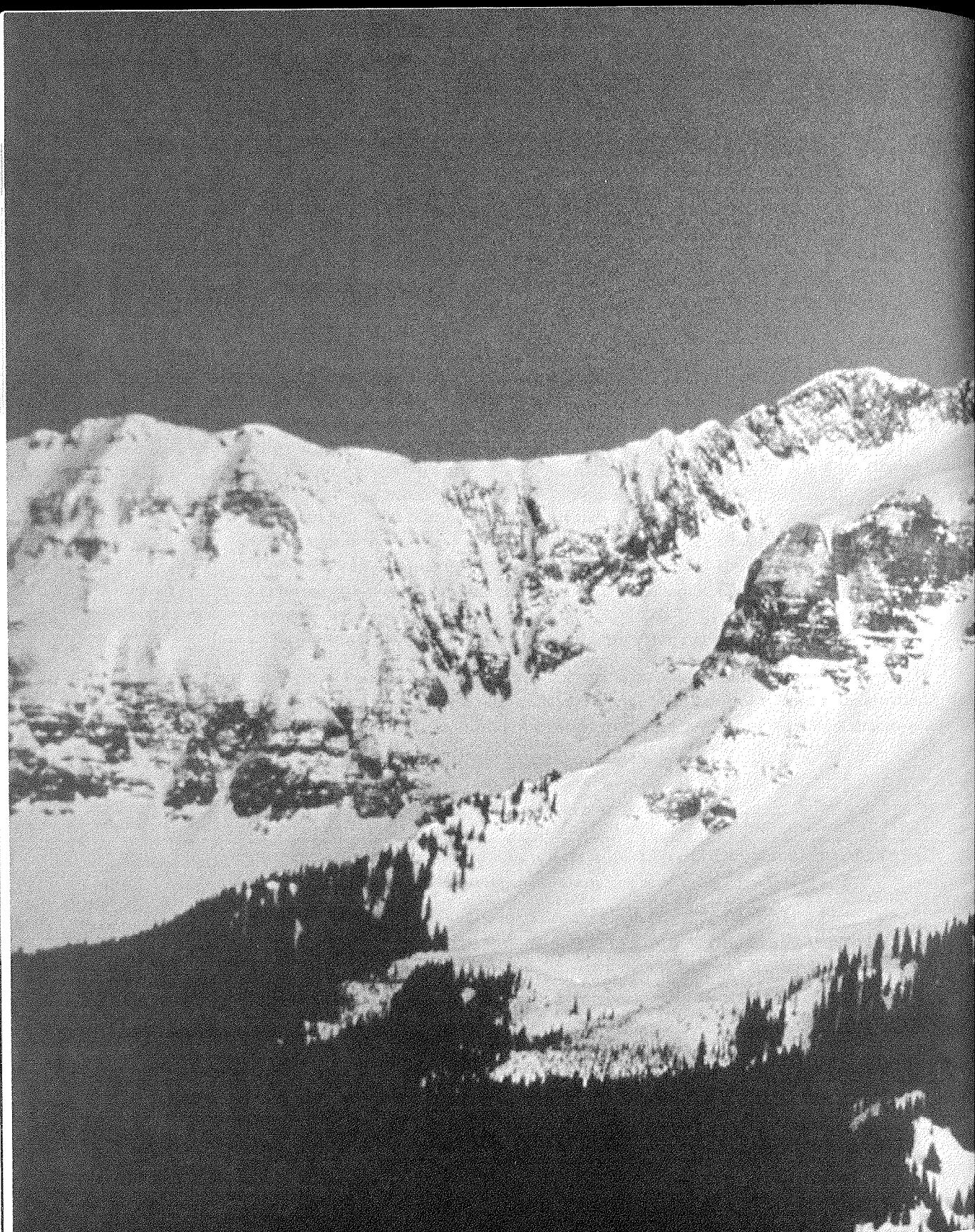
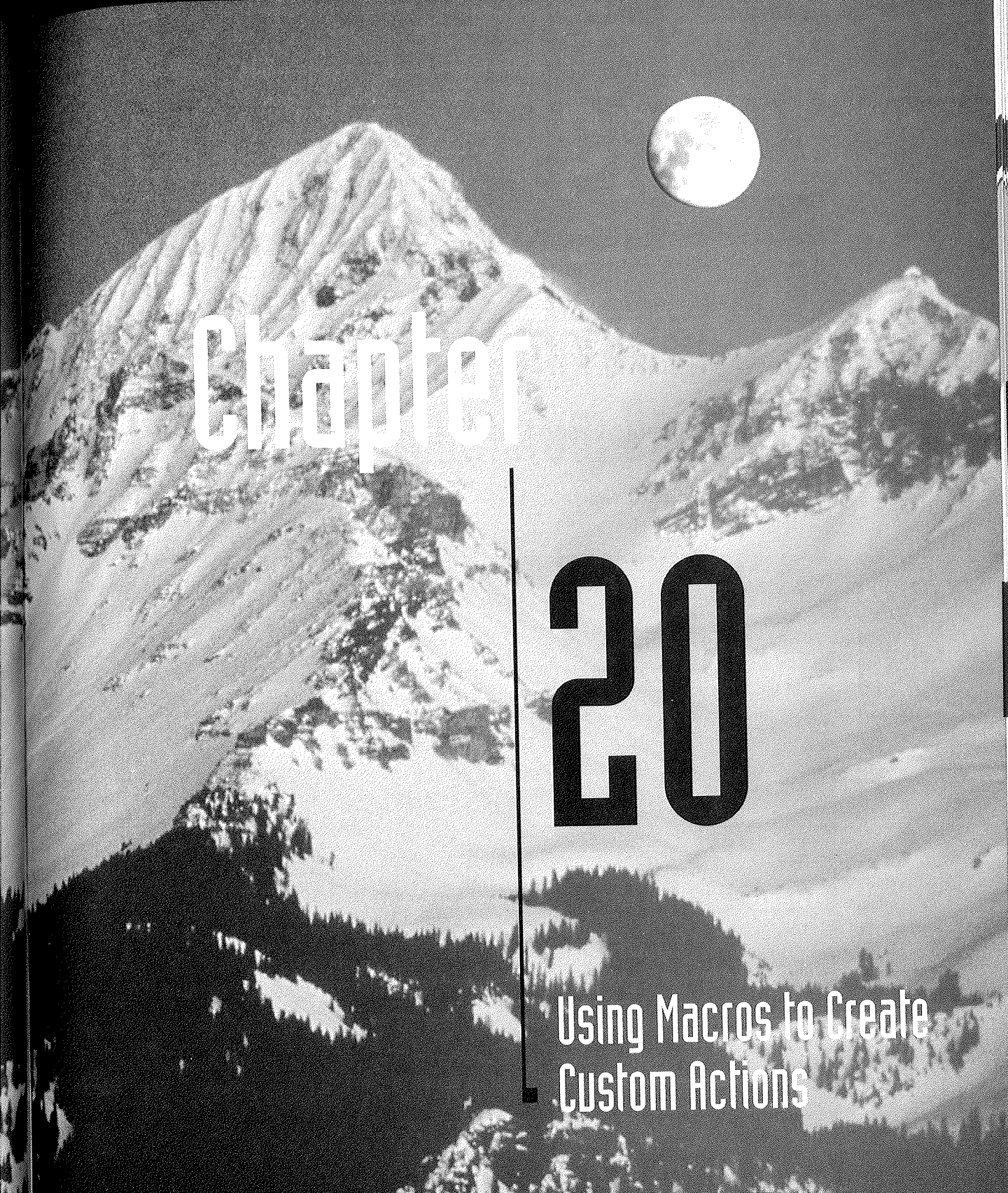
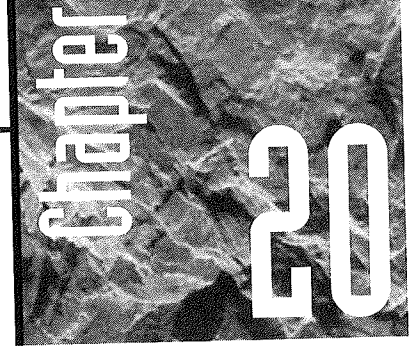




Chapter 20

Using Macros to Create Custom Actions

FEATURING



Using Macros to Create Custom Actions

The easiest way to create a control and a custom action for that control is to use the Control Wizards we discussed in Chapter 19. Adding a hyperlink to a form or report is another easy way to create a control that performs a simple action like opening a form or report. But as you develop more sophisticated applications, you'll probably want to define custom actions that are more complex than the actions you can set up with a hyperlink or the Control Wizards.

When you can't get a Control Wizard to create the exact action you want to perform, you can use either of these two alternative techniques to define a custom action:

- Create a macro
- Write a Visual Basic procedure

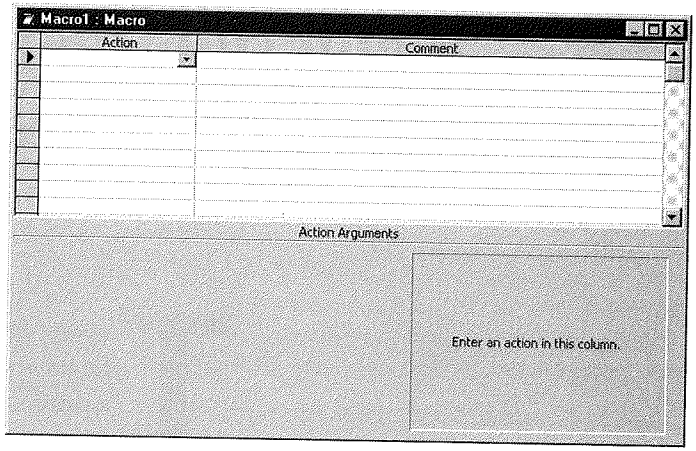
Visual Basic requires that you type long strings of commands very, very accurately. Macros, however, let you define actions using the simpler point-and-click approach. Unless you already happen to be a Visual Basic whiz, you'll probably find that macros are by far the quickest and easiest way to define a custom action in your application. In this chapter we'll focus on macros.

How to Create a Macro

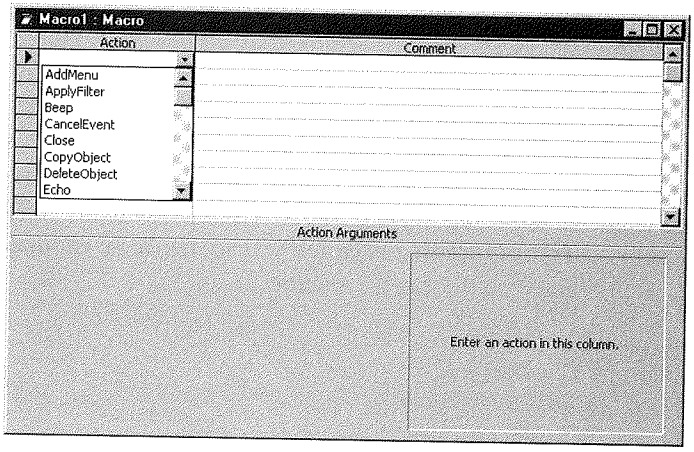
The mechanics of creating a macro are fairly straightforward:

1. Click on the Macros tab in the database window.
2. Click on the New button. You're taken to a *macro sheet* that's tentatively named Macro1, as in Figure 20.1.

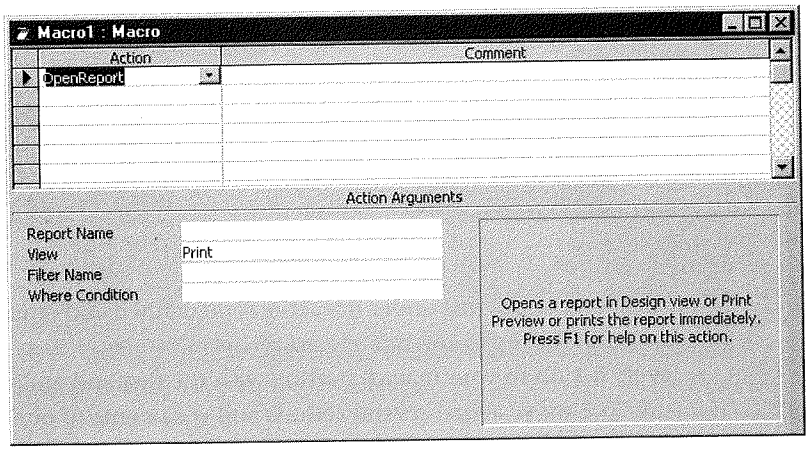
FIGURE 20.1
A new, blank macro sheet.



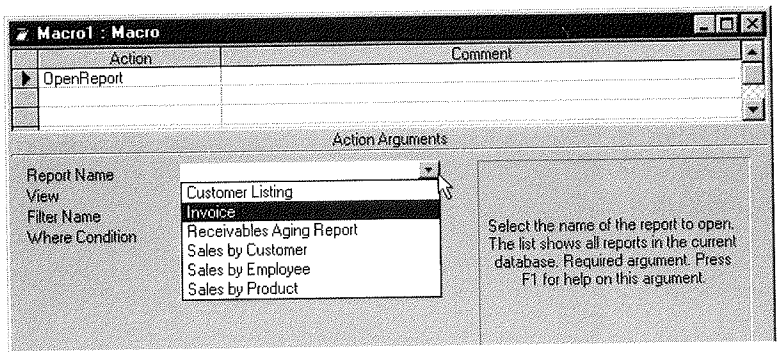
3. Click on the drop-down list button in the Action column. You'll see a partial list of possible actions, as below. (You can use the scroll bar, the ↓ key, or type a letter to scroll down the list.)



4. Choose whichever action best describes what you want the macro to do. For example, below we chose OpenReport (an action that will cause the macro to open up a report in this database). Notice that in addition to the word OpenReport appearing in the action column, the lower portion of the window shows some *action arguments* to be filled in. And the hint box tells us what the selected action will do.

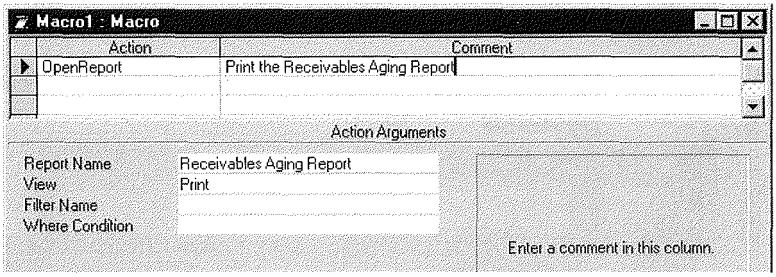


5. Fill in the selections under Action Arguments. For example, below we clicked next to Report Name and can now use the drop-down list to choose which report we want the macro to open. Note too that the hint box is now giving us information that's specific to the Report Name argument that we're filling in.



You need to fill in each *required argument* for your action. You can leave *optional arguments* blank if you wish. To determine whether an argument is required or optional, click on the argument and read the hint box to the right.

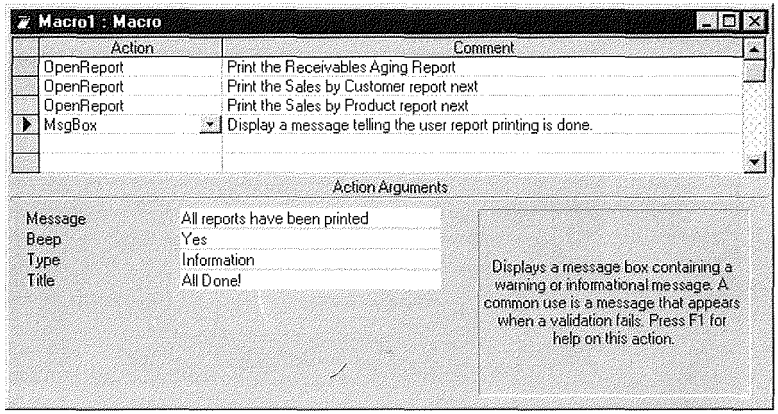
6. (Optional) Click just to the right of the action you chose and type in a plain-English description of what that action does. Note the comment next to our OpenReport action below.



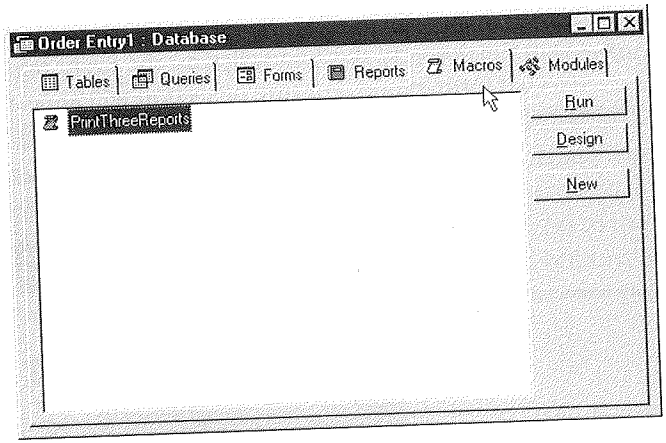
7. Click on the cell just under the action you defined and repeat steps 3 to 8 to define additional actions for this macro. When the macro is executed, it will perform every action in your macro, starting with the first and ending with the last. In Figure 20.2 we've added several actions and comments to our sample macro.
8. Close the macro when you are finished and give it a name. (Choose File ➤ Close or click the × button in the upper-right corner of the macro sheet window.) You can choose Yes to save your changes and enter a new, more descriptive name for your macro.

FIGURE 20.2

A macro with several actions defined.



The name you assigned to the macro appears in the database window whenever the Macros tab is active. For example, we named our macro PrintThreeReports when closing it. So now that name appears in the database window as you can see below:.



Determining When a Macro Plays

After you've created a macro, your next step is to determine *when* the macro will perform its actions. For example, you might want the macro to play:

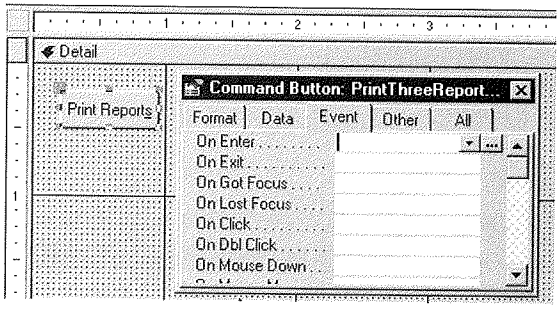
- As soon as the user clicks on a particular command button on a form
- Right after the user changes the data in some control
- As soon as the user opens a particular form or report

As we'll discuss later, you can also have the macro play when the user first opens the database (see "Creating a Macro to Run at Startup" later in this chapter). Or you can assign the macro to an option in a custom toolbar (Chapter 23) or menu (Chapter 24) that you've created. Your options for *when* the macro is triggered are virtually limitless. For now, let's just take a look at how you'd assign a macro to a report, form, or a particular control on a form:

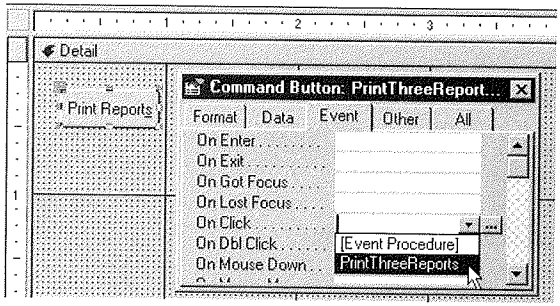
1. In design view, open the form or report that you want to have trigger the macro.
   - If you want the form or report as a whole (i.e., open/close the form, open/close the report) to trigger the macro, choose Edit ➤ Select Form or Edit ➤ Select Report.

- If you want a particular control in a form to trigger a macro, select that control by clicking on it once. (If you haven't created the control yet, you can do so right on the spot using the toolbox.)
- If you want a particular section of a form or report to trigger the macro, click on the section bar in design view.

2. Open the property sheet and click on the Event tab. All the possible events for the selected form, report, control, or section will appear, as in the example below.



3. Click on the property that you want to have trigger the macro. For example, if you're assigning the macro to a command button and want the macro to run when the user clicks on that button, click on the On Click property.
4. Choose the name of the macro you want to execute from the drop-down list that appears. For example, below we're assigning the PrintThreeReports macro to the On Click property of a button we created earlier.



And now you're done. It's a good idea to save and close the form at this point, before you test the macro. Choose File ➤ Close and Yes when asked about saving your changes.

Running the Macro

To run the macro, you need to play the role of the user by triggering whatever event activates the macro. For example, if you assigned the macro to the On Click property of a command button, you need to open the form (in form view) that holds the button, then click on that button just as the user would. If you assigned the macro to the On Open property of a form, all you need to do is open the appropriate form.

TIP

You can run a macro simply by clicking on its name in the database window and then clicking on the Run button. This technique is fine for testing a macro. But when creating a custom application, you want the user to have easier access to the macro, which is the reason that macros are typically attached to command buttons on forms.

All That in a Nutshell

Whether you're an absolute beginner or are accustomed to creating macros in other products, creating Access macros will probably take some getting used to. Here's a summary of the step-by-step instructions for creating a macro and assigning it to an event:

- Click on the Macros tab in the database window and click on New.
- Choose an action from the Action column.
- Fill in the required arguments for that action. You can create several actions within a single macro.

NOTE

Remember that unlike some other products such as Word or Excel, Access does not include a macro recorder to automate the process of defining macro actions.

- Close and save the macro, giving it a name that will be easy to remember later.
- Open the form or report that you want in design view to "trigger" the macro.
- Select the control that will trigger the macro (or choose Edit ➤ Select Form or Edit ➤ Select Report if you want a form or report event to trigger the macro).
- Open the property sheet and click on the Event tab.
- Click on the specific event that you want to have trigger the macro and then choose the macro name from the drop-down list that appears.
- Close and save the form.

Once you've done all that, the macro will play every time you trigger the event to which you assigned the macro. The macro will not run (ever) in design view. You must open the form in form view or print the report, as a user would, in order to make the macro play its actions.

Summary of Macro Actions

Once you understand the mechanics of creating a macro and attaching it to some event, you still have to work through the mind-boggling stage of figuring out what you can and *can't* do with a macro. To help you sort through the overwhelming number of possibilities, here's a summary of every macro action that is available when you click on the drop-down list in the Action column of the macro sheet.

AddMenu Adds a menu to a custom menu bar (see Chapter 24).

ApplyFilter Applies a filter, query, or SQL WHERE clause to a table, form, or report. Often used to filter records in the table underlying the form that launched the macro. You can use the ShowAllRecords action to clear the filter.

Beep Just sounds a beep.

CancelEvent Cancels the event that caused the macro to execute. For example, if a BeforeUpdate event calls a macro, that macro can test data and then execute a CancelEvent to prevent the form from accepting the new data.

Close Closes the specified window. Typically used to close a form.

CopyObject Copies the specified object to a different Access database, or to the same database but with a different name.

DeleteObject Deletes the specified object, or the currently selected object in the database window if you don't specify an object.

Echo Hides, or shows, on the screen the results of each macro action as the macro is running.

FindNext Repeats the previous FindRecord action to locate the next record that matches the same criterion.

FindRecord Locates a record meeting the specified criterion in the current table (the table underlying the form that launched the macro).

GoToControl Moves the focus (cursor) to the specified field or control on a form.

GoToPage Moves the focus to the specified page in a multipage form.

GoToRecord Moves the focus to a new record, in relation to the current record (e.g., Next, Previous, First, Last, New).

Hourglass Changes the mouse pointer to a "wait" hourglass (so the user knows to wait for the macro to finish its job).

Maximize Expands the active (current) window to full-screen size.

Minimize Shrinks the active (current) window to an icon.

MoveSize Moves and/or sizes the active window to the position and measurement you specify in inches (or centimeters if you've defined that as your unit of measure in the Windows Control Panel).

MsgBox Displays a message on the screen.

OpenForm Opens the specified form and moves the focus to that form.

OpenModule Opens, in design view, the specified Visual Basic module.

OpenQuery Opens a Select, Crosstab, or Action query. If you use this to run an Action query, the screen will display the usual warning messages, unless you precede this action with a SetWarnings action.

OpenReport Prints the specified report or opens it in print preview or design view. You can apply a filter condition with this action.

OpenTable Opens the specified table in datasheet, design, or print preview view.

OutputTo Exports data in the specified object to HTML (.html), Microsoft ActiveX Server (.asp), Microsoft Excel (.xls), Microsoft IIS (.htx; .idc), rich text (.rtf), or text (.txt) format.

PrintOut Prints the specified datasheet, form, report, or module.

Quit Exits Microsoft Access.

Rename Renames the specified or selected object.

RepaintObject Performs any pending screen updates or calculations.

Requery Forces the query underlying a specific control to be re-executed. If the specified control has no underlying query, this action will recalculate the control.

Restore Restores a minimized or maximized window to its previous size.

RunApp Starts another Windows or DOS program. That application then runs in the foreground, and the macro continues processing in the background.

RunCode Runs the specified Visual Basic Function procedure. (To run a Sub procedure, create a function procedure that calls the Sub and have the macro run that function.)

RunCommand Performs an Access menu command.

RunMacro Runs a different macro. After that macro has finished its job, execution resumes in the original macro starting with the action under the RunMacro action.

RunSQL Runs the specified SQL statement.

Save Saves the specified object, or the active object if no other object is specified.

SelectObject Selects the specified object. That is, this action mimics the act of clicking on an object to select it.

SendKeys Sends keystrokes to Access or another active program.

SendObject Includes the specified database object in an e-mail message.

SetMenuItem Sets the appearance of a command (e.g., "grayed" or "checked" in a custom menu. See Chapter 24).

SetValue Sets a value for a control, field, or property. Often used to auto-fill fields on a form based on some existing data.

SetWarnings Hides, or displays, all warning boxes such as those that appear when you run an action query.

ShowAllRecords Removes an applied filter from the table, query, or form so that no records are hidden.

ShowToolBar Shows or hides a built-in or custom toolbar (see Chapter 23).

StopAllMacros Stops all running macros, turns Echo back on (if it was off), and reinstates warning messages.

StopMacro Stops execution of the currently running macro.

TransferDatabase Imports, exports, or links data in another database.

TransferSpreadsheet Imports, exports, or links data from the specified spreadsheet.

TransferText Imports, exports, or links data from a text file, and can also be used to export data to a Microsoft Word for Windows mail merge data file.

Keep in mind that you can get much more information about each action right on your screen. Just select the action and take a look at the hint box. If you need more information after reading the hint box, press Help (F1).

Executing a Macro Action "If . . ."

You can make any action, or series of actions, in a macro be conditional on some expressions. For example, suppose you want to create a macro that adds 7.75 percent sales tax to a total sale but only if the sale is made in the state of California. That is, if the State field on the current form contains CA, then you want the macro to fill in another field, named SalesTaxRate, with .0775 and use that value in calculating the sales tax and total sale. To illustrate this concept, Figure 20.3 shows a sample form with the appropriate fields, named State, SubTotal, SalesTaxRate, SalesTax, and TotalSale.

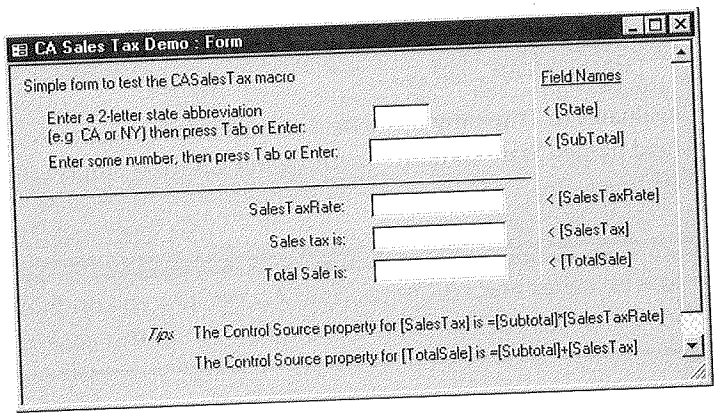
NOTE

Remember that in order to name a field on a form, you need to open the form in design view. Then click on the field you want to name, open the property sheet, and click on the All tab. Then fill in the Name property with whatever name you want to give that field. While you're at it, you can use the Format property to assign a format, such as Currency or Percent, to fields that will contain numbers.

The last two fields on the form are calculated fields. The ControlSource property for the SalesTax field contains the expression

```
= [SalesTaxRate]*[SubTotal]
```

FIGURE 20.3
A sample form containing fields named State, SubTotal, SalesTaxRate, and TotalSale.

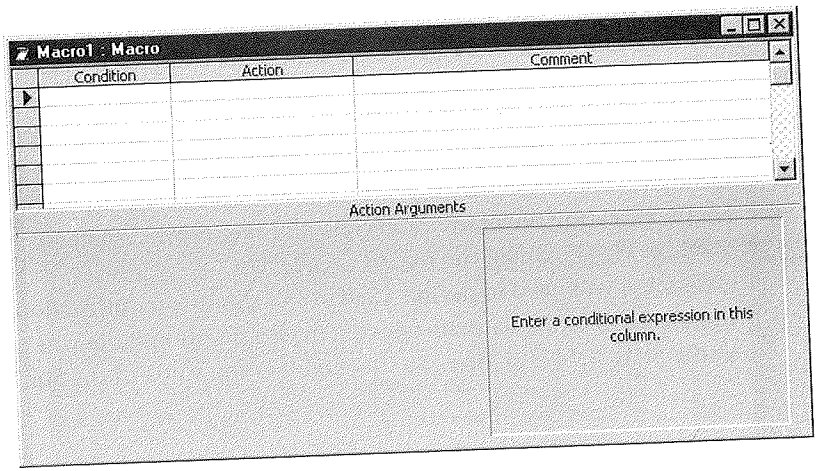


The Control Source property for the TotalSale field contains the expression

```
= [SubTotal]+[SalesTax]
```

After you've created and saved the form, you can create the macro in the normal manner. But if you want to use conditions in the macro, you need to open the Conditions column in the macro sheet. Just create (using New) or open (using Design) any macro sheet. Then click on the Conditions button in the toolbar or choose View ➤ Conditions from the menu bar. A new column titled Condition appears to the left of the existing columns, as in Figure 20.4.

FIGURE 20.4
The Conditions column now visible in the macro sheet.



The condition you type in must be an expression that evaluates to either True or False, usually in the format *something* = *something*. For example, the expression

```
[State] ="CA"
```

evaluates to True only if the field named State contains exactly the letters CA. If the field named State contains anything but CA (or is empty) the expression `[State] ="CA"` returns False.

NOTE

As with other text comparisons in Access, macro conditions are not case-sensitive. So `"ca"` or `"Ca"` or `"cA"` would all match `"CA"` in this case.

An important point to remember is that the condition you specify affects only the action immediately to the right of the condition. If the expression proves True, the action is performed. If the expression proves False, the action is completely ignored. Either way, execution then resumes at the next action in the macro.

TIP

You can repeat the condition in a row by typing three periods (...) into the condition cell immediately beneath the cell that contains the condition. The ... characters mean "apply the condition above to this action."

So let's create the CASalesTax macro now. For starters, we'll have the macro set the SalesTaxRate field to zero. Then the next action will check to see if the State field contains CA. If that's True, that action will put 0.775 into the SalesTaxRate field. The next actions will use the Repaint-Object command to recalculate the calculated controls SalesTax and TotalSale. Figure 20.5 shows the completed macro.

Since you can't see the action arguments for all three macro actions, we've listed them in Table 20.1 in the order in which they appear in the macro. (Leaving empty the action arguments for the RepaintObject action causes the entire object, the form in this example, to be recalculated.)

After creating the macro, you close and save it with whatever name you wish. In this example we've named the macro CASalesTax.

Finally, you need to decide *when* to call this macro into action. In this case we need the macro to recalculate the sales tax in two situations: after the user changes the value in the State field and after the user changes the value in the SubTotal field.

FIGURE 20.5
The CAState Tax macro.

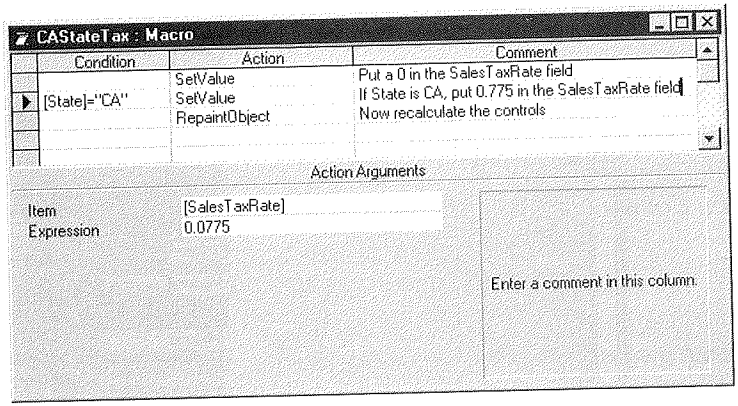
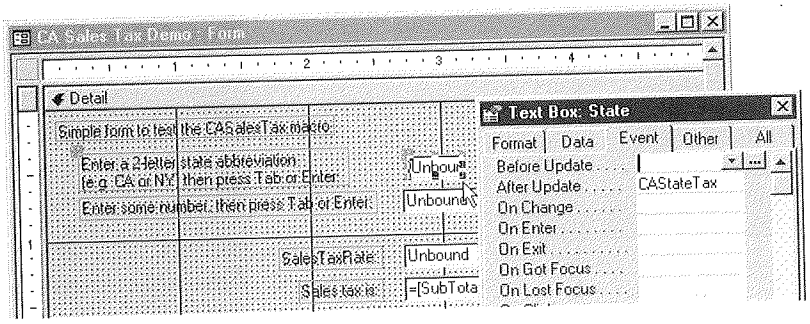


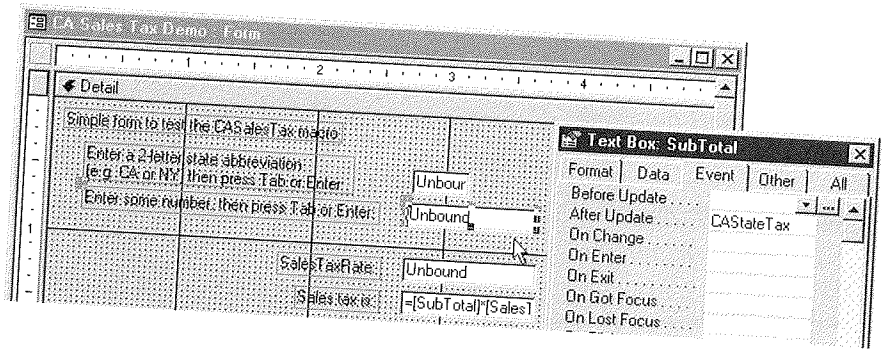
TABLE 20.1: CONDITION, ACTION, AND ACTION ARGUMENTS FOR CA STATETAX MACRO

CONDITION	ACTION	ACTION ARGUMENTS
	SetValue	Item: [SalesTaxRate] Expression: 0
[State]="CA"	SetValue	Item:[SalesTaxRate] Expression: 0.0775
	RepaintObject	

So we open the form in design view, click on the State field, open the property sheet, click on the Event tab, and then assign the CASalesTax macro to the AfterUpdate property for that field, as below:



Then we click on the SubTotal field and also set its AfterUpdate property to the CASalesTax macro.



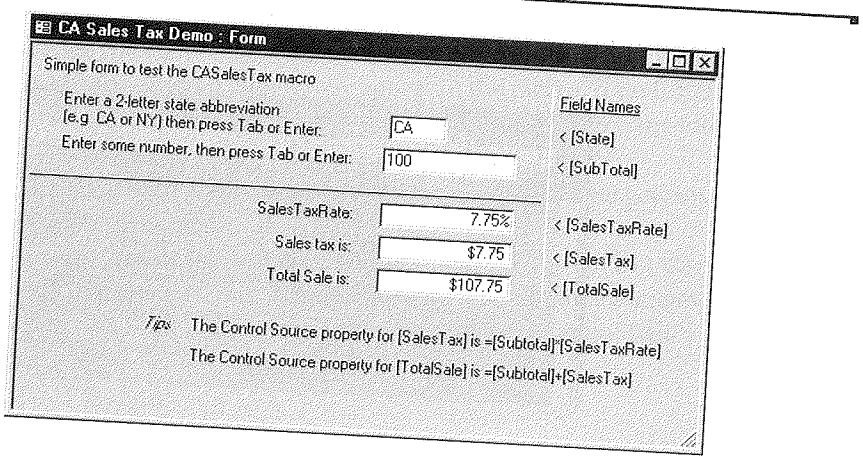
TIP

You can use Ctrl+click to select several controls, and then assign a macro to the same event on both controls at the same time.

Once those steps are complete, we can save the form and open it in form view. Then whenever we enter (or change) values in either the State or Subtotal fields (and press Tab or Enter to complete the entry), the SalesTaxRate, SalesTax, and TotalSale fields recalculate automatically. In the example shown in Figure 20.6, we entered CA in the State field and 100 in the Subtotal field. As you can see, the three fields beneath show the correct sales tax rate, sales tax amount, and total sale.

FIGURE 20.6

The macro and calculated controls automatically display the correct SalesTaxRate, SalesTax, and TotalSale.



NOTE The AfterUpdate event is triggered only when you change the contents of a field and then move to another field.

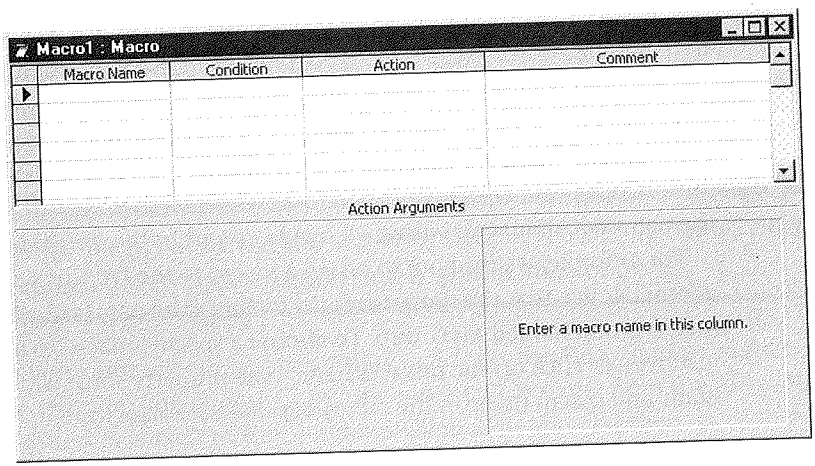
Incidentally, the field names and Tips that you see in Figure 20.6 are for your information only. They are just labels that have no effect on how the form functions. In "real life" you wouldn't have any reason to show that information to a user.

Creating Macro Groups

A macro sheet can actually contain several macros, each with its own macro name. Grouping several macros into a sheet can keep the list of macro names in the database window from becoming too lengthy and unwieldy. A good way to organize your macros is to put all the macros that go with a given form (or report) into a single macro sheet. That way, you can easily find all the macros that go with a particular form.

We often name our macro sheets for the form that triggers the macros in that sheet. For example, if we have a form named Customers, we might create a macro sheet named CustomerFormMacros that contains all the macros used by that form.

Creating a group of macros is a simple process. Just create or open a macro sheet in the usual manner. Then click on the Macro Names button in the toolbar or choose View ➤ Macro Names. A new column, titled Macro Name, appears to the left of the existing columns:

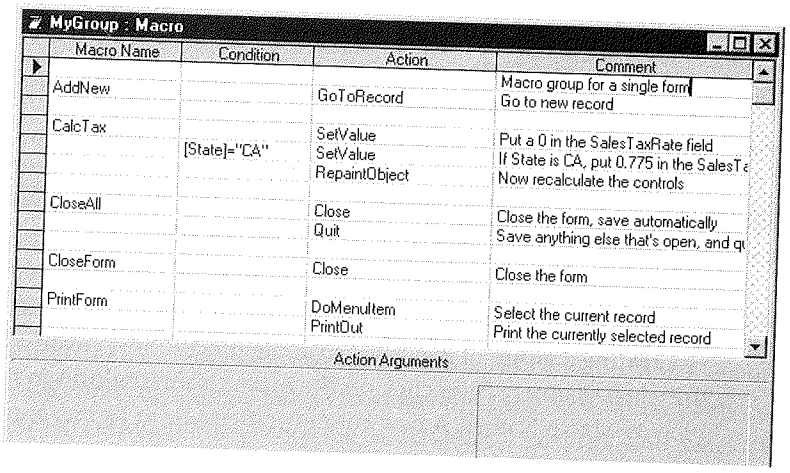


When adding a macro to the macro sheet, you need to type the macro name into the leftmost column. Then type in the first condition (if any), action, and comment in the usual manner. You can add as many actions to the macro as you wish.

Figure 20.7 shows an example with a macro sheet that contains five macros named AddNew, CalcTax, CloseAll, CloseForm, and PrintForm. Access stops running a macro when there are no more actions in the group or when it hits the name of another macro. We've added a blank line between each macro for readability.

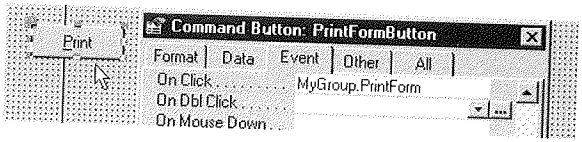
FIGURE 20.7

A macro sheet containing five macros named AddNew, CalcTax, CloseAll, CloseForm, and PrintForm.



Close and save the macro sheet in the usual manner. For example, let's say you decide to name the entire macro sheet MyGroup. Then you can assign macros to events using the standard technique—that is, open the form or report that will trigger a macro in design view. Click on the control that will trigger the macro (or choose Edit ➤ Select Form or Edit ➤ Select Report). Open the property sheet and click on the drop-down list button for the event that you want to assign a macro to. The drop-down list now shows the names of all macros within all macro groups in the format `macrogroupname.macroname`.

For example, we're about to assign a macro to the On Click property of a control on a form. Notice that the drop-down list includes the names of all the macros within the macro group named MyGroup. To choose a specific macro to assign to this event, we just need to click on the macro's name. The property sheet will show the macro group name and macro name in the `macrogroupname.macroname` format, (e.g., `MyGroup.PrintForm`).



Editing Macros

To edit an existing macro, you just need to reopen the macro sheet. Here's how:

- **If you're at the database window**, just click on the Macros tab, click on the name of the macro (or macro group) you want to edit, and click on the Design button.
- **If you're in a form's (or report's) design view**, and want to edit a macro that you've already assigned to an event, just open the property sheet, click on the Event tab, then click on the ... button next to the name of the macro that you want to edit.

When you use the latter method to open a macro group, you'll be taken to the macro group in general, not the specific macro that you assigned to the event. But once you're in the macro sheet, you can easily scroll to the macro that you want to edit.

Changing, Deleting, and Rearranging Macros

Once you're in the macro sheet, you can move, delete, and insert rows using techniques that are virtually identical to the techniques you use in a datasheet:

1. Select a row by clicking on the row selector at the left edge of the row. Or select several rows by dragging the mouse pointer through row selectors or by using Shift+Click.
2. Do any of the following:

   - **To delete the selected row(s)**, press Delete, or right-click on the selection and choose Delete Rows, or choose Edit ➤ Delete Rows from the menus.
   - **To insert a row**, press the Insert (Ins) key, or right-click on the selection and choose Insert Rows, or choose Insert ➤ Rows from the menu bar.
   - **To move the selected row(s)**, click on the row selector again, hold down the mouse button, and drag the selection to its new position.

TIP

If you arrange macros in a macro group in alphabetical order by name, when you open the macro group you can easily find a specific macro.

   - **To copy the selection**, press Ctrl+C, click on the Copy button, right-click on the selection and choose Copy, or choose Edit ➤ Copy from the menu bar. The selection is copied to the Windows Clipboard. You can then use Edit ➤ Paste (Ctrl+V) to paste the copy into the same or another macro sheet.
   - **To undo any of the above changes**, press Ctrl+Z or click on the Undo button or choose Edit ➤ Undo.

Keep in mind that any changes you make to the macro are not saved until you save the entire macro. If you close the macro without saving it, be sure to choose Yes when asked about saving your changes.

Referring to Controls from Macros

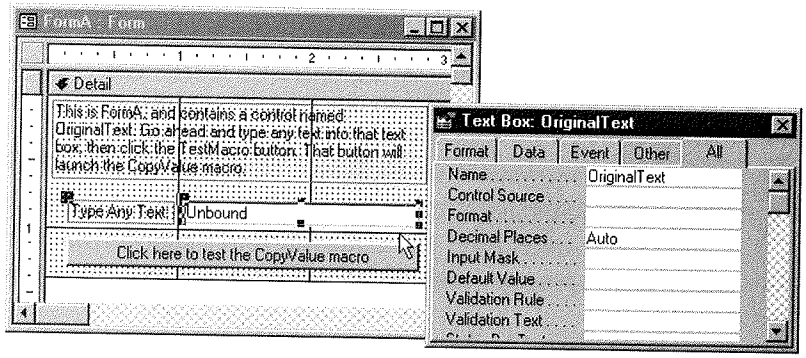
One of the most common uses of macros is to use the SetValue action to fill in a field on a form. We used the SetValue action in an earlier example in this chapter to fill in a field named SalesTaxRate.

When you start to use macros in this way on multiple forms, you need to keep a couple of very important points in mind:

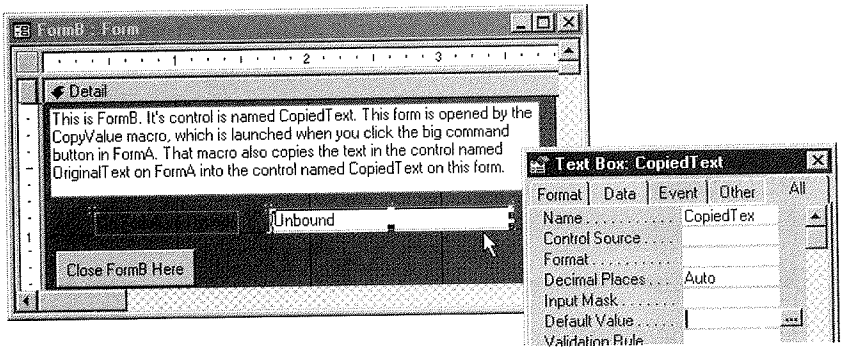
- When referring to a control on some form *other than the form that launched the macro,* you must use the full-identifier syntax (i.e., [Forms]![*formname*]![*controlname*]) to refer to the control.
- Both forms must be open.

The way in which you refer to objects on forms can be one of the most confusing aspects of using macros because if your macro opens a new form, you might think of that form as the "current form." But from Access's perspective, the form that *launched* the macro is the current form, even if that form does not have the focus at the moment. Let's look at a simple example to illustrate this situation.

Let's say you have a form named FormA. The form contains a text box control named [OriginalText], as illustrated below.



You also have a second form, named FormB, that contains a control named CopiedText, as shown below.



Let's say you want to create a macro that you'll launch from FormA. When you launch that macro, you want it to open FormB and take whatever text is in the [OriginalText] control on FormA and copy that text into the [CopiedText] control on Form B.

Figure 20.8 shows the appropriate macro (which we'll refer to as the CopyValue macro from here on out). Currently the cursor is in the SetValue action's cell so you can see the action arguments for that action. Table 20.2 shows the action arguments for both actions (where we omit an action argument, we have left the argument blank in the macro sheet as well).

FIGURE 20.8

The CopyValue macro showing the action arguments for the SetValue action.

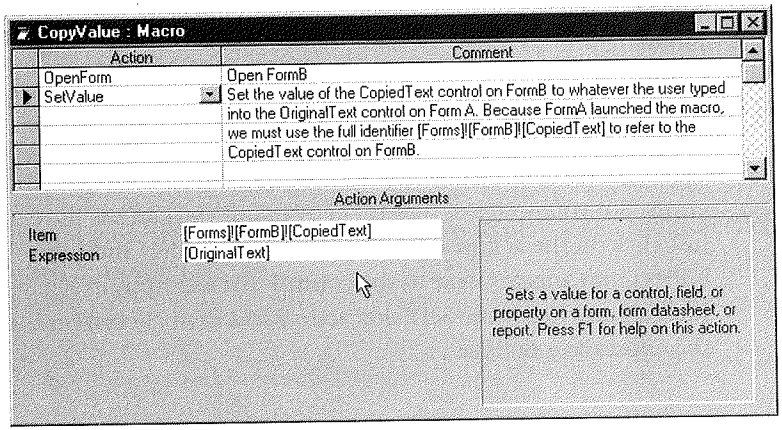


TABLE 20.2: ACTION ARGUMENTS FOR THE COPYVALUE MACRO SHOWN IN FIGURE 20.8

ACTION	ACTION ARGUMENTS
OpenForm	Form Name: FormB
	View: Form
	Data Mode: Edit
	Window Mode: Normal
SetValue	Item: [Forms]![FormB]![CopiedText]
	Expression: [OriginalText]

Notice that we must refer to the [CopiedText] control using the full formal [Forms]![*formname*]![*controlname*], even though the OpenForm action has already opened FormB and FormB has the focus. We need to do so because FormA, not FormB, is the one that *launched* the macro. We can refer to the [OriginalText] control without all the formality because [OriginalText] is the form that launched the macro.

On the other hand, you can always use the full, formal syntax. For example, we could have used these action arguments for the SetValue action, and the macro would still work just fine.

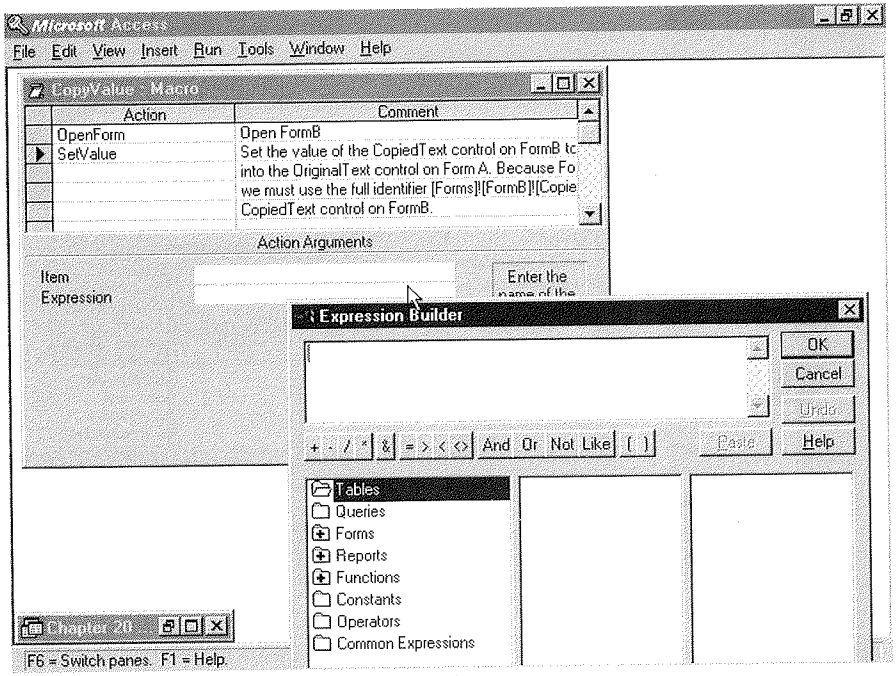
Item:	[Forms]![FormB]![CopiedText]
Expression:	[Forms]![FormA]![OriginalText]

Though a bit more cumbersome, this approach does have one advantage. Because we've referred to forms and controls specifically, we don't need to waste brain cells trying to keep track of which form opened the macro, which form has the focus at the moment, and so forth.

Typing Lengthy Identifiers

Typing those lengthy identifiers is a bit of a task, and they can be prone to typographical errors. But you need not type them by hand. You can use the expression builder instead. Just click on the action argument you want to enter and then click on the build (...) button that appears next to the control. For example, in Figure 20.9 we clicked on the Item argument for the SetValue action and then clicked on the Build button. Notice the Expression Builder.

FIGURE 20.9
Expression builder partially covering the macro sheet.



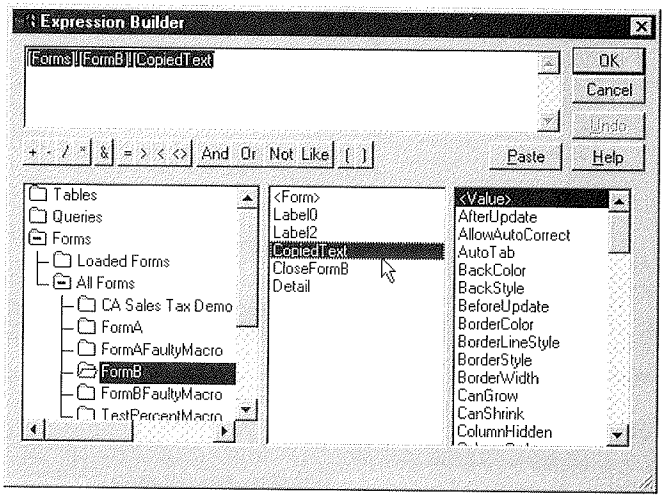
Now we can specify a control simply by working our way down to it. In this case we would double-click on Forms (since the control is on a form) and then double-click on All Forms. Then we would double-click on FormB (since that's the one that contains the control we want to fill) and double-click on CopiedText, the name of the control we want to fill. The top box in the Expression Builder now shows the proper expression for referring to the control (see Figure 20.10). When we click on the OK button, that control is copied into the Item: action argument.

Assigning CopyValue to an Event

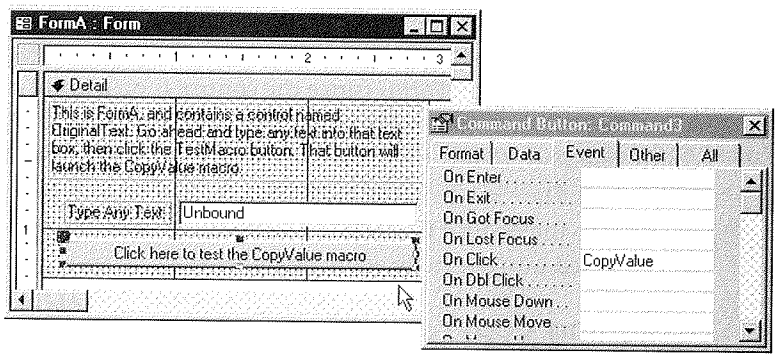
To get back to the macro shown in Figure 20.9, let's assume we save it with the name CopyValue and close it. Now we want that macro to play when the user clicks on the big command button on FormA.

FIGURE 20.10

Here we double-clicked on Forms ➤ All Forms ➤ FormB ➤ CopiedText to build the expression Forms![FormB]![CopiedText].



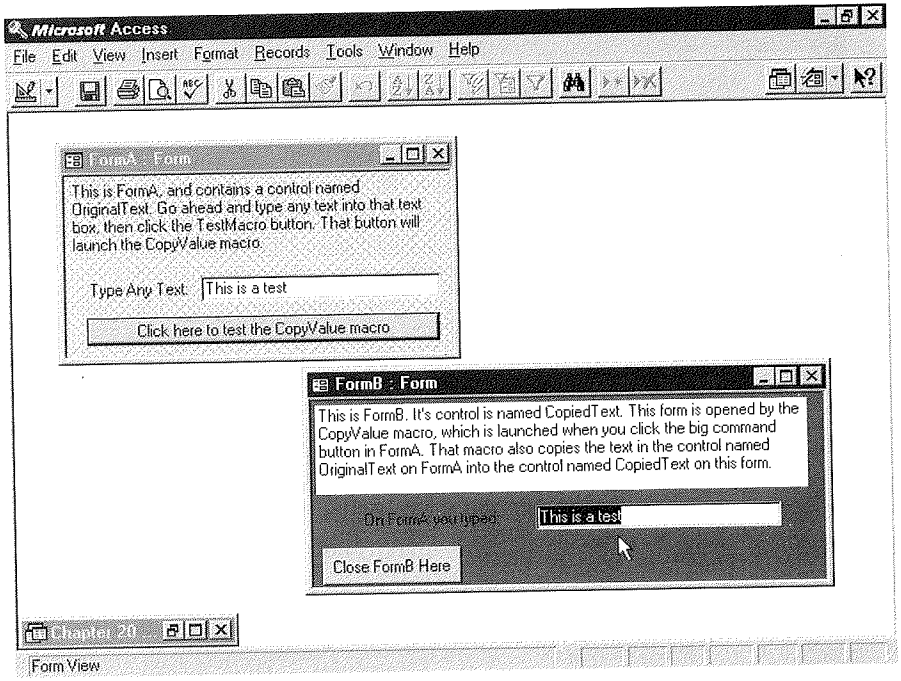
So we need to open FormA in design view, click on the command button, open the property sheet, and click on the Event tab. Then we can click on the On Click property for that control and assign the CopyValue macro to that event as shown below.



To test the macro, we then need to save FormA and close it (and close the FormB and CopyValue macros if they're open). Then open FormA, type in some text, and click on the big command button. The macro will open FormB and copy whatever we typed into the text box on FormA into the text box on FormB, as shown in Figure 20.11.

FIGURE 20.11

We typed a value into FormA and then clicked on the big command button. That button launched the CopyValue macro, which opened FormB and copied the text from FormA into the text box on FormB.



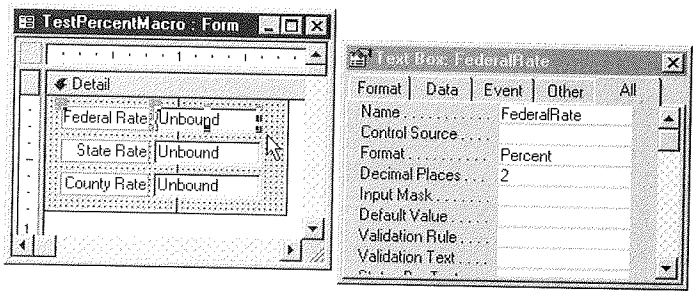
Making More "Generic" Macros

Here's another method for referring to forms and controls from within a macro. Rather than referring to a specific object, you can refer to "whatever object is current at the moment." The expressions you use are as follows:

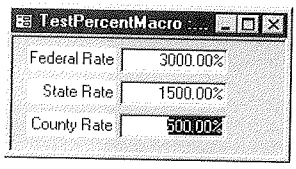
[Screen].[ActiveForm]
[Screen].[ActiveReport]
[Screen].[ActiveControl]
[Screen].[ActiveDatasheet]

Let's take a look at another fairly simple example, using the [Screen].[ActiveControl] expression. First, let's say we have a form with three controls named FederalRate, StateRate,

and CountyRate. The Format property of each control is set to Percent. (Below you can see the Format property for the FederalRate control.)



One of the problems with using the Percent format is that if the user types in a whole number, such as 30, the Percent format assumes 300 percent rather than 30 percent. For example, below you can see the results of typing in the values 30, 15, and 5 into this form in form view.



We decide to create a macro that says "If the user types in a number that's greater than or equal to one, divide that value by 100 to put it into percent format." To make things more interesting, we'll create a generic macro that will work with all three controls. That is, rather than create one macro for the [FederalRate] control, another for the [StateRate] control, and a third macro for the [CountyRate] control, we'll create a macro that refers to [Screen].[ActiveControl] that works with all three controls. Figure 20.12 shows such a macro, which I've named ConvertPercent. Notice that the macro has just one condition and one action.

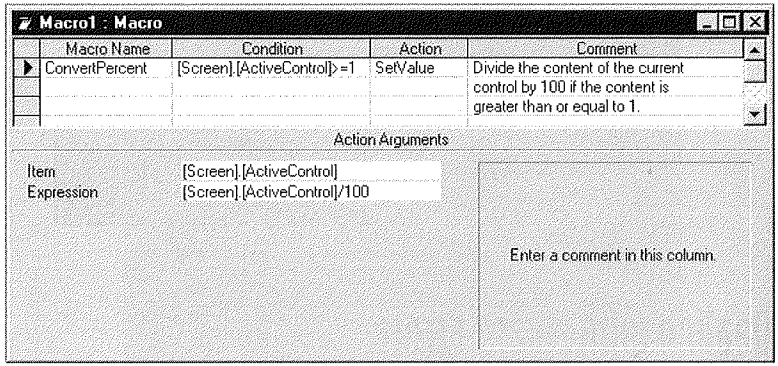
The condition

```
[Screen].[ActiveControl] > = 1
```

makes sure that the action is executed only if the content of the control is greater than or equal to one. The SetValue action arguments:

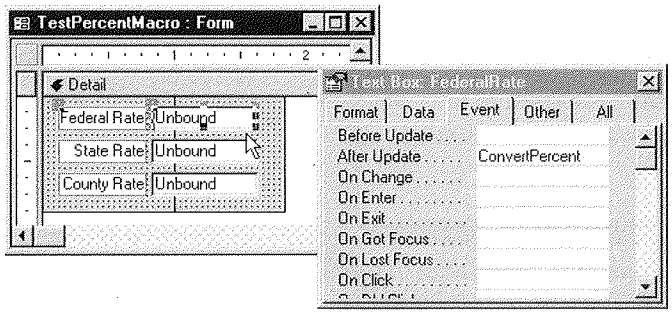
```
Item: [Screen].[ActiveControl]
Expression: [Screen].[ActiveControl]/100
```

FIGURE 20.12
The ConvertPercent macro uses [Screen].[ActiveControl] to refer to whatever control launched the macro.



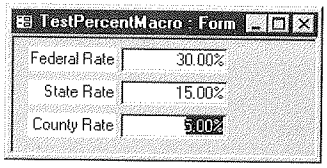
take whatever number is currently in the value and replace it with that same value divided by 100.

Next we close and save the macro. Then we need to open the TestPercentMacro form in design view and set the BeforeUpdate event property of each control to the macro name, ConvertPercent in this example. Below you can see we've set the AfterUpdate event for the FederalRate control to the macro name. We'd just need to do the same for the StateRate and CountyRate controls before closing this form.



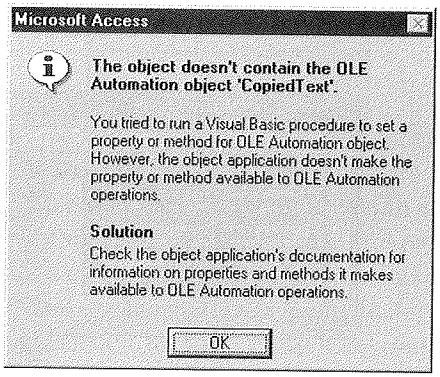
To actually test the macro, we need to go to form view. Nothing happens immediately because the AfterUpdate event occurs only after we type a new value into the control and move onto another control. In form view, let's say we again type **30** in the Federal

Rate control, **15** into the State Rate control, and **5** into the County Rate control. The macro kicks in after each entry, giving the much better result shown below.



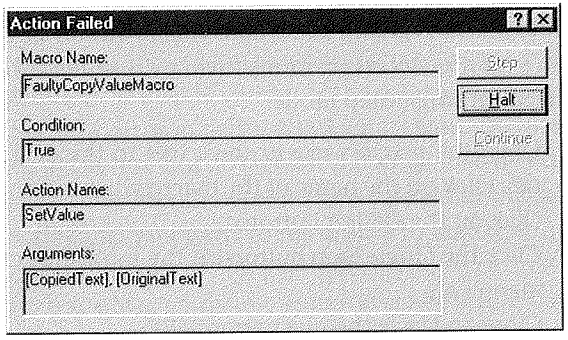
Dealing with Macro Errors

Everyone makes mistakes, especially when creating macros. As you know, when you run a macro, Access executes the first action in the macro, the second action (if any), the third, and so on until it runs out of Action cells. However, if Access has a problem executing one of the actions in your macro, it stops the macro and displays an *error message* that suggests the nature of the problem. For example, while executing a macro, you might come up with the (somewhat obscure) message below when Access hits a glitch.



After reading the message, you can click on OK. You'll see the Action Failed dialog box showing you the specific action that caused the error, as in the example shown on the next page.

This box provides the following information:



- **Macro Name** The name of the macro that contains the faulty action.
- **Condition** What the expression in the Condition column for the faulty line evaluated to (always True if the action has no condition).
- **Action Name** The specific action within this macro that caused the error.
- **Arguments** The arguments you assigned to this action.

To get rid of the error message box, you need to click on the Halt button. If you then want to edit the offending macro, just open the macro's macro sheet in the usual manner (that is, click on the Macros tab in the database window, click on the name of the macro you want to edit, and click on the Design button). Once you get to the appropriate macro and get to the offending action, you're pretty much on your own in trying to figure out why the action didn't work. You may want to check the hint box or press F1 for more detailed information about the action so that you can determine the cause and come up with a solution.

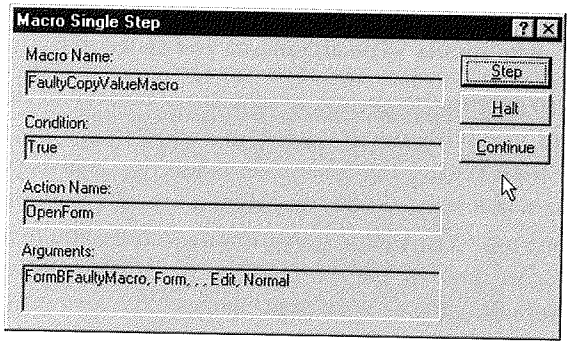
A common cause of macro errors is using faulty identifiers. For example, your macro might refer to a field named [ZipCode] that's not on the form that launched the macro. And therefore you need to add the [Forms]![*formname*]! prefix. Or perhaps your macro is referring to a control on a form that is no longer open when Access tries to execute the action.

Single-Stepping through a Macro

When you run a macro, Access whizzes through all the actions in no time at all. If a particular macro is giving you a hard time, you can slow it down and watch the results of each action as Access performs them by running the macro in *single-step* mode. To run a macro in single-step mode:

1. Open the macro's macro sheet (get to the database window, click on the Macros tab, click on the name of the macro you want to run in single-step mode, and then click on the Design button).
2. Click the Single Step button on the toolbar or choose Run ➤ Single-Step from the menu bar.
3. Close and save the macro normally.
4. Run the macro normally by causing whatever event triggers the macro.

This time when you run the macro, Access will display the Macro Single Step window, shown below, just before it executes each action.



After observing the details of the action that's about to be played, you can use the command buttons to decide what you want to do next:

Step Executes the action whose details are currently displayed in the Macro Single Step dialog box.

Halt Stops the macro and closes the Macro Single Step dialog box.

Continue Turns off the Single Step mode and runs the rest of the macro normally.

Creating a Macro to Run at Startup

As you may know, you can use the Tools ➤ Startup commands on Access's main menu to specify how you want your application to look when the user first opens the database. (Those menu commands are available whenever the database window is displayed.) In addition, you can also have a macro perform tasks automatically when the user first opens your database. All you need to do is create a normal macro and name it AutoExec.

The AutoExec macro runs after the options you defined in the Startup dialog box have been put into effect. So you want to make sure to take that into consideration when creating your AutoExec macro. For example, if you've cleared the Display Database Window option in the Startup dialog, your AutoExec macro doesn't have to hide the database window, since it will already be hidden.

You can bypass the startup options and the AutoExec macro by holding down the Shift key as your database is opening. If you are developing applications, you should keep this technique in mind because sometimes you might want to open your database from the user's perspective.

TIP

You can also press the F11 key to make the database window appear on the screen, unless you've turned off the Access Special Keys option under Tools, Startup or used the /runtime switch when launching Access.

At other times, you might want to go straight to the database window and standard toolbars so you can make changes to your application. To achieve the latter, just keep that Shift key depressed from the time you choose File ➤ Open Database until the database window appears on the screen.

Learning by Example

In this chapter, we've covered the mechanics of creating macros and assigning them to events. You'll find many practical real-world examples of macros in the chapters that follow. Exploring macros in other peoples' applications is also a good way to round out your knowledge of macros. For example, the sample Northwind database that comes with your Access programs, as well as some of the applications on the CD that comes with this book, contain several examples of macros.

To view the macros in an application, just open the database normally and get to the database window. In the database window, click on the Macros tab. Then click on any macro name and click on Design to explore the macro's contents.

In some applications, you might be surprised to see very few macros, or even no macros at all. A very sophisticated Access application might have very few macros associated with it for three reasons:

- The Control Wizards create Visual Basic code, not macros, to automate the controls you create.
- Many Access developers prefer Visual Basic code to macros because they are already familiar with Visual Basic.
- Many application developers will use the built-in macro converter to convert their macros to Visual Basic code and then delete the original macro.

MASTERING THE OPPORTUNITIES

Converting Macros to Visual Basic

Once you've created some macros and have them working properly, it's easy to convert them to Visual Basic code. To convert all the macros in a given form or report to code, first open the form or report in design view. Then choose Tools ➤ Macro ➤ Convert Form's Macros to Visual Basic (or Convert Report's Macros to Visual Basic if you're working with a report.)

If a particular set of macros isn't associated with a specific form (such as an AutoExec macro), you use a different technique to convert the macro. In design view, open the macro you want to convert. Then choose File ➤ Save As/Export ➤ Save As Visual Basic Module ➤ OK. When conversion is complete, you can find the Visual Basic version of the macro in the Modules tab of the database window.

Where to Go from Here

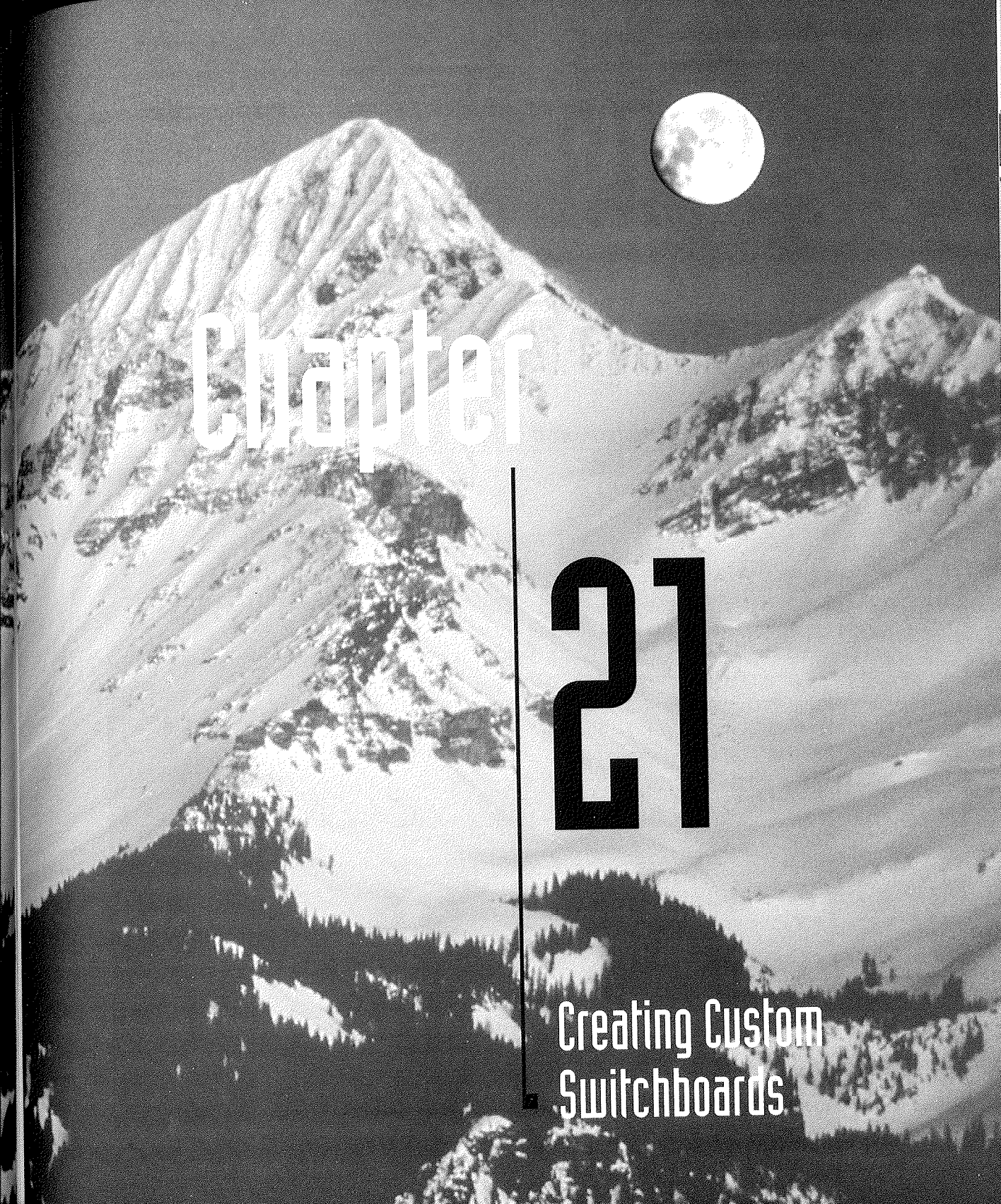
From here you can focus on different aspects of creating a custom application or learning about Visual Basic:

- To learn about creating custom switchboards and dialog boxes for your application, see Chapters 21 and 22.
- To learn how to create custom toolbars and menus for your application, see Chapters 23 and 24.
- To learn about Visual Basic, that "other way" to create custom actions, see Chapter 25.

What's New in the Access Zoo?

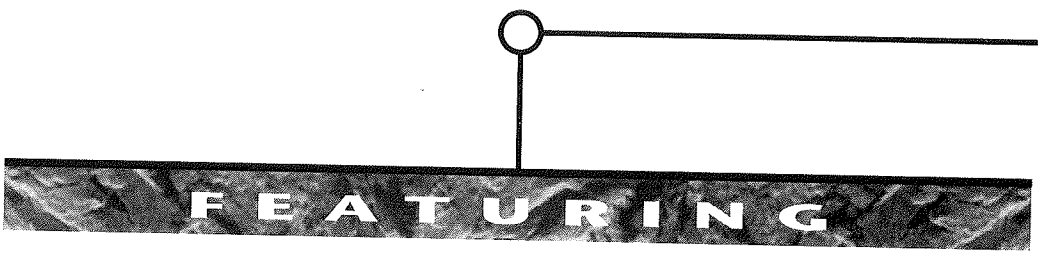
For those of you familiar with earlier versions of Access, here's what's new in the macros department:

- **RunCommand action is a** new action that replaces the old DoMenuItem action. Use it to execute an Access menu command.
- **New commands on the Tools/Macro menu** let you create menus, toolbars, and shortcut menus from macros. See Chapters 23 and 24 for more information on these new features.



Chapter 21

Creating Custom Switchboards



Creating Custom Switchboards

A switchboard is a fancy term for a form that lets the user move around in your application. When you use the Access Database Wizard to create a database application, the Wizard creates a switchboard automatically. In this chapter we're going to look at techniques for customizing the switchboard that the Wizard creates. We'll also look at techniques for creating your own completely custom switchboards with whatever appearance you like.

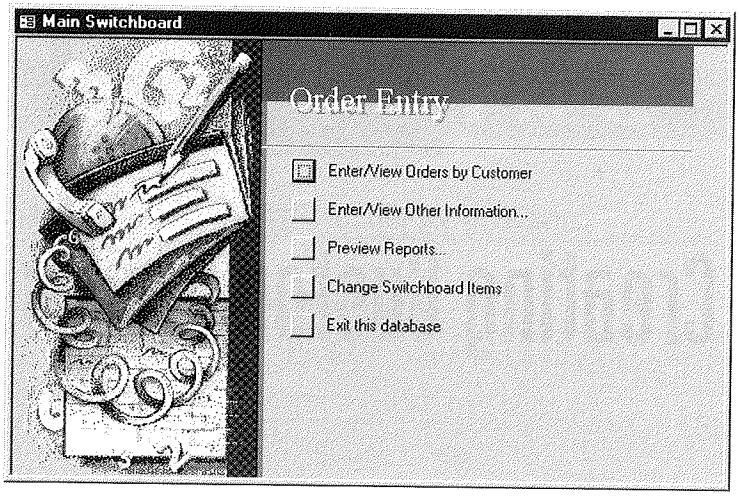
Changing a Wizard-Created Switchboard

As you know, when you use a Database Wizard to create a database, the Wizard automatically creates a switchboard for that database. For example, when you use the Order Entry Wizard to create a database, that Wizard creates the switchboard shown in Figure 21.1.

The switchboard appears automatically when you first open the database. If you happen to be at the database window, rather than at the switchboard, you can just click on the Forms tab, click on the Switchboard form name, and then click on the Open button to open the switchboard.

FIGURE 21.1

The switchboard created by the Order Entry Database Wizard.

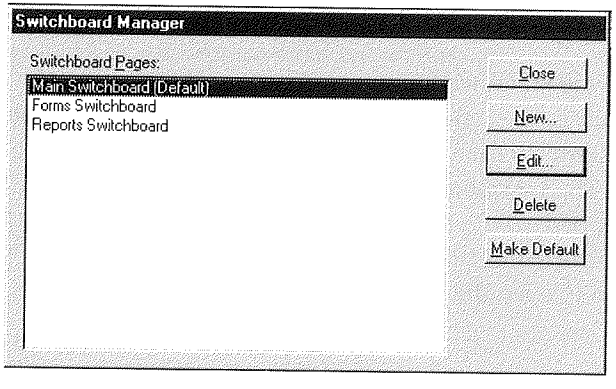


Changing Wizard-Created Switchboard Options

If you look at all the options on the Order Entry switchboard, you'll see that one option actually lets you change the switchboard itself (the fourth option down in this example). When you choose that option, you're taken to the Switchboard Manager dialog box, which will look something like Figure 21.2 (depending on the database you're using at the moment).

FIGURE 21.2

The Switchboard Manager lets you make changes to any switchboard in the current database. This database contains three switchboards.



The command buttons on the Switchboard Manager are fairly self-explanatory.

- **Close** Click on Close after you've finished exploring/modifying switchboards.
- **New** Creates a new, blank switchboard with whatever name you specify. To add options to that newly-created switchboard, click on its name and then click on the Edit button. You'll be taken to the Edit Switchboard Page dialog box described in the next section.
- **Edit** To change an existing switchboard, click on its name and then click on the Edit button. You'll be taken to the Edit Switchboard Page dialog box described in the next section.
- **Delete** To delete a switchboard, click on its name then click on the Delete button.
- **Make Default** Makes the currently selected switchboard the *default* switchboard (the one that appears automatically when the user first opens the database).

Defining and Changing Switchboard Items

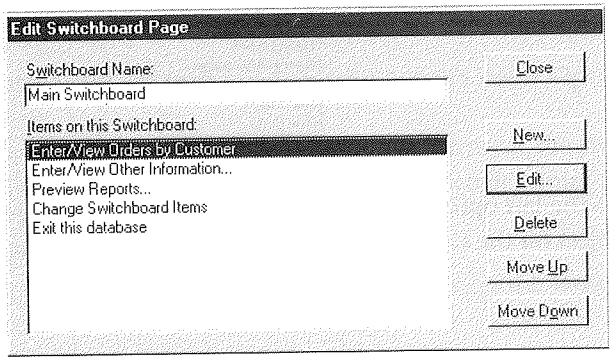
When you choose the Edit button from the Switchboard Manager, you're taken to the Edit Switchboard Page. If you are working with a new switchboard, the list under Items on this Switchboard is blank. You can use the New button to create new items. When you Edit an existing switchboard, the items on that switchboard are listed under Items on this Switchboard, as in Figure 21.3.

The command buttons in the Edit Switchboard Page dialog box are also self-explanatory.

- **Close** Choose this button when you've finished making changes to return to the Switchboard Manager dialog box.
- **New** Add a new item to this switchboard.

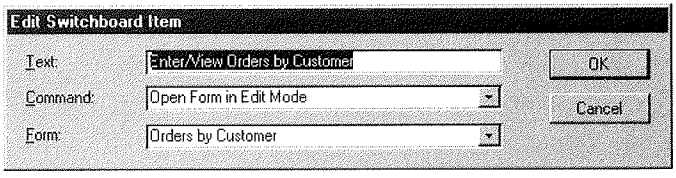
FIGURE 21.3

The Edit Switchboard Page dialog box lets you add, change, and delete individual options on the currently selected switchboard.



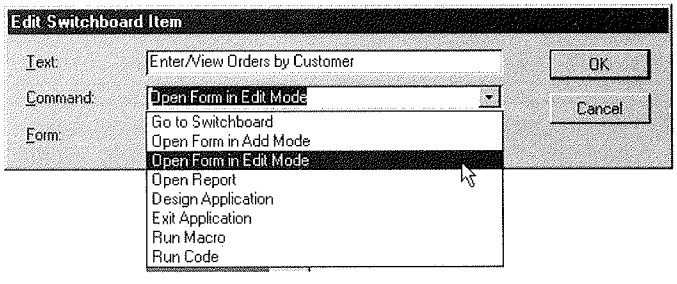
- **Edit** Change the currently selected switchboard item.
- **Delete** Delete the currently selected switchboard item.
- **Move Up** Move the currently selected switchboard item up in the list.
- **Move Down** Move the currently selected switchboard item down in the list.

When you Edit a switchboard item, you're taken to the small Edit Switchboard Item dialog box, as in the example shown below.



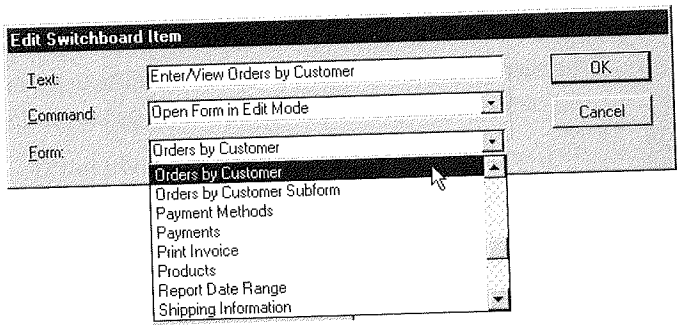
This dialog box is where you define how the item looks on the switchboard and what happens when the user selects that item. In the example shown, the Text that actually appears on the switchboard is *Enter/View Orders by Customer*. To change that text just click anywhere in the text and make your changes using standard editing techniques.

The Command box describes what will happen when the user selects the item. In the example, when the user chooses *Enter/View Orders by Customer* the action that occurs is *Open Form in Edit Mode*. But you can change that action, if you wish, simply by choosing a new option from the Command drop-down list. As you can see below, you have quite a few options for defining what happens when the user chooses the item.



The last option in the Edit Switchboard Item dialog box lets you choose a specific object for the Command to act upon. For example, when the Command is Open Form

in Edit Mode, the last option is titled Forms and you can select a specific form for the item to open, as below.



If, on the other hand, the Command box contained the action Open Report, the last option would be titled Report and you could choose a specific report for the command to open.

After making changes to a Wizard-created switchboard, you need to get back to form view to see and test the effects of those changes. Select OK and Close, as appropriate, to work your way back to the database window. If you really want to see how things will look to a person opening the database for the first time, you can close and then reopen the database. To do so, you choose File ➤ Close from the Access menu bar; click on File again and select the name of the database you just closed.

Changing Wizard-Selected Art

When you use a Database Wizard to create a database, you're also given the option of adding a picture to the database's switchboards. You can change that picture after the fact, if you wish, using any bitmap image on your hard disk. You can use an existing clip art image, a bitmap image you created yourself, or an image you digitized using a scanner.

To change the picture on a Wizard-created switchboard, follow these steps:

1. Open the database in the usual manner with Access's File ➤ Open Database menu commands.
2. Click on the Close (×) button in the upper-right corner if the switchboard is currently open. Get to the database window (if it's hidden or minimized, just press the F11 key).
3. Click on the Forms tab in the database window.
4. Click on the Switchboard form name and then click on the Design button.
5. Click on the picture that you want to change.

6. Open the property sheet if closed (click on the Properties toolbar button or choose View ➤ Properties).
7. Click on the All tab and scroll to the top of the property sheet. You should see the control name Picture, as in Figure 21.4.

   The second property in the property sheet is titled Picture and shows the location and name of the picture that's currently displayed in the switchboard.

8. Click on Picture and then click on the Build button. The Insert Picture dialog box appears as in Figure 21.5 (though initially, your Insert Picture dialog box might show the contents of some folder other than the one named Dbwiz).
9. Browse to the folder and file that contain the picture that you want to display in your switchboard and choose OK.

   The picture you chose will replace the one that's currently in your switchboard and will be stretched to fit the picture's container. You can use the Size Mode property to change the picture's sizing mode to Clip or Zoom to see which mode works best.

10. Save the switchboard with the new picture and size mode: choose File ➤ Close; click on Yes when asked if you want to save your changes.

To see the results of the change, reopen the switchboard in form view. Or if you want to be sure to view the switchboard from the user's perspective, close the entire database (File ➤ Close). Then reopen the entire database by clicking on the File menu and the name of the database you just closed.

FIGURE 21.4

Properties for the control named Picture in a Wizard-created switchboard.

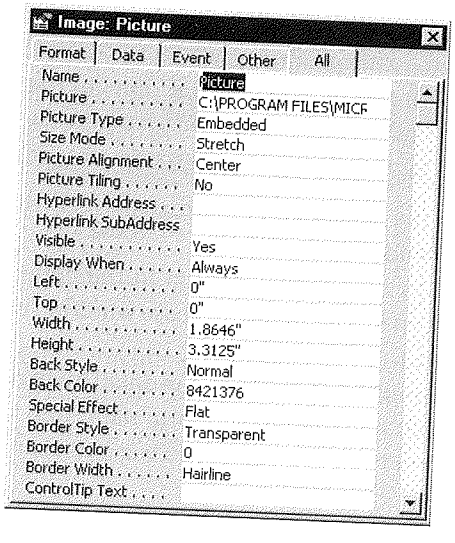
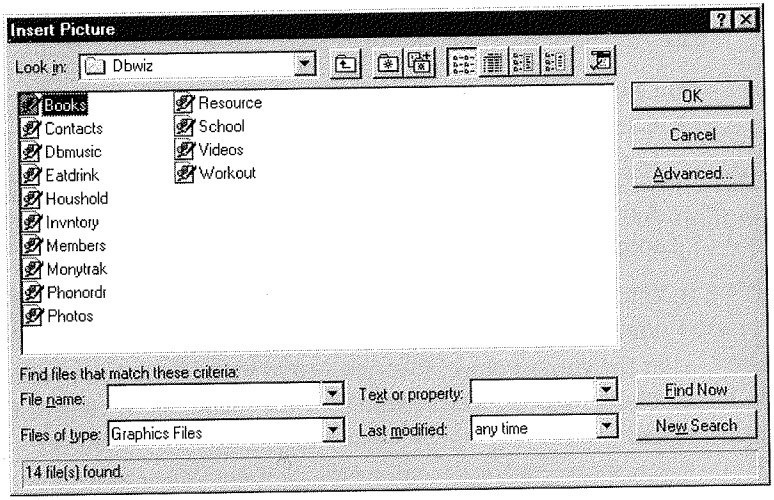


FIGURE 21.5
The Insert Picture dialog box lets you select a new picture to display in your Wizard-created switchboard.



Note that the picture you chose will appear on all the database's switchboards. The reason is that the Database Wizards actually create only one switchboard per database. When you're *using* that database, you might think you are going from one switchboard to another from time to time. But in fact, your database is just changing the title of, and items on, that one switchboard.

Creating a Switchboard from Scratch

As you know, using Database Wizards isn't the only way to create a database application. You can create all your tables, queries, forms, reports, and macros from scratch. Likewise, you can create custom switchboards for your application, completely from scratch.

To create a custom switchboard, first create a blank form that isn't bound to any table. To make it look like a switchboard rather than a bound form, you can hide the navigation buttons, record selectors, and other doo-dads that normally appear on bound forms. Then you can add hyperlinks, controls (such as command buttons), and macros to make the controls on the switchboard do whatever you want them to do. We'll take it step-by-step, starting in the next section.

Creating the Blank Switchboard Form

The basic idea behind a switchboard is to create a form that helps the user of your application do any of a variety of tasks, such as navigating from one form to another or

printing reports. So typically you'd create tables, queries, forms, and reports for your application before you work on the custom switchboards. Then, within that same database, you'd follow these steps to create a new, blank switchboard form:

1. Click on the Forms tab in the database window.
2. Click on the New button; then choose Design View from the New Form dialog box. Leave the Choose a Table or Query option blank and click on the OK button.

   A new empty form opens in design view. The first thing you'll want to do is to set some properties for it.

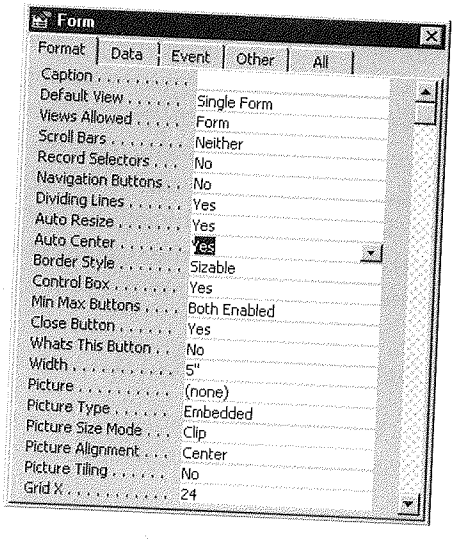
3. Click on the Properties toolbar button (if the property sheet isn't open) or choose View ➤ Properties from the menu bar. You want to be sure the entire form is selected as the current object (because you're about to set form properties).
4. Choose Edit ➤ Select Form from the menu bar or click in the box where the rulers meet.
5. Click on the Format tab in the property sheet and then set the first few Format properties as indicated below and shown in Figure 21.6. (Properties below that are marked with an asterisk are suggestions only. You might want to experiment with those properties when creating your own switchboards.)

   Default View: Single Form

   Views Allowed: Form

   Scroll Bars: Neither*

FIGURE 21.6
Suggested Form properties for a custom switchboard.



Record Selectors:	No
Navigation Buttons:	No
Auto Resize:	Yes*
Auto Center:	Yes

TIP

Remember that you can get more information about a property right on your screen. Just click on the property you're interested in and press Help (F1).

6. (Optional) Fill in the Caption property with whatever text you want to appear in the title bar of the custom form.
7. Click on the Detail band within the form if you want to color the form. Then choose a color from Back Color button on the Formatting toolbar. If that toolbar isn't visible, choose View ➤ Toolbars and click on Formatting (Form/Report). At this point you may want to size and shape the form to approximately the size you want the switchboard to be.
8. Drag the lower-right corner of the shaded area within the form design window to about the size you want to make the switchboard.

NOTE

To size the gray area, move the mouse pointer to its lower-right corner until the mouse pointer turns into a four-headed arrow. Then hold down the mouse button and drag that corner.

9. You can save and name the form now. Choose File ➤ Close, choose Yes when asked about saving the form, enter the name you want to give the form (e.g., Main Switchboard), and choose OK.

The Switchboard form is listed in the database window, in the Forms tab, just like all your other forms. And you can treat it as you would any other form:

- **To see and use the form from the user's perspective,** click on the form name and then click on the Open button. (At this point, our sample form is completely blank.)
- **To make changes to the form,** open it in design view. (Click on the form name in the database window and then click on the Design button.)

Once the form is open you can easily switch between form view and design view by clicking on the ... View button in the toolbar or by choosing either Form View or Design View from the View menu in the menu bar.

Adding Controls to Your Custom Switchboard

Currently our switchboard is empty. We need to add some hyperlinks or *controls* to allow the user to choose actions. As with all types of forms, you create controls using the toolbox in form design view. You can create any control you wish, but chances are you'll want to create mostly hyperlinks and command buttons.

As you may recall from earlier chapters, you can use the Insert Hyperlink button on the toolbar to add a hyperlink to a form. If you need a control like a command button instead, use a Control Wizard to create a control and action in one fell swoop. When creating controls on a switchboard, the decision on whether or not to use the Control Wizards centers around three factors:

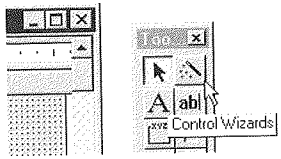
- If the control will perform a single action, such as opening a form, and that form already exists, then you can use the Control Wizard. (You may prefer to use a hyperlink for opening a form or report for the performance reasons outlined in Chapter 16.)
- If the control will open a form (or report) that you have not yet created, you can create the control without using the Control Wizard. Later, after you've created the form or report that the control will act upon, you can go back to the switchboard and assign an action to the control.
- If the control will perform two or more actions, then you'll need to define the control's action using a macro (or Visual Basic code). You can create the control without the Control Wizard. Then later create the macro and assign that macro to the control on the switchboard.

The last alternative is perhaps the most common when creating switchboard controls because typically you want the control to perform two actions: open some other form or report and then close the switchboard itself. So let's work through an example using that last approach.

Here's a quick way to create a command button and assign a macro to it. First create the macro and then just drag and drop the macro name on to the form (in design view). You'll get a command button whose On Click property launches the dropped macro!

Let's say we want to create a command button on our switchboard to open a form named AddressBook and then close the switchboard. For this example we'll also assume that we previously created the form named AddressBook and that it exists in the current database.

1. Open the switchboard in design view.
2. Click on the Toolbox toolbar button (if closed) or choose View ➤ Toolbox.
3. Make sure the Control Wizard button in the toolbox is *not* pushed in because we don't want to use the Control Wizard in this example.

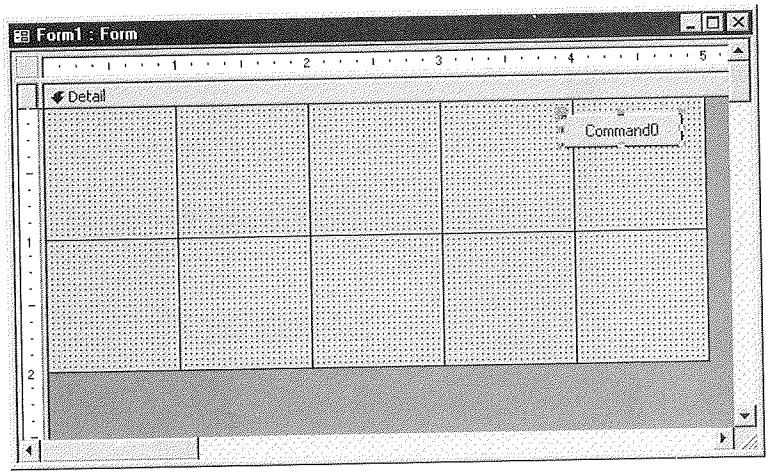


4. Click on the Command Button button in the toolbox and then click at the position where the button should appear in the switchboard. In Figure 21.7 you can see we've created a button, which is (tentatively) captioned Command0.

Though it's not absolutely necessary to do so, we could close the Main Switchboard form now, just to get it out of the way, by choosing File ➤ Close and then Yes when asked to save changes.

FIGURE 21.7

Here we've created a command button without using the Control Wizard. The button has the generic name Command0.



Creating a Macro for the New Control

Next we need to create a macro that will open the Address Book form and close the Main Switchboard form.

1. Click on the Macros tab in the database window.
2. Click on the New button. A new blank macro sheet opens. We'll probably want to put all the macros for the Main Switchboard into this macro sheet.
3. Open the Macro Name column (click on the Macro Names toolbar button). Or choose View ➤ Macro Names from the menu bar.
4. Type a name for this macro such as OpenAddressBook in the Macro Name column.
5. Choose the OpenForm action in the Action column to the right.
6. Specify the name of the form you want to open in the Action Arguments (AddressBook in this example). Figure 21.8 shows how the macro sheet would look at this point.
7. Create a second action to close the Main Switchboard by choosing the Close action in the next Action column down.
8. Complete the action arguments as shown in Figure 21.9.
9. Close the macro sheet and give it a name. In this example we would choose File ➤ Close, choose Yes when asked about saving the macro, give it a name such as MainSwitchboardMacros, and then choose OK.

FIGURE 21.8

The first action for the Open-AddressBook macro defined in the macro sheet.

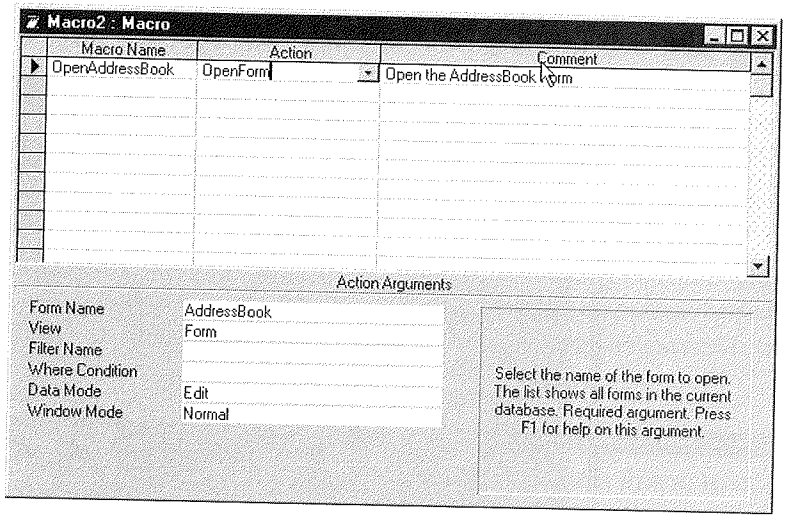
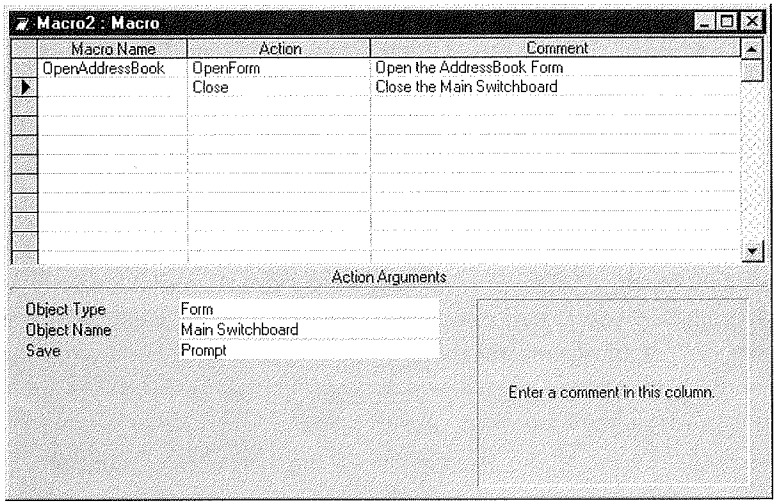
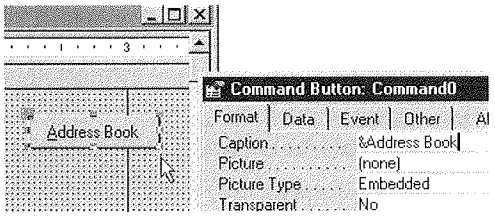


FIGURE 21.9
The second action in the OpenAddressBook macro closes the Main Switchboard form.



Finally we need to assign that new macro to the On Click property of the button we created on the switchboard. While we're at it, we can change the caption on the button. Here are the steps:

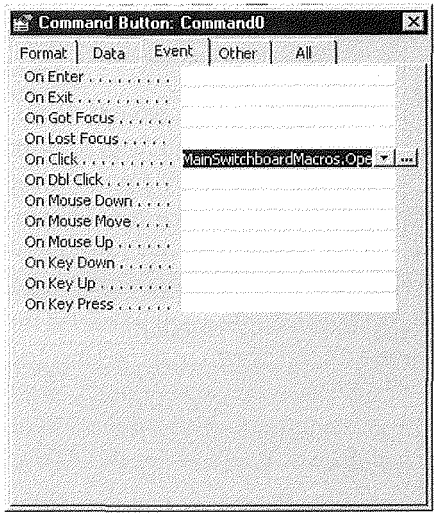
1. Click on the Forms tab in the database window.
2. Click on Main Switchboard and click on the Design button.
3. Click on the button to which we want to assign the macro (the button titled Command0 in this example).
4. Open the property sheet and click on the Event tab.
5. Choose the On Click property and choose the name of the macro you want this button to launch. In this example we want to choose the MainSwitchboard-Macros.OpenAddressBook macro (see Figure 21.10).
6. Change the caption on the command button: Click on the Format tab in the property sheet and then type in a caption such as &Address Book (which will appear as Address Book on the button).



7. Close and save the Main Switchboard form.

FIGURE 21.10

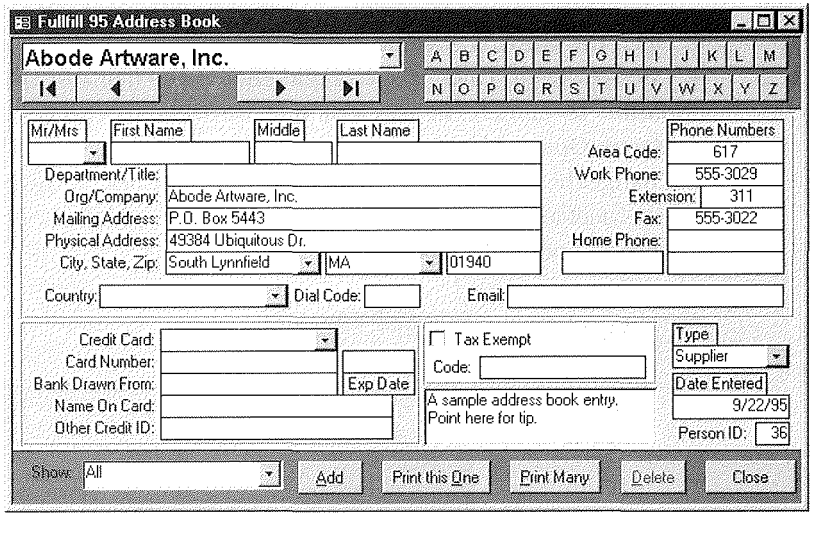
Assigning the MainSwitchboard-Macros.Open AddressBook macro to the On Click property of a button on the Main switchboard.



To test your new control and action, open the Main Switchboard in form view. Then click on the Address Book button. The macro will open the AddressBook form and close the Switchboard, as in Figure 21.11.

FIGURE 21.11

Clicking the Address Book button in the Main Switchboard opens this form and closes the switchboard.



We realize we haven't mentioned anything about the AddressBook form prior to this chapter. But our goal here is to show you how to make a switchboard button open one form and close its own form. In Chapter 28 and in Appendix C, we'll talk more about the AddressBook form and the Fulfill application.

Making AddressBook Return to the Main Switchboard

In this particular application, clicking on the Address Book button in the Main Switchboard sends users to a form named AddressBook. It stands to reason that, when users close the AddressBook form, they would expect to be returned to the Main Switchboard.

We could make a button on the AddressBook form that closes the AddressBook and then opens the Main Switchboard again. But there's just one problem. Suppose the user closes the AddressBook form by clicking on the × button in the form's window or by choosing File ➤ Close from the menu bar. Neither task would trigger the action to open the Main Switchboard. So here's what we need to do:

- Create a Close button that, when clicked, closes the AddressBook form.
- Go to the property sheet for the AddressBook form as a whole and create an action that opens the Main Switchboard form. If we attach that action to the On Close property of the AddressBook form, it doesn't matter how the user exits the form—he or she will still be returned to the main switchboard.

We'll create the Close button on the AddressBook form first. Since we want this button to do one simple act, we can use the Control Wizard to define the control and action in one fell swoop.

1. Open the AddressBook form in design view.
2. Open the toolbox if it isn't already open. (Click on the Toolbox toolbar button or choose View ➤ Toolbox.)
3. Make sure the Control Wizard button in the toolbox is pushed in because we can use its help in this case.
4. Click on the Command Button tool in the toolbox; then click on the spot where the close button should appear on the form (the lower-right corner in this example).
5. Choose Form Operations and Close Form when the Command Button Wizard appears, as shown in Figure 21.12.
6. Choose Next> from the Command Button Wizard and the next screen asks about the appearance of the button. In this example we chose to have the button show the text &Close (see Figure 21.13; once again, the & symbol is used to specify the underlined hotkey).

FIGURE 21.12

The new button we're adding to the Address Book form will close the form.

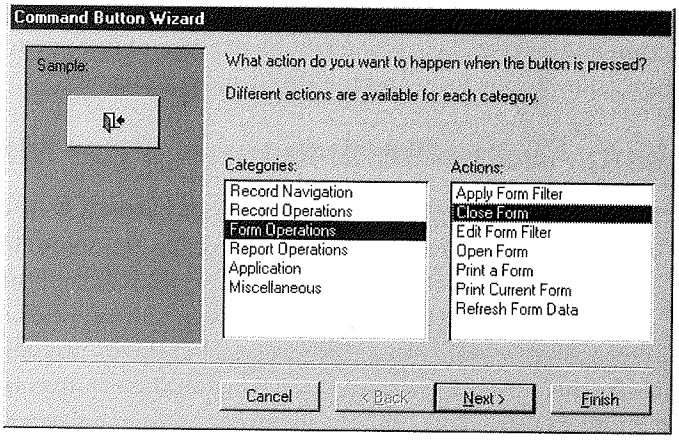
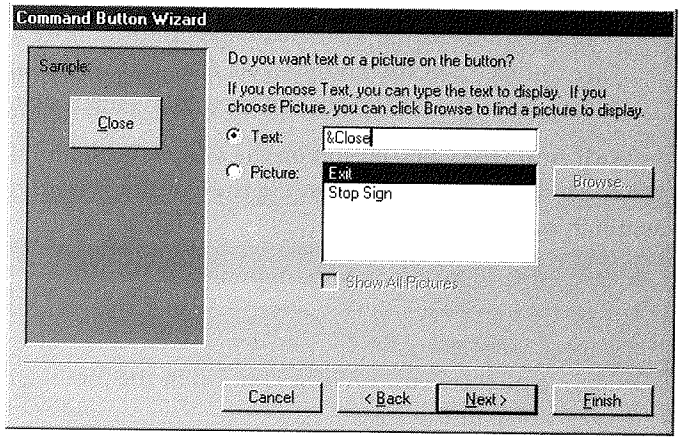


FIGURE 21.13

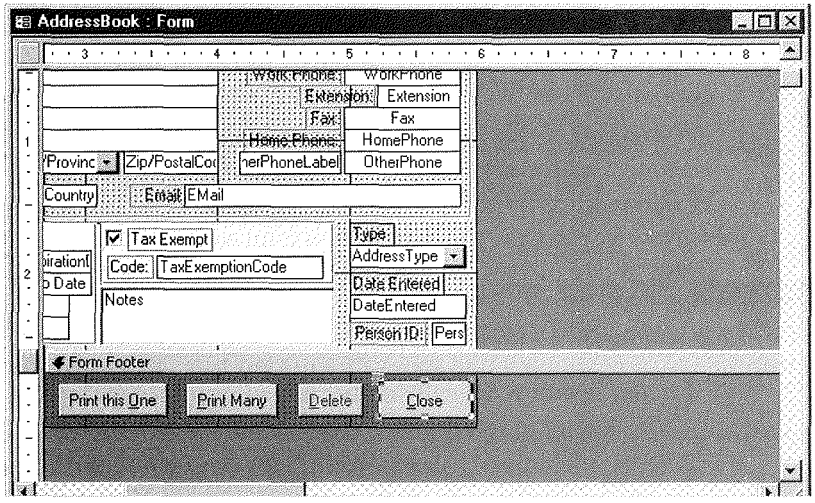
The Close button on the AddressBook form will be captioned Close.



Clicking Next > takes us to the Wizard window to name the button. This is the name used within Access, not the text caption that appears on the button. We could name the button anything we want. In this example we named the button CloseAddressBookForm and then clicked on the Finish button.

When the Command Button Wizard is done we're returned to our form, where we can see the new button. We can use the standard techniques for moving and sizing

controls to position the button precisely. In the figure below we've opted to put that button near the lower-right corner of the form.



Now we still need to make the closing of the AddressBook form automatically reopen the Main Switchboard form. Keep in mind that the user will probably have several means of closing that form—not just our new Close button. So we need to find a way of saying, "No matter how the user closes this form, open the Main Switchboard form again."

We could create a macro that opens the Main Switchboard form, but let's try a slightly different approach, using a bit of Visual Basic. How do we write a Visual Basic procedure to open a form? Let's ask the Answer Wizard:

1. Choose Help ➤ Contents and Index from Access's menu bar.
2. Click the Index tab and type **open form**.
3. Double-click on OpenForm Method in the topic list.

TIP

An *action* generally refers to macros, whereas a *method* generally refers to Visual Basic code. We chose Open-Form Method in step 3 because we want to check out the Visual Basic approach to doing this.

A great deal of information about the OpenForm Method appears, but we mainly need to know the syntax. In this case the syntax is

```
DoCmd.OpenForm formname
```

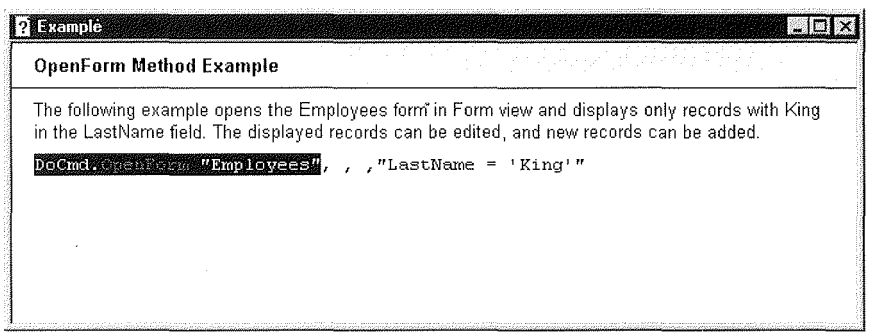
followed by some optional arguments enclosed in square brackets. When we click on the Example option, we see that the name of the form to open needs to be enclosed in quotation marks. To make life easy, we can just copy the example shown from the Help screen right into out property sheet. To do that we just drag the cursor through the part we want to copy, as in Figure 21.14, and then press Ctrl+C to copy that selection to the Windows Clipboard.

Now we can close the Help screens until we get back to the AddressBook form, which is still in design view. Now here's how we make the act of closing this form automatically open the Main Switchboard form. With the AddressBook form on the screen in design view:

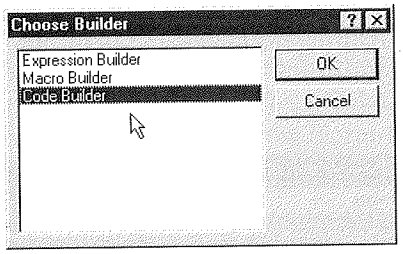
1. Choose Edit ➤ Select Form because we want to work with the form properties as a whole (not properties of individual controls).
2. Open the property sheet and click on the Event tab.

FIGURE 21.14

An Example of using Visual Basic to open a form. We've selected the part we want to copy to our form.

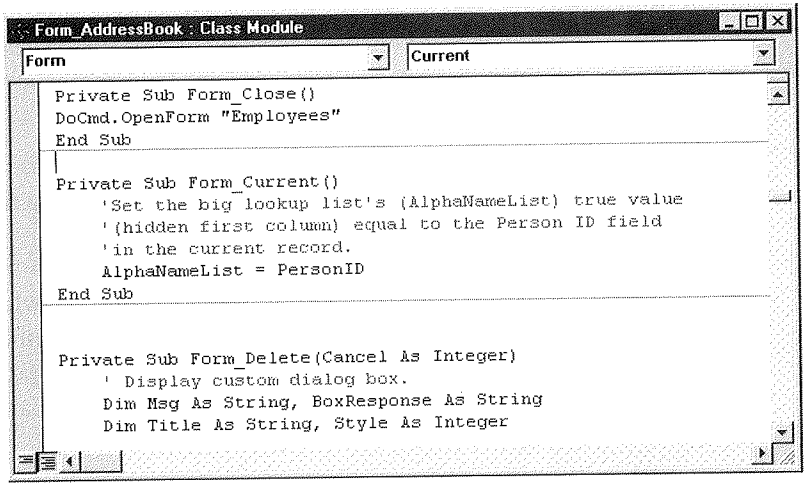


3. Click on the On Close property, and a Build button appears with these options:

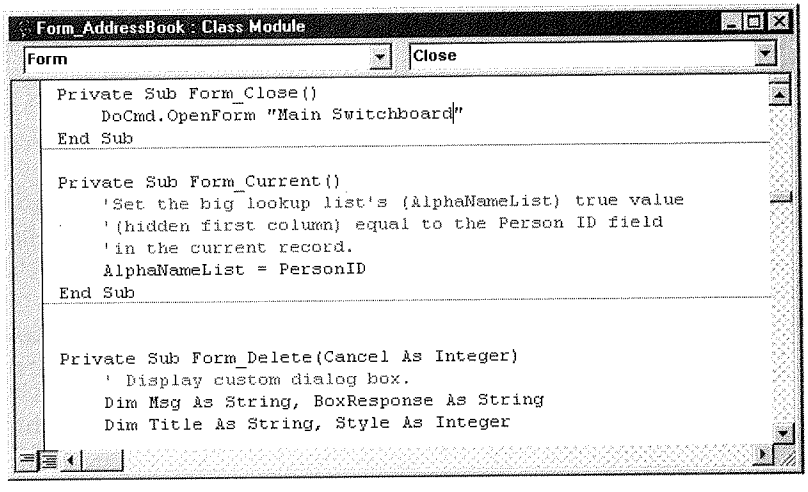


Remember, you can use either Visual Basic or macros to define many actions. Here we've used Visual Basic just because it's quick and easy to do so in this example. Chapter 25 introduces Visual Basic.

4. We're going to try our hand at some Visual Basic code, so click on Code Builder and then click on OK. A new window pops up that already contains a couple of lines of Visual Basic Code, `Private Sub Form_Close` and `End Sub`. Any code we want to add must go between those two lines.
5. Put the cursor between the two existing lines of code and then press Ctrl+V to add the copied lines of code. Initially, the pasted text looks like this:



6. Change the form name in the code, as below, so that the code opens the form named Main Switchboard, not the form named Employees. (You can also press Home to move the cursor to the start of the line and then press Tab to indent the line. Indenting the lines between the Private Sub and End Sub commands is a standard practice.)



7. Click on the Compile Loaded Modules button in the toolbar to *compile* the code quickly and check for gross errors in our Visual Basic command. If you did everything correctly, you won't see any error messages.
8. Close the module window (the one that contains the Visual Basic code) by clicking on the Close (×) button in the upper-right corner of the module window or by choosing File ➤ Close from the menu bar.

   The property sheet now shows [Event Procedure] next to the On Close property, indicating that we've assigned a Visual Basic procedure to this event.

9. Close and save the AddressBook form.

To test the effects of all this, we can now open the Main Switchboard form in form view. When we click on the Address Book button in that switchboard, the Address Book form should open and the Main Switchboard form should disappear. When we close the AddressBook form, that form should disappear and the Main Switchboard form should reopen.

Filling Out the Switchboard

We can continue work with the Main Switchboard form, adding whatever controls we think will be useful later down the road. We can also use the Label and Rectangle tools in the Toolbox to add some labels and boxes. The Back Color, Border Color, Border Width, and Special Effects buttons on the Formatting (Form/Report) toolbar can help make these embellishments even fancier.

TIP

If you create a rectangle around a group of buttons and the rectangle ends up *covering* the buttons, don't panic. Just select (click on) the rectangle and choose Format ➤ Send to Back from the form design screen's menu bar.

The Main Switchboard example we're showing you in this chapter was actually the starting point for a real application, named Fulfill 95, that's on the CD-ROM that came with this book. And as you'll see when you try that application, we've embellished the Main Switchboard.

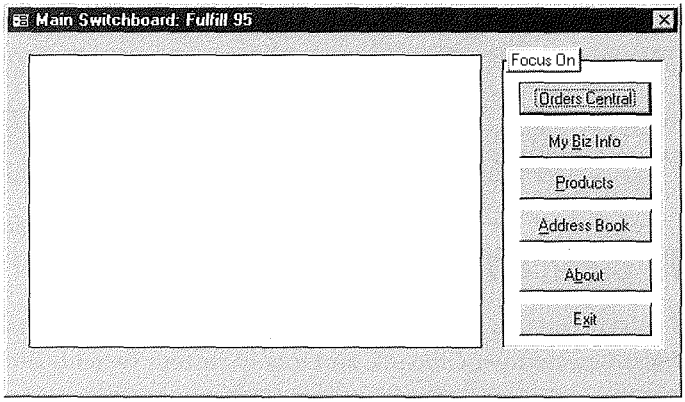
For example, we added a dark gray rectangle behind the command buttons and a label (Focus On) to the upper-right corner of that rectangle. We also added a large white rectangle as a placeholder for Fulfill's logo, which we'll create and add later. Figure 21.15 shows the Main Switchboard, in form view at this stage of Fulfill's development.

NOTE

When you explore the Fulfill application, you'll no doubt find that its main switchboard and other forms have evolved from what's shown here.

FIGURE 21.15

The sample Fulfill application's Main Switchboard, in form view, under construction.



You can start exploring Fulfill at any time by copying it from the CD and opening it up in Access for Windows 95. (If you get a message that the database cannot be opened because it is read-only, open the Explorer window. Select `Fulfill.mdb`, choose Properties from the File menu, and uncheck the Read-only attribute. Appendices B and C will help you.) Chapter 28 discusses ways of exploring Fulfill (and other custom Access applications) so you can start learning "by example" how all the pieces are put together in an Access custom database application.

Making a Switchboard Appear at Startup

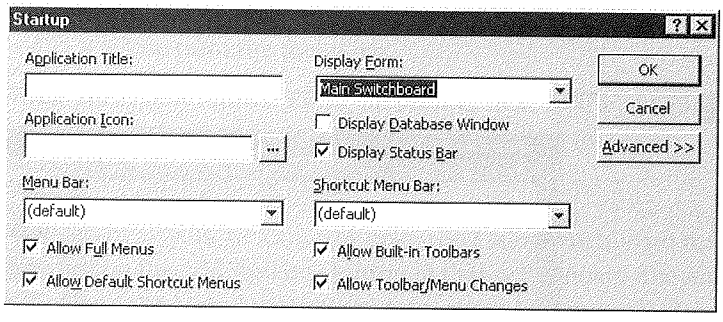
If you create a custom switchboard for your application, and want it to appear automatically when the user first opens the database, set the Display Form option in Startup to the name of your switchboard. Here are the exact steps to follow:

1. Close any open forms to get to the database window.
2. Choose Tools ➤ Startup from the menu bar.
3. Choose the name of your main switchboard from the drop-down list box next to Display Form, as I've done in Figure 21.16.
4. Choose OK.

You can leave all the other settings as they appear in the Startup dialog box until you're further along in the development process. (More on those options in Chapter 28.) The next time you open the database, your custom switchboard will appear on the screen automatically.

FIGURE 21.16

The form named Main Switchboard is the first to appear when the database opens.



Wizard-Created versus Custom Switchboards

If you've read this entire chapter, you may be confused by the vast differences between Wizard-created switchboards and totally custom switchboards. Let's take a moment

here to review the primary differences so you don't leave this chapter feeling confused on this topic.

Summary: Wizard-Created Switchboards

When you use a Database Wizard to create a database application, keep in mind the following points about the switchboard(s):

- To change items on a Wizard-created switchboard, open the switchboard in *form view* and use the Change Switchboard Items option to make your changes.
- You can make design changes to the Wizard-created switchboard by opening that switchboard in design view. However, any changes you make will affect all the switchboards in that database application.
- The reason for the above is that the Database Wizard really only creates one switchboard per database application. It just changes the items on that one switchboard, automatically, when you choose an item that takes you to a (seemingly) different switchboard.
- You can make a Wizard-created database open with a different, custom switchboard of your own design. Just create your custom switchboard. Then choose Tools ➤ Startup ➤ Display Form and set the name of the form to your new custom switchboard.

Summary: Custom Switchboards

When you don't use a Database Wizard to create a database, keep in mind the following points:

- Initially, your database application will have no switchboards at all.
- You create a switchboard by creating a new form that's not bound to any table or query.
- To ensure that the switchboard form doesn't look like a data-entry form, turn off the form's record selectors, navigation buttons, scroll bars, datasheet view, and so forth by selecting the entire form in form view, and making appropriate changes to the property sheet.
- You need to add your own controls (i.e., command buttons) to a custom switchboard, using the toolbox in form design view. You can also use hyperlinks for simple actions like opening forms and reports.
- To make a switchboard appear automatically at startup, choose Tools ➤ Startup and set the Display Form option to the name of your switchboard.

Where to Go from Here

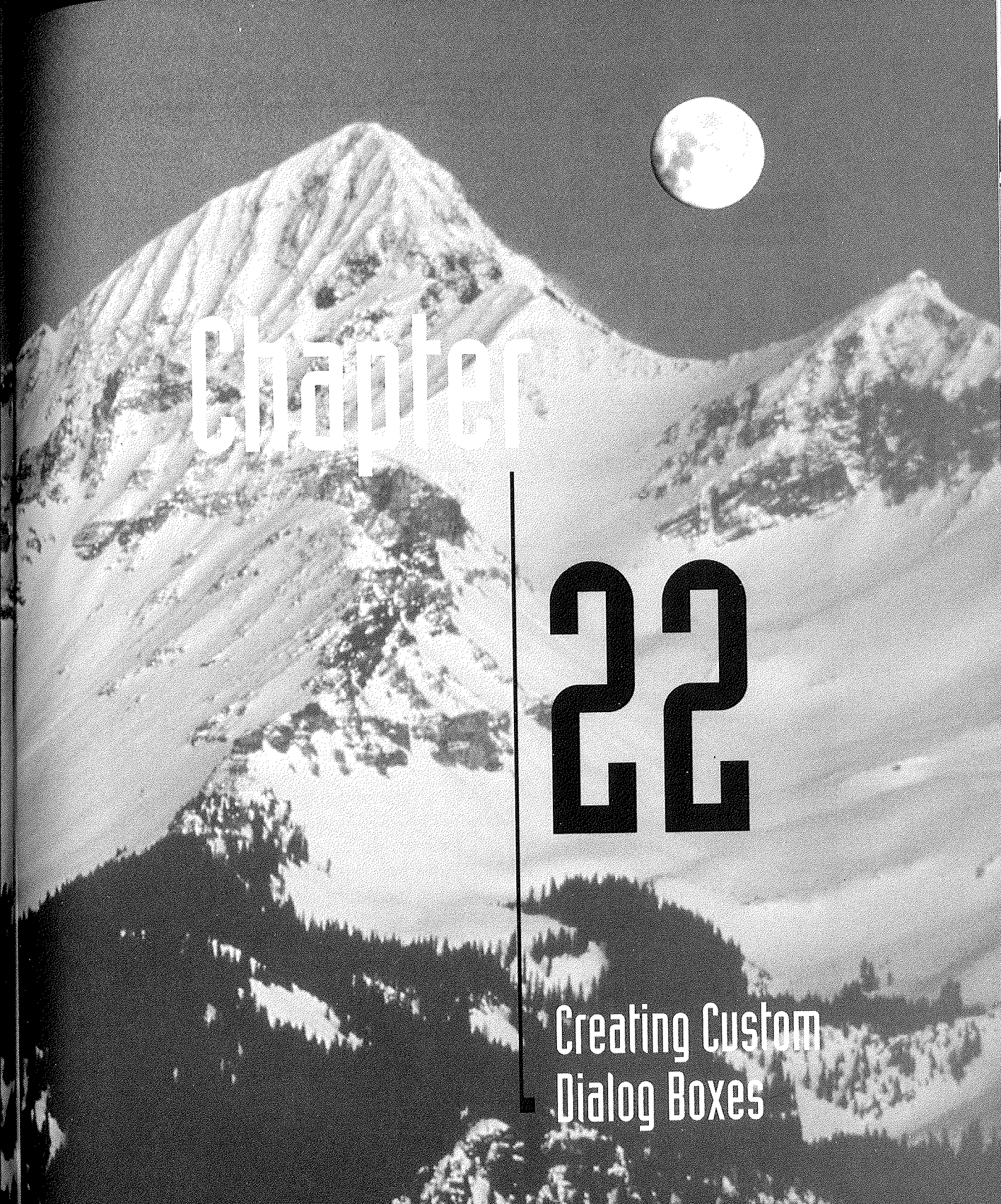
Next we'll look at ways of creating custom dialog boxes from scratch. As you'll see, the basic starting point is the same as it is for creating a custom switchboard. You create a form that's not bound to any table or query. Then you add appropriate controls and actions using the toolbox in the form design view.

If you prefer, you can explore other topics related to building custom applications:

- To view the Fulfill 95 application's final custom switchboards, see Appendix C.
- To take a look at some custom switchboards in other sample applications, see Chapter 28 for some tips.
- To learn how to create custom toolbars and menus for your application, see Chapters 23 and 24.
- To learn about Visual Basic, see Chapter 25.

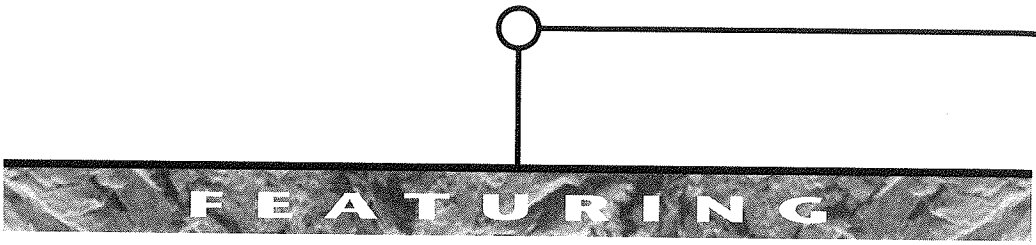
What's New in the Access Zoo?

With Access95, you could create switchboards. Now with Access 97, you have the added flexibility of being able to use hyperlinks on a switchboard to jump to forms and reports. Using hyperlinks in this way is quicker than setting up command buttons with macros and also results in better performance.



Chapter 22

Creating Custom Dialog Boxes



Creating Custom Dialog Boxes

As a Windows user, you've probably seen hundreds of dialog boxes. A dialog box is a window that pops up on the screen to give you information or to ask questions about what you want to do next. You make your selections from the box and then choose OK to proceed. Or in some cases, you can choose a Cancel button to back out of the dialog box gracefully without making any selections.

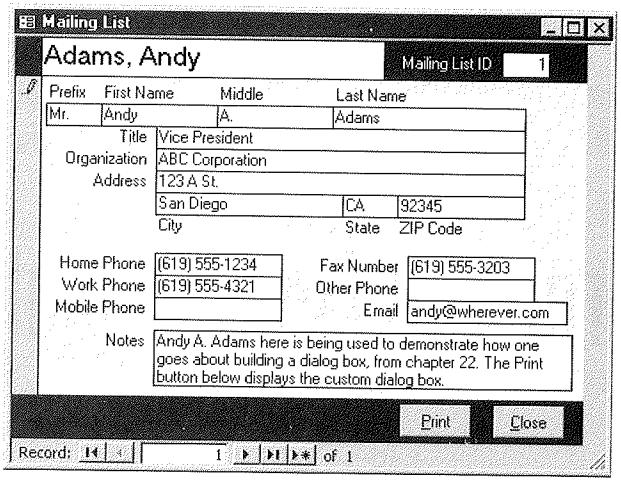
You can create your own custom dialog boxes in your Access applications. The procedure is similar to creating a switchboard: Start off with a blank, unbound form, add some controls, and develop some macros or Visual Basic code to specify what happens when the user selects a control. You can also add some finishing touches, such as OK and Cancel buttons and a special border. In this chapter we'll look at all the factors involved by creating a sample dialog box for a sample database.

Our Goal

We start with a database with a simple name and address table in it. We've also created a Mailing List form for entering and editing data in that table, as shown in Figure 22.1.

FIGURE 22.1

A sample form in a simple database.



In addition, we've created four reports for this database. You can see their names in the database window in Figure 22.2.

Now let's say our goal is to be able to hide the database window from users of this application. In order to print a report, we want the users to click on the Print button at the bottom of the form. When they do so, a pop-up dialog box (see Figure 22.3) allows them to choose one or more reports to print or preview.

FIGURE 22.2

Reports defined for the simple database.

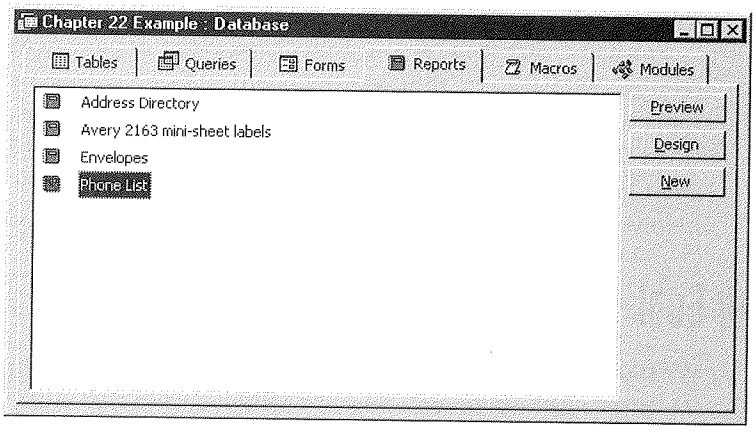
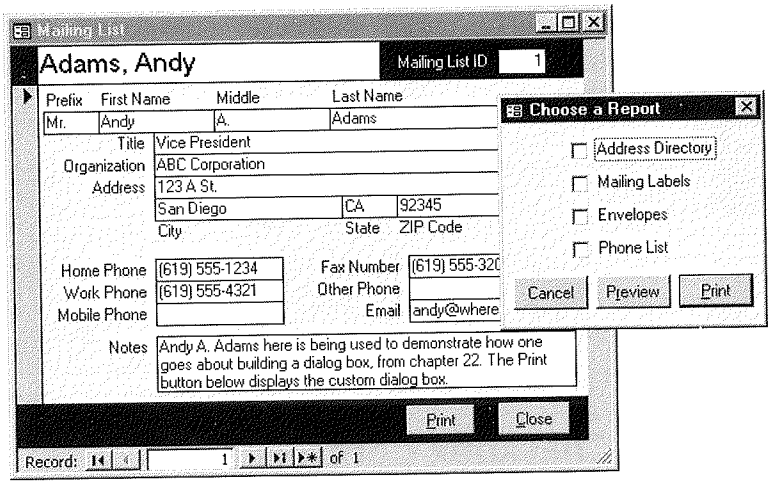


FIGURE 22.3

A custom dialog box appears when the user clicks on the Print button near the bottom of the form.



For the rest of this chapter, we'll look at the exact steps required to create such a dialog box. Remember, in this example we're assuming the table, form, and four reports have already been created. Our job here is simply to create the dialog box.

Step 1: Create the Dialog Box

Creating a blank dialog box is pretty much the same as creating a new, blank switchboard. Here are the steps to get started:

1. Click on the Forms tab in the database window and then click on New.
2. Choose Design View and leave the *Choose the Table or Query* option blank.
3. Choose OK.
4. Open the property sheet (click on the Properties button in the toolbar or choose View ➤ Properties) and click on the Format tab in the property sheet.
5. Set the first few properties in the property sheet to the values shown in Figure 22.4.

NOTE

Remember that the Caption property is the title that will appear in the title bar of your custom dialog box. So be sure to enter a caption that's suitable for the dialog box you're creating.

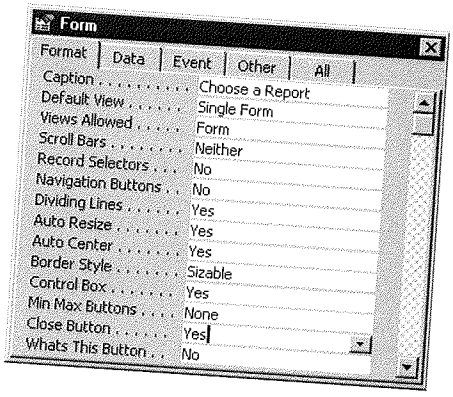


FIGURE 22.4
An unbound form with Format properties set to make the form look like a dialog box.

Add the Checkbox Controls

Now that we have a blank form to work on, we need to add the controls that the user will select from. You can use any of the controls that the toolbox offers. In this example we'll use checkboxes and command buttons. Here are the steps for adding one checkbox:

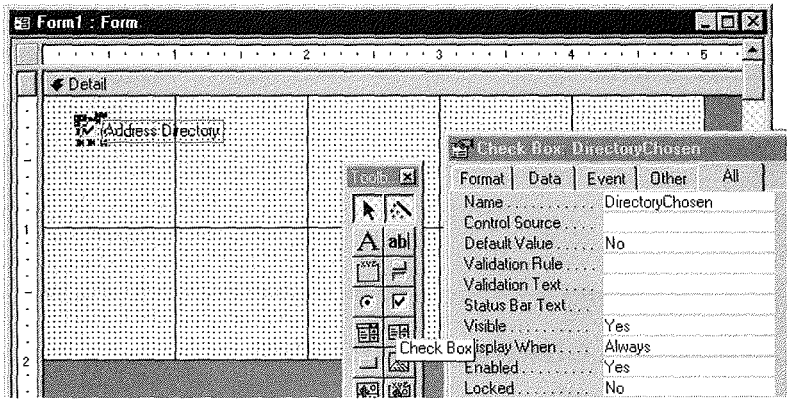
1. Open the toolbox if closed (click on the Toolbox toolbar button or choose View ➤ Toolbox).
2. Click on the Checkbox tool and then click in the form at about where you want the checkbox to appear. Access creates a checkbox with a generic name and caption (most likely Check0).
3. (Optional) Change the caption to something more descriptive, such as Address Directory (just click within the caption and type your change).
4. Open the property sheet and click on the checkbox (so it's selected). Use the All tab to give the checkbox a more descriptive name (e.g., DirectoryChosen) and, optionally, set its default value to No.

Be careful when assigning names to controls that you don't inadvertently assign the name to the control's label. Always click *directly on* the control you want to name before typing a control name into the property sheet. The top of the property sheet always shows the type and current name of the currently selected control.

Figure 22.5 shows our progress. The checkbox is on the form, and the name of that control in the property sheet is DirectoryChosen. The label (caption) for the control (on the form itself) is Address Directory.

FIGURE 22.5

The caption of the first checkbox control in the dialog box is Address Directory; its name is DirectoryChosen.



Next we follow those same steps to create three more checkboxes, one for each possible report. Figure 22.6 shows all four checkboxes in place. Table 22.1 lists the caption for each checkbox and the name we assigned to each checkbox. (You can't see the name of each checkbox because the property sheet shows properties for only one control at a time.)

TIP

Checkboxes can be difficult to align and space evenly. Try using Edit ➤ Select All to select all the controls; then use Format ➤ Size ➤ To Grid, Format ➤ Align ➤ To Grid, and Format ➤ Vertical Spacing to get things in the ballpark. Then you can use other options under Format ➤ Align, as appropriate, to tidy up.

FIGURE 22.6

Four checkbox controls added to our dialog box.

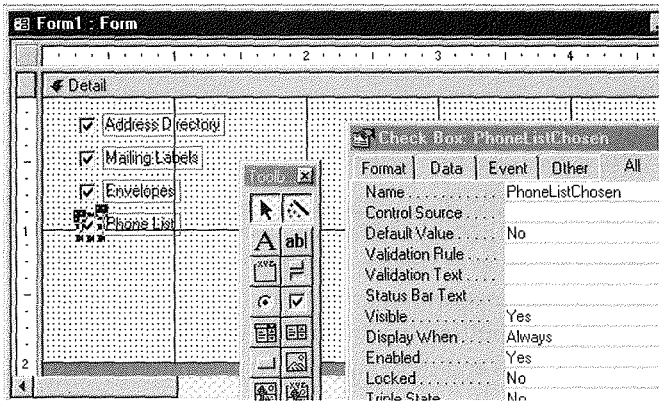


TABLE 22.1: LABELS AND NAMES OF THE CHECKBOXES SHOWN IN FIGURE 22.6

CAPTION	NAME
Address Directory	DirectoryChosen
Mailing Labels	LabelsChosen
Envelopes	EnvelopesChosen
Phone List	PhoneListChosen

Add the Command Buttons

After the checkboxes are in place, we need to add the command buttons. You probably know the routine by now, but let's go through the steps to create one of the command buttons. (The Control Wizards won't really help here because we haven't yet created the macros that will respond to the user's dialog box selections.)

1. Turn off the Control Wizards by clicking the button "out" as below.

2. Click on the Command Button tool and then click in the form where you want the command button to appear. A button with a generic name, such as Command0, appears.
3. Make sure the command button is selected and then use the All tab in the property sheet to give the button a name and caption.

In Figure 22.7 we've created a command button, named it CancelButton, and assigned the caption Cancel.

We repeat steps 1 to 3 to create two additional command buttons, captioned P&review (which shows up as Preview on the button face) and &Print (which shows up as Print). You can then use dragging techniques and the options on the Format menu to size, position, and align the buttons to your liking. Figure 22.8 shows the finished dialog box. Table 22.2 lists the names and captions assigned to those buttons.

FIGURE 22.7
The custom dialog box with a Cancel command button.

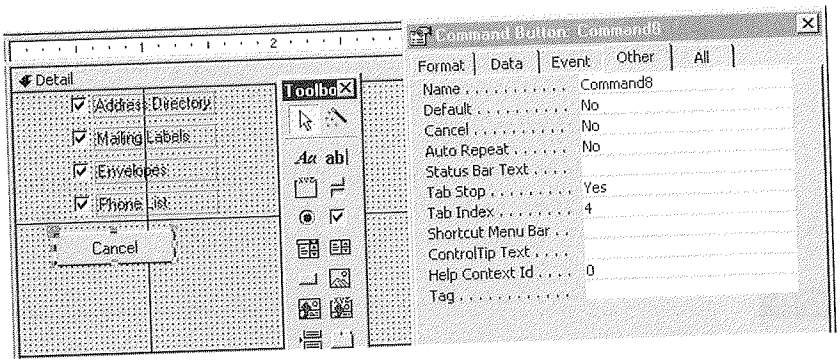


FIGURE 22.8
The custom dialog box with three command buttons.

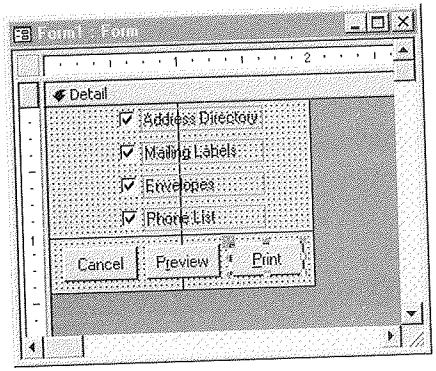


TABLE 22.2: NAMES AND CAPTIONS FOR THE COMMAND BUTTONS SHOWN IN FIGURE 22.8

NAME	CAPTION
CancelButton	Cancel
PreviewButton	P&review
PrintButton	&Print

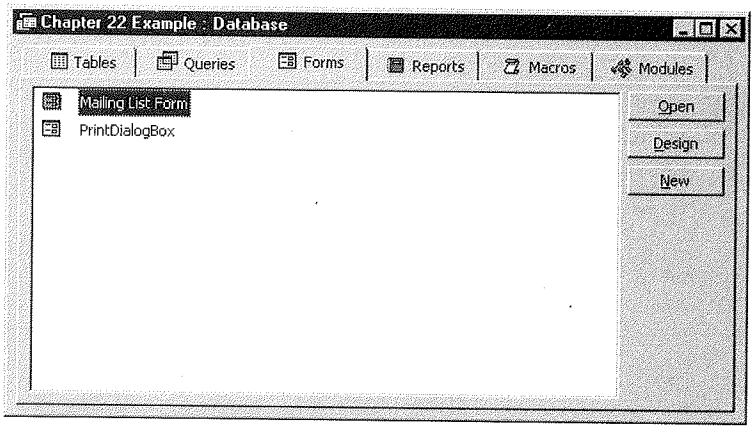
Print, Save, and Close the Form

With the controls in place we can now name and close the form and optionally print some "technical documentation" that will help us develop the macros in the next step. Here are the steps to follow:

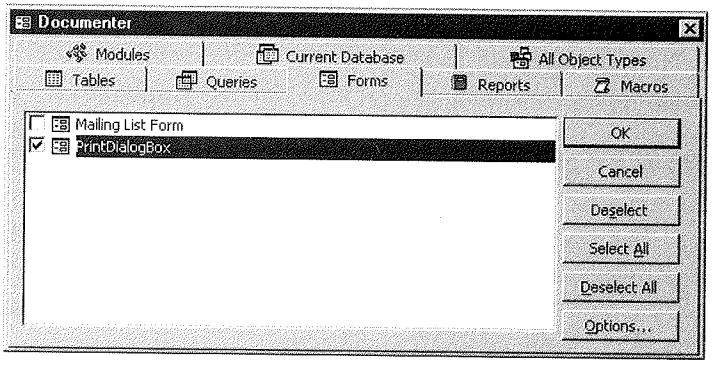
1. Choose File ➤ Close ➤ Yes and enter a name such as PrintDialogBox. The new dialog box name appears in the database window along with any other forms, as in the example shown in Figure 22.9.

FIGURE 22.9

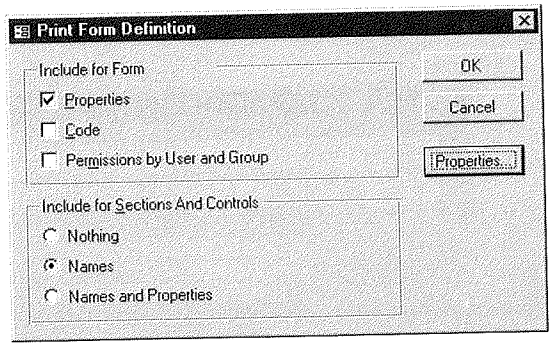
Once closed and saved, the new dialog box is listed right along with any other forms in the database window.



2. Choose Tools ➤ Analyze ➤ Documenter if you want to print the technical documentation.
3. Choose Forms under Object Type and click on the name of the form that you want to document (PrintDialogBox in this example, as shown below):



4. Click on the Options button and limit the display to the options shown below.



5. Choose OK (twice) and wait for the Object Definition window to appear.

You can then use the Print button in the toolbar to print the documentation. Then click on the Close toolbar button to close the Object Definition window and return to the database window.

We'll use the printed documentation to help us remember the exact names we gave to the controls in the dialog box. The names of the controls appear near the end of the printout and will look something like this:

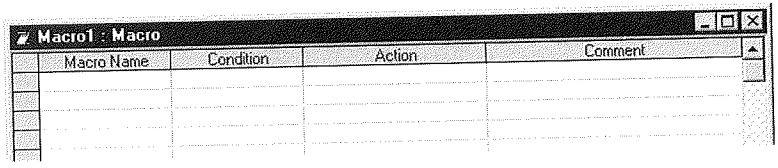
```
Command Button: CancelButton
Check Box: DirectoryChosen
Check Box: EnvelopesChosen
Label: Label1
Label: Label5
Label: Label7
Label: Label9
Check Box: LabelsChosen
Check Box: PhoneListChosen
Command Button: PreviewButton
Command Button: PrintButton
```

Step 2: Create the Macro Actions

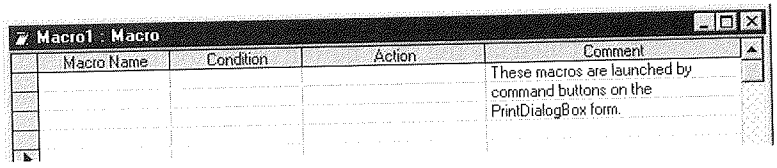
Next we need some macros to define what will happen when the user makes selections from the dialog box. We need to start with a blank macro sheet:

1. Click on the Macros tab in the database window.
2. Click on the New button to get to a new, blank macro sheet.

3. Open the Macro Names and Condition columns using the appropriate options on the toolbar or the View menu. You should see all four column headings listed across the top of the columns.



Now we're ready to start creating the individual macros. You can start off by typing just a comment into the first row(s) of the macro sheet.



Cancel Printing Macro

One of the buttons on the PrintDialogBox form lets the user Cancel—that is, bail out without doing anything. The macro we assign to that button need only close the form. So follow these steps to create that macro:

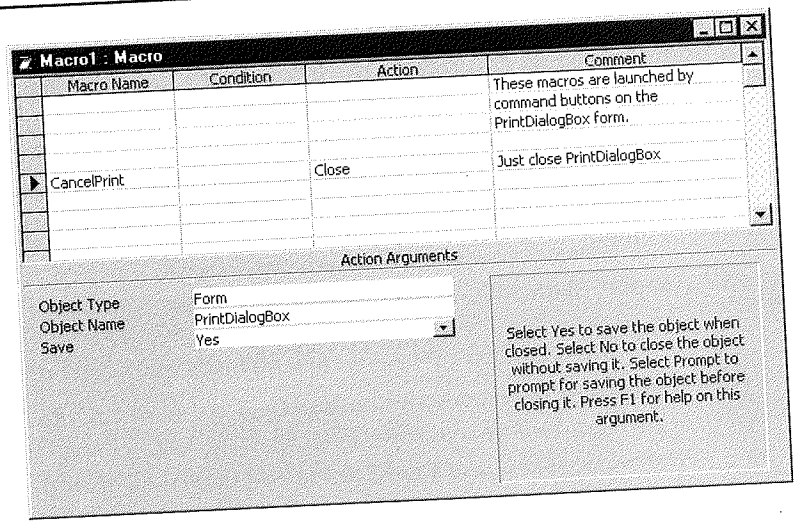
1. Enter a name such as CancelPrint in the Macro Name column in a blank row beneath the comments you typed.
2. Leave the Conditions column empty.
3. Choose Close in the Action column.
4. Fill out the action arguments as follows:

Object Type:	Form
Object Name:	PrintDialogBox
Save:	Yes

5. (Optional) Fill in the Comments column to describe what this macro does.

Figure 22.10 shows the completed first macro.

FIGURE 22.10
The first macro typed into the new macro sheet.



Preview Reports Macro

The next macro is a little trickier than the first because it needs to say, "If the Directory-Chosen checkbox is checked, preview the Address Directory report," and then "If the LabelsChosen checkbox is checked, preview the Avery 2163 minisheet labels report," and so on. So we need to explain one thing about the checkboxes before we do that.

A checkbox is a control that can contain any one of two values, either True (checked) or False (unchecked). We don't actually use the checkboxes to launch an action. Instead, we decide whether to perform some action based on whether a checkbox is checked or not. The "decision" part takes place in the Condition column of the macro. As you may recall from Chapter 20, the Condition column must contain an expression that evaluates to True or False. Since the value of a checkbox is inherently True or False, we only need to use the name of the checkbox in the Condition column of the macro. For example, if I put DirectoryChosen as the condition in a line, then DirectoryChosen proves True if the checkbox is checked and proves False if the checkbox is unchecked.

With that little tidbit in the back of your mind, let's go ahead and create the next macro in this sheet. We'll name this new macro PreviewReports. Here's how to proceed:

1. Leave one blank row beneath the CancelPrint macro. Type the name PreviewReports into the Macro Name column of the new row.
2. Type **[DirectoryChosen]** in the Condition column.

The printed documentation for the form lets you easily look up the exact spelling of the checkbox controls on the form. That's how I "remembered" the Directory-Chosen name. In lieu of using printed documentation, you can use the Expression Builder to locate names of controls on forms.

3. Choose OpenReport in the Action column and fill in the Action Arguments as follows:

Report Name:	Address Directory
View:	Print Preview

4. (Optional) Type a description into the Comment column.

At this point our macro sheet looks like Figure 22.11.

Next we need to repeat steps 2 to 4 to add three more rows to the macro. But we need to refer to different controls and report names. Figure 22.12 shows the complete macro.

Table 22.3 shows the Condition, Action, and Action Argument for each row in the PreviewReports macro.

FIGURE 22.11

Starting the second macro, which we've named Preview-Reports.

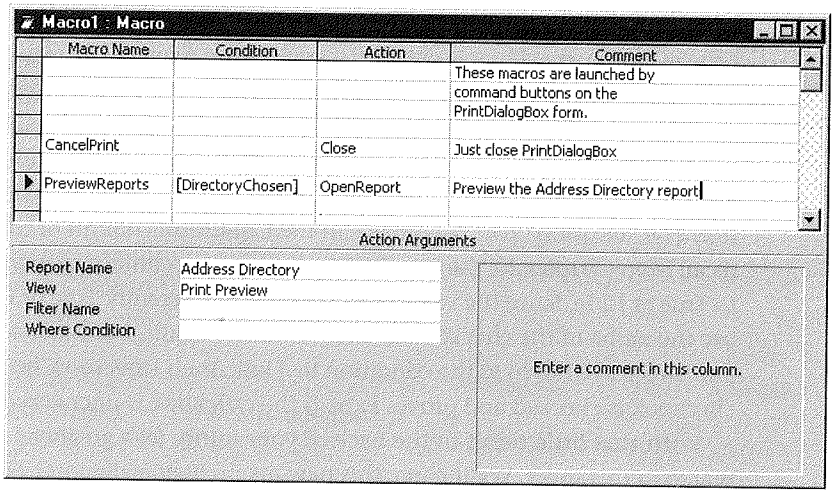


FIGURE 22.12
The Preview-Reports macro defined in our macro sheet.

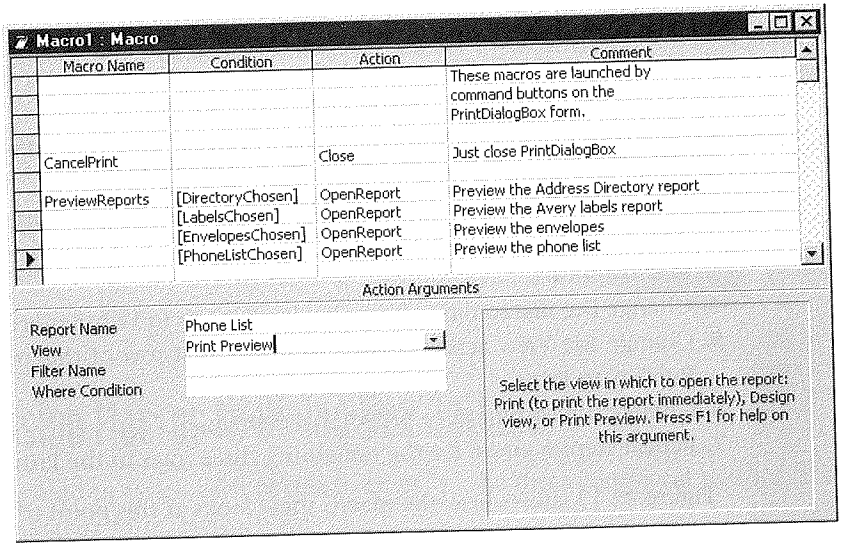


TABLE 22.3: CONDITIONS, ACTIONS, AND ARGUMENTS SHOWN IN FIGURE 22.12

CONDITION	ACTION	ACTION ARGUMENTS
[LabelsChosen]	OpenReport	**Report Name:** Avery 2163 mini-sheet labels **View:** Print Preview
[EnvelopesChosen]	OpenReport	**Report Name:** Envelopes **View:** Print Preview
[PhoneListChosen]	OpenReport	**Report Name:** Phone List **View:** Print Preview

Print Reports Macro

Next we need a macro to print reports. This macro is virtually identical to the Preview-Reports macro except that the View Action Argument for each OpenReport action

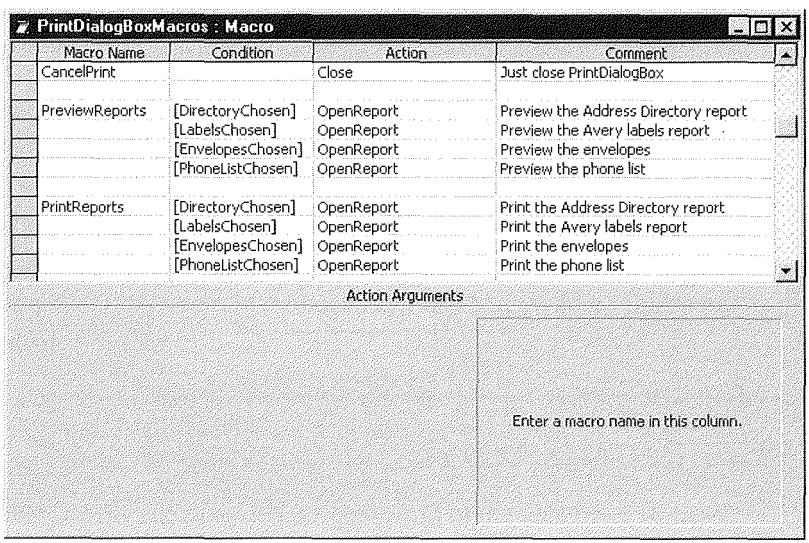
needs to be changed from Print Preview to Print. To create this macro quickly and easily, follow these steps:

1. Hold down the Ctrl key and click on each of the four rows in the PreviewReports macro so that all four rows are selected.
2. Choose Edit ➤ Copy or press Ctrl+C to copy those rows to the Clipboard (nothing happens on the screen).
3. Leave a blank row under the Preview Reports macro, click in the Macro Name column, and choose Edit ➤ Paste (or press Ctrl+V). An exact copy of the PreviewReports macro appears.
4. Change the macro name from PreviewReports to PrintReports.
5. Change the View action argument in the first row of this new macro from Print Preview to Print.
6. Change the comment to reflect this change.
7. Repeat steps 5 and 6 for the remaining three rows in the Print Reports macro.

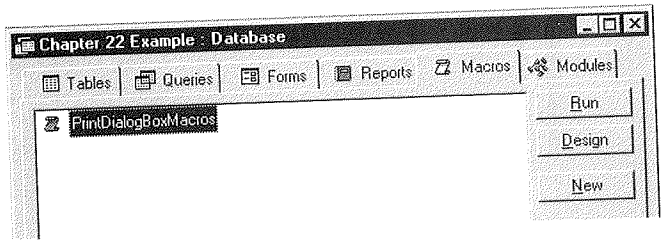
Figure 22.13 shows how the macro sheet looks at this point (though you can only see the Action Arguments for the last row in the macro).

FIGURE 22.13

The PrintReports macro added to the macro sheet.



You may now save and close the macro in the usual manner. That is, choose File ➤ Close ➤ Yes, type in a name such as PrintDialogBoxMacros, and choose OK. The macro name appears in the database window whenever the Macros tab is selected.



Step 3: Assign Macros to Dialog Box Buttons

Next we need to assign each of those macros to the three command buttons in the PrintDialogBox form. Here's how:

1. Click on the Forms tab in the database window, click on the PrintDialogBox name, and then click on the Design button to open that form in design view.
2. Open the property sheet and click on the Events tab.
3. Click on the Cancel button.
4. Click on the On Click property in the property sheet and then use the drop-down list button to choose PrintDialogBoxMacros.CancelPrint as the macro to run when the user clicks that button (Figure 22.14).

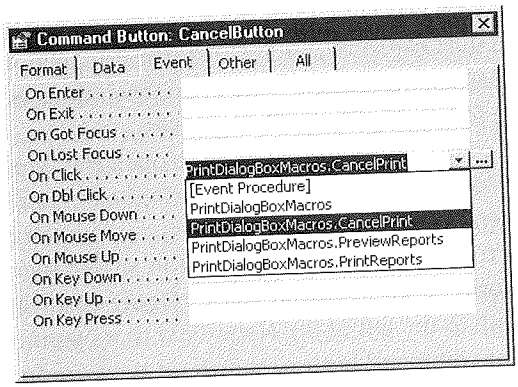
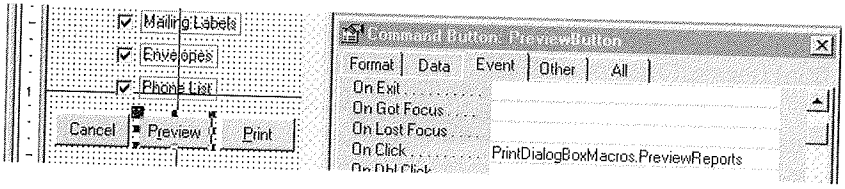
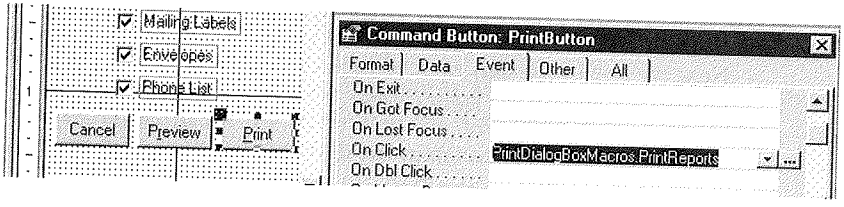


FIGURE 22.14
The PrintReportsMacros.CancelPrint macro assigned to the On Click property of the Cancel button.

5. Click on the button captioned Preview and assign the macro named PrintDialogBoxMacros.PreviewReports to that button.



6. Click on the button captioned Print and assign the PrintDialogBoxMacros.PrintReports to the On Click property of that button.



7. Close and save the form (choose File ➤ Close ➤ Yes).

You're returned to the database window. The dialog box and its macros are complete. Assuming you had created the four reports mentioned at the start of this chapter, you could test the dialog box right now simply by opening it in form view and making selections.

As you may recall from earlier in this chapter (refer to Figure 22.3), we actually assigned this dialog box to the Print button on a form we had created earlier. The simple way to do this would be to open that form in design view, open the toolbox, and turn on the Control Wizards. Create the Print command button and, when the Control Wizard asks for actions, choose Form Operations ➤ Open Form ➤ PrintDialogBox. The caption for the button would be &Print.

You could also use a hyperlink to open the PrintDialogBox form. Click the Insert Hyperlink dialog box, enter PrintDialogBox under *Named location in file*, and click on OK. Then move the hyperlink from the upper-left corner of the form to wherever you want it to appear. Click on the caption and change it to Print (instead of showing the entire form name, PrintDialogBox). Figure 22.15 shows the Mailing List form with a hyperlink to the left of the Notes field that opens the PrintDialogBox form. Note that the Print command button, shown on the form in Figure 22.1, has been removed.

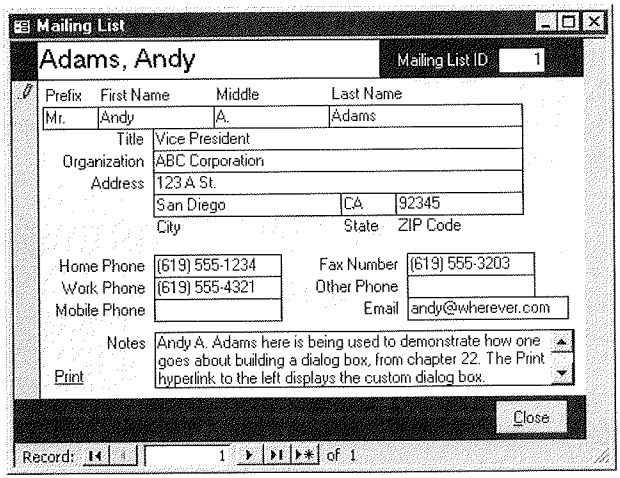


FIGURE 22.15
The Mailing List form with a Print hyperlink instead of a Print command button.

Finishing Touches

You can put a few finishing touches on your dialog box to refine its appearance and behavior, as we'll discuss in the remaining sections in this chapter. As always, these "features" are actually properties or specific controls that you assign to the form using the property sheet in the form design screen.

Modal and Pop-up Properties

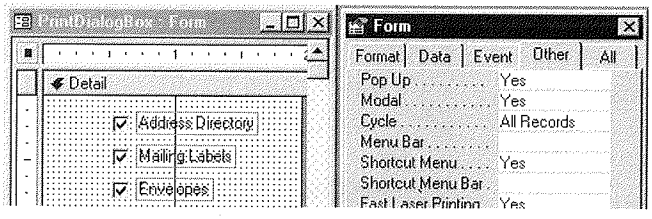
You may have noticed, in your day-to-day use of Windows, that most dialog boxes are "sticky"; that is, once the dialog box is on the screen, you can't just shoo it away by clicking on some other window. You need to specifically complete the dialog box, close the dialog box, or choose the dialog box's Cancel key to get rid of the dialog box.

The technical term for "stickiness" is *modal*. That is to say, most dialog boxes are actually modal windows. By contrast, most "regular" (i.e., application and document) windows are *modeless*, meaning that you can do work outside the window even while the window is on the screen.

A second characteristic of dialog boxes is that they are *pop-up* forms. That is to say, once the window is on the screen, no other window can cover it. You might already be familiar with the Always on Top feature of Windows Help screens. When you activate that feature, you are, in essence, making the Windows Help window a pop-up window.

If you want to give your custom dialog boxes the modal and pop-up characteristics, follow these steps:

1. Open the custom dialog box in form design view.
2. Choose Edit ➤ Select Form to select the entire form.
3. Open the property sheet and click on the Other tab.
4. Set the Modal and Pop-up properties to Yes.



To learn more about modal and pop-up properties and ways to combine them, press F1 while the cursor is on either property within the property sheet.

5. Close and save the form (File ➤ Close ➤ Yes).

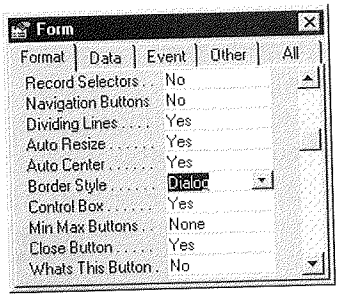
To test your efforts, open the dialog box in the normal form view. When you click outside the dialog box, nothing will happen (except, maybe, you'll hear a beep). The only way to get rid of the dialog box is to specifically close it using one of its command buttons or the Close (×) button in its upper-right corner.

Dialog Box Border Style

Another characteristic of many dialog boxes that make them different from other windows is their border. Many dialog boxes have a thick, black border that cannot be sized. If you want to give your custom dialog box that kind of border, follow these simple steps:

1. Open the custom dialog box in form design view.
2. Choose Edit ➤ Select Form to select the entire form.
3. Open the property sheet and click on the Format tab.

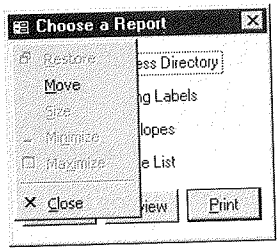
4. Set the Border property to Dialog.



TIP To learn more about border styles, press the Help key when the cursor is in the Border Style property box.

5. Close and save the form normally (File ➤ Close ➤ Yes).

To see the effects, open the dialog box in form view. Then try sizing the dialog box by dragging one of its edges or corners. Can't be done! If you try to "trick it" by using commands in the control menu (in the upper-left corner of the dialog box), no go. The menu will now offer only the Move and Close options, as illustrated below.



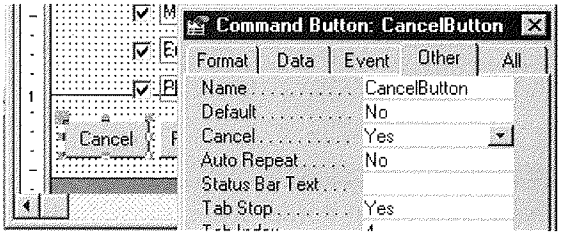
Default and Cancel Buttons

Two last features that many dialog boxes share are cancel and default buttons:

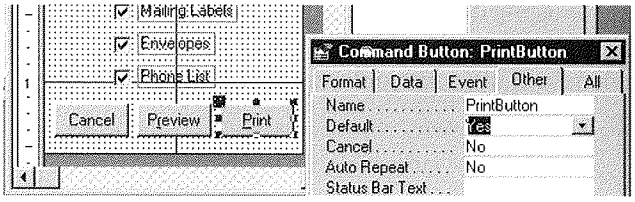
- **Cancel button** The button that gets pushed automatically when the user presses the Escape key.
- **Default button** The button that is automatically selected when the user presses Enter. This button will also have a darker border than other buttons on the same form.

You can make one (and only one) button in your dialog box the default button and any other single button the cancel button. Here's how:

1. Open your custom dialog box in design view.
2. Open the property sheet.
3. Click on the Other tab.
4. If you want to make a button into the Cancel button, first click on that button to select it. Then set its Cancel property to Yes, as below.

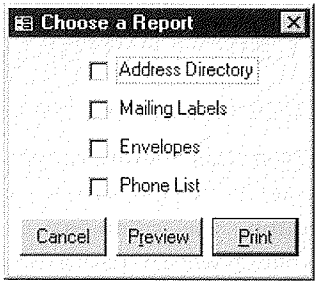


5. If you want to make some other button the default button, first click on that button to select it. Then set its Default property to Yes, as below.



6. Close and save the form normally (File ➤ Close ➤ Yes).

When you reopen the dialog box in form view, the only visual difference you'll see is the darker border around the default button (the Print button in the example below).



You can test the new properties by pressing the Escape or Enter key while the form is on the screen.

What we've learned here is the big secret to custom dialog boxes: They're really just forms that aren't bound to any particular table or query. You use the toolbox in form design to add controls, and maybe some hyperlinks, to that form. Then you create macros (or Visual Basic code) to define the actions that the dialog box will perform. You can even make your dialog box behave like the dialog boxes in bigger Windows applications by setting Modal, Pop-Up, and Border Style properties to the form as a whole. You can also assign the Cancel and Default properties to any two command buttons on the form.

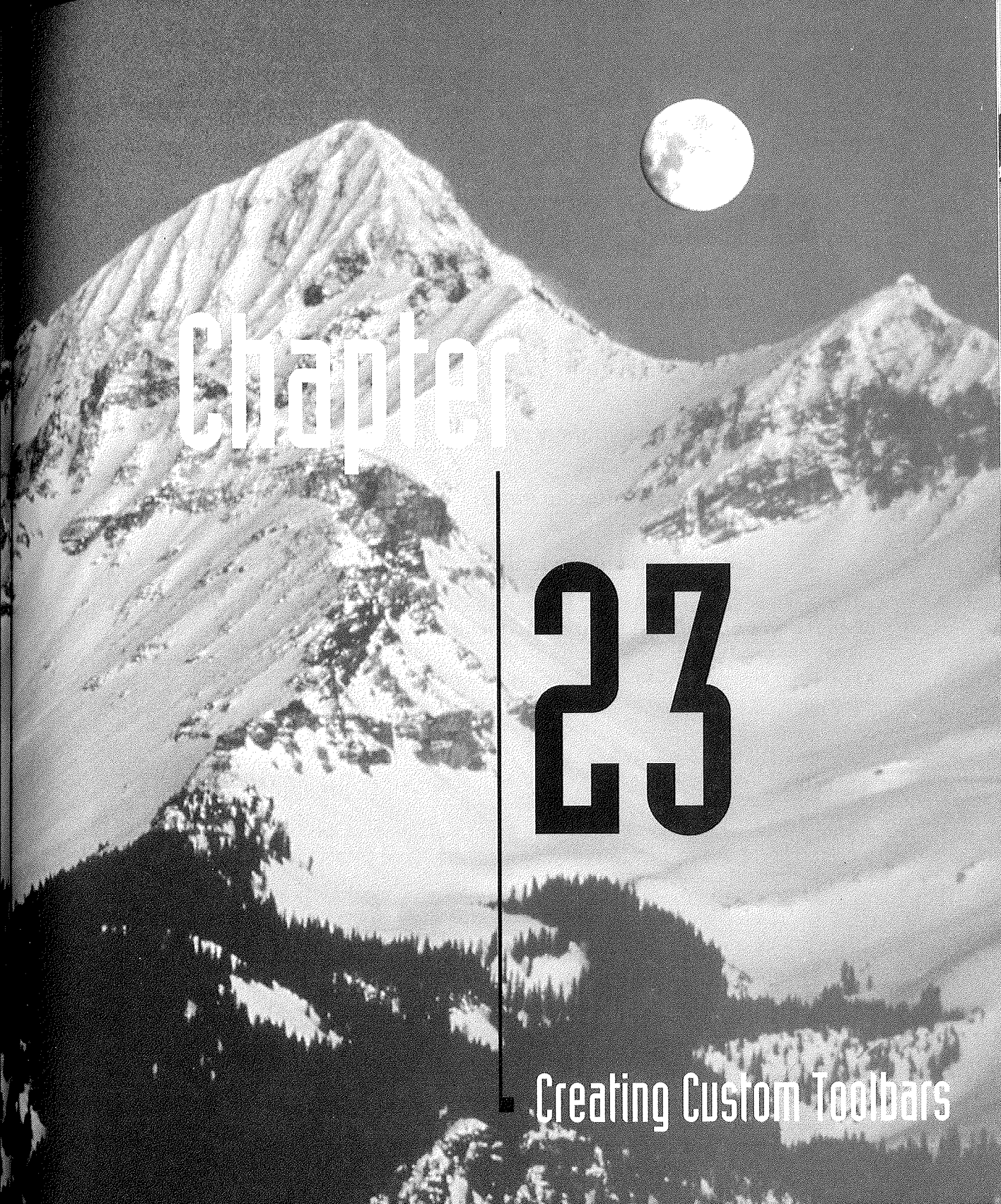
Where to Go from Here

In the next two chapters we'll look at techniques for creating custom toolbars and menus. Those two features will add even more professional polish to your custom Access applications. Here are some other chapters you might want to explore:

- To get a refresher on the mechanics of creating macros, see Chapter 20 "Using Macros to Create Custom Actions."
- To see a custom application with lots of custom dialog boxes, try the Fulfill sample database on the CD (see Appendix C).
- To learn about exploring custom applications behind the scenes, see Chapter 28.

What's New in the Access Zoo?

Creating custom dialog boxes with Access 97 is the same as it was previously with one exception: you can now use hyperlinks, as well as controls like checkboxes and command buttons, to interact with users.



Chapter 23

Creating Custom Toolbars

FEATURING

Creating Custom Toolbars

Hanna Barbera got it right in "The Jetsons"; most of us have ended up with push-button jobs. Microsoft's toolbars are a perfect example because they let you do virtually anything with the click of a button. With Access 97, you can also add menu commands to toolbars to create "command bars" of all the tools and menus you use most.

Access's Toolbars

Microsoft Access comes with many built-in toolbars. Most of them are tied to specific views and are named accordingly:

Database	Relationship
Table Design	Table Datasheet
Query Design	Query Datasheet
Form Design	Form View
Filter/Sort	Report Design
Print Preview	
Formatting (Form/Report)	Formatting (Datasheet)
Macro Design	Visual Basic

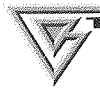
Other built-in toolbars that aren't attached to a specific view include

- **Utility 1** and **Utility 2 toolbars** For creating your own custom toolbars.
- **Web** For browsing Web documents and searching the Web.
- **Toolbox** Offers buttons for creating controls in form design and report design. It's generally free-floating, but can be docked like any other toolbar (see the next section).

Hiding/Displaying the Built-in Toolbars

You can hide or display any number of toolbars at any time. Just follow these procedures:

- **To enable or display *all* the built-in toolbars,** choose View ➤ Toolbars ➤ Customize. Then check each toolbar that you want displayed in the Toolbars dialog box. Choose Close when you are finished.
- **To hide or display a specific toolbar,** right-click on a toolbar and uncheck the name of the toolbar, or choose View ➤ Toolbars. Then check or uncheck the toolbar you want to hide or display.
- **To move a toolbar,** move the mouse pointer to any blank space in the toolbar and drag the toolbar to wherever you want to put it.
- **To dock a toolbar,** drag it to the edge of the screen until its outline expands to the width or height of the screen and then release the mouse button.
- **To undock a toolbar** so that it becomes free floating, just move it away from the edge of the screen.



You can quickly dock or undock a toolbar by double-clicking on any blank space in the toolbar. To hide a floating toolbar, click on the small close button in the toolbar's upper-right corner.

Controlling the Size and Appearance of Toolbars

You can control the size of the buttons and the appearance of any toolbar by following these steps:

1. Right-click on any toolbar and choose Customize, or choose View ➤ Toolbars ➤ Customize and click the Options tab.
2. Choose any combination of appearance features from the lower part of the dialog box:
   - **Large icons** Choose this option to make the buttons larger (handy on small laptop-size screens or on screens with resolutions higher than VGA).
   - **Show ScreenTips on toolbars** Clear this option if you don't want your toolbar to display ScreenTips.

- **Show shortcut keys in ScreenTips** Check this option if you want to show a button's shortcut key with its ScreenTip when you point to it.

3. Choose Close after making your selection(s).

Modified versus Custom Toolbars

As an application developer, you need to be aware of the difference between a modified built-in toolbar and a custom toolbar:

- **Modified existing toolbar** If you modify an existing toolbar, that version of the toolbar will appear in *all* your databases.
- **Custom toolbar** When you create a new custom toolbar, it appears only in the database in which it was created.

NOTE

The built-in Utility 1 and Utility 2 toolbars are initially blank. When you add buttons to those toolbars, that counts as modifying an existing toolbar—not as creating a new, custom toolbar. In other words, the Utility 1 and Utility 2 toolbars are accessible from all your databases.

Empowering/Limiting Your Users

You can use custom toolbars to determine what the users of your application can and can't do. For example, if you want users to be able to create and change objects, you can include design buttons on your toolbars. On the other hand, if you don't want the users to modify objects, you can keep them away from the design screens by excluding design buttons from your application's custom toolbars.

NOTE

You'll need to create custom menus, discussed in the next chapter, to determine exactly how much freedom your user has.

Creating a Custom Toolbar

Here's how to create a new custom toolbar:

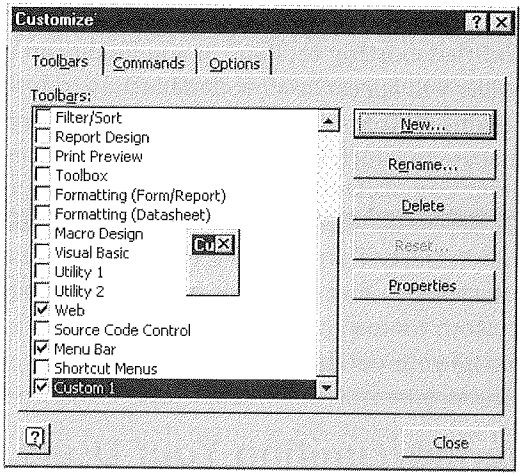
1. Make sure that the database you want to put the toolbar into is the currently open database.

2. Right-click an existing toolbar and choose Customize, or choose View ➤ Toolbars ➤ Customize. Then click New on the Toolbars page.
3. Enter a name (up to 64 characters) for your new toolbar and then choose OK.

A tiny (and sometimes hard to see) empty toolbar appears on the screen, as shown in Figure 23.1.

FIGURE 23.1

A new, blank toolbar and the Customize Toolbars dialog box.



Adding and Deleting Buttons

To add buttons to your new toolbar, you can either use the Commands tab of the Customize dialog box or copy buttons from one toolbar to another.

Using the Commands Tab to Add Buttons

If you have the Customize dialog box open, follow these steps to add buttons to a toolbar:

1. Click the Commands tab of the Customize dialog box. Choose a category of button type from the Categories list (just click on any category name).
2. Click on whichever Commands button you think you might want to add to your toolbar. Click the Description button under Selected Command to check the ScreenTip and description to make sure you know what the button will do.
3. Drag the button to your custom toolbar.

In Figure 23.2 we've already dragged a few of buttons to the custom toolbar and are now examining buttons in the File category. We've also dragged the new toolbar, Custom 1, from where it appeared on top of the Customize dialog box to a spot where it's easier to work with.

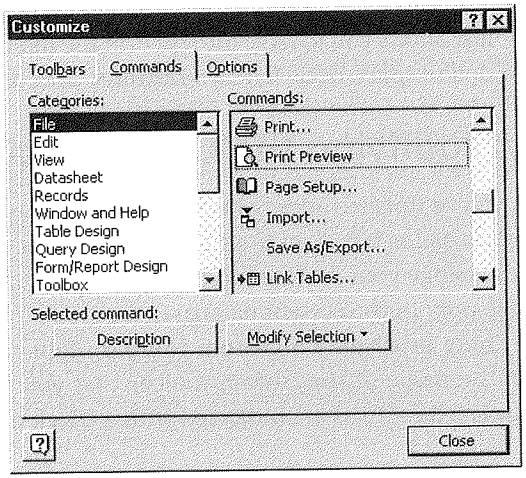


FIGURE 23.2
Here we've just dragged two buttons to our custom toolbar and are browsing the Commands list in the Customize dialog box for more buttons to add.

Copying or Moving Buttons between Toolbars

It is also possible to copy or move buttons from one toolbar to another. First make sure that both toolbars are visible. What you do next depends on whether the Customize dialog box is already open. If it is, just drag a button from one toolbar to another to move it. To copy a button with the Customize dialog box open, press Ctrl while you drag it from one toolbar to another. If the Customize dialog box is not open, press Alt while moving or copying.

Deleting Buttons

Deleting a button from a toolbar is also a simple task. First open the Customize dialog box. Next show the toolbar you want to change, if it's not already visible. (You may need to drag the Customize dialog box out of the way so you can see the button you want to delete.) Then just drag the button off the toolbar. You can also right-click the button you want to delete and choose Delete.

Refining a Toolbar

You can use any of these techniques to refine your custom toolbar while you're viewing the Customize dialog box:

- **To remove a button**, drag it off of your custom toolbar.
- **To move a button to a new location on the toolbar**, drag the button to its new location.

- **To add space between buttons**, drag the button slightly to the right (a distance a little less than half the width of the button). (Closing the dialog box and docking the toolbar allows you to have the space on the toolbar to undertake this operation.)
- **To delete space between two buttons**, drag one button slightly to the left.

Saving/Modifying the Custom Toolbar

When you've finished adding buttons to your custom toolbar, choose Close from the Customize dialog box. You can then use any of these techniques, at any time, to view, hide, or change your custom toolbar (but don't forget, your custom toolbar will be available only in the current database):

- **To hide or display a custom toolbar**, right-click on any toolbar and then click on the name of the custom toolbar that you want to hide or display. Currently displayed toolbars are indicated with a check mark.

If no toolbars are visible, choose View ➤ Toolbars and click on a toolbar name.

- **To change a custom toolbar**, first display that toolbar, right-click on it, and choose Customize to return to the Customize dialog box. There you can make changes using the same techniques that you used to create the custom toolbar.
- **To delete a custom toolbar**, choose View ➤ Toolbars ➤ Customize and click the Toolbars tab if it's not already active. Then scroll down to the name of the custom toolbar you want to delete, click on it to highlight it, click on the Delete button, and choose Yes.
- **To rename a custom toolbar**, choose View ➤ Toolbars ➤ Customize, scroll down to the name of the custom toolbar you want to rename, and click on the Rename button. Type a new, unique name for your toolbar and choose OK.

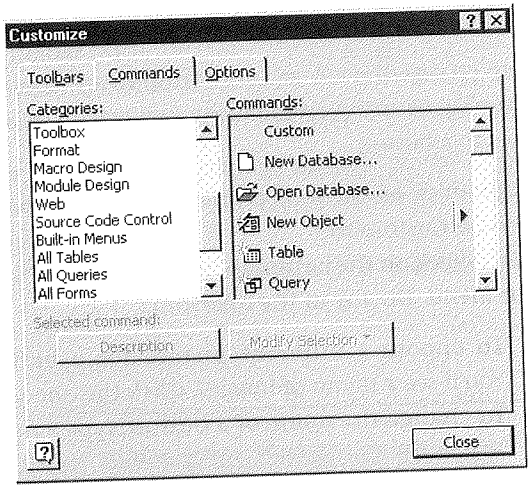
The Delete and Rename buttons aren't visible in the Toolbars dialog box when the highlight is on a built-in toolbar because you can't delete or rename those toolbars.

- **To move/dock/undock a custom toolbar**, use the same techniques you'd use with a built-in toolbar, as described earlier in this chapter.

Creating Your Own Buttons

You're not limited to creating buttons that perform built-in Access tasks. You can create your own buttons to run macros, open tables, preview reports, and more. The general procedure is the same as for "regular" buttons. You just need to choose your buttons from the categories that start with the word *All*. Here are the steps:

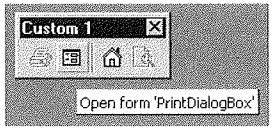
1. Display the toolbar to which you want to assign a custom button.
2. Right-click on that toolbar and choose Customize. Then click the Commands tab in the Customize dialog box.
3. Scroll down to and select one of the last few categories under Categories (beginning with the word *All*). The Objects list shows the names of all the objects in the current database that fall into that category (see below).



4. Drag the name of any object to your toolbar.
5. Repeat steps 3 and 4 to add as many buttons as you like and then choose Close.

A default button for that type of object appears on your toolbar. (You can change the button, as you'll see in the next section.)

When you move the mouse pointer to the custom button, the status bar and (in a couple of seconds) the ScreenTip describe what the button will do, as illustrated below.



TIP

You can also drag the name of any object from the database window into the toolbar to instantly create a button that displays that object.

Changing a Button's Face/Description

You can change the face of any button in any toolbar, and you can change the name or ScreenTip of any custom button you create. Here's how to make these types of changes to a button:

1. Right-click on the toolbar that contains the button you want to change and then choose Customize.
2. Right-click on the button in the toolbar that you want to change to show it's shortcut menu. Do any of the following:
   - **To choose a different picture for a button**, select Change Button Image. You'll see a menu of images. Click the one you want to use.
   - **To change the name of a button**, type a new value in the box after Name.
   - **To change the ScreenTip**, choose Properties and enter the tip you want to see in the box after ScreenTip.
   - **To show text instead of a picture**, choose Text Only (Always). The text is taken from the Name property.
   - **To show text when the button is on a menu**, choose Text Only (in Menus). As you'll see later in this chapter, Access 97 lets you add buttons to menus or add menu commands to toolbars. (You can also copy, paste, and reset button images using other items on a button's right-click menu.)

NOTE

Remember, you can change the Description only on custom buttons—not on the built-in buttons.

3. Repeat step 2 to choose a face and/or name and ScreenTip for as many buttons as you wish. Then choose Close when you're done.

Resetting a Button Face

If you change a button face on a built-in button and then decide to go back to the original button face:

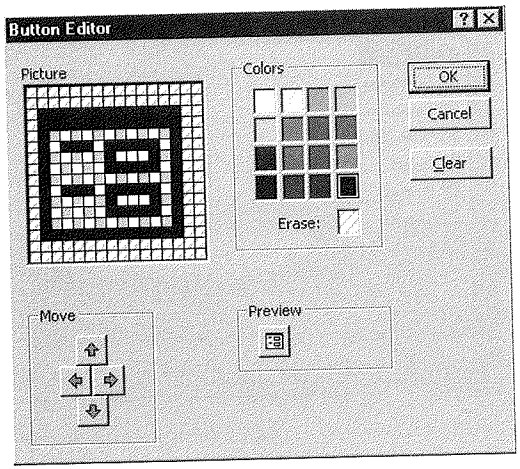
1. Right-click on the button face you want to reset and choose Customize from the shortcut menu (if the Customize dialog box is not already open).
2. Right-click on the button again and choose Reset Button Image.
3. Click on Close in the Customize dialog box.

Creating Your Own Button Face

So what do you do when you want to create your own button image? Once you have added a button to the toolbar, with the Customize dialog box open, right-click on the button. Select Edit Button Image from the context menu and the Button Editor appears (see Figure 23.3). To create your own button image, follow these steps:

1. To change the color of a pixel, first click on the color in the Colors frame and then click on the box on the Picture grid that represents the pixel. (Select the Erase color box to erase a pixel.)
2. To scroll the Picture grid (not all of it appears in the box), click on the arrows below the grid.

FIGURE 23.3
The Access Button Editor.



3. To see what your new button image looks like, check the Preview frame.
4. To clear the button face, click on the Clear button.
5. To save the button image, click on OK.

Adding Toolbars to Your Custom Application

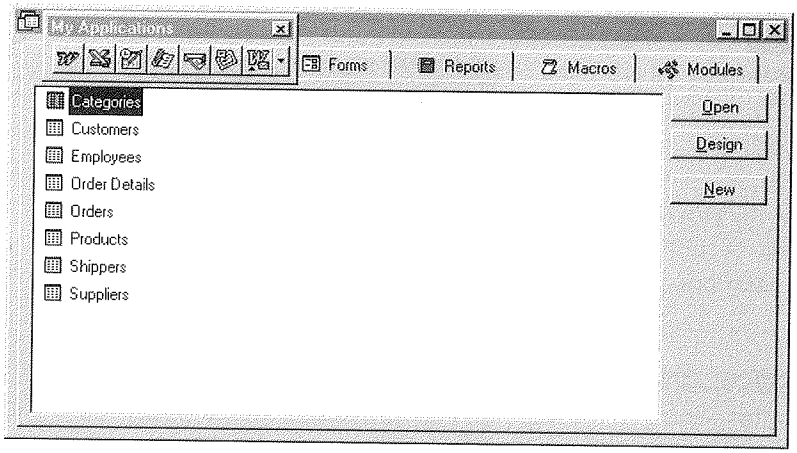
As an application developer, you'll want to control exactly *which* toolbar appears *when*. First create a database with the Database Wizard or open a database that you have already created so that you can work through a couple of examples. (We will use the Northwind Traders database included with Access for the examples in this chapter.) Create a custom toolbar and add the buttons to it that you use most often when you work with a database.

The custom toolbar we created includes tools that switch to our other applications. It reflects the fact that much of the time we are working in Access and switching to other applications to perform less frequent tasks. Figure 23.4 shows this custom toolbar, which we creatively named My Applications.

Since we obviously use Microsoft applications, we could have just used the Microsoft toolbar. However, most users have other applications. If Access doesn't provide a button for your application, you can use Visual Basic for Applications to launch it and attach that code to a button on the toolbar.

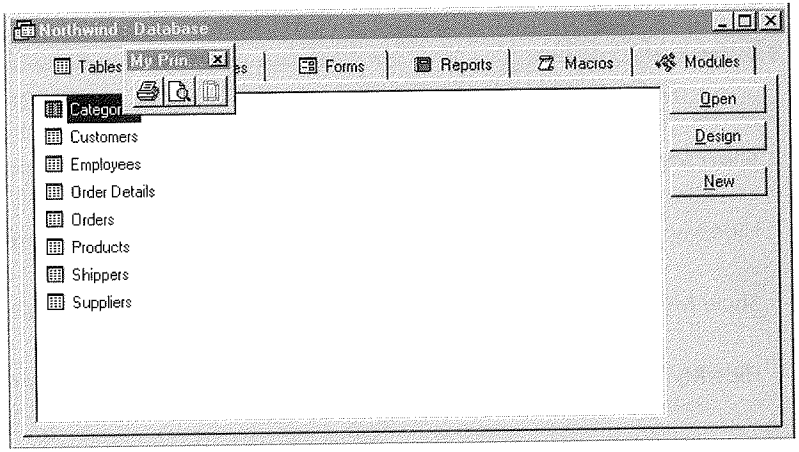
FIGURE 23.4

The custom toolbar My Applications displayed in the Northwind Traders database.



We then created a second custom toolbar, named My Printing Preview, which contains icons for printing, print preview, and page setup (see Figure 23.5).

FIGURE 23.5
The custom My Printing Preview toolbar.



Creating Macros to Show or Hide Custom Toolbars

After you've created your custom toolbars, you need to create macros to show and hide them. In the Northwind Traders application, we put all those macros into a single macro group named Global Macros, as shown in Figure 23.6.

FIGURE 23.6
The macro group named Global Macros contains the macros that show or hide the custom toolbars.

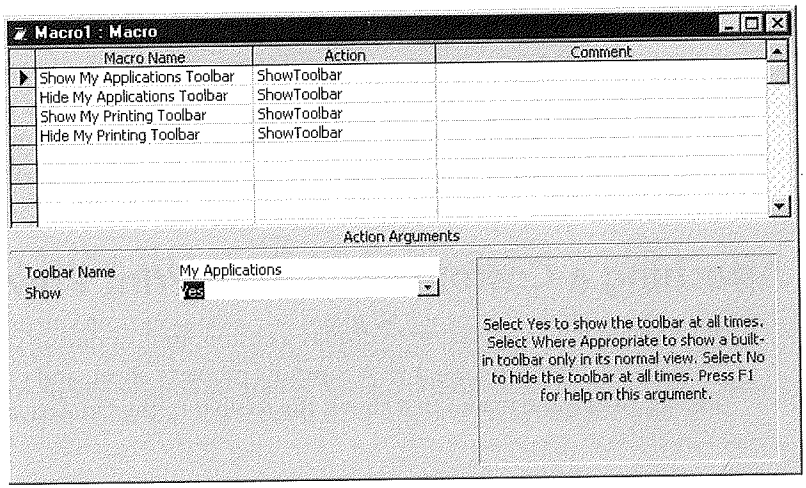


Table 23.1 shows the Macro Name, Action, and Action Argument of each macro. (Notice that this macro group does not have a Condition column.) Basically, each macro uses a single ShowToolBar action. The Action Arguments for each Action name the toolbar to show or hide and then use Yes to show the toolbar or No to hide that toolbar.

TABLE 23.1: SAMPLE MACROS TO HIDE AND SHOW CUSTOM TOOLBARS

MACRO NAME	ACTION	ACTION ARGUMENTS
Show My Applications Toolbar	ShowToolBar	**Toolbar Name:** My Applications **Show:** Yes
Hide My Applications Toolbar	ShowToolBar	**Toolbar Name:** My Applications **Show:** No
Show My Printing Toolbar	ShowToolBar	**Toolbar Name:** My Printing **Show:** Yes
Hide My Printing Toolbar	ShowToolBar	**Toolbar Name:** My Printing **Show:** No

Attaching Toolbars to Forms

In order to attach a toolbar to a particular form, you need to execute, from an event on the form, the macro that displays (or hides) the toolbar.

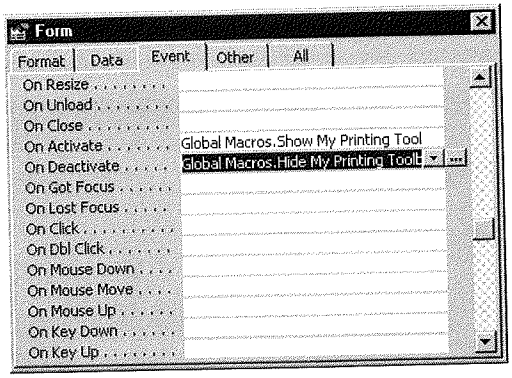
1. Open the form (in design view) that you want to display a custom toolbar.
2. Open the property sheet, select the Event tab, and choose Edit ➤ Select Form.
3. Assign the macro that *shows* the toolbar to the On Activate property.
4. Assign the macro that *hides* the toolbar to the On Deactivate properties of that form.

Figure 23.7 shows an example using the Northwind Traders application, in which we display the My Printing toolbar when the form appears and hide that toolbar when the user is done with the form. By using the On Activate and On Deactivate properties, we can make sure the toolbar is visible whenever the user is working with this form and hidden whenever she or he moves the focus to another form.

Attaching a Custom Toolbar to Print Preview

If you want your application to display a custom toolbar during print preview, you need to open the report in design view, open its property sheet, and choose Edit ➤ Select Report. Assign the macro that shows the toolbar to the On Activate event properties.

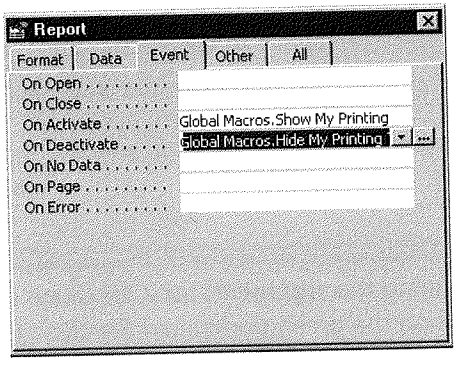
FIGURE 23.7
Form event properties for the Northwind Traders customer phone list form.



Assign the macro that hides the toolbar to the On Deactivate event properties. The example in Figure 23.8 uses a report from the Northwind Traders application.

By the way, we know that all these form and report event properties can be confusing. For help while assigning macros to these properties press F1 or search help for *Order of Events*.

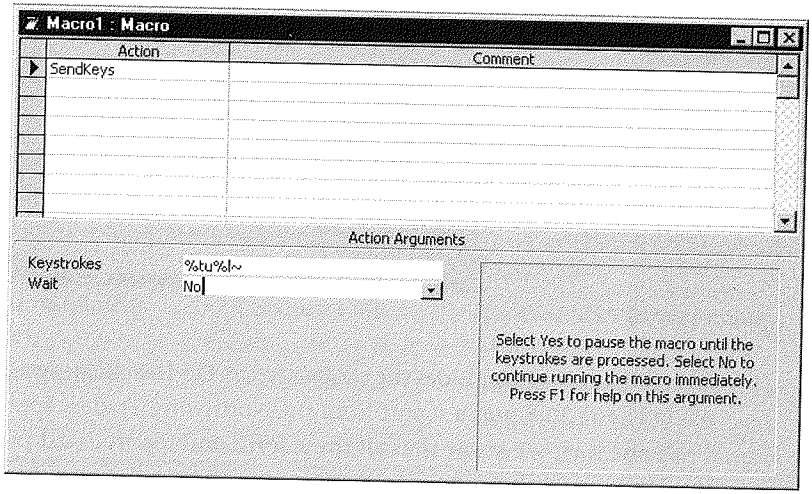
FIGURE 23.8
Macros to hide and display a custom toolbar when the user looks at the report named Alphabetical List of Products in print preview.



Macro to Hide the Built-in Toolbars

When creating an application, you might decide to hide all the built-in toolbars from the user. As you know, you can turn off the built-in toolbars manually through the Startup dialog box. If you want your application to turn off those toolbars, have your AutoExec

macro send the necessary keystrokes at startup. You can use a SendKeys action to have the macro press the appropriate keys, as in the following example.



Notice the Keystrokes entries in the action argument for the SendKeys action:

%t	Presses Alt+T to open the Tools menu
u	Types **u** to choose Startup
%l	Unchecks the Allow Built-in Toolbars checkbox
~	Presses Enter to choose OK

TIP

When defining the arguments for a SendKeys action in a macro, press F1 for help. Then click on the green underlined *SendKeys* jump word and scroll through that Help screen to find the codes you need to represent various keystrokes.

Redisplaying the Built-in Toolbars

If you want your application to redisplay the built-in menus when the user quits the application, have your "quit" macro execute a SendKeys action to restore the built-in toolbars. You might also want to have that macro redisplay the database window.

TIP The same macro that turns the built-in toolbars off will turn them back on.

If you ever need to turn on the built-in toolbars manually, go to the database window and choose Tools ➤ Startup from the menu bar. Then check the Allow Built-in Toolbars checkbox and choose OK. If no toolbar appears, choose View ➤ Toolbars and check Database.

Modifying a Built-in Toolbar

So far in this chapter we've focused on creating custom toolbars for your custom applications. But you may also want to modify Access's built-in toolbars to better suit your own needs. As mentioned earlier, when you modify a built-in toolbar, that toolbar becomes accessible in all your databases.

NOTE A custom toolbar is stored in the database it was created in and is available only in that database. Built-in toolbars (modified or not) are stored in the Access workgroup information file and are available to any database.

To modify a built-in toolbar:

1. Display the built-in toolbar that you want to modify.
2. Right-click on that toolbar and choose Customize.
3. Make changes using the techniques described under "Adding and Deleting Buttons" earlier in this chapter.
4. Choose Close when you've finished.

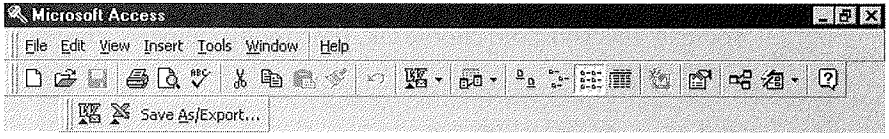
That modified version of the built-in toolbar will appear in all your databases.

Combining Menus and Toolbars

A new feature in Access 97 lets you create *command bars* by adding menu commands to toolbars and buttons to menus. These hybrid bars can have virtually any combination of menus commands and toolbar buttons you can imagine.

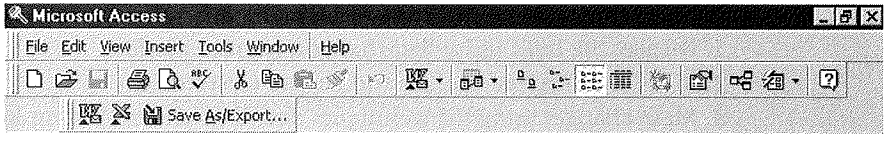
You may have already noticed this feature while experimenting with the Customize dialog box. For example, if you select the File category on the Commands tab, you'll find commands, such as Save As/Export, that are not represented by a button. You can

drag these commands to a toolbar, and they show up as text. The custom toolbar below includes two buttons and the Save As/Export command. When you click Save As/Export, Access opens the Save As dialog box, as if you had chosen File ➤ Save As/Export from the menus.



Once you add a menu command to a toolbar, you are free to show it as a picture instead of text:

1. Right-click the toolbar you want to change and choose Customize. (If the toolbar isn't visible, choose View ➤ Customize and check the desired toolbar first.)
2. Right-click the command button you want to show as a picture on the toolbar. Choose Change Button Image and select an image.
3. Right-click the button again and check Default Style if you want to show the button as a picture without text. Or leave Image and Text checked to show the button as a combination of an image and text. The Save As/Export command on the toolbar below has this property checked.



Resetting a Built-in Toolbar

If you want to reset a built-in toolbar to its original state, right-click on a toolbar and choose Customize. Make sure the Toolbars tab is selected in the Customize dialog box and click on the name of the built-in toolbar that you want to reset. Then click on the Reset button, choose OK, and click on Close in the Customize dialog box.

Where to Go from Here

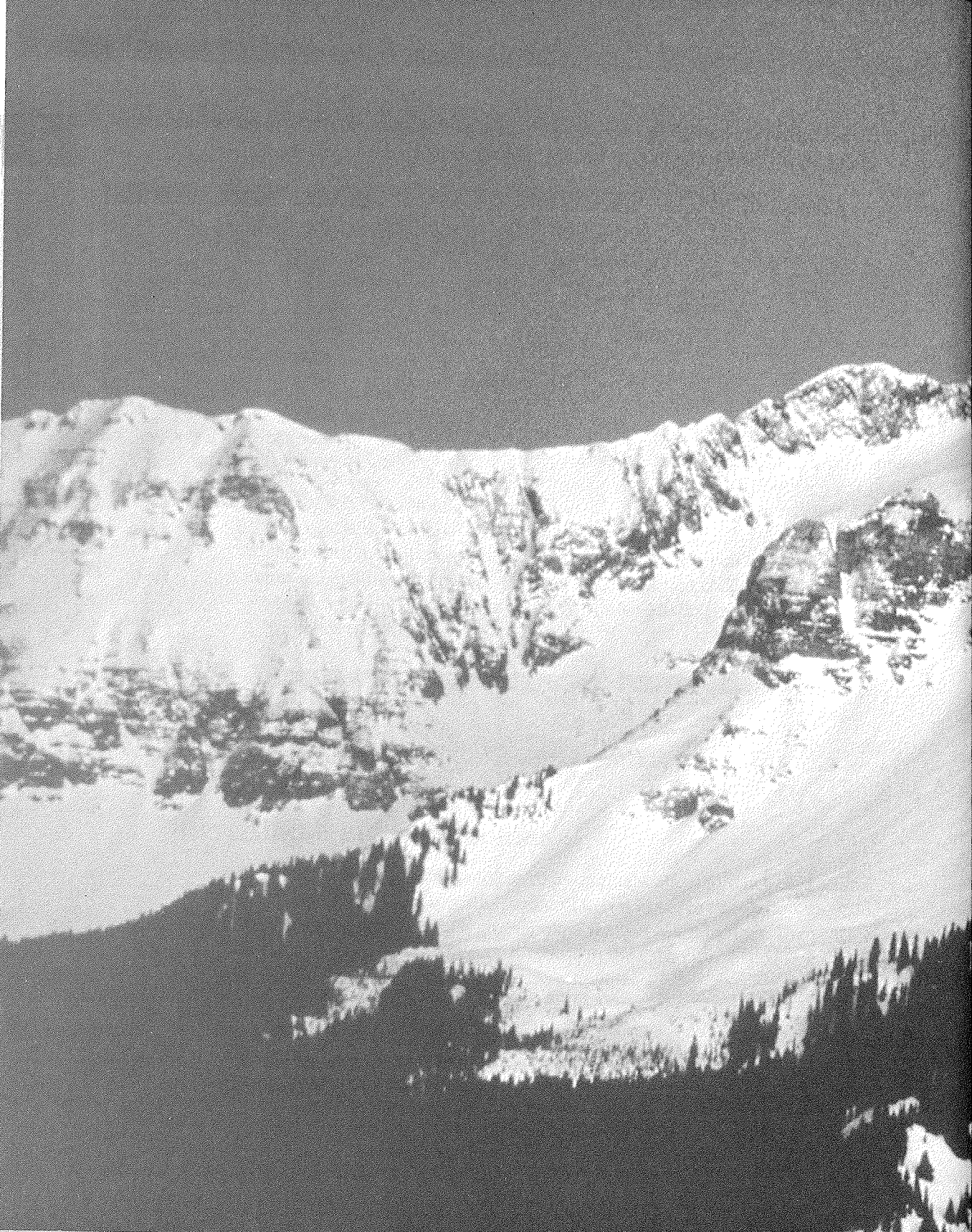
Adding custom toolbars can make your applications much more functional for users. They are truly a convenience feature in any application. However, toolbars usually need

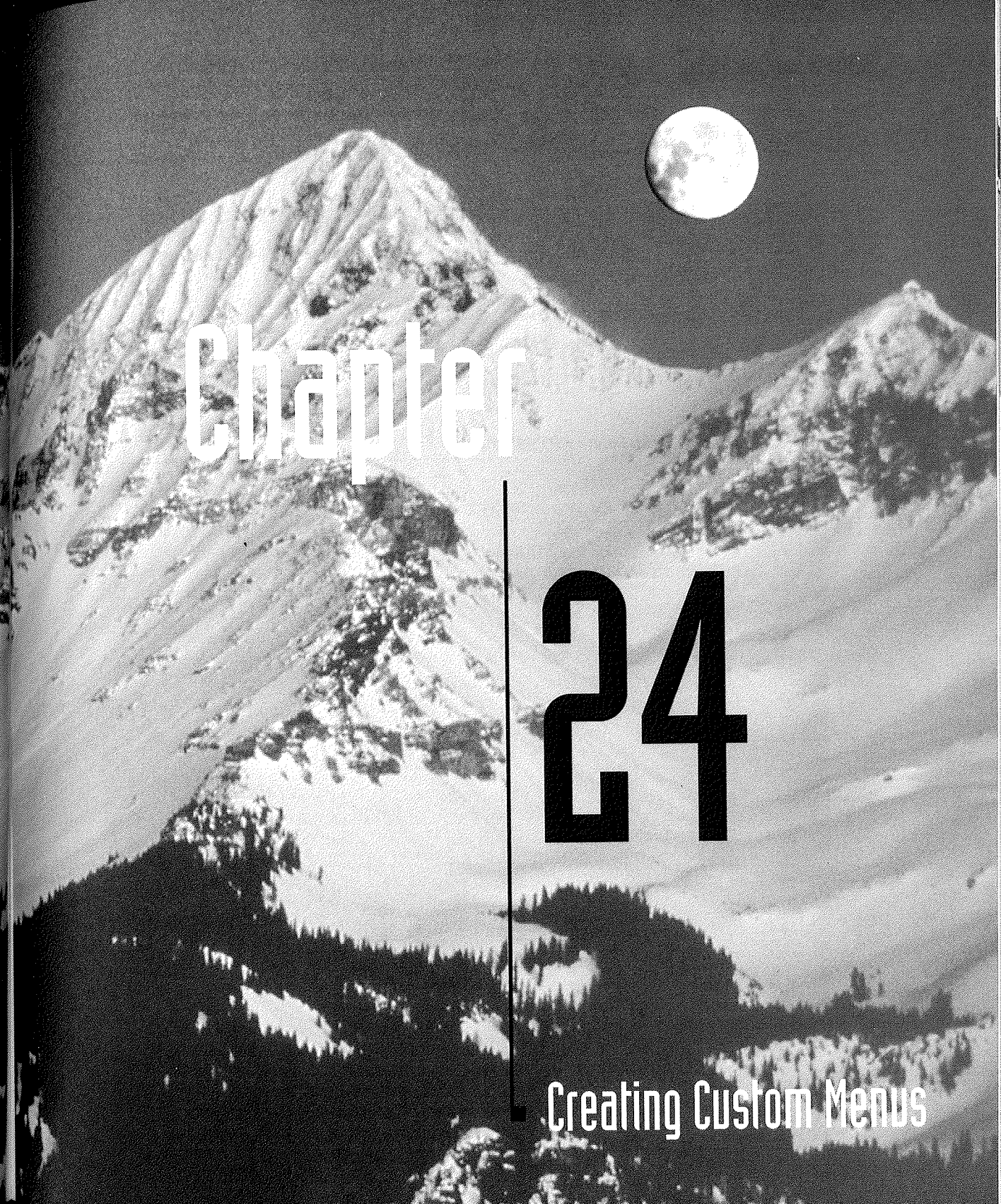
to be backed by menus. The next chapter explains how to create the custom menus that are necessary to back up custom toolbars.

What's New in the Access Zoo?

Toolbars have been seriously enhanced in this release of Access. Several new features make working with toolbars easier. These features include

- A revamped Customize dialog box that makes toolbars easier to customize.
- The ability to add menu commands to toolbars. You can also add buttons to menus, as you'll see in Chapter 24. This new feature lets you create command bars of almost limitless functionality.
- A new shortcut menu for toolbar buttons that lets you specify whether they should be shown as an image, text, or a combination of both.




Chapter
24
Creating Custom Menus

FEATURING

Creating Custom Menus

When developing an application, you'll probably want to give it some custom menus. As with custom toolbars, you can use custom menus to determine exactly what the user of your application can and can't do.

Displaying Custom Menus

You can display custom menus either with a particular form or globally within your application:

- You can attach a custom menu to a form so that the menu bar is displayed only while that form is on the screen.
- A global menu is one that appears throughout your application, though it will be replaced by any custom menus that you attach to forms.

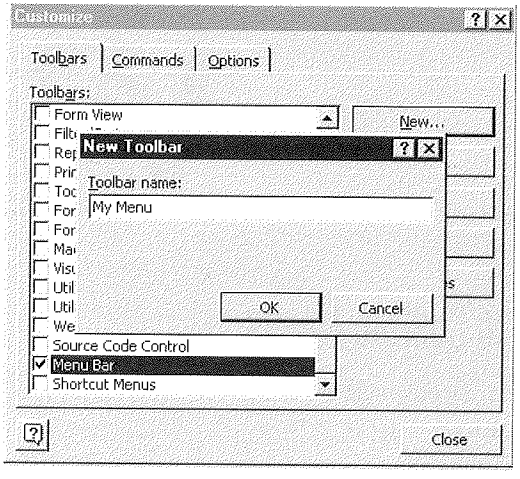
You can use the Customize dialog box, the same one we used in Chapter 23 to work with toolbars, to create either type of menu. We explain how to attach each type of menu to your application a little later in this chapter. For now, just keep in mind that you can use the Customize dialog box to create any number of custom menus for an application.

Creating Custom Menus

Access 97 has a new tool for creating and customizing menus. Instead of using the Menu Builder that was included with Access 95, you use the Customize dialog box you learned about in Chapter 23. As you work through the examples in this chapter, you'll see that the steps are almost the same as those for creating a custom toolbar.

Follow these steps to create a new menu bar:

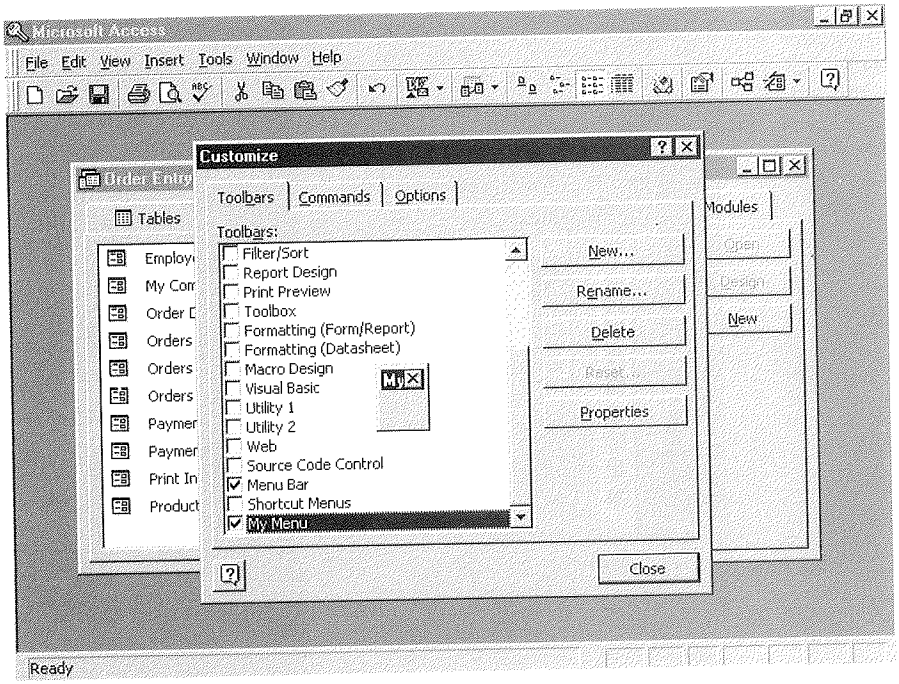
1. Open the database to which you want to add the custom menu bar.
2. Choose View ➤ Toolbars ➤ Customize.
3. Click on the Toolbars tab (if it's not already active) and choose New.
4. Enter a name in the New Toolbar dialog box shown below and click OK or press Enter. (Don't worry that it asks for a toolbar name instead of a menu bar name. With Access 97, toolbars and menu bars can be combined and, in a sense, are interchangeable.) You'll see a new toolbar, usually floating within the Customize dialog box, like the one shown in Figure 24.1.



5. Click on Properties, select Menu Bar from the Type list, and click on Close. If the new menu bar (which still looks like a toolbar) is no longer in sight, drag the Customize dialog box out of the way. Then drag the new menu bar and the Customize dialog box to new locations where you can see them at the same time.

At this point you can add a built-in menu or a custom menu to the new menu bar. Leave the Customize dialog box open to continue your work.

FIGURE 24.1
A new menu bar called My Menu is shown without any buttons or commands added to it yet.



Adding a Built-in Menu to a Menu Bar

To add a built-in menu like File or Edit, click the Commands tab in the Customize dialog box and select Built-in Menus under Categories. Then drag your choice from the Command list to the new menu bar. Figure 24.2 shows the My Menu menu bar after File and Edit menus have been added to it.

Adding a Custom Menu to a Menu Bar

To add a custom menu to a menu bar, click the Commands tab in the Customize dialog box and select New Menu under Categories. Then drag New Menu from the Command list to the menu bar you are customizing. To change the name of the command called New Menu, right-click on the menu bar it was just added to and enter something in the Name box. The menu bar below called My Menu has a custom menu that has been renamed Report Menu:

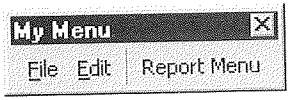
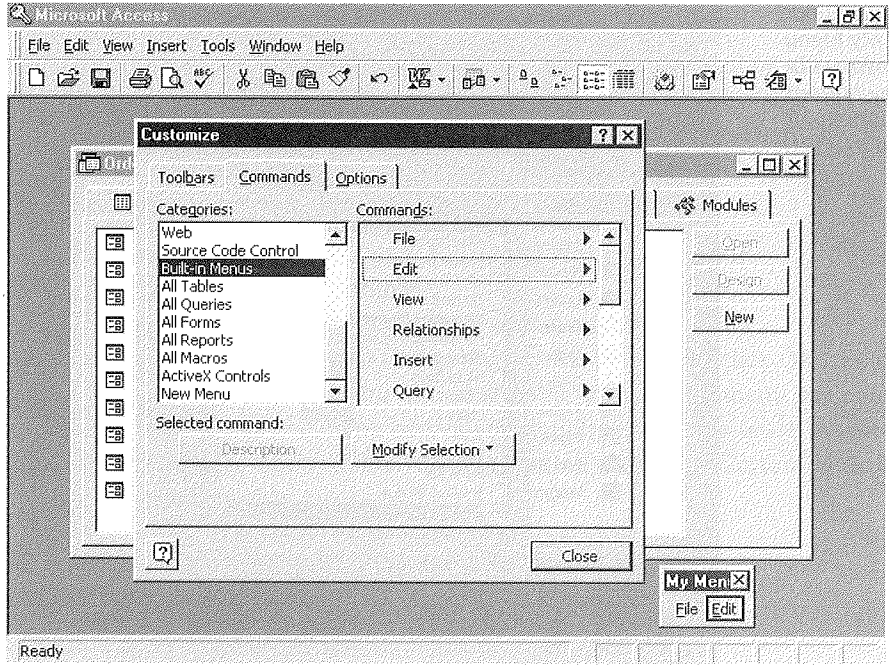


FIGURE 24.2

The menu bar My Menu after the File and Edit menu commands have been added to it.



The next step is to add commands to the custom menu. You can either use the Customize dialog box or drag commands from other menus. To work with the Customize dialog box, click the Commands tab and select the Category for the menu command. For example, click All Reports if you want to add the name of a report in the database to the new menu command. (Remember—we're adding commands to a menu command here, not to a menu bar.) Then drag a selection from the Commands list to the menu command you are defining on the menu bar. An empty box appears below the name of your custom menu if no other commands have been added. Check Figure 24.3 to see how the screen will look. If other commands have already been added to the menu, you'll see them (instead of an empty box) with a line at the insertion point. When you see the empty box or the insertion line, release the mouse button to add the selected command to the menu. Repeat this process until you have added all the desired commands to the menu. Figure 24.4 shows the menu bar My Menu with three reports added to the Report Menu command.

FIGURE 24.3
Adding commands to the custom menu.

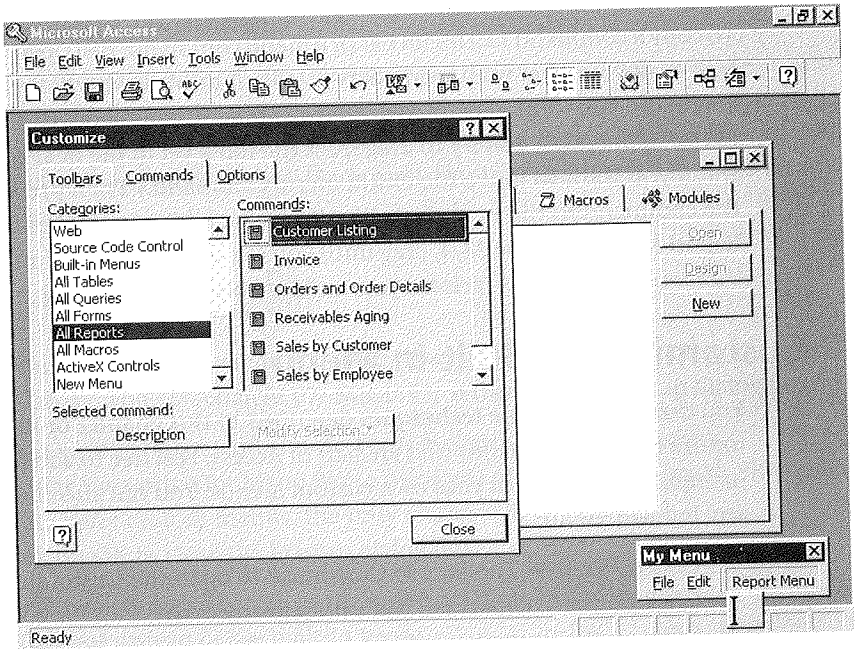
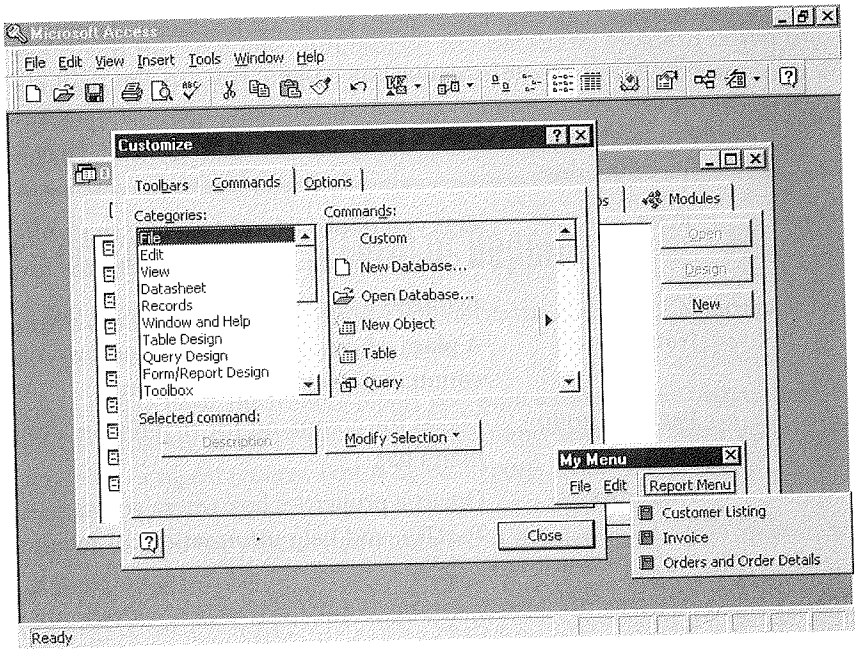


FIGURE 24.4
The My Menu menu bar with three reports added to the custom menu Report Menu.



Copying Commands from a Menu

To copy a command to a menu bar or a menu command, first open the Customize dialog box. Make sure that both menu bars you want to work with are visible. Then select the command you want to copy. (To select a submenu command, click—don't drag— to get to the submenu. If you try to drag from the top level to the next command, Access will think you want to move the entire menu command.) Then press Ctrl while you drag the command to the new menu bar. If you are adding a submenu to a command like Report Menu in Figure 24.4, drag until you see an empty box or an insertion line on the submenu list and then release your mouse button.

Customizing a Built-in Menu

You can use any of the techniques described in this chapter or in Chapter 23 to customize the built-in menus that are part of Access. Feel free to add toolbar buttons, built-in menu commands, or your own custom menus. You can also change the properties of any built-in menu:

1. Choose View ➤ Toolbars ➤ Customize to open the Customize dialog box.
2. Click on the Toolbars tab if it's not already active.
3. Click on the Properties button.
4. Select a toolbar from the Selected Toolbar drop-down list.
5. Change the properties as desired and click on Close to return to the Customize dialog box. Click on Close again if you are finished changing menus and/or toolbars.

When the Customize dialog box is open, you can also right-click on any toolbar button or menu command to get a shortcut menu. Many of the shortcut commands apply only to toolbar buttons and were described in Chapter 23. Here are a few shortcuts for menu commands:

Reset	Lets you return a built-in menu to its original state or restore a custom menu as it was the last time it was saved.
Delete	Removes a command from a menu bar.
Name	A place where you can enter your own names for menu commands, even the built-in ones.
Begin a Group	Makes the command the first one in a new group on the menu bar.
Properties	Opens a properties box where you can change the caption, ToolTip, and help information for menu items.

Saving a Custom Menu Bar

When you finish defining all the commands and actions on your custom menu, just click on Close in the Customize dialog box. If you change your mind about any changes you made to a toolbar or menu bar, you can always use the Reset button to restore a built-in command bar to its original state. If you reset a custom command bar, it will return to its last saved state.

Displaying a Global Menu Bar

If you want your custom menu to replace the built-in menus as soon as the user opens your database, you can change the Menu Bar setting in the Startup dialog box:

1. Open your database and choose Tools ➤ Startup from the menus.
2. Change the setting for Menu Bar to the name of your custom menu. (If you haven't created any custom menus, the only choice will be (default).
3. Click on OK to close the Startup dialog box.

To test the new menu bar, close the entire database and reopen it. Your custom menu bar will appear instead of the built-in menu.

If you have trouble returning to the normal built-in menus, close the database. Then hold down the Shift key, and reopen the database. Holding down the Shift key tells Access to ignore the Startup Properties; hence your custom menu won't appear. Use the Startup dialog box to return the Menu Bar setting to (default) if you need to.

Attaching a Custom Menu to a Form or Report

If you want a custom menu bar to appear whenever the user opens a particular form or previews a particular report, follow these steps:

1. Open, in design view, the form or report you want to attach the custom menu bar to.
2. Open the property sheet (View ➤ Properties) and select the form (choose Edit ➤ Select Form) or report (choose Edit ➤ Select Report). Select the Other tab in the property sheet.
3. Choose the Menu Bar property and then select the name of your custom menu bar from the drop-down list.
4. Choose File ➤ Close ➤ Yes to close and save the form or report.

If you've assigned a menu bar to a form, the menu bar you specified will appear only when the form is open in form view. If you've assigned a menu bar to a report, the menu bar you specified will appear only when the report is open in print preview. If you've defined a global menu bar for your application, the menu bar you attached to the form or report will replace the global menu bar whenever the form or report is open. When the user closes the form or report, the application's global menu will reappear.

Editing a Custom Menu Bar

If you need to change a custom menu bar that you've created:

1. Choose View ➤ Toolbars ➤ Customize.
2. Click the Toolbars tab if it's not already active. Then make sure the name of your custom menu bar is checked (so the menu bar will be displayed).
3. Make your changes using the same techniques you used to create the menu bar.
4. Click Close when you're done.

Creating Shortcut Menus

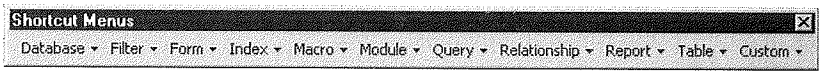
A *shortcut menu* is a menu that appears when you right-click on an object. The object can be a control on a form or a report. It can also be the form or report itself. In fact, any object that contains a Shortcut Menu Bar or Shortcut Menu property on its property sheet can take a shortcut menu.

You can create either a global or a context-specific shortcut menu. The next two sections explain how. A prerequisite, however, is to have built a menu that can serve as the shortcut menu.

Building a Shortcut Menu

To create a custom shortcut menu, follow these steps:

1. Choose Views ➤ Toolbars ➤ Customize from the menus.
2. Click the Toolbars tab and select New.
3. Enter a name in the New Toolbar box and click on OK.
4. Click on Properties on the Toolbars tab, change the Type setting to Popup, and click on Close.
5. Check Shortcut Menus on the Toolbars list to display the Shortcut Menu like this:



6. Click on Custom on the Shortcut Menus and click on the name of the new shortcut menu. An empty box appears, just to the left or right of the new shortcut menu name. Then drag a command from the Customize dialog box or another toolbar to the empty box as described earlier in this chapter. To add additional commands, drag them to the shortcut command list that you are creating.
7. Click Close in the Customize dialog box when you are finished.

Setting a Global Shortcut Menu

To set a global shortcut menu that displays when a form or object does not display its own shortcut menu, set the Shortcut Menu Bar property in the Startup dialog box (shown in Figure 24.5). To set this property, take these steps:

1. Select Tools ➤ Startup.
2. Use the Shortcut Menu Bar drop-down list box to select the shortcut menu you want to be global.
3. Choose the OK button.

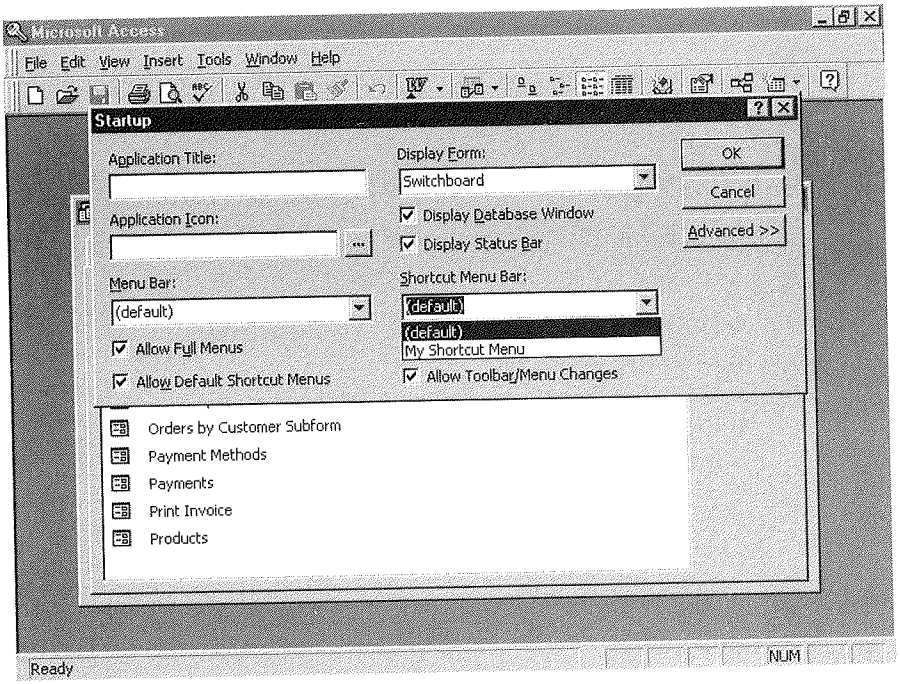


FIGURE 24.5
Creating a global shortcut menu by setting the Shortcut Menu Bar property in the Startup dialog box.

Setting a Contextual Shortcut Menu

To add a shortcut menu to a particular control on a form or to a form itself, follow these steps:

1. Open, in design view, the form you want to attach the custom menu bar to.
2. Select the object you want to display the menu or select the entire form.
3. Open the property sheet (View ➤ Properties). Select the Other tab in the property sheet.
4. Choose the Shortcut Menu Bar property and then select the name of your custom menu bar macro from the drop-down list.
5. Choose File ➤ Close ➤ Yes to close and save the form.

Controlling Whether Shortcut Menus Appear

You determine whether a shortcut menu can appear for items on a form by setting the Shortcut Menu property for the form. To set this property, follow these steps:

1. Open, in design view, the form you want to display or not display shortcut menus.
2. Open the property sheet (View ➤ Properties) and select the form (choose Edit ➤ Select Form). Select the Other tab in the property sheet.
3. Choose the Shortcut Menu property and then Yes or No from the drop-down list, depending on whether you want the menu to display or not.
4. Choose File ➤ Close ➤ Yes to close and save the form.

Converting Macro Menus to Access 97 Menus

If you have menus created from macros or with the Menu Builder in older versions of Access, you can convert them to Access 97 menus:

1. Open the Database window for your database.
2. Click the Macros tab.
3. Select the macro that defines a top-level menu.
4. Choose Tools ➤ Macro ➤ Create Menu from Macro. (Use Create Shortcut Menu from Macro if you want to convert a shortcut menu.)

Access will create a menu with the same name as the macro. You can then customize the menu, if you need to, using the Customize dialog box.

Combining Menus and Toolbars

With Access 97 you can freely combine menu commands and toolbar buttons to create hybrid command bars. With the Customize dialog box open, you can drag menu commands to toolbars and toolbar buttons to menus. For an example, see "Combining Menus and Toolbars" in Chapter 23.

Where to Go from Here

Custom menus and toolbars (see Chapter 23) can make your database function like a stand-alone application. To begin building more complex applications, you need to use the more powerful programming features of Access for Windows 95. The next three chapters show you how to take advantage of Visual Basic for Applications (VBA), Access's new programming language.

What's New in the Access Zoo?

Custom menus can add a great deal of specialized functionality to your database applications. Access 97 has simplified the menu-building process:

- Instead of using the add-in Menu Builder included with older versions of Access, you can now customize or create menus right from the Customize dialog box. This dialog box is also used to manage toolbars. Menus and toolbar buttons can be combined freely in the same command bars.
- Old-style custom menus created from macros or with the Menu Builder can be converted to Access 97 style menus.