Getting Started with ActiveX Controls

APPENDIX

B

IN THIS APPENDIX

ActiveX controls (called OLE controls before Access 97) provide a way for you to give users advanced, yet standard, interfaces, including applications developed in different languages such as VBA and xBase (FoxPro). For ActiveX controls, these languages provide containers in which the separate ActiveX server applications (OCXs) can run.

NOTE

ActiveX controls from Microsoft vary from the old OLE controls not only in name, but also in form. ActiveX controls are designed with the ActiveX control framework and are called *lightweight*, which means that they require less memory, perform faster, and are easier to distribute. Note that some third-party ActiveX controls are built on older frameworks or use *MFC (Microsoft Foundation Classes)*.

The Access retail package provides you with six ActiveX controls:

- Calendar control
- Snapshot control
- PivotTable control
- Chart control
- Spreadsheet control
- Datasource control

The Calendar and Common Dialog controls, the latter supplied in the Microsoft Office Developer (MOD), are discussed in this appendix. A more advanced discussion of the Windows Common ActiveX controls can be found in Chapter 14, "Programming for Power with ActiveX Controls."

Working with ActiveX Controls

When working with ActiveX controls in Access, you will find that they're similar to other Access controls in that they're programmed by using properties, methods, and events.

Not all ActiveX controls are data-bound like standard Access controls. The term *data-bound* means that the controls have a Control Source property that refers to Access data or can use Access record sources for row sources.

Some ActiveX controls, such as TabStrip, require little coding with VBA. Others, such as TreeView, can be programmed to be data-driven and therefore made more versatile. Using these controls to their fullest capabilities can take quite a bit of code.

TIP

You can update ActiveX control properties, like standard Access object properties, at runtime. This way, you can manipulate an ActiveX control as much as any other control.

Looking at the ActiveX Control Shipped with Access

In addition to other ActiveX controls, the following controls all ship with MOD:

Animation	ImageList	StatusBar
Common Dialog	ListView	TabStrip
FTP	ProgressBar	Toolbar
Gopher	RichTextBox	TreeView
HTTP	Slider	Winsock

In addition to these supported controls are unsupported controls available on various networks.

NOTE

Certain OCXs used internally by Access show up in the ActiveX control dialogs, but you can't use them. If you try to place these controls on your form, nothing happens. Some controls in the Access main folder—namely, Flist32.ocx and Imxgrd32.ocx—don't even show up in the ActiveX control dialogs within Access.

Placing an ActiveX Control on a Form

Although each ActiveX control varies from the next in how it's used, they're all placed on a form in the same manner. To place a control on a form from the database container, follow these steps:

1. Open an Access database.
2. Select the Forms tab. The list of forms appears.
3. Click the New button to the right of the form list. The New Form dialog appears.
4. Select the record source you want from the combo box on the bottom of the dialog.
5. Click OK. You now have a blank form, displayed in Design mode.
6. From the Insert menu, choose ActiveX Control. The Insert ActiveX Control dialog appears. (You can also use the More Controls button in the Toolbox.)

7. Choose the Calendar Control 10.0 from the list of controls, similar to Figure B.1.
8. Click OK.

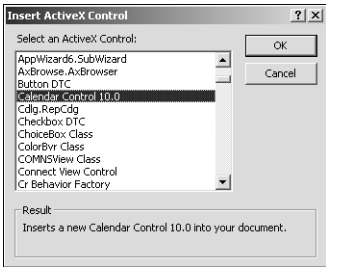

FIGURE B.1
You can insert all ActiveX controls from the Insert ActiveX Control dialog.

After the Calendar control is placed on `Form1` (see Figure B.2), you must customize it for the current application. Although this process changes for each control type—as well as for each application—the method for changing the properties stays the same.

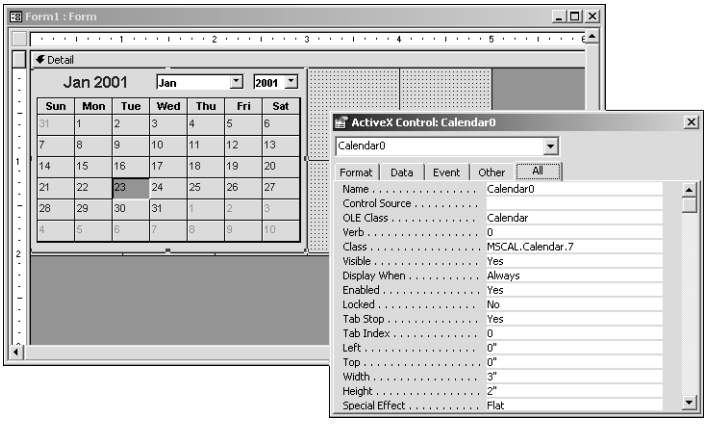

FIGURE B.2
Although you can see some ActiveX controls, such as the Calendar control, in Design mode, others display only a message.

Setting Properties on an ActiveX Control at Design Time

You can set properties in two ways at design time. The first is the same way in which all Access control properties are set—through the Access control property sheet. Figure B.3 shows the Access Control property sheet (opened by choosing Properties from the View menu) for the Calendar object created on `Form1`.

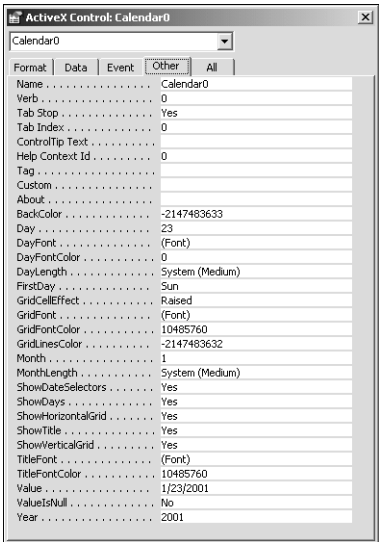

FIGURE B.3

Find the properties specific to the ActiveX control on the Other tab, just below the `Tag` *property.*

When you move into an ActiveX control property in the property sheet, the status bar displays the message `ActiveX Control specific property` because Access doesn't necessarily know what the property is without ActiveX interpreting, which happens during runtime.

TIP

The other way to get to the properties specific to an ActiveX control is to open the control's own property sheet. To do so, double-click the control or use the control's right-click menu. Keep in mind that the double-click method works only with some ActiveX controls. Others, like the current version of Calendar control, bring you to different parts of the control when you double-click. Also, the double-click method doesn't work in Visual Basic and other Office Apps. So the best method is to use the right-click menu.

The various ActiveX controls have different properties on their property sheets. Figure B.4 shows the property sheet for the Calendar control.

When changing properties of ActiveX controls at runtime, you can use either macros or VBA code. This process is the same as changing properties on any other Access object. When declaring the data type of the ActiveX control in VBA, use the `Object` data type. The following example declares an `Object` data type for a Calendar control:

```
Dim ocxCalendar as Object
```

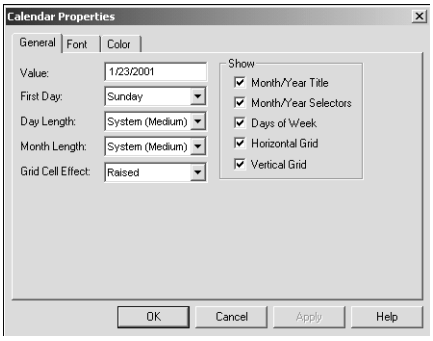

FIGURE B.4
Here's the ActiveX control property sheet for the Calendar control.

If the control is referenced (as it is by default when you add it to a form), you can use early binding on the control's `.Object` property to manipulate it, and you get all the advantages thereof:

```
Dim calCalendar as Calendar
Set calCalendar = Me!ocxCal.Object
```

By specifying the actual control type, `Calendar`, VBA knows how to handle it more specifically, and can give you Intellisense features for it.

Throughout the rest of this appendix, you see some of the different ActiveX controls available to Access developers and how to use them in applications.

Counting the Days with the Calendar Control

The Calendar control (ocxCal) presents a graphical representation of a calendar. In Figure B.5, the Calendar control used in the World Wide Video Application allows users to view when new releases are due.

The `frmNewReleases` form not only displays movies for the different days as users move through the calendar, but it also allows users to enter the name of the new release in which they're interested. The system then looks up the title in the `tblNewReleases` table and positions the calendar on the date the title is due to be released.

TIP

The power behind this form is that the Calendar control's value is a date and is actually used for the `Link Master Fields` property for the `frmNewReleasesSubform` subform. This makes it easy to show tasks or activities for a given date. `Value` is the default property, or the one bound to the underlying table field.

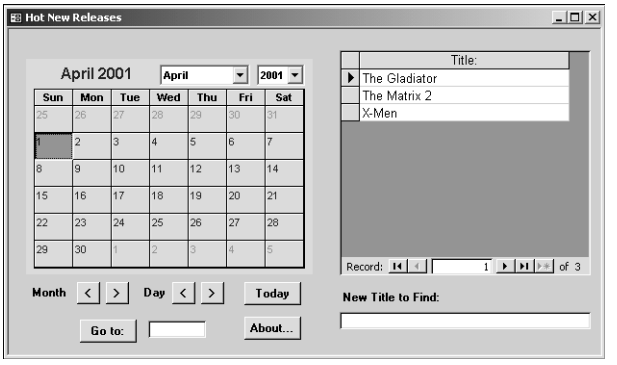

FIGURE B.5
The ActiveX Calendar control lets you create visual scheduling applications.

Figure B.6 shows the property sheet for the subform linked to the Calendar control. You can find the `frmNewReleases` form in AppB.mdb, also on this Web site.

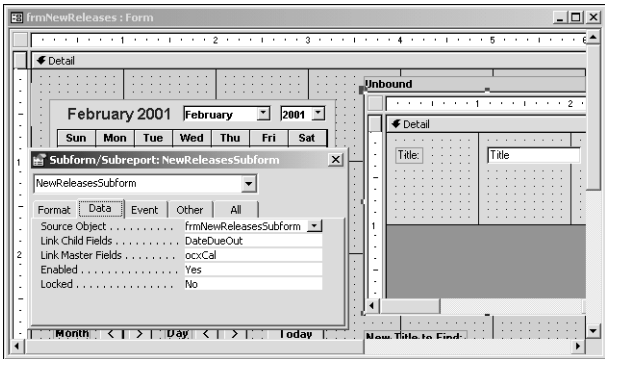

FIGURE B.6
Being able to link a subform to a Calendar control is very handy.

Before going into the sample code found in AppB.mdb, let's look at some of the properties, methods, and events found on the Calendar control.

Understanding the Calendar Control's Properties

Throughout this appendix and Chapter 14, only select properties, methods, and events are mentioned for each ActiveX control because there are too many to cover completely in two chapters. The properties, methods, and events examined here are those deemed more important—or more likely to be interesting—to you. For complete listings of the properties, methods, and events for each ActiveX control, examine each control's help file.

Table B.1 lists some of more useful properties of the Calendar control. You can follow this table by referring to the property sheet shown in Figure B.7. The next two sections list the methods and events used within the Calendar control.

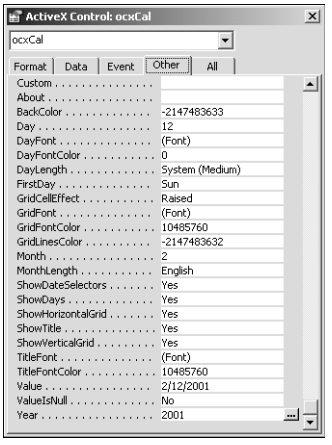

FIGURE B.7
The Calendar control has many of its own properties.

TABLE B.1 Important Calendar Controls and Their Functions

Property	*Description*

`DayFont` and `GridFont`	Set the font for the display of the day titles and grid text (respectively) on the calendar. These two properties actually consist of other properties, such as `Bold`, `Italic`, and so on. When the Builder button is clicked, the Calendar control property sheet appears. When you address the property in code—for example, the ocxCal Calendar control—uses the following syntax:
	`ocxCal.DayFont = NameOfFont`
	`ocxCal.DayFont.Name = NameOfFont`
	`ocxCal.DayFont.Size = SizeOfFont`
	`ocxCal.DayFont.Bold = True|False`
	`ocxCal.DayFont.Italic = True|False`
	`ocxCal.DayFont.Underline = True|False`
	`ocxCal.DayFont.Strikethrough = True|False`
	The same syntax can be used for `GridFont`.

`DayFontColor` and `GridFontColor`	Set the colors for the display of the day titles and the grid, respectively.

TABLE B.1 Continued

Property	*Description*
`ShowDateSelectors`	When set to `True`, displays combo boxes on the top of the calendar, allowing users to pick the month and year.
`ShowTitle`	When set to `True`, displays the month and year in whatever font and color are set by `TitleFont` and `TitleFontColor`.
`DayLength`	Lets you specify how to display the `Day` title, and whether to display it always in English (or whatever language the system is set to use). For example, for August, `English (Medium)` would be Mon and `English` would be Monday. The other two choices are System and System (Medium).
`MonthLength`	Affects the length displayed for the month in the title, and whether to display it always in English (or whatever language the system is set to use). For example, for August, `English (Medium)` would be Aug and `English` would be August. The other two choices are System and System (Medium).
`GridLinesFormat`	Specifies the format for the gridlines: `Raised`, `Sunken`, or `Flat`.
`GridLinesColor`	Specifies the color of the lines when `GridLinesFormat` is set to `Flat`.
`Day`, `Month`, and `Year`	When incremented or decremented by one, allows you to cycle through a week, a month, or a year. These properties are actually most useful to manipulate during runtime.
`Value`	Retrieves the date highlighted or moves the highlight on the calendar when set.
`ValueIsNull`	Allows you to not have a date chosen on the calendar.

TIP

When set to `True`, the `ShowDateSelectors` property is useful as the title. However, you then need to set the `ShowTitle` property to `False` to avoid redundancy. If you don't want users to be able to choose the year and month, set `ShowDateSelectors` to `False` and `ShowTitle` to `True`.

A few more properties are available for use with the Calendar control. For a complete list of properties for the Calendar control, look in the Object Browser. Follow these steps:

1. Open the VBE via Tools, Macro, Visual Basic Editor and choose View, Object Browser.
2. Select MSACAL for the library to examine.
3. Select the Calendar class from the list of classes.

NOTE

ActiveX libraries aren't loaded into the object browser unless your database/Project uses that particular ActiveX control. So if you aren't using the Calendar control and haven't specified a reference to its library, the object browser won't have MSACAL available.

Understanding the Calendar Control's Methods

The Calendar control actually uses a limited number of methods, all listed in Table B.2.

TABLE B.2 Calendar Methods and Their Functions

Method	*Description*
`AboutBox`	Displays the About box for the Calendar control
`NextDay`	Increments the `Value` property by a day, taking care of the calendar fixup for months and years (*calendar fixup* means that the calendar graphic will show the correct month and year)
`NextWeek`	Increments the `Value` property by a week, taking care of the calendar fixup for months and years
`NextMonth`	Increments the `Value` property by a month, taking care of the calendar fixup for years
`NextYear`	Increments the `Value` property by a year
`PreviousDay`	Decrements the `Value` property by a day, taking care of the calendar fixup for months and years
`PreviousWeek`	Decrements the `Value` property by a week, taking care of the calendar fixup for months and years
`PreviousMonth`	Decrements the `Value` property by a month, taking care of the calendar fixup for years
`PreviousYear`	Decrements the `Value` property by a year
`Refresh`	Repaints the Calendar control
`Today`	Sets the `Value` property to the current system date

NOTE

All methods except `AboutBox` cause the calendar to repaint.

When you use the Calendar control, forms that allow users to move around on the calendar will use most of these methods.

On the `frmNewReleases` form, many methods listed here are used in event procedures. Four methods—`PreviousDay`, `NextDay`, `PreviousMonth`, and `NextMonth`—are used with command buttons. To see an example of using the `PreviousDay` method, follow these steps:

1. Open the `frmNewReleases` form in Design mode.
2. Choose Code from the View menu. The module editor appears with the Declarations section of the `frmNewReleases` form module displayed.
3. From the Objects drop-down list, choose the `cmdPreviousDay` command button. The procedure, `cmdPreviousDay_Click()`, appears in the module editor, as follows:

```
Private Sub cmdPreviousDay_Click()
   Me!ocxCal.PreviousDay
End Sub
```

By viewing a few of the command buttons' `OnClick` events on the `frmNewReleases` form, you also can see examples of the other methods. The following examples show the command buttons that use the `Next` methods:

```
Private Sub cmdNextDay_Click()
   Me!ocxCal.NextDay
End Sub

Private Sub cmdNextMonth_Click()
  Me!ocxCal.NextMonth
End Sub
```

You can see by these examples, and when you run the form, that the Calendar control does a lot of the work by fixing up the months and years, thus repainting itself.

Understanding the Calendar Control's Events

The Calendar control supports these standard Access events: `Click`, `DblClick`, `GotFocus`, `KeyDown`, `KeyPress`, `KeyUp`, `LostFocus`, `BeforeUpdate`, and `AfterUpdate`. These events are triggered in the same way other events are triggered. Two events specific to the Calendar control, `NewMonth` and `NewYear`, are triggered when the date in a new month or new year is clicked.

Programming VBA with the Calendar Control

Some VBA code has already been shown with the events displayed. Another example of programming the Calendar control is the New Title to Find text box on the bottom of the `frmNewReleases` form. The actual manipulation of the Calendar control `ocxCal` is performed in this single line of code:

```
Me!ocxCal.Value = snpNewReleases!DateDueOut
```

Listing B.1 shows the complete code attached to the `txtTitleToFind` text box.

LISTING B.1 WebB.mdb: Updating `ocxCal` with the Date of the Title Found

```
Private Sub txtTitleToFind_AfterUpdate()
   Dim dbLocal As Database, snpNewReleases As Recordset
   On Error GoTo Error_txtTitleToFind_AfterUpdate
   Set dbLocal = CurrentDb()
   Set snpNewReleases = dbLocal.OpenRecordset("tblNewReleases", _
      dbOpenSnapshot)

   snpNewReleases.FindFirst "[Title] = '" & txtTitleToFind & "'"
   If snpNewReleases.NoMatch Then
      Beep
      MsgBox "New Title not found!", acCritical, "Find Title Error"
   Else
      Me!ocxCal.Value = snpNewReleases!DateDueOut
   End If

   snpNewReleases.Close
   dbLocal.Close

Exit_txtTitleToFind_AfterUpdate:
   Exit Sub

Error_txtTitleToFind_AfterUpdate:
   MsgBox Error$
   Resume Exit_txtTitleToFind_AfterUpdate
End Sub
```

The code in Listing B.1 performs these steps:

1. The routine opens the current database.
2. It opens the `tblNewReleases` table as a snapshot recordset.
3. The routine looks for the text entered in `txtTitleToFind`.

4. It displays an error message if the text isn't found; otherwise, it sets the `Value` property of the Calendar control `ocxCal` to the `DateDueOut` field for the title.
5. It closes the variables.

Figure B.8 shows the `frmNewReleases` form just after a title is found.

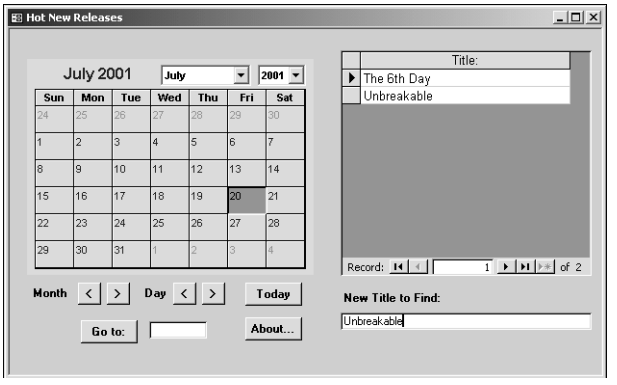

FIGURE B.8
It's easy to perform impressive tasks with very few steps by using the Calendar control.

You can use the Calendar control in several other ways. This is just one simple but powerful example, because you can control a subform with the Calendar control by using very little programming.

Using a Standard Interface with the Common Dialog Control

One ActiveX control included in the MOD is the Common Dialog. With this control, you can create the standard dialogs used for various Windows system tasks, including the following:

- Opening a file
- Saving a file under a new name
- Choosing a color
- Choosing a font
- Printing a file
- Running Winhelp.exe

You formerly had to use Windows API calls to access the Common Dialog control. As of Access 95, ActiveX controls allow you to access the same functionality without the hassle of setting up for the API call.

NOTE

To use the Common Dialog control, you must have the Microsoft Office Developer, Visual Basic 5 or 6, or Visual InterDev, Enterprise Edition. This control isn't available in the standard Microsoft Access. When you install the MOD, this control and others are registered automatically.

Let's take an overview of how to use the Common Dialog control, and then examine the following examples:

- Using the ActiveX control to locate the back end to the World Wide Video Application (Open File dialog)
- Setting a new default printer for the World Wide Video Application (Printer dialog)

After it's placed on a form, the Common Dialog control isn't very impressive visually. The Calendar control, on the other hand, looks great because you can actually see a calendar in Design mode. You can see in Figure B.9 the Common Dialog control in the upper-left corner of the `frmWindowsCommonControlsRichTextbox` form in the Chap14.mdb database, which you can find on this Web site. Even worse, at runtime you can't even see the control until it's activated with one of the `Show` methods (discussed shortly).

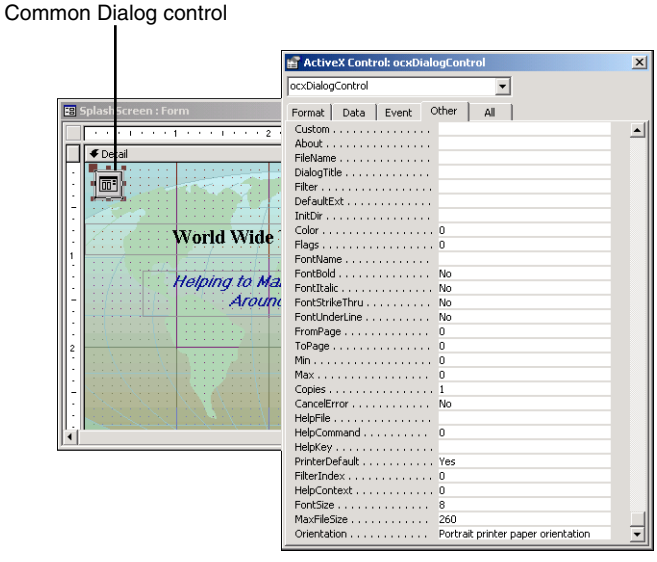

FIGURE B.9

As simple as this control looks in Design mode, the Common Dialog ActiveX control is very powerful.

You can also see from Figure B.9 that quite a few properties are specific to the Common Dialog control. Before you panic, keep in mind that different dialogs use the various properties. A better way to look at the properties is through the specific Common Dialog properties sheet. To access this property sheet, double-click the control. Figure B.10 shows a view of the tabs on the Common Dialog control-specific property sheet. You can even see the different tabs that represent the types of dialogs available. Because so many properties are available to use, specific properties are covered for two examples later in this appendix.

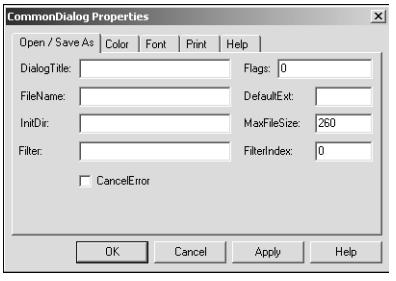

FIGURE B.10
Dealing with properties on the control's property sheet is easier with the Common Dialog control.

You can't activate the control except when using the `Show` methods, including `ShowOpen`, `ShowSave`, `ShowColor`, `ShowFont`, `ShowPrint`, and `ShowHelp`. In generic terms, these steps set up a Common Dialog ActiveX control, with the following section showing a specific example of these steps:

1. Place a Common Dialog control on the form by using the method described earlier in the section "Placing an ActiveX Control on a Form."
2. By using either the Access property sheet or the control's property sheet, set the necessary properties for the dialog type you want to use. You can also use code to set the properties so that a control can be used for more than one purpose (discussed in the next section).
3. In an event procedure, call the `Show` method of your choice.

Locating a File with the Common Dialog Control

Locating a file with the Common Dialog control is very straightforward. You can see this in the following code for the click event of the `cmdLoad` command button (with the label "Load File"), on the `frmWindowsCommonControlsRichText` form.

```
Private Sub cmdLoad_Click()
   Me!ocxCommon.Filter = "Rich Text Format files|*.rtf"
```

```
   Me!ocxCommon.ShowOpen
   Me!ocxRichText.LoadFile Me!ocxCommon.Filename, 0
End Sub
```

The filter property is the only property being set. You can add more filters just by adding the pipe (|) symbol to the end of the current filter string, followed by the additional description and skeleton. The following are some other properties that you can use with the Common Dialog's `ShowOpen` method:

Property	*Description*
`FileName`	Contains the name of the file you want to locate. This property also contains the full path of the file located with the control.
`InitDir`	Sets the initial folder at which you want the dialog to point.
`DialogTitle`	Displays the title at the top of the dialog.
`Filter`	Uses the skeletons (such as *.* or ???.doc) that DOS uses, including wildcards.
`CancelError`	Causes an error to occur when Cancel is clicked—or not clicked. `Err.Number` is set to 32755 when Cancel is clicked.

Figure B.11 shows the dialog in action. This process is just one of many of an unending number of ways to use the Common Dialog ActiveX control, even with just the `ShowOpen` method.

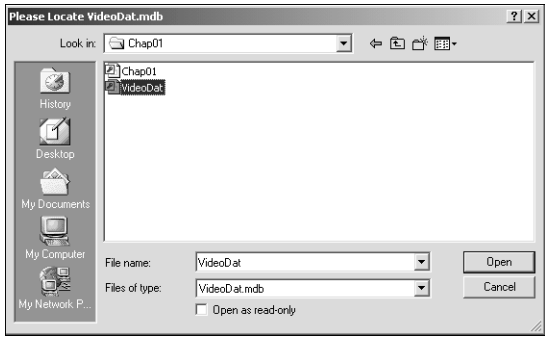

FIGURE B.11

Use the Common Dialog ActiveX control to create standard user interfaces to locate files.

Changing the Default System Printer with the Common Dialog Control

Usually when you create a report, you want it to use the default Windows printer. You can switch default printers in a number of ways. Some applications, such as Microsoft Word, allow you to change the default printer when you choose the Print command. Another way to change the default printer is by choosing Settings, Printers off the Start menu.

When dealing with users, you don't want to make them perform these other steps. The ideal way to deal with setting the printer is to have users either set a new default printer at report time or just go to a system utilities form once to handle it. Figure B.12 shows the latter choice in Design mode. This particular utility form, `ap_SystemUtilities`, is found in VideoApp.mdb(ADO) on this Web site. (A *system utility form* contains various utilities used for the system or the current application.) This example requires no setting of properties and has only one method call: `ShowPrinter`.

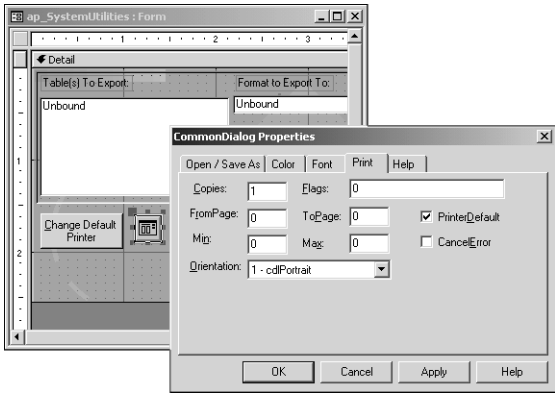

FIGURE B.12

This form contains both the command button and Common Dialog control used for changing the default printer.

By looking at the event procedure behind the `cmdChangePrinter` `OnClick` event, you see the following code:

```
Private Sub cmdChangePrinter_Click()
     Me!ocxChangePrinter.ShowPrinter
End Sub
```

That's all there is. You place the control on the form, create an event procedure on the command button's `OnClick` event, and place the preceding code lines. Figure B.13 shows the Print dialog that appears when you click the command button.

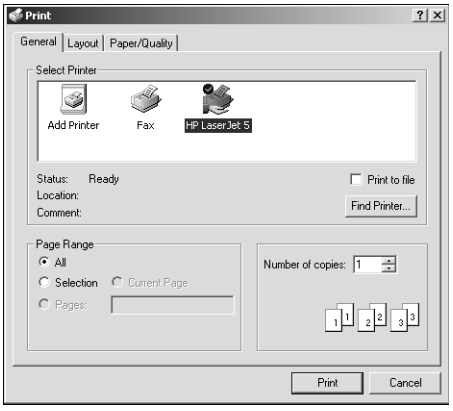

FIGURE B.13
The Common Dialog ActiveX control displays the Print dialog with just one method.

Although properties aren't needed for this example, Figure B.14 shows the Common Dialog control property sheet's Print page.

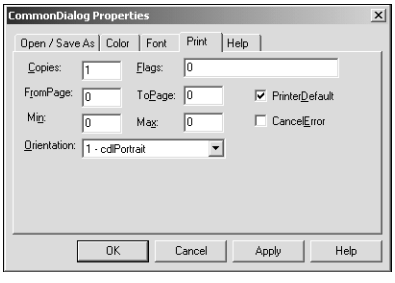

FIGURE B.14
The printer-specific properties allow users to specify how they want to set up their printer at runtime.

Now that you've seen two ways of using the Common Dialog ActiveX control, try the other `Show` methods mentioned on your own, including `ShowSave`, `ShowColor`, and `ShowFont`.