Visual Basic: MAPI Controls

Visual Studio 6.0

Delete Method (MAPIMessages Control)

See Also Example Applies To

Deletes a message, recipient, or attachment.

Syntax

object.**Delete** [*value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the item to delete, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
mapMessageDelete	0	Deletes all components of the currently indexed message, reduces the **MsgCount** property by 1, and decrements the index number by 1 for each message that follows the deleted message.
		If the deleted message was the last message in the set, this method decrements the **MsgIndex** property by 1.
mapRecipientDelete	1	Deletes the currently indexed recipient. Automatically reduces the **RecipCount** property by 1, and decrements the index number by 1 for each recipient that follows the deleted recipient.
		If the deleted recipient was the last recipient in the set, this method decrements the **RecipIndex** property by 1.
mapAttachmentDelete	2	Deletes the currently indexed attachment. Automatically reduces the **AttachmentCount** property by 1, and decrements the index by 1 for each attachment that follows the deleted attachment.

		If the deleted attachment was the last attachment in the set, this method decrements the **AttachmentIndex** by 1.

Visual Basic Reference

Visual Studio 6.0

Delete Method (OLE Container)

See Also Example Applies To

Deletes the specified object and frees the memory associated with it.

Syntax

object.**Delete**

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

Remarks

This method enables you to explicitly delete an object. Objects are automatically deleted when a form is closed or when the object is replaced with a new object.

Visual Basic: RDO Data Control

Visual Studio 6.0

Delete Method (Remote Data)

See Also Example Applies To

Deletes the current row in an updatable **rdoResultset** object.

Syntax

object.**Delete**

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

Remarks

Delete removes the current row and makes it inaccessible. The deleted row is removed from the **rdoResultset** cursor and the database. When you delete rows from an **rdoResultset**, there must be a current row in the **rdoResultset** before you use **Delete**; otherwise, a trappable error is triggered.

Once you delete a row in an **rdoResultset**, you must reposition the current row pointer to another row in the **rdoResultset** before performing an operation that accesses the current row. Although you cant edit or use the deleted row, it remains current until you reposition to another row. Once you move to another row, however, you cant make the deleted row current again.

When you position to a row in your **rdoResultset** that has been deleted by another user, or if you delete a common row in another **rdoResultset**, a trappable error occurs indicating that the row has been deleted. At this point, the current row is invalid and you must reposition to another valid row. For example, if you use a bookmark to position to a deleted row, a trappable error occurs.

You can undo a row deletion if you use transactions and the **RollbackTrans** method assuming you use **BeginTrans** before using the **Delete** method.

Using **Delete** produces an error under any of the following conditions:

- There is no current row.
- The connection or **rdoResultset** is read-only.
- No columns in the row are updatable.
- The row has already been deleted.
- Another user has locked the data page containing your row.
- The user does not have permission to perform the operation.

Visual Basic for Applications Reference

Visual Studio 6.0

Delete Method

See Also Example Applies To Specifics

Description

Deletes a specified file or folder.

Syntax

object.**Delete** *force*

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

Part	Description
object	Required. Always the name of a **File** or **Folder** object.
force	Optional. **Boolean** value that is **True** if files or folders with the read-only attribute set are to be deleted; **False** (default) if they are not.

Remarks

An error occurs if the specified file or folder does not exist.

The results of the **Delete** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.DeleteFile** or **FileSystemObject.DeleteFolder**.

The **Delete** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

Visual Studio 6.0

Visual Basic: MSChart Control

DeleteColumnLabels Method

See Also Example Applies To

Deletes levels of labels from the data columns in a data grid associated with a chart.

Syntax

object.**DeleteColumnLabels** (*labelIndex*, *count*)

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
labelIndex	Integer. Identifies the number of the first level of labels you want to delete. Column label levels are numbered bottom to top, beginning with 1.
count	Integer. Specifies the number of label levels you want to delete. The number of columns being deleted is calculated from the column identified in *labelIndex* up.

Visual Studio 6.0

Visual Basic: MSChart Control

DeleteColumns Method

See Also Example Applies To

Deletes columns of data and their associated labels from the data grid associated with a chart.

Syntax

object.**DeleteColumns** (*column, count)*

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
column	Integer. Identifies a specific data column. Columns are numbered from left to right beginning with 1.
count	Integer. Specifies the number of columns you want to delete.

Visual Basic for Applications Reference

Visual Studio 6.0

DeleteFile Method

See Also Example Applies To Specifics

Description

Deletes a specified file.

Syntax

object.**DeleteFile** *filespec*[, *force*]

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

Part	Description
object	Required. Always the name of a **FileSystemObject**.
filespec	Required. The name of the file to delete. The *filespec* can contain wildcard characters in the last path component.
force	Optional. **Boolean** value that is **True** if files with the read-only attribute set are to be deleted; **False** (default) if they are not.

Remarks

An error occurs if no matching files are found. The **DeleteFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

Visual Basic for Applications Reference

Visual Studio 6.0

DeleteFolder Method

See Also Example Applies To Specifics

Description

Deletes a specified folder and its contents.

Syntax

object.**DeleteFolder** *folderspec*[, *force*]

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

Part	Description
object	Required. Always the name of a **FileSystemObject**.
folderspec	Required. The name of the folder to delete. The *folderspec* can contain wildcard characters in the last path component.
force	Optional. **Boolean** value that is **True** if folders with the read-only attribute set are to be deleted; **False** (default) if they are not.

Remarks

The **DeleteFolder** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

An error occurs if no matching folders are found. The **DeleteFolder** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

Visual Basic Extensibility Reference

Visual Studio 6.0

DeleteLines Method

See Also Example Applies To Specifics

Deletes a single line or a specified range of lines.

Syntax

object.**DeleteLines (***startline* [, *count*]**)**

The **DeleteLines** syntax has these parts:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
startline	Required. A Long specifying the first line you want to delete.
count	Optional. A **Long** specifying the number of lines you want to delete.

Remarks

If you don't specify how many lines you want to delete, **DeleteLines** deletes one line.

Visual Basic Extensibility Reference

DeleteLines Method Example

The following example has two steps. The first **ForNext** loop uses the **InsertLines** method to insert into `CodePanes(1)` 26 ever-longer initial segments of the alphabet, starting with a. The last line inserted is the entire alphabet.

The second **ForNext** loop uses the **DeleteLines** method to delete the odd-numbered lines. Although it seems that the second loop should simply delete every other line, note that after each deletion the lines get renumbered. Therefore the deletion is advancing by two lines at each step, one line because I is increasing by one and another line because the larger line numbers are each decreasing by one.

```
For I = 1 to 26
   Application.VBE.SelectedVBComponent.CodeModule.InsertLines i, Mid$("abcdefghijklmnopqrstuvwxyz", 1, I)
Next
For I = 1 to 13
   Application.VBE.SelectedVBComponent.CodeModule.DeleteLines I
Next
```

Visual Studio 6.0

Visual Basic: MSChart Control

DeleteRowLabels Method

See Also Example Applies To

Deletes levels of labels from the data rows in a data grid associated with a chart.

Syntax

object.**DeleteRowLabels** (*labelIndex*, *count*)

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
labelIndex	Integer. Identifies the number of the first level of labels you want to delete. Row labels are numbered right to left, beginning with 1.
count	Integer. Specifies the number of label levels you want to delete. Row labels are deleted from the row identified by *labelIndex* to the left.

Visual Basic: MSChart Control

DeleteRows Method

See Also Example Applies To

Deletes rows of data and their associated labels from the data grid associated with a chart.

Syntax

object.**DeleteRows** (*row*, *count*)

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
row	Integer. Identifies a specific data row. Rows are numbered from top to bottom beginning with 1.
count	Integer. Specifies the number of rows you want to delete.

Visual Basic: Windows Controls

Visual Studio 6.0

DeselectAll Method

See Also Example Applies To

Clears all selected **Tab** objects on the **TabStrip** control.

Syntax

object.**DeselectAll**

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

Remarks

Use this method after the user has selected several tabspossible only when the **MultiSelect** property is set to **True**.

Visual Basic Extensibility Reference

Visual Studio 6.0

DesignerWindow Method

See Also Example Applies To

Returns the **Window** object that represents the component's designer.

Syntax

object.**DesignerWindow**

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

Remarks

If the component supports a designer but doesn't have an open designer, using the **DesignerWindow** method creates the designer, but it isn't visible. To make the window visible, set the **Window** object's **Visible** property to **True**.

Visual Basic Extensibility Reference

DesignerWindow Method Example

The following example uses the **DesignerWindow** method and the **Visible** property to find out whether or not a particular designer is visible. Note that the **VBComponent** object must be a form.

```
Debug.Print Application.VBE.VBProjects(1).VBComponents(1).DesignerWindow.Visible
```

Visual Basic: RichTextBox Control

Visual Studio 6.0

DoVerb Method (OLEObject Object)

See Also Example Applies To

Opens an object for an operation, such as editing.

Syntax

object.**DoVerb** (*verb*)

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
verb	Optional. The verb to execute of the **OLEObject** object within **RichTextBox** control. If not specified, the default verb is executed. The value of this argument can be one of the standard verbs supported by all objects or an index of the ObjectVerbs property array.

Remarks

The **DoVerb** method executes a verb of the specified **OLEObject** object. The verb argument is an index of one of the verbs listed in the **ObjectVerbs** property array or one of the standard verbs listed below.

Each object can support its own set of verbs. The following values represent standard verbs every object should support:

Constant	Value	Description
vbOLEPrimary	0	The default action for the object.
vbOLEShow	-1	Activates the object for editing. If the application that created the object supports in-place activation, the object is activated within the **RichTextBox** control.
vbOLEOpen	-2	Opens the object in a separate application window. If the application that created the object supports in-place activation, the object is activated in its own window.
vbOLEHide	-3	For embedded objects, hides the application that created the object.
vbOLEUIActivate	-4	If the object supports in-place activation, activates the object for in-place activation and shows any user interface tools. If the object doesn't support in-place activation, the object doesn't activate, and an error occurs.

vbOLEInPlaceActivate	-5	If the user moves the focus to the embedded object, creates a window for the object and prepares the object to be edited. An error occurs if the object doesn't support activation on a single mouse click.
vbOLEDiscardUndoState	-6	Used when the object is activated for editing to discard all record of changes that the object's application can undo.

Note These verbs may not be listed in the **ObjectVerbs** property array.

Visual Basic Reference

Visual Studio 6.0

DoVerb Method

See Also Example Applies To

Opens an object for an operation, such as editing. Doesn't support named arguments.

Syntax

object.**DoVerb** (*verb*)

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

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Verb	Optional. The verb to execute of the object within the **OLE** container control. If not specified, the default verb is executed. The value of this argument can be one of the standard verbs supported by all objects or an index of the **ObjectVerbs** property array.

Remarks

If you set the **AutoActivate** property to 2 (Double-Click), the **OLE** container control automatically activates the current object when the user double-clicks the control.

Each object can support its own set of verbs. The following values represent standard verbs every object should support:

Constant	Value	Description
VbOLEPrimary	0	The default action for the object.
VbOLEShow	-1	Activates the object for editing. If the application that created the object supports in-place activation, the object is activated within the **OLE** container control.
VbOLEOpen	-2	Opens the object in a separate application window. If the application that created the object supports in-place activation, the object is activated in its own window.
VbOLEHide	-3	For embedded objects, hides the application that created the object.
VbOLEUIActivate	-4	If the object supports in-place activation, activates the object for in-place activation and shows any user interface tools. If the object doesn't support in-place activation, the object doesn't activate, and an error occurs.

VbOLEInPlaceActivate	-5	If the user moves the focus to the **OLE** container control, creates a window for the object and prepares the object to be edited. An error occurs if the object doesn't support activation on a single mouse click.
VbOLEDiscardUndoState	-6	Used when the object is activated for editing to discard all record of changes that the object's application can undo.

Note These verbs may not be listed in the **ObjectVerbs** property array.

Visual Basic Reference

Visual Studio 6.0

Drag Method

See Also Example Applies To

Begins, ends, or cancels a drag operation of any control except the **Line**, **Menu**, **Shape**, **Timer**, or **CommonDialog** controls. Doesn't support named arguments.

Syntax

object.**Drag** *action*

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

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list. If *object* is omitted, the object whose event procedure contains the **Drag** method is assumed.
action	Optional. A constant or value that specifies the action to perform, as described in Settings. If *action* is omitted, the default is to begin dragging the object.

Settings

The settings for *action* are:

Constant	Value	Description
vbCancel	0	Cancels drag operation
vbBeginDrag	1	Begins dragging *object*
vbEndDrag	2	Ends dragging and drop *object*

Remarks

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

Using the **Drag** method to control a drag-and-drop operation is required only when the **DragMode** property of the object is set to Manual (0). However, you can use **Drag** on an object whose **DragMode** property is set to Automatic (1 or **vbAutomatic**).

If you want the mouse pointer to change shape while the object is being dragged, use either the **DragIcon** or **MousePointer** property. The **MousePointer** property is only used if no **DragIcon** is specified.

The **Drag** method generally acts synchronously, meaning that subsequent statements aren't executed until the drag action is complete. It can, however, act asynchronously if the **DragMode** property for the control is set to Manual (0 or **vbManual**).

Visual Basic Reference

Drag Method Example

This example uses the **Drag** method to drag the filename of a bitmap (.bmp) file to a picture box where the bitmap is displayed. To try this example, paste all of the code into the Declarations section of a form that contains **DriveListBox**, **DirListBox**, **FileListBox**, **PictureBox**, and **Label** controls. Use the default names for all of the controls. Size and position all controls so they can be easily seen and used. The size and position of the label is unimportant because it's changed at run time. When the program begins, you can browse your file system and load any bitmaps. Once you've located a bitmap that you want to display, click the filename of that bitmap, and drag it to the picture box.

```
Private Sub Form_Load ()
   Picture1.AutoSize = -1   ' Turn on AutoSize.
   Label1.Visible = 0   ' Make the label invisible.
   File1.Pattern = "*.BMP; *.ICO; *.WMF"   ' Set file patterns.
End Sub

Private Sub Dir1_Change ()   ' Any change in Dir1
   File1.Path = Dir1.Path   ' is reflected in File1.
End Sub

Private Sub Drive1_Change ()   ' Any change in Drive1
   Dir1.Path = Drive1.Drive   ' is reflected in Dir1.
End Sub

Private Sub File1_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim DY   ' Declare variable.
   DY = TextHeight("A")   ' Get height of one line.
   Label1.Move File1.Left, File1.Top + Y - DY /2, File1.Width, DY
   Label1.Drag   ' Drag label outline.
End Sub

Private Sub Dir1_DragOver (Source As Control, X As Single, Y As Single, State As Integer)
   ' Change pointer to no drop.
   If State = 0 Then Source.MousePointer = 12
   ' Use default mouse pointer.
   If State = 1 Then Source.MousePointer = 0
End Sub

Private Sub Drive1_DragOver (Source As Control, X As Single, Y As Single, State As Integer)
   ' Change pointer to no drop.
   If State = 0 Then Source.MousePointer = 12
   ' Use default mouse pointer.
   If State = 1 Then Source.MousePointer = 0
End Sub

Private Sub Form_DragOver (Source As Control, X As Single, Y As Single, State As Integer)
   ' Change pointer to no drop.
   If State = 0 Then Source.MousePointer = 12
   ' Use default mouse pointer.
   If State = 1 Then Source.MousePointer = 0
End Sub

Private Sub File1_DragOver (Source As Control, X As Single, Y As Single, State As Integer)
   On Error Resume Next
```

```
   If State = 0 And Right$(File1.Filename,4) = ".ICO" Then
      Label1.DragIcon = LoadPicture(File1.Path + "\" + File1.Filename)
   If Err Then MsgBox "The icon file can't be loaded."
   ElseIf State = 1 Then
      Label1.DragIcon = LoadPicture ()   ' Use no drag icon.
   End If
End Sub

Private Sub Picture1_DragDrop (Source As Control, X As Single, Y As Single)
   On Error Resume Next
   Picture1.Picture = LoadPicture(File1.Path + "\" + File1.Filename)
   If Err Then MsgBox "The picture file can't be loaded."
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

Draw Method

See Also Example Applies To

Draws an image into a destination device context, such as a **PictureBox** control, after performing a graphical operation on the image.

Syntax

object.**Draw (***hDC*, *x*,*y*, *style***)**

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

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
hDC	Required. A value set to the target object's **hDC** property.
x,*y*	Optional. The coordinates used to specify the location within the device context where the image will be drawn. If you don't specify these, the image is drawn at the origin of the device context.
style	Optional. Specifies the operation performed on the image, as described in Settings.

Settings

The settings for *style* are:

Constant	Value	Description
imlNormal	0	(Default) Normal. Draws the image with no change.
imlTransparent	1	Transparent. Draws the image using the **MaskColor** property to determine which color of the image will be transparent.
imlSelected	2	Selected. Draws the image dithered with the system highlight color.
imlFocus	3	Focus. Draws the image dithered and striped with the highlight color creating a hatched effect to indicate the image has the focus.

Remarks

The **hDC** property is a handle (a number) that the Windows operating system uses for internal reference to an object. You can paint in the internal area of any control that has an **hDC** property. In Visual Basic, these include the **Form** object, **PictureBox** control, and **Printer** object.

Because an object's **hDC** can change while an application is running, it is better to specify the **hDC** property rather than an actual value. For example, the following code ensures that the correct **hDC** value is always supplied to the **ImageList** control:

```
ImageList1.ListImages(1).Draw Form1.hDC
```

Visual Basic: Windows Controls

Draw Method Example

This example loads an image into an **ImageList** control. When you click the form, the image is drawn on the form in four different styles. To try the example, place an **ImageList** 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 X As ListImage
   'Load one image into the ImageList.
   Set X = ImageList1.ListImages. _
   Add(, , LoadPicture("bitmaps\assorted\intl_no.bmp"))
End Sub

Private Sub Form_Click()
   Dim space, intW As Integer   ' Create spacing variables.

   ' Use the ImageWidth property for spacing.
   intW = ImageList1.ImageWidth
   space = Form1.Font.Size * 2 ' Use the Font.Size for height spacing.

   ScaleMode = vbPoints   ' Set ScaleMode to points.
   Cls ' Clear the form.

   ' Draw the image with Normal style.
   ImageList1.ListImages(1).Draw Form1.hDC, , space,imlNormal
   ' Set MaskColor to red, which will become transparent.
   ImageList1.MaskColor = vbRed
   ' Draw the image with red (MaskColor) the transparent color.
   ImageList1.ListImages(1).Draw Form1.hDC, intW, space,imlTransparent
   ' Draw image with the Selected style.
   ImageList1.ListImages(1).Draw Form1.hDC, intW * 2,space,imlSelected
   ' Draw image with Focus style.
   ImageList1.ListImages(1).Draw Form1.hDC, intW * 3, space,imlFocus

   ' Print a caption for the images.
   Print _
   "Normal              Transparent          Selected         Focus"

End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

DriveExists Method

See Also Example Applies To Specifics

Description

Returns **True** if the specified drive exists; **False** if it does not.

Syntax

object.**DriveExists(***drivespec***)**

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

Part	Description
object	Required. Always the name of a **FileSystemObject**.
drivespec	Required. A drive letter or a complete path specification.

Remarks

For drives with removable media, the **DriveExists** method returns **True** even if there are no media present. Use the **IsReady** property of the **Drive** object to determine if a drive is ready.

Visual Basic: RDO Data Control

Visual Studio 6.0

Edit Method (Remote Data)

See Also Example Applies To

Enables changes to data values in the current row of an updatable **rdoResultset** object.

Syntax

object.**Edit**

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

Remarks

Before you use the Edit method, the data columns of an **rdoResultset** are read-only. Executing the **Edit** method copies the current row from an updatable **rdoResultset** object to the copy buffer for subsequent editing. Changes made to the current rows columns are copied to the copy buffer. After you make the desired changes to the row, use the **Update** method to save your changes or the **CancelUpdate** method to discard them. The current row remains current after you use **Edit**.

Caution If you edit a row, and then perform any operation that repositions the current row pointer to another row without first using **Update**, your changes to the edited row are lost without warning. In addition, if you close *object*, or end the procedure which declares the result set or the parent **rdoConnection** object, your edited row might be discarded without warning.

You cannot use the **Edit** method if the **EditMode** property of the **rdoResultset** object indicates that an **Edit** or **AddNew** operation is in progress.

When the **rdoResultset** objects **LockEdits** property setting is **True** (pessimistically locked) , all rows in the **rdoResultset** objects rowset are locked as soon as the cursor is opened and remain locked until the cursor is closed. The number of rows in the rowset is determined by the **RowsetSize** property. Since many remote data sources use page locking schemes, pessimistic locking also locks all data pages of the table(s) containing a row fetched by the **rdoResultset**.

If the **LockEdits** property setting is **False** (optimistically locked), the individual row or the data page containing the row is locked and the new row is compared with the pre-edited row just before its updated in the database. If the row has changed since you last used the **Edit** method, the **Update** operation fails with a trappable error.

Note Not all data sources use page locking schemes to manage data concurrency. In some cases, data is locked on a row-by-row basis, therefore locks only affect the specific rowset being edited.

Using **Edit** produces an error under any of the following conditions:

- There is no current row.
- The connection or **rdoResultset** is read-only.
- No columns in the row are updatable.

- The **EditMode** property indicates that an **AddNew** or **Edit** is already in progress.
- Another user has locked the row or data page containing your row and the **LockEdits** property is **True**.

Visual Studio 6.0

Visual Basic: MSChart Control

EditCopy Method

See Also Example Applies To

Copies a picture of the current chart to the clipboard in Windows metafile format. It also copies the data being used to create the chart to the clipboard.

Syntax

object.**EditCopy**

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

Remarks

This method allows you to paste the chart's data or a picture of the chart itself into another application. Since both the data and the picture of the chart are stored on the clipboard, what gets pasted into the new application varies depending on the type of application. For example, if you execute the chart's **EditCopy** method in your code and then go to an Excel spreadsheet and select **Edit Paste**, the chart data set is placed in the spreadsheet. To insert the picture of the chart into the spreadsheet, select **Edit Paste Special** and select the **Picture** type.

Visual Studio 6.0

Visual Basic: MSChart Control

EditPaste Method

See Also Example Applies To

Pastes a Windows metafile graphic or tab-delimited text from the clipboard into the current selection on a chart.

Syntax

object.**EditPaste**

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

Remarks

The chart can accept several types of information from the clipboard, depending on the currently selected chart element when **EditPaste** is called. If the entire chart is selected, the chart looks for data on the clipboard and attempts to use this new data to redraw the chart. If an item that can accept a picture, such as a bar or chart backdrop is selected, the chart looks for a metafile on the clipboard. If it finds a metafile, it uses that metafile to fill the selected object.

Visual Basic Reference

Visual Studio 6.0

EndDoc Method

See Also Example Applies To

Terminates a print operation sent to the **Printer** object, releasing the document to the print device or spooler.

Syntax

object.**EndDoc**

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

Remarks

If **EndDoc** is invoked immediately after the **NewPage** method, no additional blank page is printed.

Visual Basic Reference

EndDoc Method Example

This example uses the **EndDoc** method to end a document after printing two pages, each with a centered line of text indicating the page number. 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 half width.
      HHeight = Printer.TextHeight(Msg) /2   ' Get 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   ' Printing is finished.
   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

EndQueryEdit Method

See Also Example Applies To

Returns or sets a **DECommand** object which the query edit has completed editing.

Syntax

object.**EndQueryEdit**

The **EndQueryEdit** method syntax has one part:

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

Remarks

This property enables communication between Data View and the Data Environment designer.

Visual Basic: Windows Controls

Visual Studio 6.0

EnsureVisible Method

See Also Example Applies To

Ensures a specified **ListItem** or **Node** object is visible. If necessary, this method expands **Node** objects and scrolls the **TreeView** control. The method only scrolls the **ListView** control.

Syntax

object.**EnsureVisible**

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

Return Values

Value	Description
True	The method returns **True** if the **ListView** or **TreeView** control must scroll and/or expand to expose the object.
False	The method returns **False** if no scrolling and/or expansion is required.

Remarks

Use the **EnsureVisible** method when you want a particular **Node** or **ListItem** object, which might be hidden deep in a **TreeView** or **ListView** control, to be visible.

The method will not operate on a **TreeView** control if the **Scroll** property is set to **False**.

Visual Basic: Windows Controls

EnsureVisible Method Example

This example adds many nodes to a **TreeView** control, and uses the **EnsureVisible** method to scroll and expand the tree. 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 to see the **TreeView** expand.

```
Private Sub Form_Load()
   Dim nodX As Node
   Dim i as Integer
   TreeView1.BorderStyle = vbFixedSingle ' Show borders.

   Set nodX = TreeView1.Nodes.Add(,,,"Root")    ' Add first node.
   For i = 1 to 15    ' Add 15 nodes
      Set nodX = TreeView1.Nodes.Add(i,,,"Node " & CStr(i))
   Next i

   Set nodX = TreeView1.Nodes.Add(,,,"Bottom")    ' Add one with text.
   Set nodX = TreeView1.Nodes.Add(i,,,"Expanded") ' Add child to node.
   Set nodX = TreeView1.Nodes.Add(i+1,,,"Show me") ' Add a final child.
End Sub

Private Sub Form_Click()
   ' Tree will scroll and expand when you click the form.
   TreeView1.Nodes(TreeView1.Nodes.Count).EnsureVisible
End Sub
```

Visual Basic: RDO Data Control

Visual Studio 6.0

EstablishConnection Method (Remote Data)

See Also Example Applies To

Establishes a physical connection to an ODBC server.

Syntax

object.**EstablishConnection** *prompt*, *readonly*, *options*

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

Part	Description
object	An object expression that evaluates to an **rdoConnection** object.
prompt	Optional. Integer value indicating ODBC prompting characteristic (see the **OpenConnection** method on the **rdoEnvironment** object).
readonly	Optional. Boolean value which is **True** if intending to use connection as read-only.
options	Optional. Integer value indicating connection options. This parameter has the same rules, restrictions and possible values that it does in the **OpenConnection** method of the **rdoEnvironment** object.

Remarks

This method causes the **rdoConnection** object to physically connect to the server, if it is not so already. This method is used when creating stand-alone **rdoConnection** objects or when re-connecting **rdoConnection** objects that have been disconnected using the **Close** method.

Unlike the **OpenConnection** method, the **EstablishConnection** method does *not* automatically append the **rdoConnection** object to the **rdoConnections** collection. If you want to add the newly established connection into the **rdoConnections** collection, you must use the **Add** method. You can use the **Remove** method to remove a member from the **rdoConnections** collection.

When using the Client Batch cursor library, the **EstablishConnection** method can be used to establish a connection once the **ActiveConnection** of an **rdoResultset** or **rdoQuery** object has been set to **Nothing**.

Just as with the **OpenConnection** method, the ***prompt*** argument dictates how the ODBC driver manager prompts the user for missing arguments needed to establish the connection. You can also request that the connection be made asynchronously by using the **rdAsyncEnable** option.

In general, you must set the **Connect** property and other appropriate properties of the **rdoConnection** object prior to making an attempt at connecting to a remote server.

See the **OpenConnection** method for details on how the **rdoConnection** properties should be set prior to attempting to use the **EstablishConnection** method.

Visual Basic: RDO Data Control

Connect Property Example: DSN Connection Using Establish Connection

The following example establishes an ODBC connection using a registered DSN to provide most of the required arguments. The User ID and Password are to be provided by domain-managed security. In this case the example prints the resulting **Connect** property to the Immediate window.

```
Dim cn As New rdoConnection
Dim qd As New rdoQuery

cn.Connect = "uid=;pwd=;"DSN=WorkDB;"
cn.cursordriver = rdUseOdbc
cn.EstablishConnection rdDriverNoprompt
debug.print cn.Connect
```

Visual Basic: RDO Data Control

Visual Studio 6.0

Execute Method (Remote Data)

See Also Example Applies To

Runs an action query or executes an SQL statement that does not return rows.

Syntax

connection.**Execute** *source*[, *options*]

query.**Execute** [*options*]

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

Part	Description
connection	An object expression that evaluates to the **rdoConnection** object on which the query will run.
query	An object expression that evaluates to the **rdoQuery** object whose **SQL** property setting specifies the SQL statement to execute.
source	A string expression that contains the action query to execute or the name of an **rdoQuery**.
options	A Variant or constant that determines how the query is run, as specified in Settings.

Settings

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

It is recommended that you use the **Execute** method only for action queries. Because an action query doesnt return any rows, Execute doesnt return an **rdoResultset**. You can use the **Execute** method on queries that execute multiple statements,

but none of these batched statements should return rows. To execute multiple result set queries that are a combination of action and SELECT queries, use the **OpenResultset** method.

Use the **RowsAffected** property of the **rdoConnection** or **rdoQuery** object to determine the number of rows affected by the most recent **Execute** method. **RowsAffected** contains the number of rows deleted, updated, or inserted when executing an action query. When you use the **Execute** method to run an **rdoQuery**, the **RowsAffected** property of the **rdoQuery** object is set to the number of rows affected.

Options

To execute the query asynchronously, use the **rdAsyncEnable** option (which is set by default). If set, the data source query processor immediately begins to process the query and returns to your application before the query is complete. Use the **StillExecuting** property to determine when the query processor is ready to return the results from the query. Use the **Cancel** method to terminate processing of an asynchronous query.

To bypass creation of a temporary stored procedure to execute the query, use the **rdExecDirect** option. This option is required when the query contains references to transactions or temporary tables that only exist in the context of a single operation. For example, if you include a Begin Transaction TSQL statement in your query or reference a temporary table, you must use **rdExecDirect** to ensure that the remote engine is not confused when these objects are left pending at the end of the query.

While it is possible to execute stored procedures using the **Execute** method, it is not recommended because the procedures return value and output parameters are discarded and the procedure cannot return rows. Use the **OpenResultset** method against an **rdoQuery** to execute stored procedures.

Note When executing stored procedures that do not require parameters, do not include the parenthesis in the SQL statement. For example, to execute the "MySP" procedure use the following syntax: `{Call MySP }`.

Also, a call like:

```
rCn.Execute SqlStatement, rdAsyncEnable +
rdExecDirect
```

allows only one outstanding request and allows Visual Basic code to overlap with SQL Server processing, but doesn't allow multiple outstanding SQL Server requests.

Visual Basic: RDO Data Control

Execute Method Example

This example illustrates use of the **Execute** method to execute SQL queries against a remote data source. These action queries do not return rows, but in some cases do return the number of rows affected in the **RowsAffected** property. The example creates a work table called TestData, inserts a few rows of data in the table and proceeds to run a DELETE query against the table. Notice that the delete queries have their own embedded transaction management. Because of this, you must use the **rdExecDirect** option to prevent the creation of stored procedures which negate the use of query-provided transactions.

```
Option Explicit
Dim er As rdoError
Dim cn As New rdoConnection
Dim qy As New rdoQuery
Dim rs As rdoResultset
Dim col As rdoColumn
Dim SQL As String

Private Sub DropRows_Click()
Dim SQL As String, Ans As Integer

SQL = "Begin Transaction Delete TestData " _
   & " Where State = " & StateWanted & ""
cn.Execute SQL, rdExecDirect
Ans = MsgBox("Ok to delete these "  _
   & cn.RowsAffected & " rows?", vbOKCancel)
If Ans = vbOK Then
   cn.Execute "Commit Transaction", rdExecDirect
Else
   cn.Execute "Rollback Transaction", rdExecDirect
End If
Exit Sub
End Sub

Private Sub Form_Load()
cn.CursorDriver = rdUseOdbc
cn.Connect = "uid=;pwd=;server=sequel;" _
   & "driver={SQL Server};" _
   & "database=pubs;dsn=;"
cn.EstablishConnection
With qy
   .Name = "TestList"
   .SQL = "Select * from TestData Where State = ?"
   .RowsetSize = 1
   Set .ActiveConnection = cn
End With
SQL = "Drop Table TestData"
cn.Execute SQL

SQL = " CREATE TABLE TestData " _
   & " (ID integer identity NOT NULL, " _
   & " PName char(10) NULL," _
   & " State Char(2) NULL) " _
   & " CREATE UNIQUE INDEX " _
```

```
   &   "TestDataIndex on TestData(ID)"

cn.Execute SQL
SQL = "Insert TestData (PName,State)  " _
   & "Values('Bob', 'CA')" _
   & " Insert TestData (PName,State) " _
   & " Values('Bill', 'WA')" _
   & " Insert TestData (PName,State) " _
   & " Values('Fred', 'WA') _
   & " Insert TestData (PName,State) " _
   & " Values('George', 'CA')" _
   & " Insert TestData (PName,State) " _
   & " Values('Sam', 'TX')" _
   & " Insert TestData (PName,State) " _
   & " Values('Marilyn', 'TX')"
cn.Execute SQL
Debug.Print cn.RowsAffected
' This returns 1
'(The last INSERT statement affected 1 row)
End Sub

Private Sub SeekRows_Click()
qy(0) = StateWanted
Set rs = qy.OpenResultset(rdOpenForwardOnly, _
rdConcurReadOnly)
List1.Clear
If rs.EOF Then
   MsgBox "No hits for that state"
Exit Sub
End If
Do Until rs.EOF
   List1.AddItem rs!PName & " - " & rs!state
   rs.MoveNext
Loop
End Sub
```

Visual Basic: Internet Control

Visual Studio 6.0

Execute Method

See Also Example Applies To

Executes a request to a remote server. You can only send requests which are valid for the particular protocol.

Syntax

object.**Execute** *url, operation, data, requestHeaders*

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
url	Optional. String that specifies the URL to which the control should connect. If no URL is specified here, the URL specified in the **URL** property will be used.
operation	Optional. String that specifies the type of operation to be executed. See Settings below for a list of supported operations.
data	Optional. String that specifies the data for operations (See Settings below.)
requestHeaders	Optional. String that specifies additional headers to be sent from the remote server. The format for these is: `header name: header value vbCrLf`

Settings

Note Valid settings for *operation* are determined by the protocol being used. The tables below are organized by protocol.

Supported HTTP commands

Valid settings for *operation* are:

Operation	Description
GET	Retrieve data from the URL specified in the **URL** property.
HEAD	Sends the Request headers.

POST	Posts data to the server. The data is located in the *data* argument. This is an alternate method to **GET**, for which additional instructions are specified in the *data* argument.
PUT	Put operation. The name of the page to be replaced is located in the *data* argument.

Supported FTP commands

Important The FTP protocol uses a single string that includes the operation name and any other parameters needed by the operation. In other words, the *data* and *requestHeaders* arguments are not used; all of the operations and their parameters are passed as a single string in the *operation* argument. Parameters are separated by a space. In the descriptions below, do not confuse the terms "file1" and "file2" with the *data* and *requestHeaders* arguments.

The syntax for FTP operations is:

operationName file1 file2.

For example, to get a file, the following code invokes the **Execute** method, which includes the operation name ("GET"), and the two file names required by the operation:

```
Inet1.Execute "FTP://ftp.microsoft.com", _
"GET Disclaimer.txt C:\Temp\Disclaimer.txt"
```

Note File names that include embedded spaces are not supported.

Valid FTP settings for *operation* are:

Operation	Description
CD *file1*	Change Directory. Changes to the directory specified in *file1*.
CDUP	Change to parent directory. Equivalent to "CD.."
CLOSE	Closes the current FTP connection.
DELETE *file1*	Deletes the file specified in *file1*.
DIR *file1*	Directory. Searches the directory specified in *file1*. (Wildcards are permitted but the remote host dictates the syntax.) If no file1 is specified, a full directory of the current working directory is returned. Use the **GetChunk** method to return the directory data.
GET *file1 file2*	Retrieves the remote file specified in *file1*, and creates a new local file specified in *file2*.
LS *file1*	List. Searches the directory specified in *file1*. (Wildcards are permitted but the remote host dictates the syntax.) Use the **GetChunk** method to return the file directory data.
MKDIR *file1*	Make Directory. Creates a directory as specified in *file1*. Success is dependent on user privileges on the remote host.
PUT *file1 file2*	Copies a local file specified in *file1* to the remote host specified in *file2*.
PWD	Print Working Directory. Returns the current directory name. Use the **GetChunk** method to return the data.

QUIT	Terminates the current user.
RECV *file1 file2*	Retrieves the remote file specified in *file1*, and creates a new local file specified in *file2*. Equivalent to **GET**.
RENAME *file1 file2*	Renames the remote file named in *file1* to the new name specified in *file2*. Success is dependent on user privileges on the remote host.
RMDIR *file1*	Remove Directory. Removes the remote directory specified in *file1*. Success is dependent on user privileges on the remote host.
SEND *file1 file2*	Copies a local file, specified in *file1*, to the remote host, specified in *file2*. Equivalent to **PUT**.
SIZE *file1*	Returns the size of the directory specified in *file1*.

Return Type

None

Remarks

Many commands listed above can be carried out only if the user has privileges on the host server. For example, anonymous FTP sites will not allow anyone to delete files or directories.

Visual Basic: Internet Control

Execute Method Example

The example shows a series of common FTP operations using the **Execute** method. The example assumes that three **TextBox** controls exist on the form. The first, **txtURL** contains the URL of the FTP server. The second, **txtRemotePath**, contains additional information needed by the particular command. The third, **txtResponse**, contains the response of the server.

```
Private Sub cmdChangeDirectory_Click()
   ' Change directory to txtRemotePath.
   Inet1.Execute txtURL.Text, "CD " & _
   txtRemotePath.Text
End Sub

Private Sub cmdDELETE_Click()
   ' Delete the directory in txtRemotePath.
   Inet1.Execute txtURL.Text, "DELETE " & _
   txtRemotePath.Text
End Sub

Private Sub cmdDIR_Click()
   Inet1.Execute txtURL.Text, "DIR FindThis.txt"
End Sub

Private Sub cmdGET_Click()
   Inet1.Execute txtURL.Text, _
   "GET GetThis.txt C:\MyDocuments\GotThis.txt"
End Sub

Private Sub cmdSEND_Click()
   Inet1.Execute txtURL.Text, _
   "SEND C:\MyDocuments\Send.txt SentDocs\Sent.txt"
End Sub

Private Sub Inet1_StateChanged(ByVal State As Integer)
   ' Retrieve server response using the GetChunk
   ' method when State = 12.

   Dim vtData As Variant ' Data variable.
   Select Case State
   ' ... Other cases not shown.
   Case icError ' 11
      ' In case of error, return ResponseCode and
      ' ResponseInfo.
      vtData = Inet1.ResponseCode & ":" & _
      Inet1.ResponseInfo
   Case icResponseCompleted ' 12
      Dim vtData As Variant
      Dim strData As String
      Dim bDone As Boolean: bDone = False

      ' Get first chunk.
      vtData = Inet1.GetChunk(1024, icString)
      DoEvents
```

```
      Do While Not bDone
         strData = strData & vtData
         ' Get next chunk.
         vtData = Inet1.GetChunk(1024, icString)
         DoEvents

         If Len(vtData) = 0 Then
            bDone = True
         End If
      Loop
      txtData.Text = strData
   End Select

End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

Exists Method

See Also Example Applies To Specifics

Description

Returns **True** if a specified key exists in the **Dictionary** object; **False** if it does not.

Syntax

object.**Exists(***key***)**

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

Part	Description
object	Required. Always the name of a **Dictionary** object.
key	Required. *Key* value being searched for in the **Dictionary** object.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ExpandAll Method (MSHFlexGrid)

SeeAlso Example Applies To

Expands all rows of the specified band within the **MSHFlexGrid**.

Syntax

object.**ExpandAll(***number***)**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
number	Optional. A Long value that specifies the band that contains the rows to expand. If not specified, the default value is 1.

Visual Basic Extensibility Reference

Visual Studio 6.0

Export Method (VBA Add-In Object Model)

See Also Example Applies To Specifics

Saves a component as a separate file or files.

Syntax

object.**Export(***filename***)**

The **Export** syntax has these parts:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
filename	Required. A String specifying the name of the file that you want to export the component to.

Remarks

When you use the **Export** method to save a component as a separate file or files, use a file name that doesn't already exist; otherwise, an error occurs.

Visual Basic Extensibility Reference

Export Method Example

The following example creates a file named `test.bas` and uses the **Export** method to copy the contents of the `VBComponents(1)` code module into the file.

```
Application.VBE.ActiveVBProject.VBComponents(1).Export("test.bas")
```

Visual Basic Reference

Visual Studio 6.0

ExportReport Method

See Also Example Applies To

Exports the text of a report to a file using a specified **ExportFormat** object. Images and shapes cannot be exported.

Syntax

object.**ExportReport(***ExportFormat*, *filename*, *Overwrite*, *ShowDialog*, *Range*, *PageFrom*, *PageTo***)**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
ExportFormat	Optional. The **ExportFormat** object to be used. The argument can be an object reference, or a string key that specifies one of the members of the **ExportFormats** collection as shown in Settings. If not specified, the Export dialog box will display.
filename	Optional. A string expression that evaluates to the name of the file. If not specified, the Export dialog box will display.
overwrite	Optional. A boolean expression that determines if the file will be overwritten.
showDialog	Optional. A boolean expression that determines if the Save As dialog box will be displayed. If no **ExportFormat** object or *filename* is specified, the Export dialog box will be displayed, even if this argument is set to **False**.
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. A numeric expression that specifies the page where the export will start.
PageTo	Optional. A numeric expression that specifies the page where the export will end.

Settings

The settings for *ExportFormat* are:

Constant	Value	Description

rptKeyHTML	key_def_HTML	Specifies the HTML default member of the ExportFormats collection.
rptKeyUnicodeHTML	key_def_UnicodeHTML_UTF8	Specifies the Unicode HTML default member of the ExportFormats collection
rptKeyText	key_def_Text	Specifies the text default member of the ExportFormats collection
rptKeyUnicodeText	key_def_UnicodeText	Specifies the Unicode text default member of the ExportFormats collection

When adding an **ExportFormat** object to the **ExportFormats** collection, you must specify a *Key* for the object. That key can be used in the *ExportFormat* argument.

The settings for *Range* are:

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

Return Value

Long

Remarks

If all necessary arguments are not supplied with the method, a dialog box is displayed, prompting the user for appropriate information (such as filename).

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

Important The range of pages specified will not match the pages seen in the Print Preview mode. Whereas export page numbers are based on font attributes of the **ExportFormat** object's **ExportType** property, print and preview pages are based on the current printer object used by the computer.

Visual Basic Reference

ExportReport Method Example

The first example uses the **ExportReport** method to display the Export dialog box. The second example exports the file without displaying the Export dialog box. The third example specifies an **ExportFormat** object to use when exporting the report.

```
Private Sub ExportTheReport()
   DataReport1.ExportReport , , True, True
End Sub

Private Sub ExportWithoutDialog()
   ' Export to a file named Output.htm, overwriting if needed.
   DataReport1.ExportReport rptKeyHTML, "C:\Temp\Output", True, False
End Sub

Private Sub ExportMyReport()
   ' Export to a file named Daily.htm using the MyReport ExportFormat.
   DataReport1.ExportReport "MyReport", "C:\Temp\Daily", True, False
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

ExtractIcon Method

See Also Example Applies To

Creates an icon from a bitmap in a **ListImage** object of an **ImageList** control and returns a reference to the newly created icon.

Syntax

object.**ExtractIcon**

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

Remarks

You can use the icon created with the **ExtractIcon** method like any other icon. For example, you can use it as a setting for the **MouseIcon** property, as the following code illustrates:

```
Set Command1.MouseIcon = ImageList1.ListImages(1).ExtractIcon
```

Visual Basic: Windows Controls

ExtractIcon Method Example

This example loads a bitmap into an **ImageList** control. When the user clicks the form, the **ExtractIcon** method is used to create an icon from the bitmap, and that icon is used as a setting in the **Form** object's **MouseIcon** property. To try the example, place an **ImageList** 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 imgX As ListImage
   Set imgX = ImageList1.ListImages. _
   Add(, , LoadPicture("bitmaps\assorted\balloon.bmp"))
End Sub

Private Sub Form_Click()
   Dim picX As Picture
   Set picX = ImageList1.ListImages(1).ExtractIcon  ' Make an icon.

   With Form1
   .MouseIcon =  picX   ' Set new icon.
   .MousePointer = vbCustom    ' Set to custom icon.
   End With
End Sub
```

Visual Basic: MAPI Controls

Visual Studio 6.0

Fetch Method

See Also Example Applies To

Creates a message set from selected messages in the Inbox.

Syntax

object.**Fetch**

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

Remarks

The message set includes all messages in the Inbox which are of the types specified by the **FetchMsgType** property. They are sorted as specified by the **FetchSorted** property. If the **FetchUnreadOnly** property is set to **True**, only unread messages are included in the message set.

Any attachment files in the read buffer are deleted when a subsequent fetch action occurs.

Visual Basic Reference

Visual Studio 6.0

FetchVerbs Method (ActiveX Controls)

See Also Example Applies To

Updates the list of verbs an object supports.

Syntax

object.**FetchVerbs**

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

Remarks

You can read the updated list of verbs using the **ObjectVerbs** property.

Visual Basic Reference

Visual Studio 6.0

FetchVerbs Method

See Also Example Applies To

Updates the list of verbs an object supports.

Syntax

object.**FetchVerbs**

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

Remarks

You can read the updated list of verbs using the **ObjectVerbs** property.

Visual Basic for Applications Reference

Visual Studio 6.0

FileExists Method

See Also Example Applies To Specifics

Description

Returns **True** if a specified file exists; **False** if it does not.

Syntax

object.**FileExists(***filespec***)**

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

Part	Description
object	Required. Always the name of a **FileSystemObject**.
filespec	Required. The name of the file whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the file isn't expected to exist in the current folder.

Visual Basic Reference

Visual Studio 6.0

Files Method (ActiveX Controls)

See Also Example Applies To

Returns a collection of filenames used by the vbCFFiles format (a **DataObjectFiles** collection) which in turn contains a list of all filenames used by a **DataObject** object; for example, the names of files that a user drags to or from the Windows File Explorer.

Syntax

object.**Files(***index***)**

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

Part	Description
object	An object expression that evaluates to a **DataObject** object.
index	An integer which is an index to an array of filenames.

Remarks

The **Files** collection is filled with filenames only when the **DataObject** object contains data of type **vbCFFiles**. The **DataObject** object can contain several different types of data. You can iterate through the collection to retrieve the list of file names.

The **Files** collection can be filled to allow Visual Basic applications to act as a drag source for a list of files.

Visual Basic Reference

Visual Studio 6.0

Files Method

See Also Example Applies To

Returns a collection of filenames used by the vbCFFiles format (a **DataObjectFiles** collection) which in turn contains a list of all filenames used by a **DataObject** object; for example, the names of files that a user drags to or from the Windows File Explorer.

Syntax

object.**Files(***index***)**

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

Part	Description
object	An object expression that evaluates to a **DataObject** object.
index	An integer which is an index to an array of filenames.

Remarks

The **Files** collection is filled with filenames only when the **DataObject** object contains data of type **vbCFFiles**. The **DataObject** object can contain several different types of data. You can iterate through the collection to retrieve the list of file names.

The **Files** collection can be filled to allow Visual Basic applications to act as a drag source for a list of files.

Visual Basic: RichTextBox Control

Visual Studio 6.0

Find Method

See Also Example Applies To

Searches the text in a **RichTextBox** control for a given string.

Syntax

object.**Find**(*string, start, end, options*)

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

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
string	Required. A string expression you want to find in the control.
start	Optional. An integer character index that determines where to begin the search. Each character in the control has an integer index that uniquely identifies it. The first character of text in the control has an index of 0.
end	Optional. An integer character index that determines where to end the search.
options	Optional. One or more constants used to specify optional features, as described in Settings.

Settings

The setting for *options* can include:

Constant	Value	Description
rtfWholeWord	2	Determines if a match is based on a whole word or a fragment of a word.
rtfMatchCase	4	Determines if a match is based on the case of the specified string as well as the text of the string.
rtfNoHighlight	8	Determines if a match appears highlighted in the **RichTextBox** control.

You can combine multiple options by using the **Or** operator.

Remarks

If the text searched for is found, the **Find** method highlights the specified text and returns the index of the first character highlighted. If the specified text is not found, the **Find** method returns 1.

If you use the **Find** method without the **rtfNoHighlight** option while the **HideSelection** property is **True** and the **RichTextBox** control does not have the focus, the control still highlights the found text. Subsequent uses of the **Find** method will search only for the highlighted text until the insertion point moves.

The search behavior of the **Find** method varies based on the combination of values specified for the *start* and *end* arguments. This table describes the possible behaviors:

Start	End	Search Behavior
Specified	Specified	Searches from the specified start location to the specified end location.
Specified	Omitted	Searches from the specified start location to the end of the text in the control.
Omitted	Specified	Searches from the current insertion point to the specified end location.
Omitted	Omitted	Searches the current selection if text is selected or the entire contents of the control if no text is selected.

Visual Basic: RichTextBox Control

Find Method Example

This example finds a string in a **RichTextBox** control based on a word entered in a **TextBox** control. After it finds the specified string, it displays a message box that shows the number of the line containing the specified word. To try this example, put a **RichTextBox** control, a **CommandButton** control and a **TextBox** control on a form. Load a file into the **RichTextBox,** and paste this code into the General Declarations section of the form. Then run the example, enter a word in the **TextBox**, and click the **CommandButton**.

```
Private Sub Command1_Click()
   Dim FoundPos As Integer
   Dim FoundLine As Integer
   ' Find the text specified in the TextBox control.
   FoundPos = RichTextBox1.Find(Text1.Text, , , rtfWholeWord)

   ' Show message based on whether the text was found or not.

   If FoundPos <> -1 Then
      ' Returns number of line containing found text.
      FoundLine = RichTextBox1.GetLineFromChar(FoundPos)
      MsgBox "Word found on line " & CStr(FoundLine)
   Else
      MsgBox "Word not found."
   End If
End Sub
```

Visual Basic Extensibility Reference

Visual Studio 6.0

Find Method (VBA Add-In Object Model)

See Also Example Applies To Specifics

Searches the active module for a specified string.

Syntax

object.**Find(***target*, *startline*, *startcol*, *endline*, *endcol* [, *wholeword*] [, *matchcase*] [, *patternsearch*]**) As Boolean**

The **Find** syntax has these parts:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
target	Required. A String containing the text or pattern you want to find.
startline	Required. A Long specifying the line at which you want to start the search; will be set to the line of the match if one is found. The first line is number 1.
startcol	Required. A **Long** specifying the column at which you want to start the search; will be set to the column containing the match if one is found. The first column is 1.
endline	Required. A **Long** specifying the last line of the match if one is found. The last line may be specified as 1.
endcol	Required. A **Long** specifying the last line of the match if one is found. The last column may be designated as 1.
wholeword	Optional. A Boolean value specifying whether to only match whole words. If **True**, only matches whole words. **False** is the default.
matchcase	Optional. A **Boolean** value specifying whether to match case. If **True**, the search is case sensitive. **False** is the default.
patternsearch	Optional. A **Boolean** value specifying whether or not the target string is a regular expression pattern. If **True**, the target string is a regular expression pattern. **False** is the default.

Remarks

Find returns **True** if a match is found and **False** if a match isn't found.

The *matchcase* and *patternsearch* arguments are mutually exclusive; if both arguments are passed as **True**, an error occurs.

The content of the **Find** dialog box isn't affected by the **Find** method.

The specified range of lines and columns is inclusive, so a search can find the pattern on the specified last line if *endcol* is supplied as either 1 or the length of the line.

Visual Basic Extensibility Reference

Find Method Example

The following example uses the **Find** method to verify that the specified block of lines, lines 1261 through 1279, of a particular code pane does contain the string "Tabs.Clear."

```
Application.VBE.CodePanes(2).CodeModule.Find ("Tabs.Clear", 1261, 1, 1280, 1, False, False)
```

Visual Basic: Windows Controls

Visual Studio 6.0

FindItem Method (ListView Control)

See Also Example Applies To

Finds and returns a reference to a **ListItem** object in a **ListView** control.

Syntax

object.**FindItem** (*string, value, index, match*)

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

Part	Description
object	Required. An object expression that evaluates to a **ListView** control.
string	Required. A string expression indicating the **ListItem** object to be found.
value	Optional. An integer or constant specifying whether the string will be matched to the **ListItem** object's **Text**, **Subitems**, or **Tag** property, as described in Settings.
index	Optional. An integer or string that uniquely identifies a member of an object collection and specifies the location from which to begin the search. The integer is the value of the **Index** property; the string is the value of the **Key** property. If no index is specified, the default is 1.
match	Optional. An integer or constant specifying that a match will occur if the item's **Text** property is the same as the string, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
lvwText	0	(Default) Matches the string with a **ListItem** object's **Text** property.
lvwSubitem	1	Matches the string with any string in a **ListItem** object's **SubItems** property.
lvwTag	2	Matches the string with any **ListItem** object's **Tag** property.

The settings for *match* are:

Constant	Value	Description
lvwWholeWord	0	(Default) An integer or constant specifying that a match will occur if the item's **Text** property begins with the whole word being searched. Ignored if the criteria is not text.
lvwPartial	1	An integer or constant specifying that a match will occur if the item's **Text** property begins with the string being searched. Ignored if the criteria is not text.

Remarks

If you specify Text as the search criteria, you can use **lvwPartial** so that a match occurs when the **ListItem** object's **Text** property begins with the string you are searching for. For example, to find the **ListItem** whose text is "Autoexec.bat", use:

```
'Create a ListItem variable.
Dim itmX As ListItem
'Set the variable to the found item.
Set itmX = ListView1.FindItem("Auto",,,lvwpartial)
```

Visual Basic: Windows Controls

FindItem Method Example

This example populates a **ListView** control with the contents of the Publishers table of the Biblio.mdb database. A **ComboBox** control is also populated with three options for the **FindItem** method. A **CommandButton** contains the code for the **FindItem** method; when you click on the button, you are prompted to enter the string to search for, and the **FindItem** method searches the **ListView** control for the string. If the string is found, the control is scrolled using the **EnsureVisible** method to show the found **ListItem** object. To try the example, place a **ListView**, **ComboBox**, and a **CommandButton** control on a form and paste the code into the form's Declarations section. Run the example and click on the command button.

Note The example will not run unless you add a reference to the Microsoft DAO 3.51 Object Library by using the References command from the Tools menu.

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

   ListView1.BorderStyle = ccFixedSingle  ' Set BorderStyle property.
   ListView1.View = lvwReport  ' Set View property to Report.
   Command1.Caption = "&FindItem"

   ' Label OptionButton controls with FindItem options.
      Option1(0).Caption = "Text"
      Option1(1).Caption = "SubItem"
      Option1(2).Caption = "Tag"
      ListView1.FindItem = 0 ' Set the ListView FindItem property to Text.
   End With

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

   ' While the record is not the last record, add a ListItem object.
   ' Use the reference to the new object to set properties.
   ' Set the Text property to the Name field (myRS!Name).
   ' Set SubItem(1) to the Address field (myRS!Address).
   ' Set SubItem(7) to the Phone field (myRS!Telephone).

   While Not myRs.EOF
      Dim itmX As ListItem   ' A ListItem variable.
      Dim intCount As Integer   ' A counter variable.
```

```
      ' Use the Add method to add a new ListItem and set an object
      ' variable to the new reference. Use the reference to set
      ' properties.
      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))
      intCount = intCount + 1 ' Increment counter for the Tag property.
      itmX.Tag = "ListItem " & intCount  ' Set Tag with counter.

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

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

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

Private Sub Command1_Click()
   ' FindItem method.
   'Create an integer variable called intSelectedOption
   ' to store the index of the selected button
   ' Create a string variable called strFindMe. Use the InputBox
   ' to store the string to be found in the variable. Use the
   ' FindItem method to find the string. Option1 is used to
   ' switch the FindItem argument that determines where to look.

   Dim intSelectedOption as Integer
   Dim strFindMe As String
   If Option1(0).Value = True then
      strFindMe = InputBox("Find in " & Option1(0).Caption)
      intSelectedOption = lvwText
   End If
   If Option1(1).Value = True then
      strFindMe = InputBox("Find in " & Option1(1).Caption)
      intSelectedOption = lvwSubItem
   End If
   If Option1(2).Value = True then
      strFindMe = InputBox("Find in " & Option1(2).Caption)
      intSelectedOption = lvwTag
   End If

   ' FindItem method returns a reference to the found item, so
   ' you must create an object variable and set the found item
   ' to it.
   Dim itmFound As ListItem   ' FoundItem variable.

   Set itmFound = ListView1. _
   FindItem(strFindMe, intSelectedOption, , lvwPartial)

   ' If no ListItem is found, then inform user and exit. If a
   ' ListItem is found, scroll the control using the EnsureVisible
   ' method, and select the ListItem.
   If itmFound Is Nothing Then  ' If no match, inform user and exit.
      MsgBox "No match found"
      Exit Sub
   Else
       itmFound.EnsureVisible ' Scroll ListView to show found ListItem.
```

```
        itmFound.Selected = True    ' Select the ListItem.
       ' Return focus to the control to see selection.
        ListView1.SetFocus
    End If
End Sub

Private Sub ListView1_LostFocus()
    ' After the control loses focus, reset the Selected property
    ' of each ListItem to False.
    Dim i As Integer
    For i = 1 to ListView1.ListItems.Count
       ListView1.ListItems.Item(i).Selected = False
    Next i
End Sub
```

Visual Basic for Applications Reference

Visual Studio 6.0

FolderExists Method

See Also Example Applies To Specifics

Description

Returns **True** if a specified folder exists; **False** if it does not.

Syntax

object.**FolderExists(***folderspec*)

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

Part	Description
object	Required. Always the name of a **FileSystemObject**.
folderspec	Required. The name of the folder whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the folder isn't expected to exist in the current folder.

Visual Basic: MAPI Controls

Visual Studio 6.0

Forward Method

See Also Example Applies To

Forwards a message.

Syntax

object.**Forward**

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

Remarks

This method copies the currently indexed message to the compose buffer as a forwarded message and adds **FW:** to the beginning of the Subject line. It also sets the **MsgIndex** property to -1.