Visual Basic: MAPI Controls

Visual Studio 6.0

MAPIMessages Control

See Also Example Properties Methods Events

The messaging application program interface (MAPI) controls allow you to create mail-enabled Visual Basic MAPI applications. There are two MAPI controls:

- MAPISession
- MAPIMessages

The **MAPISession** control signs on and establishes a MAPI session. It is also used to sign off from a MAPI session. The **MAPIMessages** control allows the user to perform a variety of messaging system functions.

The MAPI controls are invisible at run time. In addition, there are no events for the controls. To use them, you must specify the appropriate methods.

The **MAPIMessages** control performs a variety of messaging system functions after a messaging session is established with the **MAPISession** control.

For these controls to work, MAPI services must be present. MAPI services are provided in MAPI compliant electronic mail systems.

Note If you attempt to run a program that uses the MAPI controls, make sure that you have the 32-bit MAPI DLLs installed properly or you may not be able to perform simple MAPI functions such as SignOn. For example, on Windows 95 or later, you must install Mail during the operating system setup, or install it separately from the control panel to correctly use MAPI functions or MAPI custom controls from Visual Basic.

Syntax

```
MAPIMessages
```

Remarks

With the MAPIMessages control, you can:

- Access messages currently in the Inbox.
- Compose a new message.
- Add and delete message recipients and attachments.
- Send messages (with or without a supporting user interface).
- Save, copy, and delete messages.

- Display the Address Book dialog box.
- Display the Details dialog box.
- Access attachments, including Object Linking and Embedding (OLE) attachments.
- Resolve a recipient name during addressing.
- Perform reply, reply-all, and forward actions on messages.

Most of the properties of the **MAPIMessages** control can be categorized into four functional areas: address book, file attachment, message, and recipient properties. The file attachment, message, and recipient properties are controlled by the **AttachmentIndex**, **MsgIndex**, and **RecipIndex** properties, respectively.

Message Buffers

When using the **MAPIMessages** control, you need to keep track of two buffers, the *compose buffer* and the *read buffer*. The read buffer is made up of an indexed set of messages fetched from a user's Inbox. The **MsgIndex** property is used to access individual messages within this set, starting with a value of 0 for the first message and incrementing by one for each message through the end of the set.

The message set is built using the **Fetch** method. The set includes all messages of type **FetchMsgType** and is sorted as specified by the **FetchSorted** property. Previously read messages can be included or left out of the message set with the **FetchUnreadOnly** property. Messages in the read buffer can't be altered by the user, but can be copied to the compose buffer for alteration.

Messages can be created or edited in the compose buffer. The compose buffer is the active buffer when the **MsgIndex** property is set to -1. Many of the messaging actions are valid only within the compose buffer, such as sending messages, sending messages with a dialog box, saving messages, or deleting recipients and attachments.

Refer to the object library in the Object Browser for property and error constants for the control.

Visual Basic: MAPI Controls

Visual Studio 6.0

MAPISession Control

See Also Example Properties Methods Events

The messaging application program interface (MAPI) controls allow you to create mail-enabled Visual Basic MAPI applications. There are two MAPI controls:

- MAPISession
- MAPIMessages

The **MAPISession** control signs on and establishes a MAPI session. It is also used to sign off from a MAPI session. The **MAPIMessages** control allows the user to perform a variety of messaging system functions.

Syntax

```
MAPISession
```

Remarks

After sign-on is successful, the **SessionID** property contains the handle to the MAPI session. The session handle must then be passed to the **MAPIMessages** control or an error results when using the **MAPIMessages** control.

The **MAPISession** control is invisible at run time. In addition, there are no events for the control. To use it, you must specify the appropriate properties and methods.

For these controls to work, MAPI services must be present. MAPI services are provided in MAPI compliant electronic mail systems.

Note If you attempt to run a program that uses the MAPI controls, make sure that you have the 32-bit MAPI DLLs installed properly or you may not be able to perform simple MAPI functions such as SignOn. For example, on Windows 95 or later, you must install Exchange during the operating system setup, or install it separately from the control panel to correctly use MAPI functions or MAPI custom controls from Visual Basic.

Visual Studio 6.0

Visual Basic: MSChart Control

Marker Object

See Also Example Properties Methods Events

A marker that identifies a data point on a chart.

Syntax

Marker

Visual Basic: MSChart Control

Marker Object Example

The following example sets a blue X marker style for a chart series.

```
Option Explicit
Option Base 1
Private Sub Command1_Click()
   ' Display Markers for Series 1.
   Dim i As Integer
   For i = 1 To MSChart1.Plot.SeriesCollection.Count
      With MSChart1.Plot.SeriesCollection _
      .Item(i).DataPoints.Item(-1).Marker
         .Visible = True
         .Size = 20
         .Style = VtMarkerStyleX
         .FillColor.Automatic = False
         .FillColor.Set 0, 0, 255
      End With
   Next i
End Sub
```

Visual Basic: MaskedEdit Control

Visual Studio 6.0

Masked Edit Control

See Also Example Properties Methods Events

The **Masked Edit** control provides restricted data input as well as formatted data output. This control supplies visual cues about the type of data being entered or displayed. This is what the control looks like as an icon in the Toolbox:

##|

File Name

MSMASK32.OCX

Class Name

MaskEdBox

Remarks

The **Masked Edit** control generally behaves as a standard text box control with enhancements for optional masked input and formatted output. If you don't use an input mask, the **Masked Edit** control behaves much like a standard text box, except for its dynamic data exchange (DDE) capability.

If you define an input mask using the **Mask** property, each character position in the **Masked Edit** control maps to either a placeholder of a specified type or a literal character. Literal characters, or *literals*, can give visual cues about the type of data being used. For example, the parentheses surrounding the area code of a telephone number are literals: `(206)`.

If you attempt to enter a character that conflicts with the input mask, the control generates a ValidationError event. The input mask prevents you from entering invalid characters into the control.

The **Masked Edit** control has three bound properties: **DataChanged**, **DataField**, and **DataSource**. This means that it can be linked to a data control and display field values for the current record in the recordset. The **Masked Edit** control can also write out values to the recordset.

When the value of the field referenced by the **DataField** property is read, it is converted to a **Text** property string, if possible. If the recordset is updatable, the string is converted to the data type of the field.

To clear the **Text** property when you have a mask defined, you first need to set the **Mask** property to an empty string, and then the **Text** property to an empty string:

```
MaskedEdit1.Mask = ""
MaskedEdit1.Text = ""
```

When you define an input mask, the **Masked Edit** control behaves differently from the standard text box. The insertion point automatically skips over literals as you enter data or move the insertion point.

When you insert or delete a character, all nonliteral characters to the right of the insertion point are shifted, as necessary. If shifting these characters leads to a validation error, the insertion or deletion is prevented, and a ValidationError event is triggered.

Suppose the **Mask** property is defined as "?###", and the current value of the **Text** property is "A12." If you attempt to insert the letter "B" to the left of the letter "A," the "A" would shift to the right. Since the second value of the input mask requires a number, the letter "A" would cause the control to generate a ValidationError event.

The **Masked Edit** control also validates the values of the **Text** property at run time. If you set the **Text** property so that it conflicts with the input mask, the control generates a run-time error.

You may select text in the same way as for a standard text box control. When selected text is deleted, the control attempts to shift the remaining characters to the left of the selection.

Normally, when a selection in the **Masked Edit** control is copied onto the Clipboard, the entire selection, including literals, is transferred onto the Clipboard. You can use the **ClipMode** property to transfer only user-entered data onto the Clipboard literal characters that are part of the input mask are not copied.

Visual Basic Reference

Visual Studio 6.0

MDIForm Object

See Also Example Properties Methods Events

An MDI (multiple-document interface) form is a window that acts as the background of an application and is the container for forms that have their **MDIChild** property set to **True**.

Syntax

MDIForm

Remarks

You create an **MDIForm** object by choosing MDI Form from the Insert menu.

An application can have only one **MDIForm** object but many MDI child forms. If an MDI child form has menus, the child form's menu bar automatically replaces the **MDIForm** object's menu bar when the MDI child form is active. A minimized MDI child form is displayed as an icon within the **MDIForm**.

An **MDIForm** object can contain only **Menu** and **PictureBox** controls and custom controls that have an **Align** property. To place other controls on an **MDIForm**, you can draw a picture box on the form, and then draw other controls inside the picture box. You can use the **Print** method to display text in a picture box on an **MDIForm**, but you can't use this method to display text on the **MDIForm** itself.

An **MDIForm** object can't be modal.

MDI child forms are designed independently of the **MDIForm**, but are always contained within the **MDIForm** at run time.

You can access the collection of controls on an **MDIForm** using the **Controls** collection. For example, to hide all the controls on an MDIForm you can use code similar to the following:

```
For Each Control in MDIForm1.Controls
   Control.Visible = False
Next Control
```

The **Count** property of the **MDIForm** tells you the number of controls in the **Controls** collection.

Visual Basic Extensibility Reference

Visual Studio 6.0

Member Object

See Also Example Properties Methods Events

The **Member** object represents a mixture of code-based properties and type library-based attributes of members.

Syntax

Member

Remarks

Code-based properties like **Name** are read-only, so the add-in must modify the code to change these properties.

Visual Basic Extensibility Reference

Visual Studio 6.0

Members Collection

See Also Example Properties Methods Events

Returns a collection of code module-level members.

Syntax

Members

Remarks

A member of a code module is an identifier that has module-level scope and which can be considered a property, method, or event of that code module.

Visual Basic Reference

Visual Studio 6.0

Menu Control

See Also Example Properties Methods Events

A **Menu** control displays a custom menu for your application. A menu can include commands, submenus, and separator bars. Each menu you create can have up to four levels of submenus.

Syntax

Menu

Remarks

To create a **Menu** control, use the Menu Editor. Enter the name of the **Menu** control in the Caption box. To create a separator bar, enter a single hyphen (-) in the Caption box. To display a check mark to the left of a menu item, select the Checked box.

While you can set some **Menu** control properties using the Menu Editor, all **Menu** control properties are displayed in the Properties window. To display the properties of a **Menu** control, select the menu name in the Objects list at the top of the Properties window.

When you create an MDI application, the menu bar on the MDI child form replaces the menu bar on the **MDIForm** object when the child form is active.

Visual Basic: Internet Control

Visual Studio 6.0

Microsoft Internet Transfer Control

See Also Example Properties Methods Events

The **Internet Transfer** control provides implementation of two of the most widely used protocols on the Internet, HyperText Transfer Protocol (HTTP) and File Transfer Protocol (FTP).

Using the HTTP protocol, you can connect to World Wide Web servers to retrieve HTML documents. With the FTP protocol, you can log on to FTP servers to download and upload files. The **UserName** and **Password** properties allow you to log on to private servers that require authentication. Otherwise, you can connect to public FTP servers and download files. Common FTP commands, such as CD and GET, are supported through the **Execute** method.

Visual Basic: Windows Controls

Visual Studio 6.0

MonthView Control

See Also Example Properties Methods Events

The **MonthView** control enables you to create applications that let users view and set date information via a calendar-like interface.

Syntax

MonthView

Remarks

The **Value** property of the **MonthView** control returns the currently selected date.

You can allow end users to select a contiguous range of dates by setting the **MultiSelect** property to **True**, and specifying the number of selectable days with the **MaxSelCount** property. The **SelStart** and **SelEnd** properties return the start and end dates of a selection.

You can customize a **MonthView** control's appearance in many ways. Various color attributes such as **MonthBackColor**, **TitleBackColor**, **TitleForeColor** and **TrailingForeColor** enable you to create a unique color scheme for the control.

You can display more than one month at a time (up to 12) in a **MonthView** control by setting the **MonthRows** and **MonthColumns** properties. The total of the **MonthRows** and **MonthColumns** properties must be less than or equal to 12.

Note The **MonthView** control is part of a group of ActiveX controls that are found in the MSCOMCT2.OCX file. To use the **MonthView** control in your application, you must add the MSCOMCT2.OCX file to the project. When distributing your application, install the MSCOMCT2.OCX file in the user's Microsoft Windows System or System32 directory. For more information on how to add an ActiveX control to a project, see "Adding Controls to a Project" in the *Programmer's Guide*.

Visual Basic: MSComm Control

Visual Studio 6.0

MSComm Control

See Also Example Properties Methods Events

The **MSComm** control provides serial communications for your application by allowing the transmission and reception of data through a serial port.

Syntax

MSComm

Remarks

The **MSComm** control provides the following two ways for handling communications:

- Event-driven communications is a very powerful method for handling serial port interactions. In many situations you want to be notified the moment an event takes place, such as when a character arrives or a change occurs in the Carrier Detect (CD) or Request To Send (RTS) lines. In such cases, use the **MSComm** control's **OnComm** event to trap and handle these communications events. The **OnComm** event also detects and handles communications errors. For a list of all possible events and communications errors, see the **CommEvent** property.

- You can also poll for events and errors by checking the value of the **CommEvent** property after each critical function of your program. This may be preferable if your application is small and self-contained. For example, if you are writing a simple phone dialer, it may not make sense to generate an event after receiving every character, because the only characters you plan to receive are the OK response from the modem.

Each **MSComm** control you use corresponds to one serial port. If you need to access more than one serial port in your application, you must use more than one **MSComm** control. The port address and interrupt address can be changed from the Windows Control Panel.

Although the **MSComm** control has many important properties, there are a few that you should be familiar with first.

Properties	**Description**
CommPort	Sets and returns the communications port number.
Settings	Sets and returns the baud rate, parity, data bits, and stop bits as a string.
PortOpen	Sets and returns the state of a communications port. Also opens and closes a port.
Input	Returns and removes characters from the receive buffer.
Output	Writes a string of characters to the transmit buffer.

Visual Basic: MSComm Control

MSComm Control Example

The following simple example shows basic serial communications using a modem:

```
Private Sub Form_Load ()
   ' Buffer to hold input string
   Dim Instring As String
   ' Use COM1.
   MSComm1.CommPort = 1
   ' 9600 baud, no parity, 8 data, and 1 stop bit.
   MSComm1.Settings = "9600,N,8,1"
   ' Tell the control to read entire buffer when Input
   ' is used.
   MSComm1.InputLen = 0
   ' Open the port.
   MSComm1.PortOpen = True
   ' Send the attention command to the modem.
   MSComm1.Output = "ATV1Q0" & Chr$(13) ' Ensure that
   ' the modem responds with "OK".
   ' Wait for data to come back to the serial port.
   Do
      DoEvents
   Buffer$ = Buffer$ & MSComm1.Input
   Loop Until InStr(Buffer$, "OK" & vbCRLF)
   ' Read the "OK" response data in the serial port.
   ' Close the serial port.
   MSComm1.PortOpen = False
End Sub
```

Note The **MSComm** control can use polling or an event-driven method to retrieve data from the port. This simple example uses the polling method. For an example of the event-driven method, see help for the OnComm event.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

MSFlexGrid Control

See Also Example Properties Methods Events

The Microsoft FlexGrid (**MSFlexGrid**) control displays and operates on tabular data. It allows complete flexibility to sort, merge, and format tables containing strings and pictures. When bound to a **Data** control, **MSFlexGrid** displays read-only data.

Syntax

MSFlexGrid

Remarks

You can put text, a picture, or both, in any cell of an **MSFlexGrid**. The **Row** and **Col** properties specify the current cell in an **MSFlexGrid**. You can specify the current cell in code, or the user can change it at run time using the mouse or the arrow keys. The **Text** property references the contents of the current cell.

If the text in a cell is too long to display in the cell, and the **WordWrap** property is set to **True**, the text wraps to the next line within the same cell. To display the wrapped text, you may need to increase the cells column width (**ColWidth** property) or row height (**RowHeight** property).

Use the **Cols** and **Rows** properties to determine the number of columns and rows in an **MSFlexGrid**.

Note Before you can use an **MSFlexGrid** in your application, you must add the MSFlxGrd.ocx file to your project. To automatically include the file in your project, put it in the Autoload file. When distributing your application, you should install the MSFlxGrd.ocx file in the users Microsoft Windows System directory. For more information about adding an ActiveX control to a project, see "Standard ActiveX Controls" in the *Visual Basic Programmers Guide*.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

MSHFlexGrid Control

See Also Example Properties Methods Events

The Microsoft Hierarchical FlexGrid (**MSHFlexGrid**) control displays and operates on tabular data. It allows complete flexibility to sort, merge, and format tables containing strings and pictures. When bound to a data control, **MSHFlexGrid** displays read-only data.

Syntax

MSHFlexGrid

Remarks

You can place text, a picture, or both in any cell of an **MSHFlexGrid**. The **Row** and **Col** properties specify the current cell in an **MSHFlexGrid**. You can specify the current cell using code, or the user can change it at run time using the mouse or the arrow keys. The **Text** property references the contents of the current cell.

If the text in a cell is too long to display in the cell, and the **WordWrap** property is set to **True**, the text wraps to the next line within the same cell. To display the wrapped text, you may need to increase the cells column width (**ColWidth** property) or row height (**RowHeight** property).

Use the **Col** and **Row** properties to determine the number of columns and rows in an **MSHFlexGrid**. Use the **Band** properties to determine the band styles in an **MSHFlexGrid**.

Displaying Hierarchical Recordsets

A major feature of the MSHFlexGrid control is its ability to display hierarchical recordsetsrelational tables displayed in a hierarchical fashion. The easiest way to create a hierarchical recordset is to use the Data Environment designer and assign the **DataSource** property of the **MSHFlexGrid** control to the Data Environment. You can also create a hierarchical recordset in code using a Shape command as the **RecordSource** for an **ADO Data Control**, as shown in the example below:

```
' Create a ConnectionString.
Dim strCn As String
strCn = "Provider=MSDataShape.1;Data Source=Nwind;" & _
"Connect Timeout=15;Data Provider=MSDASQL"

' Create a Shape command.
Dim strSh As String
strSh = "SHAPE {SELECT * FROM `Customers`}  AS Customers " & _
"APPEND ({SELECT * FROM `Orders`}  AS Orders RELATE " & _
"CustomerID TO CustomerID) AS Orders"

' Assign the ConnectionString to an ADO Data Control's
' ConnectionString property, and the Shape command to the
' control's RecordSource property.
With Adodc1
   .ConnectionString = strCn
   .RecordSource = strSh
End With
' Set the HflexGrid control's DataSource property to the
' ADO Data control.
Set HFlexGrid1.DataSource = Adodc1
```

For More Information To find out more about hierarchical recordsets, see Hierarchical Cursors and Data Shaping Summary. A description of Shape commands can also be found in Shape Commands in General.

Note Before you can use an **MSHFlexGrid** in your application, you must add the MSHFlxGd.ocx file to your project. To automatically include the file in your project, put it in the Autoload file. When distributing your application, you should install the MSHFlxGd.ocx file in the users Microsoft Windows System directory. For more information about adding an ActiveX control to a project, see "Standard ActiveX Controls" in the *Visual Basic Programmers Guide*.

Visual Studio 6.0

Visual Basic: MSChart Control

MSChart Control

See Also Example Properties Methods Events

A chart that graphically displays data.

Syntax

MSChart

Remarks

The **MSChart** control supports the following features:

- True three-dimensional representation.
- Support for all major chart types.
- Data grid population via random data and data arrays.

The **MSChart** control is associated with a data grid (**DataGrid** object). This data grid is a table that holds the data being charted. The data grid can also include labels used to identify series and categories on the chart. The person who designs your chart application fills the data grid with information by inserting data or by importing data from a spreadsheet or array.

Visual Basic: MSChart Control

MSChart Control Example

The following example displays a three-dimensional chart with eight columns and rows of data and sets the legend parameters.

```
Private Sub Command1_Click()
   With MSChart1
      ' Displays a 3d chart with 8 columns and 8 rows
      ' data.
      .ChartType = VtChChartType3dBar
      .ColumnCount = 8
      .RowCount = 8
      For column = 1 To 8
         For row = 1 To 8
            .Column = column
            .Row = row
            .Data = row * 10
         Next row
      Next column
      ' Use the chart as the backdrop of the legend.
      .ShowLegend = True
      .SelectPart VtChPartTypePlot, index1, index2, _
      index3, index4
      .EditCopy
      .SelectPart VtChPartTypeLegend, index1, _
      index2, index3, index4
      .EditPaste
   End With
End Sub
```

Visual Basic: Multimedia MCI Control

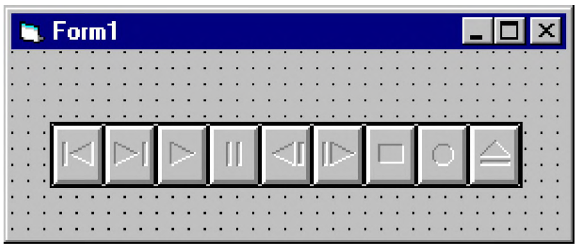
Visual Studio 6.0

Multimedia MCI Control

See Also Example Properties Methods Events

The **Multimedia MCI** control manages the recording and playback of multimedia files on Media Control Interface (MCI) devices. Conceptually, this control is a set of push buttons that issues MCI commands to devices such as audio boards, MIDI sequencers, CD-ROM drives, audio CD players, videodisc players, and videotape recorders and players. The MCI control also supports the playback of Video for Windows (*.avi) files.

When you add the **Multimedia MCI** control to a form at design time, the control appears on the form as follows:



The buttons are defined as Prev, Next, Play, Pause, Back, Step, Stop, Record, and Eject, respectively.

Remarks

Your application should already have the MCI device open and the appropriate buttons in the **Multimedia MCI** control enabled before the user is allowed to choose a button from the **Multimedia MCI** control. In Visual Basic, place the MCI Open command in the Form_Load event.

When you intend to record audio with the **Multimedia MCI** control, open a new file. This action ensures that the data file containing the recorded sound will be in a format compatible with your system's recording capabilities. Also, issue the MCI Save command before closing the MCI device to store the recorded data in the file.

The **Multimedia MCI** control is programmable in several ways:

- The control can be visible or invisible at run time.
- You can augment or completely redefine the functionality of the buttons in the control.
- You can control multiple devices in a form.

If you want to use the buttons in the **Multimedia MCI** control, set the **Visible** and **Enabled** properties to **True**. If you do not want to use the buttons in the control, but want to use the **Multimedia MCI** control for its multimedia functionality, set the **Visible** and **Enabled** properties to **False**. An application can control MCI devices with or without user interaction.

The events (button definitions) of the **Multimedia MCI** control are programmable. You can augment or completely redefine the functionality of these buttons by developing code for the button events.

The MCI extensions support multiple instances of the **Multimedia MCI** control in a single form to provide concurrent control of several MCI devices. You use one control per device.

Distribution Note When you create and distribute applications that use the **Multimedia MCI** control, you should install and register the appropriate files in the customer's Microsoft Windows System or System32 directory. The Package and Deployment Wizard included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

Visual Basic: Multimedia MCI Control

Examples (Multimedia MCI Control)

Visual Basic Example

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

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

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

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

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