Microsoft Visual FoxPro

ActiveX Controls and Visual FoxPro 5.0

Overview

ActiveX™ controls provide additional functionality to a Microsoft® Visual FoxPro™ application. Visual FoxPro includes a set of ActiveX controls (.OCX files) you can add to and distribute with your applications. ActiveX controls are added to a form in an application by using the Visual FoxPro OLE Container control.

ActiveX controls are installed by default if you choose the Complete installation option when you install Visual FoxPro. If you did not choose this option, you can run Visual FoxPro Setup again at any time and install only the ActiveX controls. ActiveX controls are installed in the SYSTEM directory in Windows 95 or the SYSTEM32 directory in Windows NT®.

ActiveX Controls included in Visual FoxPro 5.0

The following table lists the .OCX files shipped with Visual FoxPro and the ActiveX controls contained in each file.

File	Controls
COMCTRL32.OCX	ImageList ListView ProgressBar Slider StatusBar TabStrip Toolbar TreeView
COMDLG32.OCX	Common Dialogs
DBLIST32.OCX	MSDataCombo MSDataList
FOXHWND.OCX	Visual FoxPro HWND
FOXTLIB.OCX	Visual FoxPro Foxtlib
GRID32.OCX	Grid
MCI32.OCX	Microsoft Multimedia
MSACAL70.OCX	Calendar
MSCOMM32.OCX	Microsoft Comm
MSMAPI32.OCX	Microsoft MAPI Message Microsoft MAPI Session
MSOUTL32.OCX	Outline
PICCLP32.OCX	PicClip
RICHTX32.OCX	Rich Textbox
SYSINFO.OCX	SysInfo
TABCTL32.OCX	SSTab
THREED32.OCX	Threed Checkbox Threed Command Button Threed Frame Threed Group Push Button Threed Option Button Threed Panel

For additional information about adding ActiveX controls to your applications, see Chapter 16, "Adding OLE," in the *Developer's Guide*.

Distributing ActiveX Controls included with Visual FoxPro

Distribution rights of an ActiveX control is determined by the control's creator. Be sure you have distribution rights for any ActiveX controls you distribute with a Visual FoxPro application. All of the ActiveX controls included with Visual FoxPro can be distributed with the applications you create.

Note The MediaView 1.41 ActiveX control (MEDV141N.OCX), installed with the Visual FoxPro online documentation, cannot be added to or distributed with your applications. This ActiveX control is intended only for browsing the Visual FoxPro online documentation in an interactive Visual FoxPro session.

Using ActiveX Controls in Visual FoxPro

The OLE Container Control

ActiveX controls can be placed on a Visual FoxPro form through the OLE Container control. When you place an OLE Container control on a form, the Insert Object dialog box appears. To add an ActiveX control to the form, choose the Insert Control option button. The Control Type list appears. From this list you can choose an ActiveX control to place on the form. If the ActiveX control you'd like to place on the form isn't in the Control Type list, choose the Add Control button. Choosing the Add Control button displays the Browse dialog box, allowing you to locate the ActiveX control you'd like to place on the form.

ActiveX controls can also be added to a form by choosing the View Classes button on the Form Controls toolbar, and then choosing ActiveX Controls. An icon for each registered ActiveX control appears on the toolbar. (To register an installed ActiveX control, choose Options from the Tools menu, then choose the Controls tab. Check the box next to the ActiveX control to register, then choose the Set As Default button.) To place an ActiveX control on the form, click its icon and then click the form where you'd like the ActiveX control to appear.

ActiveX Tips and Tricks

The following sections discuss tips and tricks for applications that utilize ActiveX controls.

Object Keyword

A new Object keyword has been added to Visual FoxPro 5.0 so you can specify properties for ActiveX controls in class definitions defined programmatically. Class definitions are processed before the ActiveX control is instantiated; the Object keyword indicates to Visual FoxPro that a property value should be applied when the ActiveX control is instantiated. If the Object keyword is not included, a custom property for the ActiveX control is created when the control is instantiated. Note that the Object keyword must be immediately preceded by a period.

The following example adds the ActiveX Outline control (MSOUTL32.OCX) to a form. The Object keyword is used to specify the BackColor property for the Outline control before the control is instantiated. Run the example, remove the Object keyword from the program line containing the BackColor property, and run the example again to see the effect of the Object keyword on the Outline control.

```
PUBLIC frmOLETest

frmOLETest = CREATEOBJECT('Form')  && Create a form
frmOLETest.Visible = .T.  && Display the form

* The following line adds the Outline control to the form
frmOLETest.ADDOBJECT ( 'OCXTest', 'BlueOLEControl',
    ; 'MSOutl.Outline')

frmOLETest.OCXTest.AddItem( 'Item One' )   && Add an item to the control
frmOLETest.OCXTest.AddItem( 'Item Two' )   && Add an item to the control
```

```
DEFINE CLASS BlueOLEControl AS OLEControl
    * Use the Object keyword to set the Backcolor
    * property for the Outline control

    .Object.Backcolor = 16776960 && Light blue

       Set properties of the OLE Container Control
       Height = 100       Width = 200       Visible = .T.
ENDDEFINE
```

Passing Arrays to ActiveX Controls

If you pass an array to an ActiveX control - for example, if you call one of the control's methods and want to pass it an array - you must pass the array by reference. If you pass the array by value (the default), only the first element will be passed.

Programmatically Adding ActiveX Controls to an OLE Container Control

You can use ADDOBJECT( ) to add an ActiveX control to an OLE Container control programmatically at run time. The third argument in ADDOBJECT( ) is the OLE class ID for the ActiveX control. To determine the OLE class ID for an ActiveX control, add the ActiveX control to a form. The class ID for the ActiveX control appears in the OLEClass property in the Properties window.

The following example demonstrates how you can programmatically add the ActiveX ProgressBar control contained in COMCTRL32.OCX to a form. Note the third argument of ADDOBJECT( ) (COMCTL.ProgCtrl.1) that specifies the OLE class ID for the ActiveX ProgressBar control.

```
PUBLIC oForm
oForm = CREATEOBJECT("Form")   && Create a form

* The following line adds the ProgessBar control to the form

oForm.ADDOBJECT("OleProgress", "OLEControl", "COMCTL.ProgCtrl.1")

* The following lines specify the location
* and size of the ProgressBar control

oForm.OleProgress.Top = 5
oForm.OleProgress.Left = 5
oForm.OleProgress.Height = 25
oForm.OleProgress.Width = 289

oForm.OleProgress.Visible = .T. && Display the ProgessBar
oForm.Visible = .T. && Display the Form

FOR i = 1 to 10
  oForm.OleProgress.Value=i*10 && Update the ProgressBar
  = INKEY(1)   && Wait one second
ENDFOR
```

If an ActiveX control is programmatically added to an OLE Container control, the ActiveX control must be properly registered on the computer on which the Visual FoxPro application is run. Proper registration of the ActiveX control is the responsibility of the developer who distributes an application utilizing ActiveX controls. The ActiveX controls included with Visual FoxPro (with the exception of MEDV141N.OCX, the MediaView 1.41 ActiveX control) are licensed to be distributed with any application.

You can register an ActiveX control used by your Visual FoxPro application during the installation of that application. Include the ActiveX control and any supporting files it requires in the list of distributed files specified in the Setup Wizard. In the "Change File Settings" step of the Setup Wizard, check the ActiveX checkbox for each ActiveX control that you distribute. This instructs Setup to register the control when the distributed application is installed. Note that the list of

supporting files required for each of the ActiveX Control shipped with Visual FoxPro is listed in the control's help file or help topic.

The preferred method for distributing an application utilizing the ActiveX controls included with Visual FoxPro is to add ActiveX controls to .VCX visual class libraries. The .VCX visual class libraries are included in the Setup Wizard distribution directory, and the setup created by the Setup Wizard will automatically register the ActiveX controls on the computer on which the distributed application is installed.

For more information about distributing ActiveX controls with your applications, see "Including ActiveX Components" in Chapter 25, "Building an Application for Distribution" in the *Developer's Guide*.

Licensing

Many ActiveX Controls, including those shipped with Visual FoxPro 5.0, require both a design time and runtime license. The design time license is located and read by the control when it is instantiated. The runtime license is provided to Visual FoxPro by the control when the control is saved with a form or visual class. The legal agreement provided with the ActiveX controls specifies that the design time license may not be distributed with custom applications developed in Visual FoxPro 5.0.

To distribute ActiveX Controls which enforce this type of licensing scheme with your application, create a visual class library which contains a visual class for each of the controls you wish to distribute. Include this visual class and the ActiveX controls (.OCX files) and the supporting files they require with your custom application. In your code, instantiate the ActiveX controls you add to your forms from these class libraries. Because the visual class libraries are created and saved on your computer have a design time license, they also contain the correct runtime license, and your code will run without errors in your distributed applications. Note that ActiveX controls saved in visual class libraries and forms do not need to be registered to run correctly. It is not necessary to specify to the Setup Wizard that the ActiveX controls you distribute are ActiveX components.

Working with ActiveX controls

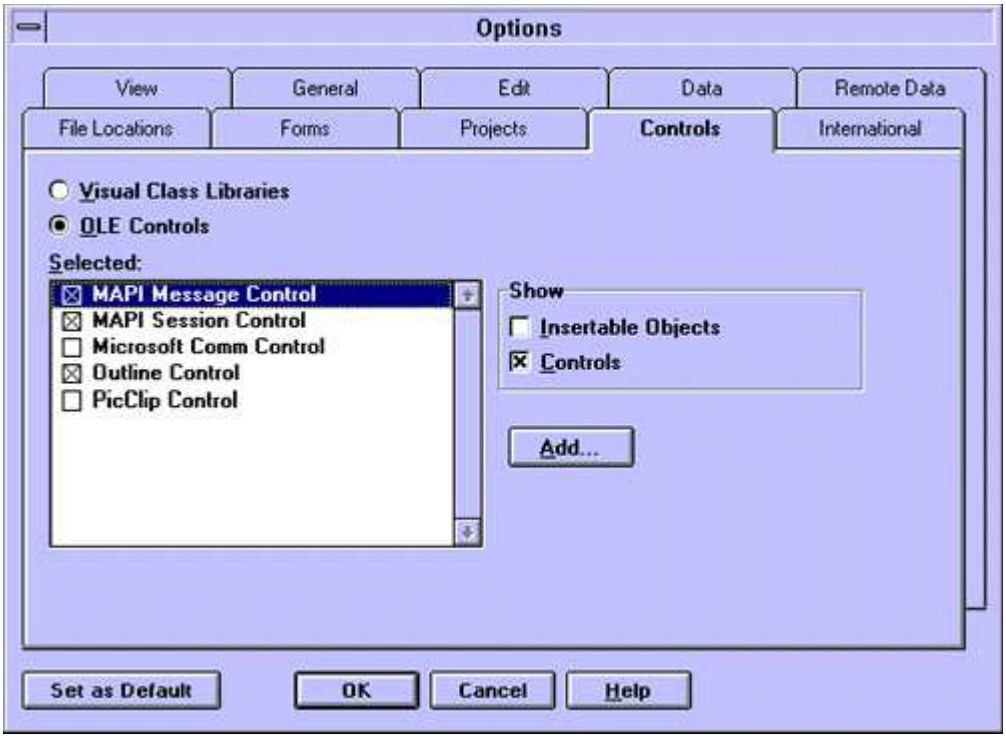
This section discusses three ActiveX controls that ship with Visual FoxPro: the Outline control, the MAPI Session control and the MAPI Messages control.

The Outline Control

The Outline control is a special type of ListBox control that shows list information in hierarchical order. It is useful for showing a myriad of information from files and directories (much like File Manager in Microsoft Windows) to sales by territory, and much more. Basically, any information based on ownership or containership can be shown in hierarchical order with the Outline control.

Registering ActiveX Controls

ActiveX controls can be used like any other control, provided they have been *registered* with Visual FoxPro. You register ActiveX controls in the Options dialog box as shown below.



Options Dialog Box

To see the list of the ActiveX controls available on your system, select the ActiveX Controls option button and make sure that the Controls check box is selected. In this case, the Insertable Objects check box (which refers to OLE applications such as Microsoft Word for Windows) is cleared to keep the list more readable and manageable.

Note, in this example, the five controls shown. All of these controls ship with Visual FoxPro and are installed with the full installation. You can, of course, opt not to install these. If you do not see these controls in your list, you can install them from the Visual FoxPro installation diskettes. See the Visual FoxPro *Installation Guide* for more information.

To permanently register a control with Visual FoxPro (so that it is always available in designers), select the desired controls and then click Set As Default. The controls will then always appear in the list of ActiveX controls in the designers.

If you acquire additional ActiveX controls and install them, use the Options dialog box to register them for use with Visual FoxPro.

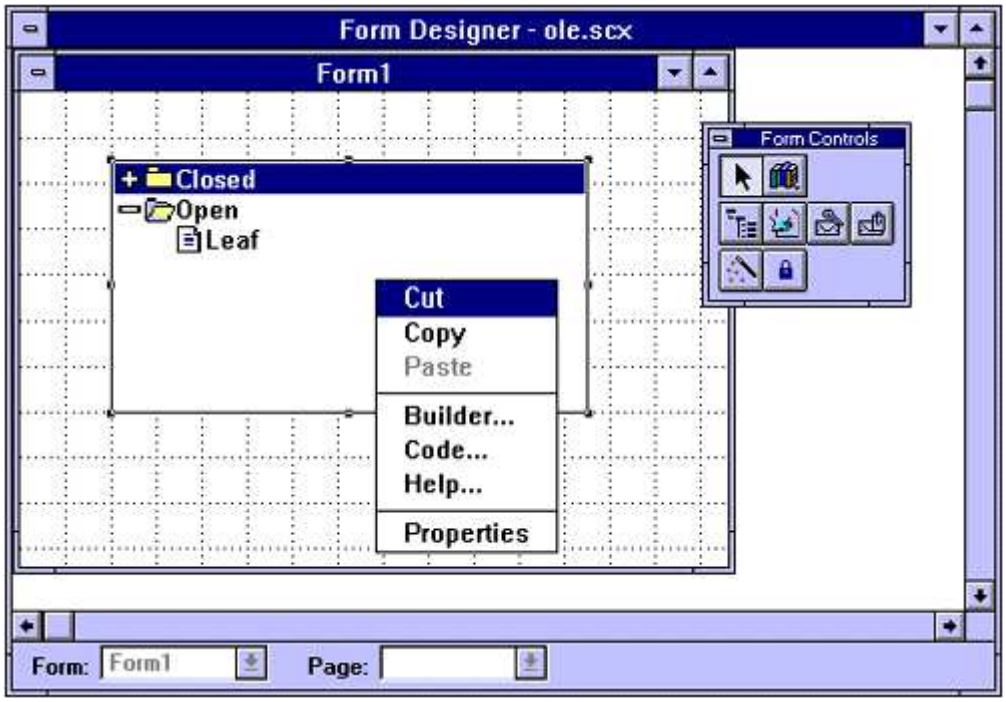
Using ActiveX Controls

Once the ActiveX controls are registered, you can access them with the Form Controls toolbar. To show the registered controls, click the View Classes button on the toolbar and select ActiveX Controls. The registered ActiveX controls appear on the toolbar. Click the desired object and drop it on the container (for example, a form, a page frame or a grid). From that point on, you can work with the ActiveX control like any other control.

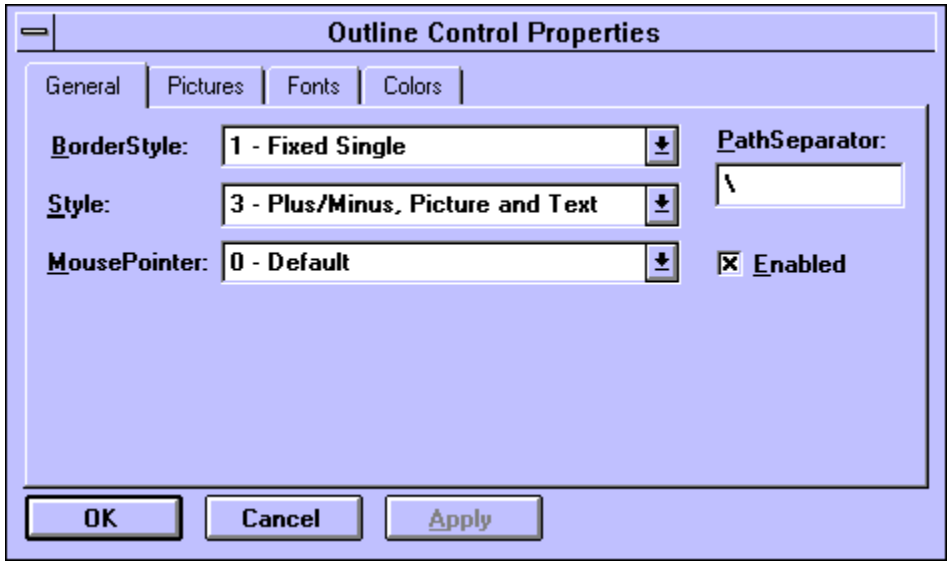
Working with ActiveX Control Properties

The properties of an ActiveX control can be accessed in two ways:

- Using the Visual FoxPro Properties window. This dialog box shows the control's properties, events and methods much like any other Visual FoxPro control.
- Using the Properties window associated with an ActiveX control, which can be accessed with a right mouse click from the shortcut menu shown below. Note that the Properties option is now at the bottom of the list.



ActiveX Control Shortcut Menu
The Properties window brought up by selecting the Properties option from this menu shows *only those properties programmed into the control*. The dialog below shows the Properties window for the Outline control.



Outline Control Properties Window
Note that this dialog only allows access to *properties* of the control; events and methods are not accessible. The standard Properties dialog box, on the other hand, allows access to all Visual FoxPro standard properties, events and methods, as well as the properties specific to the control. The reason for this is interesting. Visual FoxPro wraps the ActiveX control in an OLE container that allows the developer access to the standard properties, events, and methods as well as the properties specific to the ActiveX control. This wrapping means that you can subclass ActiveX controls. This feature is *unique* to Visual FoxPro.

Now that we have discussed the issues relating to working with ActiveX controls in general, the next step is to discuss the particulars of the Outline control.

An Outline Example

The following figure is an example of a form using the Outline control. It shows customers and

the products they have ordered. The data for this example comes from the TESTDATA database that ships with Visual FoxPro.

Customer Orders Form
As you can see, this is a powerful way of looking at data. Only the detail for the companies the user specifically expands are shown. This gives the user the ability to limit the information they see on the form.

Here's the code for this form:

```
* Class.............: Frmcustord*
* Notes.............: Exported code from Class Browser.
************************************************
 *-- Form:        frmcustord
 *-- ParentClass: form
 *-- BaseClass:   form
 *

DEFINE CLASS frmcustord AS form

DataSession = 2
Height = 358
Width = 588
DoCreate = .T.
AutoCenter = .T.
BackColor = RGB(192,192,192)
BorderStyle = 2
Caption = "Customer Orders"
Name = "frmcustord"

ADD OBJECT oleoutline AS olecontrol WITH ;
  Top = 12, ;
  Left = 12, ;
  Height = 301, ;
  Width = 565, ;
  Name = "oleOutline"
```

```
ADD OBJECT cmdok AS commandbutton WITH ;
  Top = 324, ;
  Left = 240, ;
  Height = 29, ;
  Width = 94, ;
  Cancel = .T., ;
  Caption = "\<OK", ;
  Default = .T., ;
  Name = "cmdOK"
*-- Loads the customer orders into the Outline Control
PROCEDURE LoadData
  SET TALK OFF
  WAIT WINDOW NOWAIT "Loading information. Please stand by..."
  SELECT _cCustOrders
  GO TOP

  LOCAL lcCompany
  SCAN
    lcCompany = _cCustOrders.Company
    THISFORM.OleOutline.AddItem(lcCompany)

    DO WHILE lcCompany == _cCustOrders.Company
      IF !EMPTY(_cCustOrders.Product)
       THIS.OleOutline.AddItem(Product)
       THIS.OleOutline.Indent(THIS.OleOutline.listcount-1) = 2
       THIS.oleOutline.PictureType(THIS.OleOutline.listcount-1)= 2
      ENDIF
      SKIP
    ENDDO
  ENDSCAN
ENDPROC

PROCEDURE Init
  THISFORM.LoadData()
ENDPROC

PROCEDURE Load
  OPEN DATABASE home()+"SAMPLES\DATA\TESTDATA"

 SELECT Customer.company, ;
        Products.prod_name AS Product;
   FROM testdata!customer, ;
        testdata!orders, ;
        testdata!orditems, ;
        testdata!products ;
 WHERE Customer.cust_id = Orders.cust_id ;
    AND Orders.order_id = Orditems.order_id ;
    AND Products.product_id = Orditems.product_id ;
    AND Customer.Country = "Germany" ;
  UNION ALL ;
        SELECTCustomer.company, ;
               "" ;
          FROMtestdata!customer ;
        WHERECustomer.cust_id NOT IN ;
               (SELECT Orders.cust_id FROM testdata!orders ;)
           ANDCustomer.Country = "Germany" ;
```

```
 ORDER BY 1,2 ;
   INTO CURSOR _cCustOrders


 GO TOP

ENDPROC

PROCEDURE oleoutline.Collapse
 *** OLE Control Event ***
 LPARAMETERS listindex
 IF this.HasSubitems(listindex)
   this.PictureType(listindex) = 0
 ENDIF
ENDPROC

PROCEDURE oleoutline.Expand
 *** OLE Control Event ***
 LPARAMETERS listindex
 IF THIS.HasSubitems(listindex)
   THIS.PictureType(listindex) = 1
 ENDIF
ENDPROC

PROCEDURE cmdok.Click
 RELEASE THISFORM
ENDPROC

ENDDEFINE
*
*-- EndDefine: frmcustord
**************************************************
```

The Load event run a SELECT - SQL statement from the information in the TESTDATA database to extract the names of customer in Germany and the products they have purchased. The list is ordered by company name and then by product name.

The list is populated in the LoadData method, which is a custom method to this form. This method simply scans through the result set. Customers are added at the default indent level (1) whereas the products they ordered are added at level 2. Indenting items to level 2 makes them subitems of the immediately prior level 1 item. Products are also set to have the "document" icon by setting the PictureType property for that element to 2.

When the list is expanded and contracted, the picture for the main item has to be changed from the "closed" folder to the "open" folder icon. This is accomplished in the Expand and Collapse event methods which are part of the Outline control.

As you can see, the amount of code needed to work with Outline control lists is minimal.

Outline Control: A Final Word

The Outline control is a wonderful example of how an ActiveX control can enhance a system. The ability to translate data into a visual, hierarchical display is a powerful way to allow users to view and work with data. However, ActiveX controls can add much more than display features to an application. The two MAPI controls that ship with Visual FoxPro provide an example of how ActiveX controls can add functionality to your applications.

Creating Mail-Enabled Applications with the MAPI Controls

Visual FoxPro ships two controls designed to enable applications to interact with MAPI-compliant mail systems. These two controls, the MAPI Session control and the MAPI Messages control, work

in tandem to allow an application to send, respond and reply to messages in addition to managing the e-mail system.

Creating mail-enabled applications is becoming a requirement in many business settings. The MAPI controls make this requirement easy to accomplish.

Managing E-Mail Sessions

Before you can work with e-mail, you have to establish a *session* with the e-mail system. You can think of a session in e-mail as being similar to working with an Automated Teller Machine (ATM). In order to work with the ATM, you first have to establish a session. Part of establishing a session includes logging into the system. Once you have a session established, you can initiate and complete many transactions. The same is true of an e-mail system.

Establishing the Session

Establishing a session with the MAPI Session control is a simple matter. Place the control on a form and put the following code in its Init( ) event:

```
THIS.LogonUI = .T.
THIS.SignOn()
```

The Signon( ) method accesses the mail system and attempts to establish a mail session. Since the LogonUI property is set to .T., if the user is not logged in yet, a dialog box is displayed for the user to enter a user name and password.

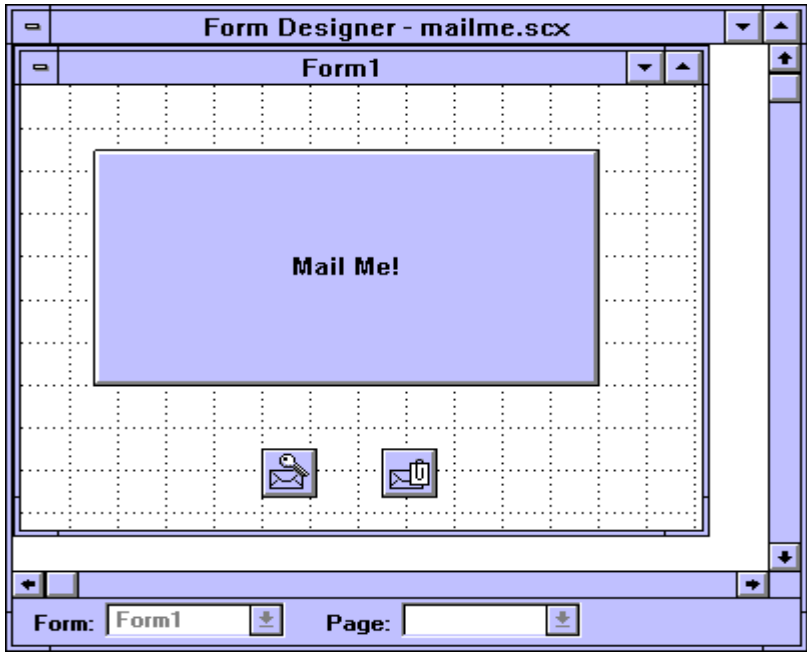
The session is terminated with the SignOff( ) method.

Each MAPI session has an ID. The ID of the session is stored in the SessionId property of the Session control. The ID is important for the Message control to be able to communicate with the session.

Composing Messages

Composing message with the MAPI Message control is straightforward. The control has a series of methods and properties that a developer can modify and work with in order to compose messages to be sent through the mail system. For example, the MsgNoteText property holds the text of the message. Placing text in that property will put the text into the body of the message. The control also allows you to attach files, work with address books, and read messages as well.

MAILME.SCX is a sample form that illustrates well how messages with attachments can be composed and sent using the MAPI Message control. The figure below shows the form in the designer followed by the code to the form.



Form MAILME * Form.....: MAILME.SCX * Notes.....:
Exported code from Class Browser.

```
**************************************************
*-- Form:        form1
*-- ParentClass: form
*-- BaseClass:   form
*
DEFINE CLASS form1 AS form

 DoCreate = .T.
 Caption = "Form1"
 Name = "Form1"

ADD OBJECT olesession AS olecontrol WITH ;
 Top = 204, ;
 Left = 120, ;
 Height = 100, ;
 Width = 100, ;
 Name = "oleSession"

ADD OBJECT olemessage AS olecontrol WITH ;
Top = 204, ;
Left = 180, ;
Height = 100, ;
Width = 100, ;
AddressModifiable = .T., ;
AddressResolveUI = .T., ;
Name = "oleMessage"

ADD OBJECT cmdmailme AS commandbutton WITH ;
Top = 36, ;
Left = 36, ;
Height = 133, ;
Width = 253, ;
Caption = "Mail Me!", ;
Name = "cmdMailMe"
```

```
PROCEDURE olesession.Init
 *-- Establish a session

 THIS.LogonUI = .T.
 THIS.Signon()
ENDPROC

PROCEDURE olesession.Destroy
 THIS.signoff()
ENDPROC

PROCEDURE olemessage.Init
 THIS.SessionID = THISFORM.oleSession.SessionID
ENDPROC

PROCEDURE cmdmailme.Click  *-- First, save this as a class to
JUNKME.VCX

 LOCAL lcOldSafety
 lcOldSafety = SET("safety")
 SET SAFETY OFF

 THISFORM.SaveAsClass("JunkMe.VCX", "MailMe")

 *-- Set the address book caption
 THISFORM.oleMessage.AddressCaption = "Select Recipient(s)"

 *-- Clear the compose buffer.
 THISFORM.oleMessage.Compose()

 *-- Get the recipient's address.
 THISFORM.oleMessage.Show()

 *-- Set the message subject and the body text.
 THISFORM.oleMessage.msgNoteText = ;
    "Please see that these files get into the class libraries " + ;
     REPL(chr(13)+chr(10), 3) + " "
 THISFORM.oleMessage.msgSubject = "Attached Files"

*-- Add first attachment  THISFORM m.oleMessage.AttachmentIndex = ;
   THISFORM.oleMessage.AttachmentCount
 THISFORM.oleMessage.AttachmentType = 0
 THISFORM.oleMessage.AttachmentPathName = "JUNKME.VCX"
 THISFORM.oleMessage.AttachmentPosition = ;
   LEN(THISFORM.oleMessage.msgNoteText)-2

 *-- Now the second attachment
 THISFORM.oleMessage.AttachmentIndex =
THISFORM.oleMessage.AttachmentCount
 THISFORM.oleMessage.AttachmentType = 0
THISFORM.oleMessage.AttachmentPathName =
"JUNKME.VCT"
 THISFORM.oleMessage.AttachmentPosition = ;
   LEN(THISFORM.oleMessage.msgNoteText)-1

 *-- Diplay the Send dialog before sending
 THISFORM.oleMessage.Send(.T.)
```

```
 *-- Reset SET SAFETY and finish
 SET SAFETY &lcOldSafety
 ENDPROC

ENDDEFINE
*
*-- EndDefine: form1
**************************************************
```

This form has both a MAPI Session and a MAPI Message control on it.

The Init( ) event of the MAPI Message control sets the SessionId property of the Message control to the SessionId of the Session control. In this way, our messages are linked to the current open mail session. Most of the work happens in the Click( ) event of the cmdMailMe command button:

- The form is saved to a .VCX file using the SaveAsClass( ) method. This creates a couple of files to attach to the message.
- The AddressCaption property is set. When the address book is displayed with the Show method later on, this will be the caption in the dialog box.
- The Compose method is called. This clears everything in the current compose buffer, providing a clean buffer.
- The user is presented with the list of addresses so they can select a recipient.
- The message body and subject are set using the msgNoteText and msgSubject properties respectively.
- The attachments are added. The process of adding the attachments work as follows:

  1. The AttachmentIndex property is set to a value equal to or greater than the AttachmentCount property. Attachments are numbered 0-n in the MAPI Message control. If the AttachmentIndex property is set to 0, you are working with the *first* attachment in the message. When the AttachmentIndex property is set to 0, for example, the AttachmentCount property is automatically set to 1 (because there is one attachment in the message).

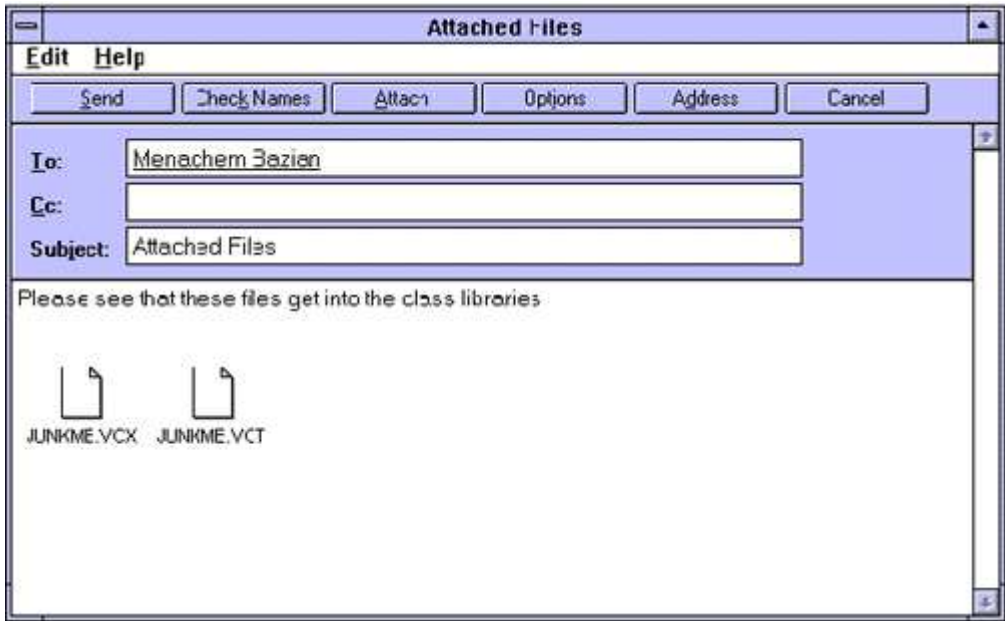
  2. The type of attachment is specified. Type 0 is a plain data file.

  3. The name of the file is specified (JUNKME.VCT).

  4. The position of the attachment in the message is specified. The attachment shows as an icon in the message. The AttachmentPosition property, which refers to the currently indexed attachment, defines which character position in the message the attachment will show up in. It is very important to allocate text (in this case, we allocated a few spaces at the end of the message) for the attachment icons and to place the icons properly. If you do not, you might overwrite characters in the message or even get an error, if you did not specify enough characters in the msgNoteText property to allow for the attachment icons.

- The Send( ) method is called with a parameter of .T.,so that a dialog box is displayed with the message, allowing the user to edit it. If the logical parameter is not provided, the message is sent directly.

That's all there is to it. The figure below shows the dialog box that displays at the end of this process.



Composed Message

MAPI Controls: Conclusion

The MAPI controls show another side to working with ActiveX controls. They provide a host of possible additions to the capabilities of Visual FoxPro applications. These controls, although non-visual, reduce the complexity and tedium of mail-enabling applications to setting a few properties and calling a few methods. This is, in essence, what easy development is all about.

ActiveX Troubleshooting

The following sections discuss problems you might encounter with ActiveX controls and suggest workarounds for the problems.

Support for ActiveX Controls

Because ActiveX controls are independent modules, not developed in or specifically for Visual FoxPro, they might exhibit behavior different than that of native Visual FoxPro controls. You might also experience problems when using them in Visual FoxPro or find that they work differently than they do in other programs.

For example, Visual FoxPro treats ISimpleFrame controls as true containers. Visual FoxPro also supports the ability to subclass ActiveX controls. Both of these features affect the language used when working with ActiveX controls. Some of the samples that you find in the control's documentation might need to be modified to work correctly in Visual FoxPro.

If you are experiencing a problem with a control, you might find that a workaround or a more recent version of the control is available. For the most up-to-date information, search the Visual FoxPro Knowledge Base using the keyword "ActiveX." The Knowledge Base contains sample code, version information, and known issues concerning ActiveX controls.

Automatically Resizing Panel Controls in Forms

If you add an ActiveX Threed Panel Control (as an OLE Container control) to a form and then set its AutoSize property to 2 - AutoSize Panel Height To Caption, you might experience problems in redrawing the control. The control might appear very thin, and might not allow you to reset its height. To correct this problem, reset its Caption property by double-clicking it in the Properties window. This causes the Caption property to be set to "(None)," and allows you to resize the panel as required.

Binding RichTextBox Controls to Memo Fields

If you attempt to bind a Microsoft RichTextBox control directly to a memo field in a table (by setting the control's ControlSource property to the name of the memo field), Visual FoxPro

displays an error when you run the form, and unbinds the control. To work around this problem, you can copy the contents of the field to a variable that is updated whenever the record pointer is moved. Then bind the control to the variable instead of directly to the memo field. For an example of how to create this type of binding, see "Use the RichText Control" in the Solutions Sample.

ToolTips for Controls Inside ISimpleFrame Controls

To display ToolTip text for a control inside an ISimpleFrame ActiveX control (for example, for a command button inside an ActiveX SSTab control) set the ShowTips property of the ActiveX control itself to True. For controls outside of ISimpleFrame ActiveX controls, set the ShowTips property of the form to True.

Getting Help for Controls Inside ISimpleFrame Controls

If you put a native Visual FoxPro control inside an ISimpleFrame control (for example, if you put a command button inside an ActiveX SSTab control), you cannot display context-sensitive Help (What's This Help) for that control.

Moving Between SSTab Control Tabs in Design Mode

If you create a form that contains an SSTab control, then run the form from the Form Designer, and then return to design mode, you might not be able to display the first tab in the control by clicking it. If this occurs, you can try clicking it several times, or you can click on the very edge of the tab.

Using ISimpleFrame Controls in Page Frames

You might experience the following problems when working with ISimpleFrame controls (such as a Threed Frame Control) in pages in a page frame:

If you place an ISimpleFrame control on one page in a page frame and then display the form, the control might initially appear on all pages. To prevent the control from appearing on any page except the page on which you placed it, set the control's Visible property to true in the page's Activate method and to false in its Deactivate method. Depending on what page is initially active when the form is run, you might also have to set the control's Visible property to false in the Activate method of the page that first appears.

If you place an ISimpleFrame control on a page in a form-based class, the control might appear on the Windows® desktop when you create an instance of the class and show it. If so, you can correct the problem using this code in the Init method of the form:

```
 * On the following line,2 represents  * the page on which the control appears

THIS.PageFrame1.ActivePage = 2
THIS.Draw
THIS.PageFrame1.ActivePage = 1 && default page
```

Dragging and Dropping ActiveX Controls

You might not be able to establish drag and drop behavior for certain ActiveX controls. For example, you cannot drag ISimpleFrame controls. In other cases, you might find that establishing drag and drop between ActiveX controls results in an error.

Disabling or Enabling VTABLE Binding

Some ActiveX controls are documented to support VTABLE binding but do not fully do so. VTABLE binding allows the control's host to bind "early" to its members (properties and methods) for better performance. If the control does not support VTABLE binding, Visual FoxPro might experience a fatal error in the module OLEAUT32.DLL, or you might see a message such as "Not enough storage space is available to complete this operation."

If so, you can temporarily disable VTABLE binding by calling the new SYS(2333) function. The syntax is:

```
SYS(2333, 0)
```

After disabling VTABLE binding, you can create an instance of the control. Then reset VTABLE binding by calling SYS(2333,1) so that other ActiveX controls can take advantage of the speed increases offered by VTABLE binding.

Note If you are testing ActiveX controls in Visual Basic® 4.0, you will not encounter this problem because that version of Visual Basic does not support VTABLE binding.

You can also create an entry in the Windows Registry to disable VTABLE binding for individual controls, which saves you the trouble of disabling it each time you want to create another instance of the control. To do so, create an entry using the following Registry key, in which clsid represents the class id of the control for which you are disabling VTABLE binding:

HKEY_CLASSES_ROOT\NoDualInterface\{clsid}
For example, you can disable VTABLE binding for the Internet Explorer Shell control by making this entry in the Registry:

HKEY_CLASSES_ROOT\NoDualInterface\{EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B}

ActiveX Control Help

Many of the ActiveX controls that ship with Visual FoxPro were originally created for use with other applications such as Microsoft Visual Basic® and Microsoft Access. When an ActiveX control is placed on a form and you press F1 for Help on the control, the Help that displays is from the original application, not Visual FoxPro.

If you place the SysInfo ActiveX control on a form, there is no Help available for the control through F1. For information about this control, see the SysInfo control Help file (SYSINFO.HLP) in the SYSTEM or SYSTEM32 directory.

The Visual FoxPro HWND (FOXHWND.OCX) and Visual FoxPro Foxtlib (FOXTLIB.OCX) ActiveX controls were created for use with Visual FoxPro. Visual FoxPro Help contains information about these controls.

ActiveX Interfaces Supported by Visual FoxPro

The following table lists the ActiveX features and interfaces supported by Visual FoxPro. These features and interfaces will be of interest to developers of ActiveX controls designed for use with Visual FoxPro.

For additional information about creating ActiveX controls for applications such as Visual FoxPro, see "For Developer's Only" at the Microsoft Internet web site at www.microsoft.com/devonly/.

Supported Features

- Embedded objects from in-process servers
- Inside-out activation
- Visual Editing of embedded objects
- Idispatch for events
- Simple Data Binding
- Extended control - Visible
- Extended control - Default
- ACTIVATEWHENVISIBLE
- ACTSLIKEBUTTON
- ACTSLIKELABEL
- NOUIACTIVATE
- ALIGNABLE
- TAG
- SIMPLEFRAME
- SETCLIENTSITEFIRST
- Keyboard : Tab handling including tab order
- Keyboard: Default Button

- INSIDEOUT
- INVISIBLEATRUNTIME
- ALWAYSRUN
- Keyboard: Mnemonics
- Keyboard: Escape Button

Supported Interfaces

- IadviseSink:: OnRename
- IadviseSink:: OnSave
- IadviseSink:: OnClose
- IOleInPlaceSite:: CanInPlaceActivate
- IOleInPlaceSite:: OnInPlaceActivate
- IOleInPlaceSite:: OnUIActivate
- IOleInPlaceSite:: GetWindowContext
- IOleInPlaceSite:: OnUIDeactivate
- IOleInPlaceSite:: OnInPlaceDeactivate
- IOleInPlaceSite:: OnPosRectChange
- IOleInPlaceSite:: GetWindow
- IOleInPlaceSite:: ContextSensitiveHelp
- IoleControlSite:: LockInPlaceServer
- IoleControlSite:: OnControlInfoChanged
- IoleControlSite:: TransformCoords
- IoleControlSite:: TranslateAccelerator
- IoleControlSite:: OnFocus
- IoleInPlaceFrame:: GetWindow
- IoleInPlaceFrame:: ContextSensitiveHelp
- IoleInPlaceFrame:: GetBorder
- IoleInPlaceFrame:: RequestBorderSpace
- IoleInPlaceFrame:: SetBorderSpace
- IoleInPlaceFrame:: SetActiveObject
- IoleInPlaceFrame:: InsertMenus
- IoleInPlaceFrame:: SetMenu
- IoleInPlaceFrame:: RemoveMenus
- IoleInPlaceFrame:: SetStatusText
- IoleInPlaceFrame:: EnableModeless
- IoleInPlaceFrame:: TranslateAccelerator
- IoleContainer:: EnumObjects
- Idispatch - Backcolor
- Idispatch - DisplayName
- Idispatch - Font
- Idispatch - ForeColor
- Idispatch - LocaleID
- Idispatch - UserMode
- Idispatch - UIDead
- Idispatch - ShowGrabHandles
- Idispatch - ShowHatching
- Idispatch - DisplayAsDefaultButton
- Idispatch - SupportsMnemonics
- IpropertyNotifySink:: OnChanged
- IpropertyNotifySink:: OnRequestEdit
- IerrorInfo - OLE Automation Exceptions
- IpersistStorage
- IpersistStream or IPersistStreamInit

ActiveX Controls: Conclusion

Extensibility. Once it was an option in a development language. Today, it's the key to success. As technology has become more advanced, users expect more from their applications. No single software vendor can anticipate all the needs and desires of the user market. However, many vendors can address a much broader range of needs.

By adding ActiveX support to Visual FoxPro, Microsoft has opened the door to a flood of additional functionality. With support for this powerful feature, Visual FoxPro has no limits on what it can do.