

Remarks

When you successfully execute the **OpenConnection** method, a new **rdoConnection** object is instantiated and added to the **rdoConnections** collection, and a network connection is established to the remote server. If the connection cannot be established, no object is created and a trappable error is fired.

Note RDO 2.0 behaves differently than RDO 1.0 in how it handles orphaned references to **rdoConnection** objects. When you Set a variable already assigned to an **rdoConnection** object with another **rdoConnection** object using the **OpenConnection** method, the existing **rdoConnection** object is closed and dropped from the **rdoConnections** collection. In RDO 1.0, the existing object remained open and was left in the **rdoConnections** collection.

If you set the ***options*** argument to **rdAsyncEnable**, the connection operation is executed asynchronously. That is, control returns to your application before the connection has been established to prevent your application from blocking while the connection is being made. You can check for completion of the connection by polling the **rdoConnection** objects **StillConnecting** property, which returns **False** when the connection operation is complete. You can also code an event procedure for the Connect event which is fired when the connect operation is complete. If you use the **Cancel** method while waiting for an asynchronous connection to be established, the connection attempt is abandoned.

Before the process of establishing a connection is started, the BeforeConnect event is fired. This event procedure permits you to examine and modify the connect string and prompt levels as needed.

There are a variety of reasons why a connection might not be made. These include but are not limited to the following:

- Lack of proper user ID and password.
- Incorrect driver or options configuration.
- Lack of correct network or server permissions.
- The remote server could not be found on the network, or is not operating.
- The remote server did not have sufficient resources or connections to permit another user to connect.

DSN-Less Connections

In some cases, it might not be necessary to create and register a Data Source Name (DSN) before attempting to open a connection to a data source. If your remote server uses the named pipes LAN protocol and the default OEMTOANSI settings, you can simply provide the name of the server and ODBC driver in the connect string. You must provide an empty DSN entry in the connect string, or in the dsName parameter as the ***last*** argument. If the ODBC driver manager finds a null DSN entry, it attempts to locate it unless it has already determined the driver and server values. The connect string shown below is used to establish a DSN-less connection to a SQL Server named BETAV486:

```
Connect = "UID=;PWD=;Database=WorkDB;" _
& "Server=BETAV486;Driver={SQL Server}" _
& "DSN='';"
```

Other Connect String Options

Establishing an **rdoConnection** may require that the user specified by the **UserName** property, or UID connect string argument have permission to access the network, the specific data source server, and the chosen database on that server. Failure to meet these qualifications might result in failure to connect.

If you do not specify a database either through the DATABASE parameter of the ***connect*** argument or through the data source entry, the database opened when you establish a connection is determined by the default database assigned to the user by the database administrator. In some cases, you can change the default database by executing an action query containing an SQL command such as the Transact SQL USE *database* statement.

Note The *connect* part of the **OpenConnection** method is coded differently than the *source* part of the **OpenDatabase** method as used with DAO. The *connect* part neither requires nor supports use of the ODBC; keyword at the beginning of the connect string. In addition, the *connect* part does not support use of the LOGINTIMEOUT argument use the **LoginTimeout** property of the **rdoEnvironment** object instead.

Use the **Close** method on the object to close a database associated with an **rdoConnection**, remove the connection from the **rdoConnections** collection, and disconnect from the data source.

You can also declare a new **rdoConnection** object using the **Dim** statement as follows:

```
Dim myCn as New rdoConnection
```

Once instantiated in this manner, you can set the **rdoConnection** properties as required, and use the **EstablishConnection** method to open the connection.

For more information about ODBC drivers and the specific connect string arguments they require, see the Help file provided with the driver.

Visual Basic: RDO Data Control

Connect Property and OpenConnection Example: DSN Connection Using OpenConnection

The following example establishes an ODBC connection using the **OpenConnection** method but requires the user to provide all connection information. In this case the example prints the resulting **Connect** property to the Immediate window.

```
Dim cn As rdoConnection
Dim en As rdoEnvironment

Set en = rdoEnvironments(0)
Set cn = en.OpenConnection(dsName:="WorkDB", _
    Prompt:=rdDriverCompleteRequired)
debug.print cn.Connect
```

Visual Basic: RDO Data Control

Visual Studio 6.0

OpenResultset Method (Remote Data)

See Also Example Applies To

Creates a new **rdoResultset** object.

Syntax

Set *variable* = *connection*.**OpenResultset**(*name* [,*type* [,*locktype* [,*option*]]])

Set *variable* = *object*.**OpenResultset**([*type* [,*locktype* [, *option*]]])

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

Part	Description
variable	An object expression that evaluates to an **rdoResultset** object.
connection	An object expression that evaluates to an existing **rdoConnection** object you want to use to create the new **rdoResultset**.
object	An object expression that evaluates to an existing **rdoQuery** or **rdoTable** object you want to use to create the new **rdoResultset**.
name	A String that specifies the source of the rows for the new **rdoResultset**. This argument can specify the name of an **rdoTable** object, the name of an **rdoQuery**, or an SQL statement that might return rows.
type	A Variant or constant that specifies the type of cursor to create as indicated in Settings.
locktype	A Variant or constant that specifies the type of concurrency control. If you dont specify a *locktype*, **rdConcurReadOnly** is assumed.
option	A Variant or constant that specifies characteristics of the new **rdoResultset**.

Settings

- ***name***
  The ***name*** argument is used when the **OpenResultset** method is used against the **rdoConnection** object, and no query has been pre-defined. In this case, name typically contains a row-returning SQL query. The query can contain more than one SELECT statement, or a combination of action queries and SELECT statements, but not just action queries, or a trappable error will result. See the **SQL** property for additional details.

- Cursor ***type***

Note Not all types of cursors and concurrency are supported by every ODBC data source driver. See **rdoResultset** for more information. In addition, not all types of cursor drivers support SQL statements that return more than one set of results. For example, server-side cursors do not support queries that contain more than one SELECT statement.

The ***type*** argument specifies the type of cursor used to manage the result set. If you dont specify a type, **OpenResultset** creates a forward-only **rdoResultset**. Not all ODBC data sources or drivers can implement all of the cursor types. If your driver cannot implement the type chosen, a warning message is generated and placed in the **rdoErrors** collection. Use one of the following result set type constants that defines the cursor type of the new **rdoResultset** object. For additional details on types of cursors, see the **CursorType** property.

***type* Constant**	**Value**	**Description**
rdOpenForwardOnly	0	(Default) Opens a forward-only-type **rdoResultset** object.
rdOpenKeyset	1	Opens a keyset-type **rdoResultset** object.
rdOpenDynamic	2	Opens a dynamic-type **rdoResultset** object.
rdOpenStatic	3	Opens a static-type **rdoResultset** object.

- Concurrency ***LockType***
  In order to maintain adequate control over the data being updated, RDO provides a number of concurrency options that control how other users are granted, or refused access to the data being updated. In many cases, when you lock a particular row using one of the **LockType** settings, the remote engine might also lock the entire page containing the row. If too many pages are locked, the remote engine might also escalate the page lock to a table lock to improve overall system performance.

  Not all lock types are supported on all data sources. For example, for SQL Server and Oracle servers, static-type **rdoResultset** objects can only support **rdConcurValues** or **rdConcurReadOnly**. For additional details on the types of concurrency, see the **LockType** property.

***locktype* Constant**	**Value**	**Description**
rdConcurReadOnly	1	(Default) Read-only .
rdConcurLock	2	Pessimistic concurrency.
rdConcurRowVer	3	Optimistic concurrency based on row ID.
rdConcurValues	4	Optimistic concurrency based on row values.
rdConcurBatch	5	Optimistic concurrency using batch mode updates. **Status** values returned for each row successfully updated.

- Other ***options***
  If you use the **rdAsyncEnable** option, control returns to your application as soon as the query is begun, but before a result set is available. To test for completion of the query, use the **StillExecuting** property. The **rdoResultset** object is not valid until **StillExecuting** returns **False**. You can also use the QueryComplete event to determine when the query

is ready to process. Until the **StillExecuting** property returns **True**, you cannot reference any other property of the uninitialized **rdoResultset** object and only the **Cancel** and **Close** methods are valid.

If you use the **rdExecDirect** option, RDO uses the *SQLExecDirect* ODBC API function to execute the query. In this case, no temporary stored procedure is created to execute the query. This option can save time if you dont expect to execute the query more than a few times in the course of your application. In addition, when working with queries that should not be run as stored procedures but executed directly, this option is mandatory. For example, in queries that create temporary tables for use by subsequent queries, you must use the **rdExecDirect** option.

You can use the following constants for the ***options*** argument:

Constant	Value	Description
rdAsyncEnable	32	Execute operation asynchronously.
rdExecDirect	64	(Default.) Bypass creation of a stored procedure to execute the query. Uses SQLExecDirect instead of SQLPrepare and SQLExecute.

Remarks

If the **OpenResultset** method succeeds, RDO instantiates a new **rdoResultset** object and appends it to the **rdoResultsets** collection even if no rows are returned by the query. If the query fails to compile or execute due to a syntax error, permissions problem or other error, the **rdoResultset** is not created and a trappable error is fired. The **rdoResultset** topic contains additional details on **rdoResultset** behavior and managing the **rdoResultsets** collection.

Note RDO 2.0 behaves differently than RDO 1.0 in how it handles orphaned references to **rdoResultset** objects. When you Set a variable already assigned to an **rdoResultset** object with another **rdoResultset** object using the **OpenResultset** method, the existing **rdoResultset** object is closed and dropped from the **rdoResultsets** collection. In RDO 1.0, the existing object remained open and was left in the **rdoResultsets** collection.

Note Before you can use the name of a base table in the ***name*** argument, you must first use the **Refresh** method against the **rdoTables** collection to populate it. You can also populate the **rdoTables** collection by referencing one of its members by its ordinal number. For example, referencing **rdoTables**(0) will populate the entire collection.

Executing Multiple Operations on a Connection

If there is an unpopulated **rdoResultset** pending on a data source that can only support a single operation on an **rdoConnection** object, you cannot create additional **rdoQuery** or **rdoResultset** objects using the **OpenResultset** method**,** or use the **Refresh** method on the **rdoTable** object until the **rdoResultset** is flushed, closed, or fully populated. For example, when using SQL Server 4.2 as a data source, you cannot create an additional **rdoResultset** object until you move to the last row of the last result set of the current **rdoResultset** object. To populate the result set, use the **MoreResults** method to move through all pending result sets, or use the **Cancel** or **Close** method on the **rdoResultset** to flush all pending result sets.

Visual Basic: RDO Data Control

ActiveConnection Property Example

The following examples illustrates use of the **ActiveConnection** property to select an **rdoConnection**. In this case, the application opens two separate connections and uses the same **rdoQuery** against each.

```
Dim rdoCn As New rdoConnection
Dim rdoCn2 As New rdoConnection
Dim rdoQy As New rdoQuery
Dim rdoRs As rdoResultset
Dim rdoCol As rdoColumn
Dim rdoEn As rdoEnvironment

Private Sub Form_Load()
On Error GoTo CnEh

Set rdoEn = rdoEnvironments(0)

With rdoCn

    .Connect = "UID=;PWD=;Database=WorkDB;" _
        & "Server=Betav486;Driver={SQL Server}" _
        & "DSN='';"
    .LoginTimeout = 5
    .EstablishConnection rdDriverNoPrompt, True
    rdoEn.rdoConnections.Add rdoCn
End With

With rdoCn2
    .Connect = "UID=;PWD=;Database=Pubs;" _
        & "Server=Betav486;Driver={SQL Server}" _
        & "DSN='';"
    .LoginTimeout = 5
    .EstablishConnection rdDriverNoPrompt, True
    rdoEn.rdoConnections.Add rdoCn2
End With

With rdoQy
    Set .ActiveConnection = rdoCn
    .SQL = "Select Name, refDate " _
      & " from Sysobjects where type = 'U' "
    .LockType = rdConcurReadOnly
    .RowsetSize = 1
    .CursorType = rdUseServer
End With

For Each rdoCn In rdoEn.rdoConnections
    Set rdoQy.ActiveConnection = rdoCn
    Set rdoRs = rdoQy.OpenResultset(rdOpenForwardOnly)
    With rdoRs
        For Each rdoCol In rdoRs.rdoColumns
            Debug.Print rdoCol.Name,
        Next
        Debug.Print
```

```
        Do Until rdoRs.EOF
                For Each rdoCol In rdoRs.rdoColumns
                    Debug.Print rdoCol
                Next
            rdoRs.MoveNext
        Loop
    End With
Next                    ' Next Connection

Exit Sub
CnEh:
Dim er As rdoError
    Debug.Print Err, Error
    For Each er In rdoErrors
        Debug.Print er.Description, er.Number
    Next er
    Resume Next
End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

OpenTextFile Method

See Also Example Applies To Specifics

Description

Opens a specified file and returns a **TextStream** object that can be used to read from or append to the file.

Syntax

object.**OpenTextFile(***filename*[**,** *iomode*[**,** *create*[**,** *format*]]]**)**

The **OpenTextFile** method has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
filename	Required. String expression that identifies the file to open.
iomode	Optional. Indicates input/output mode. Can be one of two constants, either **ForReading** or **ForAppending**.
create	Optional. **Boolean** value that indicates whether a new file can be created if the specified *filename* doesn't exist. The value is **True** if a new file is created; **False** if it isn't created. The default is **False**.
format	Optional. One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.

Settings

The *iomode* argument can have either of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForAppending	8	Open a file and write to the end of the file.

The *format* argument can have any of the following settings:

Constant	Value	Description
TristateUseDefault	2	Opens the file using the system default.
TristateTrue	1	Opens the file as Unicode.
TristateFalse	0	Opens the file as ASCII.

Remarks

The following code illustrates the use of the **OpenTextFile** method to open a file for appending text:

```
Sub OpenTextFileTest
    Const ForReading = 1, ForWriting = 2, ForAppending = 3
    Dim fs, f
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.OpenTextFile("c:\testfile.txt", ForAppending,TristateFalse)
    f.Write "Hello world!"
    f.Close
End Sub
```

Visual Basic: Internet Control

Visual Studio 6.0

OpenURL Method

See Also Example Applies To

Opens and returns the document at the specified URL. The document is returned as a variant. When the method has completed, the URL properties (and portions of the URL such as the protocol) are updated to reflect the current URL.

Syntax

object.**OpenUrl** *url [,datatype]*

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
url	Required. The URL of the document to be retrieved.
datatype	Optional. Integer that specifies the type of the data, as shown in Settings.

Settings

The settings for *datatype* are:

Constant	Value	Description
icString	0	Default. Retrieves data as string.
icByteArray	1	Retrieves data as a byte array.

Return Type

Variant

Remarks

The **OpenURL** method's return value depends on the target of the URL. For example, if the target URL is the directory of an FTP server, the directory will be returned. On the other hand, if the target is a file, the file will be retrieved.

The **OpenURL** method is equivalent to invoking the **Execute** method with a GET operation, followed by a **GetChunk** method invoked in the StateChanged event. The **OpenURL** method, however, results in a synchronous stream of data being returned from the site.

If you are retrieving a binary file, be sure to use a byte array as a temporary variable before writing it to disk, as shown below:

```
Dim b() As Byte
Dim strURL As String
' Set the strURL to a valid address.
strURL = "FTP://ftp.GreatSite.com/China.exe"
b() = Inet1.OpenURL(strURL, icByteArray)

Open "C:\Temp\China.exe" For Binary Access _
Write As #1
Put #1, , b()
Close #1
```

Note When using the **OpenURL** method, set the **URL** property before you set the **Password** and **UserName** properties. If you set the **URL** property last, the **UserName** and **Password** properties will be set to "".

Visual Basic: Internet Control

OpenURL Method Example

The example uses the **OpenURL** method to retrieve the directory of an FTP server. To try the example, place an **Internet Transfer** control and a **RichTextBox** control on a form. Paste the code into the Declarations section. Press F5 to run the example, and double-click on the form.

```
Private Sub Form_DblClick()
   Inet1.AccessType = icUseDefault
   RichTextBox1.Text = Inet1.OpenURL _
   (InputBox("URL", , "ftp://ftp.microsoft.com"))
End Sub
```

This example presumes the data is a binary file. Using a byte array, the file is retrieved and written to the disk using the **Open**, **Put**, and **Close** statements. To try the example, place an **Internet Transfer** control on a form and paste the code into the Declarations section. Press F5 and double-click on the form.

```
Private Sub Form_DblClick()
   Inet1.AccessType = icUseDefault
   Dim b() As Byte
   Dim strURL As String

   ' Presuming this is still a valid URL.
   strURL = "ftp://ftp.microsoft.com/" & _
   "developr/drg/Win32/Autorun.zip"

   ' Retrieve the file as a byte array.
   b() = Inet1.OpenURL(strURL, icByteArray)

   Open "C:\Temp\Autorun.zip" For Binary Access _
   Write As #1
   Put #1, , b()
   Close #1
   MsgBox "Done"
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

Overlay Method

See Also Example Applies To

Draws one image from a **ListImages** collection over another, and returns the result.

Syntax

object.**Overlay (***index1*, *index2***)**

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

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
index1	Required. An integer (**Index** property) or unique string (**Key** property) that specifies the image to be overlaid.
index2	Required. An integer (**Index** property) or unique string (**Key** property) that specifies the image to be drawn over the object specified in *index1*. The color of the image that matches the **MaskColor** property is made transparent. If no color matches, the image is drawn opaquely over the other image.

Remarks

Use the **Overlay** method in conjunction with the **MaskColor** property to create a single image from two disparate images. The **Overlay** method imposes one bitmap over another to create a third, composite image. The **MaskColor** property determines which color of the overlaying image is transparent.

The *index* can be either an index or a key. For example, to overlay the first picture in the collection with the second:

```
Set Picture1.Picture = ImageList1.Overlay(1,2) ' Reference by Index.
   'Or reference by Key property.
Set Picture1.Picture = ImageList1.Overlay("First", "Second")
```

Visual Basic: Windows Controls

Overlay Method Example

This example loads five **ListImage** objects into an **ImageList** control and displays any two images in two **PictureBox** controls. For each **PictureBox**, select an image to display from one of the two **ComboBox** controls. When you click the form, the code uses the **Overlay** method to create a third image that is displayed in a third **PictureBox** control. To try the example, place an **ImageList** control, two **ComboBox** controls, and three **PictureBox** 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()
   Dim X As ListImage
   ' Add 5 images to a ListImages collection.
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\elements\moon05.ico"))
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\elements\snow.ico"))
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\writing\erase02.ico"))
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\writing\note06.ico"))
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\flags\flgfran.ico"))

   With combo1   ' Populate the first ComboBox.
      .AddItem "Moon"
      .AddItem "Snowflake"
      .AddItem "Pencil"
      .AddItem "Note"
      .AddItem "Flag"
      .ListIndex = 0
   End With

   With combo2   ' Populate the second ComboBox.
      .AddItem "Moon"
      .AddItem "Snowflake"
      .AddItem "Pencil"
      .AddItem "Note"
      .AddItem "Flag"
      .ListIndex = 2
   End With

   Picture1.BackColor = vbWhite   ' Make BackColor white.
   Picture2.BackColor = vbWhite
   Picture3.BackColor = vbWhite
End Sub

Private Sub Form_Click()
   ' Overlay the two images, and display in PictureBox3.
   Set Picture3.Picture = ImageList1. _
    Overlay(combo1.ListIndex + 1, combo2.ListIndex + 1)
End Sub

Private Sub combo1_Click()
   ' Change PictureBox to reflect ComboBox selection.
```

```
    Set Picture1.Picture = ImageList1. _
     ListImages(combo1.ListIndex + 1).ExtractIcon
End Sub

Private Sub combo2_Click()
    ' Change PictureBox to reflect ComboBox selection.
    Set Picture2.Picture = ImageList1. _
     ListImages(combo2.ListIndex + 1).ExtractIcon
End Sub
```

Visual Basic Reference

Visual Studio 6.0

PaintPicture Method

See Also Example Applies To

Draws the contents of a graphics file (.bmp, .wmf, .emf, .cur, .ico, or .dib) on a **Form**, **PictureBox**, or **Printer**. Doesn't support named arguments.

Syntax

object.**PaintPicture** *picture*, *x1*, *y1*, *width1*, *height1*, *x2*, *y2*, *width2*, *height2*, *opcode*

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

Part	Description
object	Optional. An object expression that evaluates to an object in the Applies To list. If *object* is omitted, the **Form** object with the focus is assumed to be *object*.
Picture	Required. The source of the graphic to be drawn onto *object*. Must be the **Picture** property of a **Form** or **PictureBox**.
x1, *y1*	Required. Single-precision values indicating the destination coordinates (x-axis and y-axis) on *object* for *picture* to be drawn. The **ScaleMode** property of *object* determines the unit of measure used.
Width1	Optional. Single-precision value indicating the destination width of *picture*. The **ScaleMode** property of *object* determines the unit of measure used. If the destination width is larger or smaller than the source width (*width2*), *picture* is stretched or compressed to fit. If omitted, the source width is used.
Height1	Optional. Single-precision value indicating the destination height of *picture*. The **ScaleMode** property of *object* determines the unit of measure used. If the destination height is larger or smaller than the source height (*height2*), *picture* is stretched or compressed to fit. If omitted, the source height is used.
x2, *y2*	Optional. Single-precision values indicating the coordinates (x-axis and y-axis) of a clipping region within *picture*. The **ScaleMode** property of *object* determines the unit of measure used. If omitted, 0 is assumed.
Width2	Optional. Single-precision value indicating the source width of a clipping region within *picture*. The **ScaleMode** property of *object* determines the unit of measure used. If omitted, the entire source width is used.
Height2	Optional. Single-precision value indicating the source height of a clipping region within *picture*. The **ScaleMode** property of *object* determines the unit of measure used. If omitted, the entire source height is used.
Opcode	Optional. Long value or code that is used only with bitmaps. It defines a bit-wise operation (such as **vbMergeCopy** or **vbSrcAnd**) that is performed on *picture* as it's drawn on *object*. For a complete list of bit-wise operator constants, see the RasterOp Constants topic in Visual Basic Help.

	There are some limitations in the usage of opcodes. For example, you can't use any opcode other than **vbSrcCopy** if the source is an icon or metafile, and the opcodes that interact with the pattern (or "brush" in SDK terms) such as MERGECOPY, PATCOPY, PATPAINT, and PATINVERT actually interact with the **FillStyle** property of the destination. **Note** *Opcode* is used to pass a bitwise operation on a bitmap. Placing a value in this argument when passing other image types will cause an "Invalid procedure call or argument" error. This is by design. To avoid this error, leave the *Opcode* argument blank for any image other than a bitmap.

Remarks

You can flip a bitmap horizontally or vertically by using negative values for the destination height (*height1*) and/or the destination width (*width1*).

You can omit as many optional trailing arguments as you want. If you omit an optional trailing argument or arguments, don't use any commas following the last argument you specify. If you want to specify an optional argument, you must specify all optional arguments that appear in the syntax before it.

Note that there is a difference between loading a .Bmp into a **PictureBox** control, and using the Windows API function BitBlt() to add a picture to it. When you BitBlt an image, the **PictureBox** control doesn't know to resize like it does if you use the **LoadPicture** method. Setting the **ScaleWidth** and **ScaleHeight** properties to the size of the image also doesn't work. If you want the **PictureBox** to resize to the new picture after BitBlt'ing, you must do so manually by code, converting units and dealing with borders. Below is a simple example of how to do this:

```
Sub ResizePictureBoxToImage(pic as PictureBox, twipWd _
  as Integer, twipHt as Integer)
 ' This code assumes that all units are in twips.  If
 ' not, you must convert it to twips before calling
 ' this routine.  This also assumes that the image
 ' was blt'ed to 0,0.
 Dim BorderHt as Integer, BorderWd as Integer
 BorderWd = Pic.Width - Pic.ScaleWidth
 BorderHt = Pic.Height - Pic.ScaleHeight
 pic.Move pic.Left, pic.Top, twipWd + BorderWd, _
   twipHt + BorderHt
End Sub
```

Visual Basic Reference

Visual Studio 6.0

Paste Method

See Also Example Applies To

Copies data from the system Clipboard to an **OLE** container control.

Syntax

object.**Paste**

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

Remarks

To use this method, set the **OLETypeAllowed** property, and then check the value of the **PasteOK** property. You can't paste successfully unless **PasteOK** returns a value of **True**.

If the **Paste** method was carried out, the **OLEType** property is set to **vbOLELinked** (0) or **vbOLEEmbedded** (1). If the **Paste** method wasn't carried out, the **OLEType** property is set to **vbOLENone** (3).

You can use this method to support an Edit Paste command on a menu.

If the **PasteOK** property setting is **True** and Visual Basic can't paste the object, the **OLE** container control deletes any object already in the control.

Visual Basic Reference

Visual Studio 6.0

PasteSpecialDlg Method

See Also Example Applies To

Displays the Paste Special dialog box.

Syntax

object.**PasteSpecialDlg**

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

Remarks

At run time, you display this dialog box to enable the user to paste an object from the system Clipboard. This dialog box displays several options to the user, including pasting either a linked or embedded object.

Use the **OLETypeAllowed** property to determine the type of object that can be created (linked, embedded, or either) using this dialog box.

If the **PasteOK** property setting is **True** and Visual Basic can't paste the object, the **OLE** container control deletes any object already in the control.

Visual Basic: Winsock Control

Visual Studio 6.0

PeekData Method

See Also Example Applies To

Similar to **GetData** except **PeekData** does not remove data from the input queue. This method works only for TCP connections.

Syntax

object.**PeekData** *data*, [*type*,] [*maxLen*]

The **PeekData** method syntax has these parts

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	Stores retrieved data after the method returns successfully. If there is not enough data available for requested type, *data* will be set to Empty.
type	Optional. Type of data to be retrieved, as described in Settings. Default Value: **vbArray** + **vbByte**.
maxLen	Optional. Length specifies the desired size when receiving a byte array or a string. If this argument is missing for byte array or string, all available data will be retrieved. If provided for data types other than byte array and string, this argument is ignored.

Settings

The settings for *type* are:

Type	Constant
Byte	**vbByte**
Integer	**vbInteger**
Long	**vbLong**
Single	**vbSingle**
Double	**vbDouble**

Currency	**vbCurrency**
Date	**vbDate**
Boolean	**vbBoolean**
SCODE	**vbError**
String	**vbString**
Byte Array	**vbArray** + **vbByte**

Return Value

Void

Remarks

If the type is specified as **vbString**, string data is converted to UNICODE before returning to the user.

Visual Basic: Windows Controls

Visual Studio 6.0

Play Method

See Also Example Applies To

Plays an .avi file in the **Animation** control.

Syntax

object.**Play** [= *repeat, start, end*]

Part	Description
object	Required. An object expression that evaluates to an **Animation** control.
Repeat	Optional. Integer that specifies the number of times the clip will be repeated. The default is -1, which causes the clip to repeat indefinitely.
Start	Optional. Integer that specifies the starting frame. The default value is 0, which starts the clip on the first frame. The maximum value is 65535.
end	Optional. Integer that specifies the ending frame. The default value is -1, which indicates the last frame of the clip. The maximum value is 65535.

Data Type

Integer

Remarks

Use the **Stop** method to stop an .avi file from playing.

Visual Basic Reference

Visual Studio 6.0

Point Method

See Also Example Applies To

Returns, as a long integer, the red-green-blue (RGB) color of the specified point on a **Form** or **PictureBox**. Doesn't support named arguments.

Syntax

object.**Point(***x*, *y***)**

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

Part	Description
object	Optional. An object expression that evaluates to an object in the Applies To list. If *object* is omitted, the **Form** object with the focus is assumed to be *object*.
x, *y*	Required. Single-precision values indicating the horizontal (x-axis) and vertical (y-axis) coordinates of the point in the **ScaleMode** property of the **Form** or **PictureBox**. Parentheses must enclose the values.

Remarks

If the point referred to by the *x* and *y* coordinates is outside *object*, the **Point** method returns -1.

Visual Basic Reference

Point Method Example

This example uses the **Point** method to determine the color of a specific point on a form. To try this example, paste the code into the Declarations section of a form, and then press F5 and click the form.

```
Private Sub Form_Click ()
   Dim LeftColor, MidColor, Msg, RightColor   ' Declare variables.
   AutoRedraw = -1   ' Turn on AutoRedraw.
   Height = 3 * 1440   ' Set height to 3 inches.
   Width = 5 * 1440   ' Set width to 5 inches.
   BackColor = QBColor(1)   ' Set background to blue.
   ForeColor = QBColor(4)   ' Set foreground to red.
   Line (0, 0)-(Width / 3, Height), , BF   ' Red box.
   ForeColor = QBColor(15)   ' Set foreground to white.
   Line (Width / 3, 0)-((Width / 3) * 2, Height), , BF
   LeftColor = Point(0, 0)   ' Find color of left box,
   MidColor = Point(Width / 2, Height / 2)   ' middle box, and
   RightColor = Point(Width, Height)   ' right box.
   Msg = "The color number for the red box on the left side of "
   Msg = Msg & "the form is " & LeftColor & ". The "
   Msg = Msg & "color of the white box in the center is "
   Msg = Msg & MidColor & ". The color of the blue "
   Msg = Msg & "box on the right is " & RightColor & "."
   MsgBox Msg   ' Display message.
End Sub
```

Visual Basic Reference

Visual Studio 6.0

PopupMenu Method

See Also Example Applies To

Displays a pop-up menu on an **MDIForm** or **Form** object at the current mouse location or at specified coordinates. Doesn't support named arguments.

Syntax

object.**PopupMenu** *menuname*, *flags*, *x*, *y*, *boldcommand*

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

Part	Description
object	Optional. An object expression that evaluates to an object in the Applies To list. If *object* is omitted, the form with the focus is assumed to be *object*.
Menuname	Required. The name of the pop-up menu to be displayed. The specified menu must have at least one submenu.
Flags	Optional. A value or constant that specifies the location and behavior of a pop-up menu, as described in Settings.
X	Optional. Specifies the x-coordinate where the pop-up menu is displayed. If omitted, the mouse coordinate is used.
Y	Optional. Specifies the y-coordinate where the pop-up menu is displayed. If omitted, the mouse coordinate is used.
boldcommand	Optional. Specifies the name of a menu control in the pop-up menu to display its caption in bold text. If omitted, no controls in the pop-up menu appear in bold.

Settings

The settings for *flags* are:

Constant (location)	Value	Description
vbPopupMenuLeftAlign	0	(Default) The left side of the pop-up menu is located at *x*.

vbPopupMenuCenterAlign	4	The pop-up menu is centered at *x*.
vbPopupMenuRightAlign	8	The right side of the pop-up menu is located at *x*.

Constant (behavior)	Value	Description
vbPopupMenuLeftButton	0	(Default) An item on the pop-up menu reacts to a mouse click only when you use the left mouse button.
vbPopupMenuRightButton	2	An item on the pop-up menu reacts to a mouse click when you use either the right or the left mouse button.

Remarks

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

You specify the unit of measure for the x and y coordinates using the **ScaleMode** property. The x and y coordinates define where the pop-up is displayed relative to the specified form. If the x and y coordinates aren't included, the pop-up menu is displayed at the current location of the mouse pointer.

When you display a pop-up menu, the code following the call to the **PopupMenu** method isn't executed until the user either chooses a command from the menu (in which case the code for that command's Click event is executed before the code following the **PopupMenu** statement) or cancels the menu. In addition, only one pop-up menu can be displayed at a time; therefore, calls to this method are ignored if a pop-up menu is already displayed or if a pull-down menu is open.

Visual Basic Reference

PopupMenu Method Example

This example displays a pop-up menu at the cursor location when the user clicks the right mouse button over a form. To try this example, create a form that includes a **Menu** control named mnuFile (mnuFile must have at least one submenu). Copy the code into the Declarations section of the form, and press F5.

```
Private Sub Form_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
   If Button = 2 Then
      PopupMenu mnuFile
   End If
End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

Print Method

See Also Example Applies To Specifics

Prints text in the **Immediate** window.

Syntax

object.**Print** [*outputlist*]

The **Print** method syntax has the following object qualifier and part:

Part	Description
object	Optional. An object expression that evaluates to an object in the Applies To list.
outputlist	Optional. Expression or list of expressions to print. If omitted, a blank line is printed.

The *outputlist* argument has the following syntax and parts:

{**Spc(***n***)** | **Tab(***n***)**} *expression charpos*

Part	Description
Spc(*n***)**	Optional. Used to insert space characters in the output, where *n* is the number of space characters to insert.
Tab(*n***)**	Optional. Used to position the insertion point at an absolute column number where *n* is the column number. Use **Tab** with no argument to position the insertion point at the beginning of the next print zone.
expression	Optional. Numeric expression or string expression to print.
charpos	Optional. Specifies the insertion point for the next character. Use a semicolon (**;**) to position the insertion point immediately following the last character displayed. Use **Tab(***n***)** to position the insertion point at an absolute column number. Use **Tab** with no argument to position the insertion point at the beginning of the next print zone. If *charpos* is omitted, the next character is printed on the next line.

Remarks

Multiple expressions can be separated with either a space or a semicolon.

All data printed to the **Immediate** window is properly formatted using the decimal separator for the locale settings specified for your system. The keywords are output in the appropriate language for the host application.

For Boolean data, either `True` or `False` is printed. The **True** and **False** keywords are translated according to the locale setting for the host application.

Date data is written using the standard short date format recognized by your system. When either the date or the time component is missing or zero, only the data provided is written.

Nothing is written if *outputlist* data is Empty. However, if *outputlist* data is Null, `Null` is output. The **Null** keyword is appropriately translated when it is output.

For error data, the output is written as `Error errorcode`. The **Error** keyword is appropriately translated when it is output.

The *object* is required if the method is used outside a module having a default display space. For example an error occurs if the method is called in a standard module without specifying an *object*, but if called in a form module, *outputlist* is displayed on the form.

Note Because the **Print** method typically prints with proportionally-spaced characters, there is no correlation between the number of characters printed and the number of fixed-width columns those characters occupy. For example, a wide letter, such as a "W", occupies more than one fixed-width column, and a narrow letter, such as an "i", occupies less. To allow for cases where wider than average characters are used, your tabular columns must be positioned far enough apart. Alternatively, you can print using a fixed-pitch font (such as Courier) to ensure that each character uses only one column.

Visual Basic for Applications Reference

Print Method Example

Using the **Print** method, this example displays the value of the variable `MyVar` in the **Immediate** window. Note that the **Print** method only applies to objects that can display text.

```
Dim MyVar
MyVar = "Come see me in the Immediate pane."
Debug.Print MyVar
```

Visual Basic Reference

Visual Studio 6.0

PrintForm Method

See Also Example Applies To

Sends a bit-by-bit image of a **Form** object to the printer.

Syntax

object.**PrintForm**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list. If *object* is omitted, the **Form** with the focus is assumed to be *object*.

Remarks

PrintForm prints all visible objects and bitmaps of the **Form** object. **PrintForm** also prints graphics added to a **Form** object or **PictureBox** control at run time if the **AutoRedraw** property is **True** when the graphics are drawn.

The printer used by **PrintForm** is determined by the operating system's Control Panel settings.

Visual Basic Reference

PrintForm Method Example

This example uses the **PrintForm** method to print the current form. To try this example, paste the code into the Declarations section of a form. Place on the form any controls you want to see on the printed form, and then press F5 and click the form.

```
Private Sub Form_Click ()
   Dim Msg   ' Declare variable.
   On Error GoTo ErrorHandler   ' Set up error handler.
   PrintForm   ' Print form.
   Exit Sub
ErrorHandler:
   Msg = "The form can't be printed."
   MsgBox Msg   ' Display message.
   Resume Next
End Sub
```

Visual Basic Reference

Visual Studio 6.0

PrintReport Method

See Also Example Applies To

At run time, prints the data report created with the Data Report designer.

Syntax

object.**PrintReport(***ShowDialog*, *Range*, *PageFrom*, *PageTo*)

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

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
ShowDialog	Optional. A boolean expression that determines if the Print dialog box is shown.
Range	Optional. Sets an integer that determines if all the pages in the report will be executed or a range of pages, as shown in Settings.
PageFrom	Optional. An integer that sets the page from which to start printing.
PageTo	Optional. An integer that sets the page at which to stop printing.

Settings

Constant	Value	Description
rptRangeAllPages	0	(Default) All pages will be printed.
rptRangeFromTo	1	Only the specified range of pages will be exported.

Return Type

Long

Remarks

If no arguments are supplied with the method, a dialog box will be displayed prompting the user for appropriate information.

The **PrintReport** method performs an asynchronous operation. The **PrintReport** method returns the identifier of the "cookie" that identifies the asynchronous operation.

Visual Basic Reference

PrintReport Method Example

The first example displays the Print dialog box, allowing the user to specify the file name and page range. The second example prints the report without displaying the dialog box.

```
Private Sub DisplayPrintDialog()
   DataReport1.PrintReport True
End Sub

Private Sub PrintWithoutDialog()
   ' Prints all pages in the report.
   DataReport1.PrintReport False, rptRangeAllPages
End Sub
```

Visual Basic Reference

Visual Studio 6.0

PropertyChanged Method

See Also Example Applies To

Notifies the container that a propertys value has been changed.

Syntax

object.**PropertyChanged** *PropertyName*

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

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

Remarks

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

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

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

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

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

Visual Basic Reference

Visual Studio 6.0

PSet Method

See Also Example Applies To

Sets a point on an object to a specified color.

Syntax

object.**PSet** [**Step**] **(***x***,** *y***),** [*color*]

The **PSet** method syntax has the following object qualifier and parts:

Part	Description
object	Optional. Object expression that evaluates to an object in the Applies To list. If *object* is omitted, the **Form** with the focus is assumed to be *object*.
Step	Optional. Keyword specifying that the coordinates are relative to the current graphics position given by the **CurrentX** and **CurrentY** properties.
(*x***,** *y***)**	Required. **Single** values indicating the horizontal (x-axis) and vertical (y-axis) coordinates of the point to set.
color	Optional. **Long** integer value indicating the RGB color specified for point. If omitted, the current **ForeColor** property setting is used. You can use the **RGB** function or **QBColor** function to specify the color.

Remarks

The size of the point drawn depends on the setting of the **DrawWidth** property. When **DrawWidth** is 1, **PSet** sets a single pixel to the specified color. When **DrawWidth** is greater than 1, the point is centered on the specified coordinates.

The way the point is drawn depends on the setting of the **DrawMode** and **DrawStyle** properties.

When **PSet** executes, the **CurrentX** and **CurrentY** properties are set to the point specified by the arguments.

To clear a single pixel with the **PSet** method, specify the coordinates of the pixel and use the **BackColor** property setting as the *color* argument.

This method cannot be used in an **WithEnd With** block.

Visual Basic Reference

PSet Method Example

This example uses the **PSet** method to draw confetti on a form. To try this example, paste the code into the General section of a form. Then press F5 and click the form.

```
Sub Form_Click ()
   Dim CX, CY, Msg, XPos, YPos   ' Declare variables.
   ScaleMode = 3   ' Set ScaleMode to
         ' pixels.
   DrawWidth = 5   ' Set DrawWidth.
   ForeColor = QBColor(4)   ' Set foreground to red.
   FontSize = 24   ' Set point size.
   CX = ScaleWidth / 2   ' Get horizontal center.
   CY = ScaleHeight / 2   ' Get vertical center.
   Cls   ' Clear form.
   Msg = "Happy New Year!"
   CurrentX = CX - TextWidth(Msg) / 2   ' Horizontal position.
   CurrentY = CY - TextHeight(Msg)   ' Vertical position.
   Print Msg   ' Print message.
   Do
      XPos = Rnd * ScaleWidth   ' Get horizontal position.
      YPos = Rnd * ScaleHeight   ' Get vertical position.
      PSet (XPos, YPos), QBColor(Rnd * 15)   ' Draw confetti.
      DoEvents   ' Yield to other
   Loop   ' processing.
End Sub
```