Chapter 14

OLE and the Data Control

G O A L S

After working through this chapter, you will be able to:

Use Visual Basic to create containers for objects created in other applications.

Understand the differences and similarities between linked and embedded objects.

Use the Data Manager program, supplied with Visual Basic, to create databases compatible with Microsoft Access.

Use Visual Basic applications to display and manipulate database information.

Use the Structured Query Language to initiate searches in the Animal Species database.

O V E R V I E W

Object linking and embedding, or OLE (oh-lay), is a powerful and exciting option you can use to enhance your Visual Basic programs. With OLE, you can use applications such as Excel as custom controls. You click and drag the Visual Basic OLE control to place it on a Visual Basic form, just as you would for any other control. Then, you choose which file to display in the control, or which application to associate with the control.

Choosing Excel or a specific Excel spreadsheet, for example, gives you access to that application from within the form. You couldn't ask for a better spreadsheet control than Excel. OLE lets you use the full functionality of Excel while you are in a Visual Basic program.

This ability to open one application within the other is an example of an extraordinary level of connectivity between applications. You can see the results of this enhanced communication as you work in many of the popular programs that may be loaded on your computer. Today, for example, you can paste a spreadsheet into a word processing document, next to a graph or table from a database program. As a result, you bring to bear the strengths of different programs on the same topic (the subject of the document).

Another dimension of OLE is OLE Automation. Windows applications that support OLE Automation are said to "expose" objects and their func-

tionality to other programs. Using OLE Automation is similar to borrowing application programs from the library or video store.

While on loan, the applications, or parts of the applications, are available for your use. When you are done using the programs, you return them. The parts of the applications available for your use are called "objects." For instance, if your Visual Basic application "borrows" an Excel spreadsheet object, the full power of the Excel program is available ("exposed") to your program.

OLE Automation is supported by Versions 3 and 4 of Visual Basic. It is not covered in this text, but you can find details and samples in the Visual Basic manuals and on line help.

The second half of the chapter explores Visual Basic's role as a front end to a database. This term—a front end to a database—means that Visual Basic provides the user interface that lets people access the data stored in the database. Imagine, for example, that you want to look up information on a particular musical artist in a database. You need a way of asking, or querying, for that information. Visual Basic forms are often used to provide that "face" to the database, so that users can request specific information from the database or add information to it.

Visual Basic reads and writes databases; in fact, one of the most common tasks for which programmers use Visual Basic is to develop databases or applications that interact with databases. In this chapter, you will use the Data Manager program, which comes with Visual Basic, to create a database. This database will be compatible with Microsoft Access, which is a powerful and sophisticated program for creating databases, editing their data, and refining their design. Your program will use Visual Basic's Data control to give users access to the data in this database.

Object Linking and Embedding

In some ways, OLE is as simple as cutting or copying and then pasting material. You are familiar with the process of marking a portion of text, cutting it, and pasting it to another part of the document. You have used a similar method to copy code from one procedure to another. A logical extension of this process is copying and pasting between different applications, not just between documents from the same application.

Think about the possibilities available when you copy a graphic from one application to another. In a simple paste, you would end up with a copy of the graphic in your document. That copy would no longer be connected in any way to the original; if you edited the original graphic, for example, none of those changes would appear in the pasted image.

Object linking and embedding takes this process of sharing material another step. Instead of merely pasting an image, you can create a "live" linked object. The data contained in a "linked" object remains in the application that created the object. The object itself may be pasted in any number of other applications. If a user changes the object in any of its host applications, the changes affect all the "instances" of the object. (An "instance" is an occurrence of the object in another application.)

In an "embedded" object, the data that makes up the object is contained in the application in which it is pasted. Any change made to the object within a particular application stays in that application. In addition, if you double-click on an embedded (or linked) object, you can actually run the application in which it was created in order to edit the data.

Before working with OLE in Visual Basic, let's experiment with these different concepts of pasting, linking, and embedding. OLE is not unique to Visual Basic; as you will see, other applications also give you access to its power.

To create a live Paintbrush object in a Write program:

1. From the Program Manager in Windows, run Paintbrush (the icon is often found in the Accessories program group).
2. Sketch a picture. Save the picture with the name: **p1.bmp** in the **c:\vbfiles** directory or on your disk. You will use the figure in a future demonstration. See Figure 14-1.

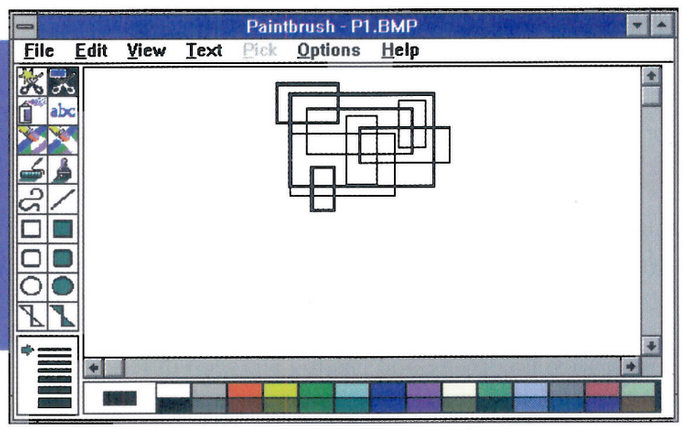


Figure 14-1
A simple picture saved as p1.bmp

3 Use the Scissors tool to select the drawing. See Figure 14-2.

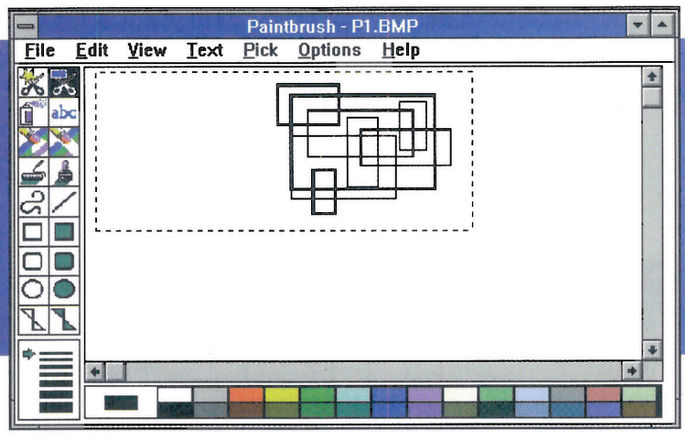


Figure 14-2
Click and drag the box around the picture to select it

4 Select Copy from the Edit menu.

5 Return to the Program Manager.

6 Double-click on the icon for the Write program to start it (this icon is also usually in the Accessories program group).

7 Select Paste from the Edit menu. The drawing appears in the Write document, as an embedded object. See Figure 14-3.
If you wanted to link to the drawing, you would choose Paste Link from Write's Edit menu. Applications that support OLE, such as Paintbrush and Write, use object linking and embedding in ordinary cut and pastes. As you will see, this allows you to edit the object in the original application that created the object. Save the file as **p1.wri**.

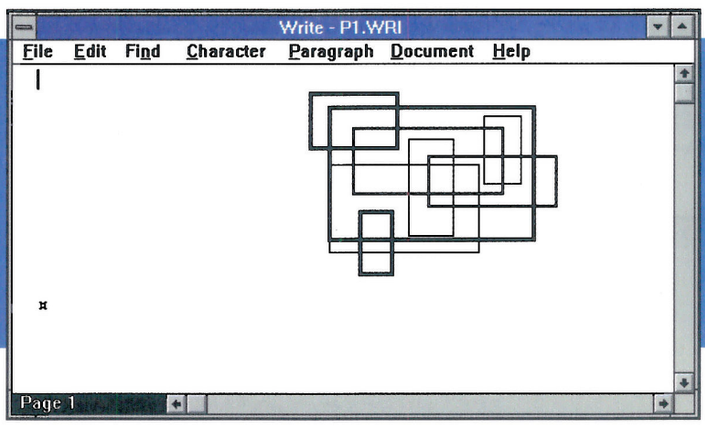


Figure 14-3
Paintbrush picture pasted into a Write document

8 Double-click on the drawing in the Write document. This object is live: it retains its identity as a Paintbrush object. This property of being "live" is a consequence of OLE. The Paintbrush program opens so that you can edit the picture.

9 Close Paintbrush. Note the special Exit command in the menu: Exit and Return to **p1.wri** (Figure 14-4). You have opened a copy of Paintbrush to edit the picture while you are working in Write. The original picture is still open in another copy of Paintbrush.

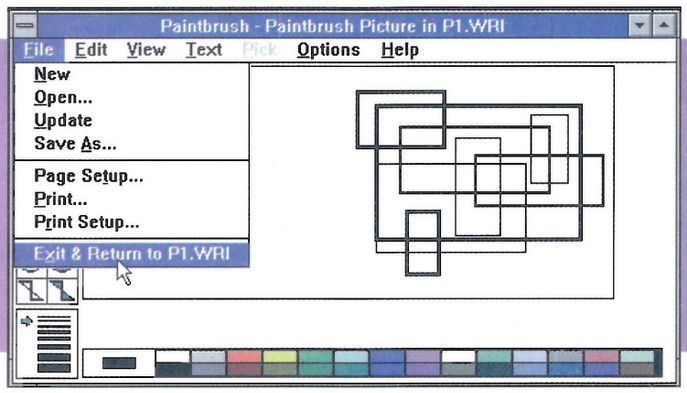


Figure 14-4
Exiting out of the copy of Paintbrush

10 Return to the original picture by pressing Alt+Tab.

11 Select the picture again, select Copy from the Edit menu, then press Alt+Tab to return to Write.

12 Position the cursor at the end of the document and select Paste Special from the Edit menu. The message box in Figure 14-5 opens.

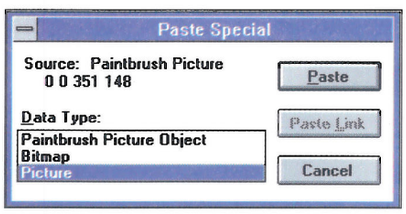


Figure 14-5
Message box for Paste Special

13 Select Picture from the Data Type list, then click on Paste. A second copy of your picture is pasted into the Write document.

14 Double-click on the second drawing. As you can see from the message box that opens (Figure 14-6), this second copy of your graphic is not an OLE object; it is not a Paintbrush object. You cannot edit it.

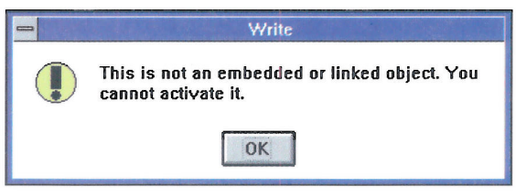


Figure 14-6
Message box explaining that the pasted object is not live

Creating a Linked Object

1. Switch back to the Paintbrush program, select the drawing with the scissors tools, and copy the image with Ctrl+C or with the Copy command from the Edit menu.
2. Return to the Write document, **p1.wri**.
3. Move the cursor to the end of the document and choose Paste Link from the Edit menu. A third copy of the drawing appears. The data making up this object is still in the Paintbrush program. The Write document merely contains a "link" pointing to the picture. You can see this by altering the drawing in Paintbrush.
4. Return to Paintbrush and add an additional shape to the drawing.
5. Return to Write and notice the third drawing changes but the other two do not. Only the linked object is sensitive to changes in the original.

Loading the OLE Control

The OLE control is a custom Visual Basic control. In versions 2 and 3 of Visual Basic, the OLE controls are found in the Windows **\System** directory with a **.vbx** extension. Before using one of these controls, you may have to take steps to add it to your project. In version 4 of Visual Basic, the OLE control is always included in the toolbox.

The OLE icons for versions 2 and 3 are shown in Figure 14-7.

The Project window shows the filenames of all the custom controls included in the toolbox. The Project window shown in Figure 14-8, for example, includes the objects shown in the toolbox in Figure 14-7.

The file named **msole2.vbx** is the version of the OLE control that comes with version 3 of Visual Basic. **oleclien.vbx** is the version that comes with version 2 of Visual Basic. There are significant differences in the ways the two controls work. The text that follows, for the most part, applies to the version 3 control, **msole2.vbx**.

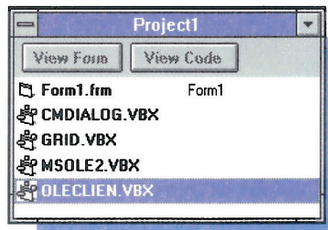


Figure 14-8 Project window showing custom controls available from the Toolbox

The OLE control is, by default, included in new projects. When you create a new project in VB3, the controls that appear in the Toolbox are the usual controls (such as label and editbox) together with those controls that have been added to a special project named **autoload.mak**.

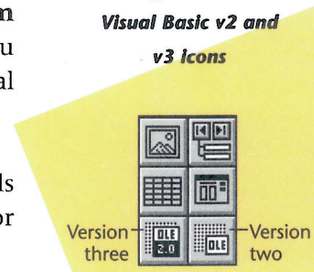


Figure 14-7 Visual Basic v2 and v3 icons

This project contains all the defaults that are used when you create a new project; these controls are loaded into the toolbox, together with the options you can set using the Project command from the Options menu.

When you install Visual Basic, an **autoload.mak** project is created in the Visual Basic directory. It must be located there in order for Visual Basic to apply its settings to your new projects. The **autoload.mak** project that is automatically created includes **msole2.vbx** (as well as the grid control).

When you choose Add File from the File menu, the VBX you select is added to the current project. If you load the **autoload.mak** project, choose Add File to add VBX controls to the project, and save it, then all the controls added to this project will automatically be included in the toolbox for each new project.

To use the OLE control in programs, you must make the control available in the toolbox by adding the **.vbx** file to projects. Note that this may already have been done by someone else using Visual Basic on your computer. In addition, you may need to load a file called **share.exe**, which lets data be transferred from one program to another. If you are running Windows for Workgroups or Windows 95, then you do not need to load **share.exe**. If you are running Windows 3.1, however, then you should ensure that you load that program whenever you start your computer. The **share.exe** program is located in the DOS directory of your computer. Make sure that your **autoexec.bat** file contains the following line:

```
C:\DOS\SHARE.EXE /L:500
```

For more information about **share.exe**, type the following command at the MS-DOS Prompt:

```
HELP SHARE
```

To add the OLE object to the toolbox:

1. Open Visual Basic. If Visual Basic is already open, select New Project from the File menu.
2. Add the OLE icon to the toolbox by selecting Add File from the File menu. Look for **msole2.vbx** in the Windows **\System** directory. The control in the VBX is now added to the current project.

Placing an OLE Control on a Form

An OLE control is a window, or a container, in which you can display a portion of an application created in a different program. The OLE control can show a portion of a spreadsheet, a graph, a document, or a piece of clip art. The difference between pasting and object linking and embedding is that the data in a live object can be modified using the original application.

In the previous Paintbrush and Write example, you implemented OLE by embedding a Paintbrush object in a Write document. The data making up the object is maintained within the Write document: it is saved with the document as part of the Write file. When you pasted using the normal Paste command, Write (which is OLE-aware, as is Paintbrush) created an embedded object. OLE-aware applications do not implement a "naive" paste (naive pastes are not OLE-aware unless that option is specifically chosen). In the previous exercise, the Paste Special command was used to paste a copy of the Paintbrush picture that could not be edited by the Paintbrush program.

The Paste Link command created an object whose data remains in Paintbrush. The Write document maintains a "link" to the original object in Paintbrush, where it may be edited. The object itself is saved by the Paintbrush application.

To place an OLE control:

1. Start Visual Basic. If Visual Basic is running, choose New Project from the File menu.
2. Change the caption of the default form to **OLE Demo**.
3. Double-click on the OLE icon to place the object on the form. Alternatively, single-click on the icon and place the object by clicking and dragging. As soon as you have placed the object on the form, a dialog box opens.

After you have placed an OLE control on a form, you are given a choice between two option buttons: open an application in the OLE control or open a specific file in the control. To create the OLE object from scratch, you would open the application, such as Word, then create the new file. By double-clicking on the control, you can open the application from Visual Basic. Alternatively, you can insert an existing file into the control. The file must be one that can be edited by an OLE-aware application. Your choice of option button determines which of two sets of choices appear in the dialog box.

Creating Another Linked Object

You have placed an OLE object on the form (see previous steps). By default, the Insert Object dialog box opens with the Create New option button selected. See Figure 14-9.

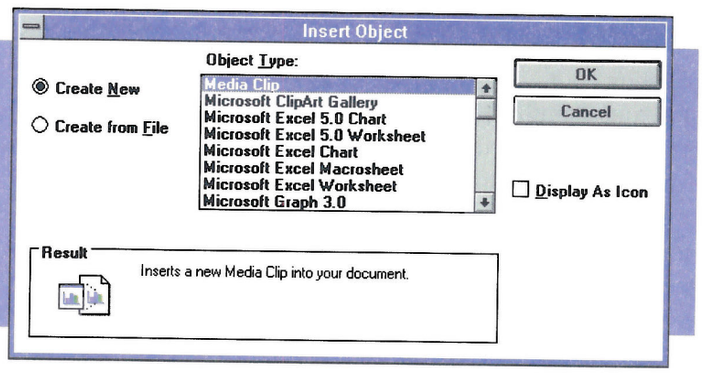


Figure 14-9
Using the Insert Object dialog box to create an OLE object from scratch

The Object Type drop-down list displays programs you can select to create OLE objects.

To place a linked object:

1. Click on the Create from File option button to display the alternative Insert Object dialog box. The choices that now appear are the ones you use to choose an *existing* file containing an object to insert into your Visual Basic form.

2. Click on the Browse button to look through different directories for a file to connect to the OLE object. The extension of the file tells the OLE control what kind of application is being imported. The Link check box allows the data in the object to be dynamically linked to the original source of the data. In this demonstration, you will create an embedded object from the drawing used above. Don't click on the Link check box.

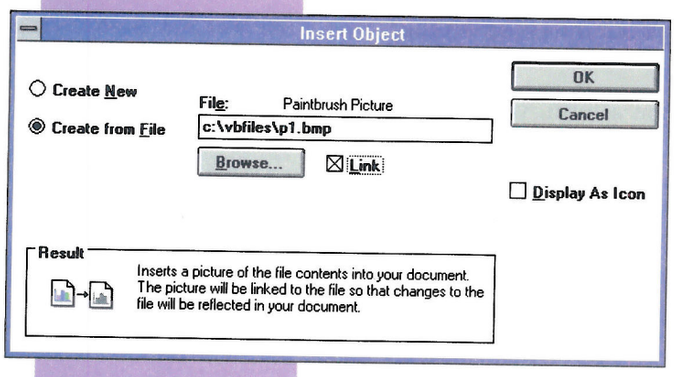


Figure 14-10
Selected Paintbrush file in the Insert Object dialog box

3. Select the name of the Paintbrush drawing you saved (**p1.bmp**). The filename now appears in the Insert Object window (Figure 14-10). Click the Link check box to create a linked object.

4 Click OK to display the Paintbrush file in the OLE object on your Visual Basic form.
Notice that the name of the file is now the value of the SourceDoc property of the OLE control (Figure 14-11).

Figure 14-11 New value for the SourceDoc property of the OLE control

5 Look at the other properties of the OLE control. The default name for the first OLE control you place on a form is OLE1. The control has a "type" of object associated with it: the Class property shows the OLE object is a Paintbrush object. See Figure 14-12.

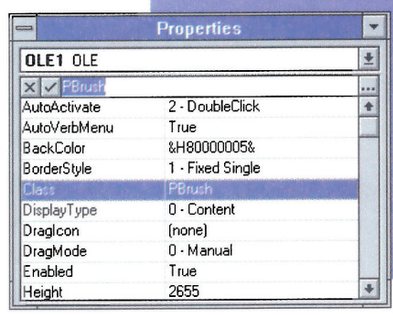


Figure 14-12 Class of the object associated with the control

6 Look at the AutoActivate property. When this property has the value of **True**, a user can start the underlying application by double-clicking on the OLE control. Test this: run the program, then double-click on the drawing to open Paintbrush. Make changes to the drawing.

7 Save the altered drawing and close Paintbrush. Do you see the changes in the drawing on your form?

Creating an Embedded Object

For an embedded object, you place the OLE control on the form in exactly the same way as you did for a linked object (see previous steps). With a second OLE object placed on the form, you are ready for these steps:

1 In the Insert Object dialog box, check that the Create New option is checked. Also make sure the Link check box is not checked.

2 Select Microsoft Excel Worksheet and click OK.

3 When Excel opens, enter the data shown here:

Computer Component	*Prices*
540MB hard drive	$165
Mini tower case	67
486 motherboard	175
Monitor	247
Keyboard	89
Mouse	59

continued

Computer Component	*Prices*
Modem	179
Sound card	100
Video card	139
Disk drive	40
I/O card	40
Memory	300
Misc.	400
Total	**2000**

4 In the row *total*, where the amount *2000* appears, enter a formula to add the values above. For instance:

```
+Sum(D3:D15)
```

5 Select Update from the File menu, then close Excel. The spreadsheet is visible in the OLE control.

6 Change the SizeMode property of the OLE control to Stretch. Note the results. The metafile image of the spreadsheet cells is stretched to fill the OLE control on the form.

7 Change the SizeMode property of the OLE control to AutoSize. The OLE window assumes the shape of the displayed spreadsheet.

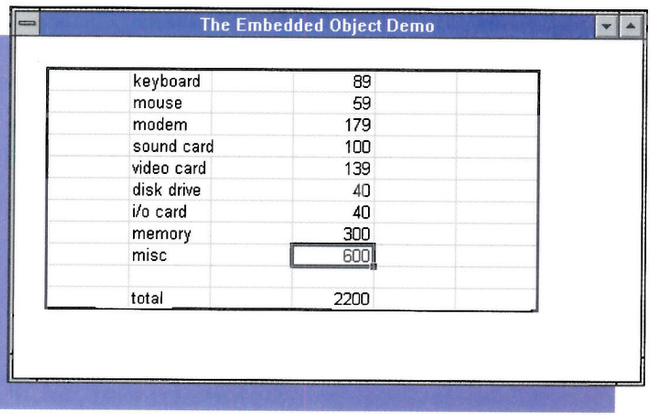


Figure 14-13
Changing an entry in an embedded Excel spreadsheet

8 Run the program. Double-click on the OLE control. Change the entry in Misc from 400 to 600. Note the results. See Figure 14-13.

9 Save the project and form files.

Working with OLE Objects

If you place the mouse pointer on the OLE control, then click on the *right* mouse button, a pop-up menu opens (Figure 14-14). Try this now.

The commands that are available from this pop-up menu vary. For example, try placing an OLE control on the form, but do not associate an application or file with it (click Cancel in the Insert Object dialog box). What commands are available from the pop-up menu, if you right-click on this OLE control?

Figure 14-14
Pop-up menu that opens when you right-click on the OLE control

Experiment with the options in the pop-up menu using your three OLE controls. What happens when you delete an embedded object? You may wonder what the Paste Special command lets you do. This command is available if you have cut or copied something from another application onto the Clipboard. By choosing this command, you can paste a "live" display into the OLE control, retaining a link to the original application and the original source of data being displayed.

Now, let's look at some of the properties of OLE controls. If you have deleted your embedded object, select another file, then embed it. (In the Insert Object dialog box, select the file, leaving the Linked check box unchecked, then click OK.) What happens when you do that? Why does the application associated with the file open? The reason is the setting of the AutoActivate property. When that property has the value of 2-Dblclick, the application opens when you double-click on the control at run-time. As a result, you (or the user of your program) can modify the contents contained in the OLE control using all the features of the original application.

If you double-click on the SourceDoc property of an OLE object, you find yourself in the Browse dialog box. This box lets you choose a file to associate with an object and decide, using the Link check box, whether the object is embedded or linked. Double-click on the SourceDoc property now and find a bitmap file (**.bmp**), a spreadsheet file (**.xls**) or a document file (**.doc**) and insert the file into your OLE object on the form. The data from that file appears in the OLE object.

The SourceItem property, if it is used, will specify the subrange of the object copied from the SourceDoc. In the case of a spreadsheet, the SourceItem property indicates a named range of the spreadsheet, or it can explicitly specify a range. If a SourceItem property is defined, it will be added to the SourceDoc property, separated with an exclamation point:

```
SourceDoc:      C:\vbfiles\Mustang.xls
SourceItem:     R1C1:R8C7
```

After the SourceItem has been defined, the SourceDoc becomes:

```
C:\vbfiles\Mustang.xls!R1C1:R8C7
```

Creating Objects at Run-Time

You don't have to create or associate an object with an OLE control during design time. You can also decide to have that happen under program control while the program is running. To do so, you use a property that is not available at design time and, therefore, is not displayed in the Properties window of the OLE control. The Action property is used after the Class, SourceDoc, and SourceItem (if used) properties are set. The integer value assigned to the Action property determines what action will take place. When the Action property is assigned the value 0, an embedded object will be created, using the information provided in the Class, SourceDoc, and SourceItem (if used) properties.

When the Action property is assigned the value 1, a linked object is created within the OLE control.

A number of actions are available through this property. For a complete list, check the Visual Basic Help file under the topic Action (OLE). You will find commands to paste from the Clipboard, save to a file, collect from a file, and delete, among a number of others.

Constant symbols for the permissible Action values can be found in the file **constant.txt**, located in the Visual Basic directory. For example, OLE_CREATE_LINK is defined as 1. You can use these constants in your code instead of numerals by adding them to a code module of your project. To do so, choose the Load Text command from the File menu, then select the file **constant.txt**.

QUESTIONS AND ACTIVITIES

1. Describe the process of cutting (or copying) and pasting within a document. If you were going to use the keyboard of the mouse to perform these commands, what keyboard combinations would you use?
   a) Marking text
   b) Cutting text
   c) Copying text
   d) Pasting text
2. Where is the data held in a linked OLE object?
3. What is the difference between a linked and an embedded object?
4. Describe the following design-time properties of the OLE object:
   a) AutoActivate
   b) Class
   c) OLETypeAllowed
   d) SizeMode
   e) SourceDoc
   f) SourceItem

5. Give a good reason to include the Stretch option in the SizeMode property.
6. How can you associate an application object to an OLE object after the OLE object has been created?
7. What action is performed when 0 is assigned to the Action property of an OLE object? What action is performed when 1 is assigned to that property?

The Mustang Data Container Program

The first step in OLE mastery begins with using an OLE control to display data under program control. A program that uses OLE to display another program's data is called a container application. In this section, you will use the OLE control to create a Visual Basic OLE container application that displays an Excel spreadsheet.

The application is a simple one: a Visual Basic program that displays a spreadsheet containing information with production figures, original prices, and current prices for 1964 to 1966 Mustangs from one of two years. See Figure 14-15.

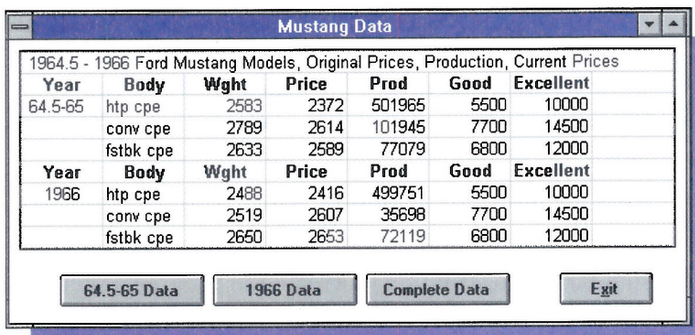


Figure 14-15
Data on Ford Mustangs

Starting Out

In this project, you are going to place an OLE object on the form, then select an Excel spreadsheet file to associate with it. The object will be linked, meaning that the data displayed in the OLE object is maintained in Excel rather than in the Visual Basic program.

The Mustang Data program displays information about the 1964.5 through 1966 Ford Mustang. The information covers the first two and a half years of production. Included in the information is the year, the body style, the weight, the original list price, the number of units produced, the current price for a used vehicle in good condition, and the current price of a vehicle in excellent condition.

Building a Spreadsheet

In Microsoft Excel, you can build a spreadsheet with multiple layers, called worksheets. For this program, you build a single worksheet from data provided to you on the disk accompanying this textbook. The command buttons display the 1964-1965 information, the 1966 information, or the entire spreadsheet. For the first command button, the range displayed is determined using the row and column notation familiar to spreadsheet users. For the second command button, the range displayed is named within the Excel application.

You can use the file **mustang.xls** from the Template disk or create your own. To create the spreadsheet:

1. Start Excel.
2. Place information from the Mustang Data table shown in Figure 14-15 into the worksheet.
3. Once the data is entered, select the rectangular area starting in row 6 column 1 and ending in row 9, column 7. This frames the data for 1966.
4. Click on the Insert menu, then click on Name and Define. Name the selected area **sixtysix**. This named range will be displayed when the 1966 Data command button is clicked.
5. Save the file in the **vbfiles** directory with the name **mustang.xls**.

Using Visual Basic as a container provides direct access from within a Visual Basic program to data created in programs such as Excel. With a Visual Basic container, the user doesn't need to explicitly open Excel in order to view and edit data that the Visual Basic program uses; instead, the data is accessed directly within the Visual Basic application.

Setting up the Form

Follow these steps to set up the form.

1. Create a new directory for the Mustang Data project.

2 Start Visual Basic. If Visual Basic is running, choose New Project from the File menu.

3 Change the caption of the form to **Mustang Data**.

4 If the OLE control is not available in the toolbox, add it to the project, by adding the file **msole2.vbx** from the Windows **\System** directory.

5 Click on the OLE icon and drag to put an OLE window on the form. The Insert Object dialog box opens.

6 Use the Browse command button to find the **mustang.xls** file in the **vbfiles** directory. If the file is available on the disk that comes with the text, move it into the **vbfiles** directory.

7 Click the Link check box to create a linked object. Leave the dialog box by clicking OK. The spreadsheet should appear within the OLE control.
The result of this dialog box is that the SourceDoc property of the OLE control is set to **C:\vbfiles\mustang.xls**. If the SourceItem property is not set, the entire spreadsheet is selected and displayed in the OLE control.

8 Place four command buttons on the form. Change their captions and names as shown in Figure 14-15.

9 Select the OLE control. In the Property window, confirm the OLETypeAllowed property is set to 2 (Either), its default value.

10 Change the SizeMode property to AutoSize.

11 Save the form and project files.

NOTE:

The Visual Basic version 4 OLE control is backward compatible with the version 3 control, allowing the program to use the Action property as described above. In version 4, though, most of these actions have become methods that apply to the OLE control. The Visual Basic version 2 implementation of the OLE control is quite different.

Writing the Code

The command buttons display portions of the entire spreadsheet. The first displays just the data for 1964-1965. The second displays data for 1966. The third displays the entire sheet.

Follow these steps to write the code:

1 Select the default form in the Project window. Click on the View Code button to open the Code window. Select cmdEarly from the Object drop-down list. Enter the code to display the first year and a half of data. The phrase "R2C1:R5C7" causes the part of the spreadsheet from row 2 to row 5 and from column 1 to column 7 to be displayed in the OLE control on the form. Setting the Action property to 1 reestablishes the link between the OLE control and the spreadsheet.

```
Ole1.SourceItem = "R2C1:R5C7"
Ole1.Action = 1
```

2. Enter the Click event handler for cmdLater. The code in this command button takes advantage of the named range in the Mustang data spreadsheet. Set the SourceItem property of the OLE object to the range name, sixtysix. Once the SourceItem property is set, it is appended to the SourceDoc property, set when the form was designed, and a link is established between the OLE container and the spreadsheet.

```
Ole1.SourceItem = "sixtysix"
Ole1.Action = 1
```

3. Enter the Click event handler for cmdAllData. The code in this button deletes the contents of the SourceItem property of the OLE control. The SourceDoc property, still set to the file **mustang.xls**, becomes the default and the entire spreadsheet is again selected. Enter the code for cmdAllData:

```
Ole1.SourceItem = ""
Ole1.Action = 1
```

4. Enter **End** in the Click event handler for cmdExit.
5. Save the form and project files.
6. Run the program. Click each command button and note the results.

Looking at Some Issues

Figuring out how to set the SourceItem property is not always easy. The Visual Basic Programmer's Guide makes it clear that a spreadsheet range is to be specified by row and column.

The row and column method of specifying a ranges marks each row with an "R" followed by a row number. Each column is marked with a "C" followed by a column number. For instance, the upper-left corner of the spreadsheet is designated r1c1, or row 1, column 1. The cell at the fifth row, third column is designated r5c3. This notation is called R1C1 notation, or sometimes RC (Row, Column) notation.

When using this system to designate a range, specify the upper-left corner and the lower-right corner of the range separated by a colon: *r1c1:r3c5*. This is the system used with a Visual Basic application to specify a range of an Excel spreadsheet. The only modification of the system is the ability to specify a particular *sheet* of the spreadsheet. To specify the range as a part of the first sheet, you add the name of a sheet, such as *sheet1*, as a prefix to the range specification: *sheet1!r1c1:r3c5*. The sheet

designation is separated from the rest of the range string with an exclamation point.

Excel spreadsheets use the notation used by Lotus 1-2-3, in which the columns are labeled with letters and the rows with numbers. The top-left corner cell in this notation is *A1*. The range above, *sheet1!r1c1:r3c5*, in the alternate Excel notation is *sheet1!a1:e3*. *A* is the first column and *e* is the fifth column. *1* is the first row, and *3* is the third row. This notation is often called A1 notation.

Although Excel lets you use either notation to designate ranges when working within the program, Visual Basic requires you to use R1C1 notation when referring to a range of cells.

Often cells are specified in positions relative to other cells. To "nail down" a reference to specific cells so the relative references will not change when the formula is copied, put a dollar sign ($) in front of the column letter and/or the row number. The cell designation, *a5* is nailed down like this: *a5*. When you see the dollar sign in a range specification, it always means the cells so marked are not to be changed when formulas are copied.

In the Mustang Data program, the information for 1966 was named *sixtysix*. The Define Name dialog box shown in Figure 14-16 shows the Excel range of cells used in the definition:

Sheet1!$A$6:$G$9

In R1 notation the range would look like:
Sheet1!$R6$C1:$R9$C7

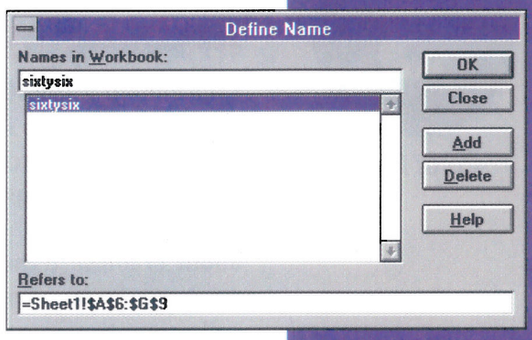


Figure 14-16
The Define Name dialog box from Excel

This range was named when the spreadsheet was created. Naming a range of cells lets you refer to the range of cells symbolically, rather than using the normal, rather cryptic, range specifications.

If you try to set the SourceItem property using A1 notation, Visual Basic generates an error message.

QUESTIONS AND ACTIVITIES

1. What are the multiple layers of an Excel spreadsheet called?
2. What are the two ways to associate an object with an OLE control?
3. A well-designed menu is a way of organizing a programming project. Design a menu that would be appropriate for a program that figures out the amount of a payroll check. Commands should include the number of regular hours worked, the number of overtime hours, and the hourly wage. The display should include both

a facsimile of a check and an itemized listing of the deductions. Don't write the code for this program, just design the menu.

4. Describe the following OLE properties:
   a) Class property
   b) SourceDoc property
   c) SourceItem property
   d) Action property
5. Use the row-and-column method of expressing a spreadsheet range to encode these areas:
   a) a7:e20
   b) a1:z26
   c) c4:g12
6. What do you do to designate a range in sheet 2 of a spreadsheet?
7. Explain the significance of the dollar sign in spreadsheet ranges.
8. If, in the last lines of the code, the Action property were set to 0 instead of 1, what changes would occur in the application?

Section

Exchanging Data with Databases

Information is the substance and product of computing. A collection of related information is called a database. Schools use databases to record course, student, teacher, and financial information. Businesses use databases to record transactions, inventories, personnel files, and customer files. A home computer user might use a database to record information about financial transactions, to keep track of a checkbook, or to trace stock prices. Museums and historical societies use databases to catalog collections or to record genealogies. Users find new and creative ways to use databases all the time.

Database files are created and maintained by database programs. The Microsoft products Access and FoxPro as well as Borland's Paradox and dBASE are examples of database programs.

Database programs have a number of distinct tasks, no matter which program you are using. Creating a database means designing the layout and format of the information contained in the database. A database program lets a user add, edit, and delete information from a database file. A database file is the file that actually holds the information in the database.

A database program also lets the user display the information contained in the database in a variety of ways. Early database programs

could do little more than display tables of data. Today's programs can do much more, attaching pictures and sounds to information and displaying tables and graphs. A user can select the graphs from a huge selection of available types.

A database engine is a group of routines that allow programs to perform the functions listed above with database files of a particular format. These routines allow you to write programs that query and manipulate database files in a high-level way, without your having to know the physical layout of data within the files. The Access database engine is part of Visual Basic as well as being a part of the Access database application. As a result, Visual Basic can manipulate database files with the same format as Access files. This ability to perform database functions under program control makes Visual Basic an extremely powerful tool. Automating database functions under program control makes it easy for users uncomfortable with computers to interact with database files.

Many, and perhaps most, professional Visual Basic programmers use Visual Basic to develop database or database-aware applications. This is one of Visual Basic's most frequently used features.

In these sections, you use Visual Basic to display, add, edit, search for, and delete information from a database.

The Student Schedule Project

The Student Schedule project uses the Data Manager program to create a simple database. Before going through the actual steps needed to create the database, you are introduced to terms and concepts you'll need to understand databases.

STARTING OUT

Imagine a database application that keeps track of a student's daily schedule. The information necessary to create and maintain this schedule might be stored in a number of separate, though related, tables of information.

Take a look at a sample student schedule. The information for Erin Sprague might be stored in a table named Schedule.

Period	*Course*	*Teacher*	*Room*
1	cs101	johnson	203
2	math102	shenk	104
4	lit101	dwyer	204
6	chem101	peterson	211
8	bus104	minor	155

More detailed information about each course might be stored in a table named Courses:

Course #	*Dept.*	*Name*	*Instructor*
cs101	cs	intro to computers	johnson
math102	math	foundations of math	shenk
lit101	english	introduction to lit	dwyer
chem101	science	intro to chemistry	peterson
bus104	business	business math	minor

Information about the teachers can be stored in another table. All in all, a complete software package to manage the affairs of even a small school district is a formidable piece of programming.

Use the Data Manager program to create a database in Access format; start by naming the tables. A simple database may have only a single table; nevertheless, it must be named.

Once the information is arranged in tables, you must design the format of each table. The categories of information contained in a table are called fields. A field has both a name and a datatype associated with it. The datatype of a field will remind of you of the datatypes available to variables in Visual Basic—the list of available datatypes is very similar.

After you name the tables and determine the fields of each table, you must add information to the database. Each discrete piece of related information is called a record. For instance, in the student schedule database described above, the description of each period's class is a record. The fields are the period, the course title, the teacher's name, and the room number.

Each piece of information stored about each class in the schedule goes into its own field. Separating the data into fields lets us search for records that meet specific criteria. You might want to list all the classes that meet in a particular room, or all the classes taught by a certain teacher. Because these pieces of information are stored in separate fields, you can search for records based on these criteria.

INSERTING DATA

After you have designed the database, it's time to insert data. Visual Basic is capable of adding new information to the database. Right now, though, to gain familiarity with the other functions of the Data Manager program, you will use it to insert data.

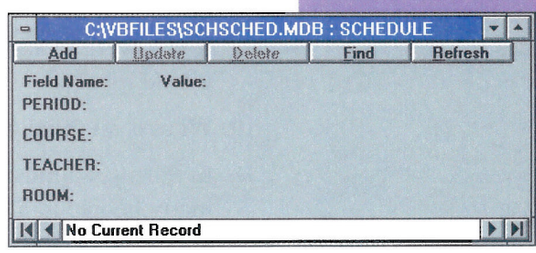
To insert data:

1. Close the Fields and Indexes window by double-clicking on the close box in the upper-left corner of the window. Closing this window returns you to the table window.

2 In the table window, click on the Open button to add information to the database. In the window that opens, you will see a list of the fields defined as a part of the database. Notice that you have the options of adding, finding, or refreshing, records of the database. See Figure 14-17.

Refreshing the database means getting new copies of the records from the database. This allows any recent changes in the data to be reflected in Data Manager windows. At the bottom of the window is a new kind of scroll bar. This is the Data control. The ends of the control look like the buttons on a compact disc player. The caption in the middle of the bar reflects the current status of the database—there are no records in the file.

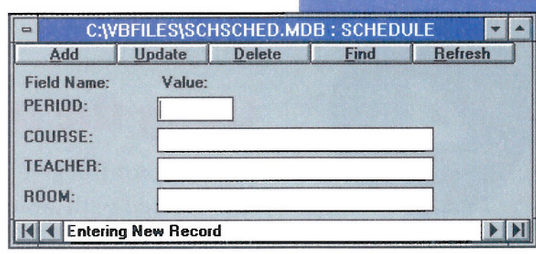
Figure 14-17
Options for working with records in the database



3 Click on the Add button and notice that the caption changes. See Figure 14-18.

Now each of the command buttons at the top of the form have been enabled. From this window, you can add new data, update data stored in the file, delete records, find records with certain values in particular fields, and refresh the listing of the data held in memory. Textboxes to enter data for the file are included, along with labels indicating the names of the fields. The top caption of the window shows the path and filename along with the table name.

Figure 14-18
Effect of clicking on the Add button



4 Make up and enter data for a couple of entries. Close the Data Manager program.

QUESTIONS AND ACTIVITIES

1. Define the term "database." Be sure to describe some of the jobs a database program performs.
2. What kind of database applications would store pictures? Name and write a sentence about at least two.
3. What is a database engine?
4. What job does the Data Manager program perform?
5. Why is Data Manager not a replacement for Access?
6. Explain the relationship between database tables and fields.

7. Write the field names and a few sample entries for a database that would save information about the stuff around your house. Assume the database will be used to help make claims against the insurance company if your possessions are stolen.
8. Every field of a database has at least two properties. What are they?
9. If you were to add a field to the Student Schedule database, what would it be?
10. What is an index to a database?
11. In Portugal almost everyone has about five names. In addition, many people in a classroom may have the same first name. To differentiate between one student and another, often two names are used. Usually the first name is used along with one of the middle names. Design the fields of a database to handle names of Portuguese students.
12. What does refreshing the database mean?

Section

Using the Data Control

The Data control gives versions 3 and 4 of Visual Basic access to database files created with Access or with the Data Manager program. In addition to Access databases, Visual Basic can process data stored in a number of other database formats in widespread use. However, Visual Basic lets you create and manipulate Access databases with the greatest ease. In this section, you go step by step through the procedures necessary to use the data control to display and process the information in an Access-format database of information about animal species.

The Data control connects a database to other controls on a form in which you display record fields. As you will see, the Data control lets you create a powerful and easy-to-use database front end with very little code.

You could easily use these same procedures in many different classes. The same programming techniques apply whether you are creating a database of video tapes, classifying species, or storing vocabulary words for a foreign language course.

NOTE:

If you are using Visual Basic version 2, you will not be able to create this database or work with the Data control.

Creating the Database

The information stored in the Animal Species database is a summary of some important characteristics of particular animal species. Figure 14-19

shows the database definition as it is displayed in the Fields and Indexes window of the Data Manager program. A number of fields contained in the database do not appear in the figure.

The Fields and Indexes window shows the fields that make up the Animals table of the database:

- The species name of the animal, called here the ScientificName
- The order to which the animal belongs
- The animal's common name
- The animal's habitat
- The animal's family name
- The animal's height and weight
- The animal's lifespan
- Whether the animal is warm or cold blooded
- Whether the animal lays eggs or bears live young
- Whether the animal is domesticated
- Whether the animal is on the endangered species list and, if so, what date the animal was put on the list
- Whether the animal is nocturnal
- A description of the animal

For instance, Figure 14-19 shows a portion of a typical entry. It is a screen shot of the Data Manager program's listing of an entry in the Animal Species database.

The choices you make about the organization of the database determine how the database will be used. To build a database about animals,

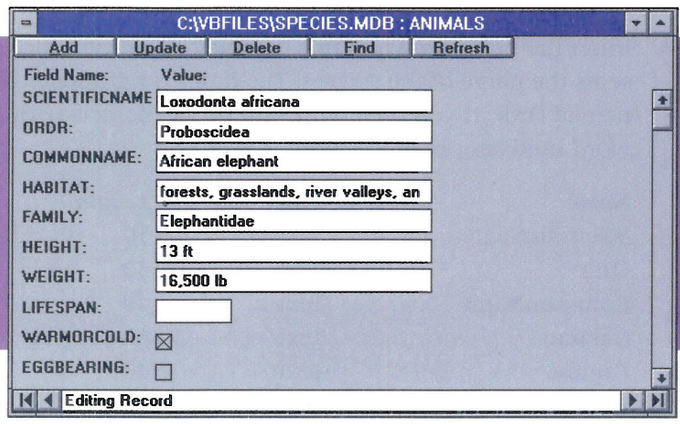


Figure 14-19
Data on animal species

you need to know what facts about the animals are important. If your database is to have an ecological slant and will contain information about whether an animal is on the endangered species list, certainly information related to that topic must be included.

As you investigate what kind of information is available concerning your subject, you must make decisions about what to include and what to exclude. When you decide to include a particular topic, you then must decide what form that information will take. Should the lifespan be represented by an integer, allowing a single number representing the number of years the animal lives, or should the lifespan be represented with a string, so that ranges of lifespans, such as 15 to 20 years, can appear? The importance you assign to a topic will partially determine how long that particular field is. How detailed a description is necessary? If your database deals with endangered species, a lengthy description of the habitat, or a separate field for changes in the habitat, may be necessary.

After you have organized the database, it can be searched. For example, you could create a list of all animals whose weight exceeds a certain value or that are cold-blooded. You could produce lists using any single field or any combination of fields.

Follow these steps to design the Animal Species database. If you are using the **species.mdb** file included on the Template disk, there is no need to enter the information. If you are not using the disk, follow these steps. Enough information is supplied for two entries in the database. You should use an encyclopedia to look up information for several more entries.

1. Start the Data Manager program.
2. Name the file **species.mdb**. Place the file in the **c:\vbfiles** directory.
3. Name the table **Animals**.
4. Enter the fields shown here. The field called ScientificName represents the name of the species. The order is represented by the field named Ordr. If you spell out this term, it conflicts with a reserved word used later in the section.

Name	*Type*	*Length*
ScientificName	Text	50
Ordr	Text	30
CommonName	Text	40
Habitat	Text	50
Family	Text	40
Height	Text	10

Name	*Type*	*Length*
Weight	Text	10
Lifespan	Integer	
WarmOrCold	Boolean	
EggBearing	Boolean	
Domestic	Boolean	
Endangered	Boolean	
DateEndangered	Date/Time	
Nocturnal	Boolean	
Description	Text	80

The Boolean data type sets up the field as a check box. Either the box is checked or it is not. No indexes are defined.

5 Use the Data Manager to enter some of the data shown in Figure 14-20. The program automatically saves the information without you having to choose Save from the File menu.

ScientificName: Giraffa camelopardalis
Order: Artiodactyla
CommonName: giraffe
Habitat: grasslands of the Saraha
Family: Giraffidae
Height: 18 ft
Weight: 3,000 lb
Lifespan: 18
☒ WarmOrCold
☐ EggBearing
☐ Domestic
☐ Endangered
DateEndangered:
☐ Nocturnal
Description:
long neck, three horns, dark patches on a tawny coat

ScientificName: Loxodonta africana
Order: Proboscidea
CommonName: African elephant
Habitat: forests, grasslands, river valleys, and deserts
Family: Elephantidae
Height: 13 ft
Weight: 16,500 lb
Lifespan:
☒ WarmOrCold
☐ EggBearing
☐ Domestic
☐ Endangered
DateEndangered:
☐ Nocturnal
Description:
largest living land animals

Figure 14-20
Two sample records for the Species database

Working with the Data Control

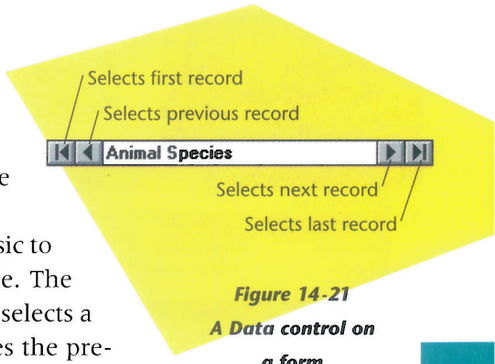


Figure 14-21
A Data control on a form

Look at the Data control. On a form, the Data control may appear as shown in Figure 14-21. The control is resizable to allow more or less room for the caption, but the icons on the end remain the same.

You use the Data control to establish a link from Visual Basic to a set of records of the database—for instance, an entire table. The Data control opens an imaginary window into the table, then selects a single record from it. Clicking on the left arrow button fetches the previous record, if one exists. Clicking on the left-most button fetches the

first entry of the table. In a similar fashion, clicking on the right arrow button fetches the next record, if one exists. Clicking on the right-most button fetches the last record of the table.

The Data control itself does not display the contents of the database. As you can see in Figure 14-22, the Data control is the source of information about each record, but this information is displayed in other controls from the Visual Basic toolbox. The Data control, then, is the conduit through which the records of the database are accessed, one by one. You use other controls to let the user view and edit the fields of these records. This two-tiered architecture gives you great flexibility when designing forms.

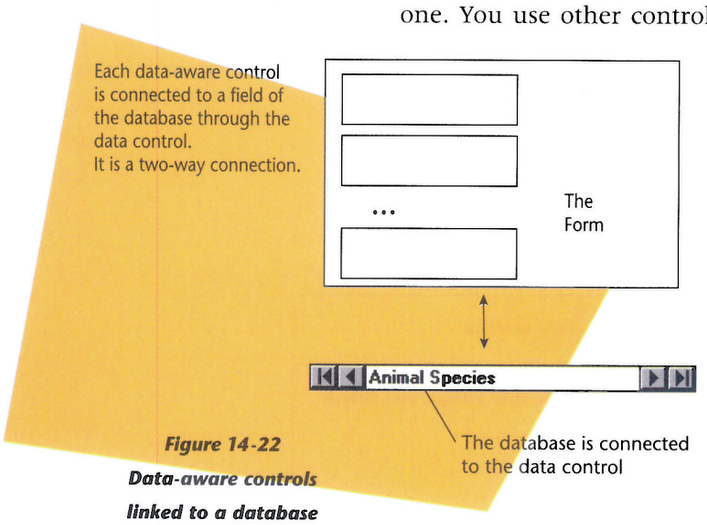


Figure 14-22
Data-aware controls linked to a database

For other controls to display information from a database, they must be (a) data-aware, which means they are able to display information from a database, and (b) linked to a Data control. The textbox, the check box, the picture box, the image control, and the label are some of the data-aware controls from the toolbox. A textbox, for instance, can be linked to display a particular field of the database linked to the Data control.

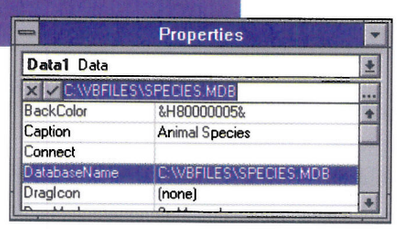
Setting up the Form

Now you need to set up a form to display information from the database. This form acts as a container for the database information.

To set up the form:

1. Run Visual Basic. If Visual Basic is running, choose New Project from the File menu.
2. Change the caption of the default form to **Animal Species**.
3. Put a Data control on the form. Double-click on the icon and resize the control to stretch the area for the control's caption.
4. Select the Data control, Data1. In the Properties win dow, link the Data control to a particular database by setting the DatabaseName property to **c:\vbfiles\species.mdb**. See Figure 14-23.
5. Set the Caption property of the control to **Animal Species**.

Figure 14-23
Linking the Data control to a database



6 Set the RecordSource property of the data control to the table name of the database. See Figure 14-24. This property, when selected for editing, displays a menu of the table names of the database. The database has only one table defined. As soon as the DatabaseName property was set, the data control establishes a link to the database.

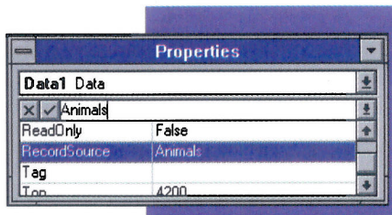


Figure 14-24
Setting the RecordSource property

7 Determine what information of the database you wish to display or edit. In most cases, you want to put at least one textbox on the form to display or edit text from the database. The textbox is our primary vehicle for interaction with the database. The Animal Species project displays all the fields of the database.

8 Use the information in Figure 14-25 to put textboxes and check boxes (with the proper labels) on the form.

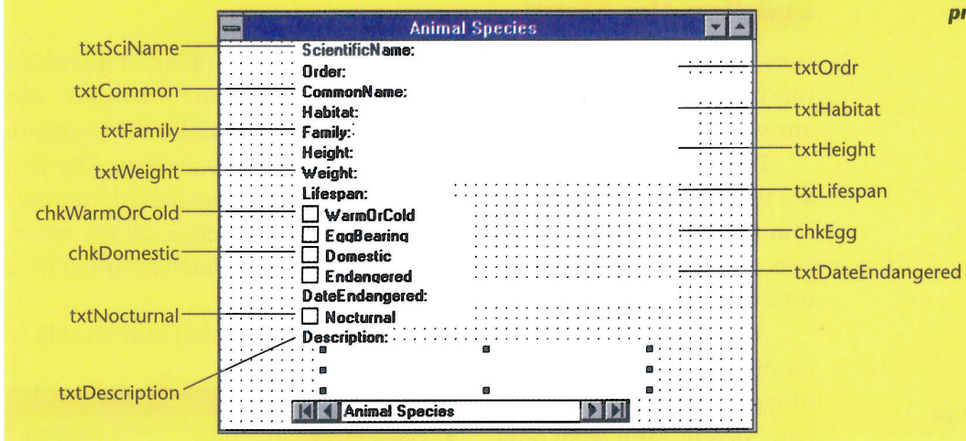


Figure 14-25
Setting the DataSource property

9 Link each textbox you placed on the form with a field from the database. This is a two-step process:

- ◎ Set the DataSource property of the textbox to indicate the source of the data (Data1, the default name of the Data control). Double-clicking on the DataSource property sets up the link.
- ◎ Set the DataField property of the textbox, to indicate which field of the DataSource is to be displayed. This edit field provides a list of all the possible choices for the fields of the database to be linked to this textbox.

10 Repeat steps 8 and 9, putting textboxes on the form, binding them to the Data control *Data1* through the DataSource property, and linking them to particular fields of the database through the DataField property. The txtDescription textbox accommodates the longest text field. This textbox's MultiLine property is set to **True** to allow the data to spread to the second or third line if necessary.

11 Save the form and project files in the **c:\vbfiles** directory.

If you ran this program now, the Data control would bind itself to the proper database. The Data control would latch on to the first record of the database, giving Visual Basic access to the information. The textboxes, through their DataField property, would each grab a piece of data from the record and display that information.

If all you want to do is establish a link between data-aware controls and the information stored in the database, you are done. There is no code to write. If you want to process that information and do something with it, however, there is still a ways to go.

Building the Menu

How will you use the Animal Species database? What kind of questions can be answered with the Animal Species information? When you can answer these questions, you are ready to design a menu for the program.

Two basic uses for any database are display and searching. The two are usually interrelated. Often you will want to display lists of information that meet specific criteria: all the endangered species, all the members of a particular order or family. A search of the database provides a list of all the entries that meet those criteria.

A program that displays records of a database should also be able to modify those records. The program should be able to edit, delete, and add new records to the database.

The kind of searches and displays you choose are determined by your needs. A program that tries to meet every need can be confusing, so you need to organize your menus carefully.

See Figure 14-26 for the structure of the menu.

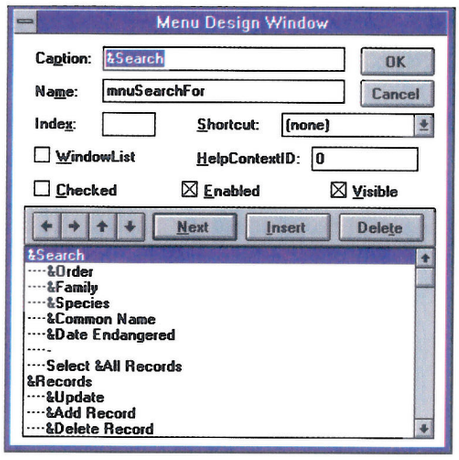


Figure 14-26
Menu structure

To build the menu:

1. Select the form in the Project window.
2. Open the Menu Design window.
3. Construct the menu using Figure 14-26 and the table below.

Caption	*Name*	*Index*
&Search	mnuSearchFor	
&Order	mnuSearch	1
&Family	mnuSearch	2
&Species	mnuSearch	3
&Common Name	mnuSearch	4
&Date Endangered	mnuSearch	5
-	mnuSearch	6
Select &All Records	mnuSearch	7
&Records	mnuRecords	
&Update	mnuUpdate	
&Add Record	mnuAdd	
&Delete Record	mnuDelete	
E&xit!	mnuExit	

4. Save the form and project files.

The code contained in the event procedures for these menu commands allows the user to search the database, choosing the records that meet certain criteria based on the content of the fields. The program selects and displays each record of an animal belonging to a particular order, family, or species, as well as displaying the record of an animal with a particular common name.

SQL, the Structured Query Language, is the reason the form you have created can act as a "face" to a database. This is a marvelously rich language that allows a multitude of different operations. Besides search operations, the SQL language lets us join sets of records together, delete large numbers of records, or find the first or last occurrence of a particular piece of data.

Don't look in the Visual Basic *User's Manual* for a discussion of SQL. Most of the information about SQL can be found in the Visual Basic Help files. In addition, any bookstore with a computer section will probably have books that describe this language. SQL was not designed by Microsoft. It was designed to generate search criteria independent of the details of internal file structures or specific computer architectures. SQL is device- and software-independent. Many programs use it as a query

language, which is a language in which users can ask questions of a database.

Editing Current Entries of the Database

Users can interact with database information through the form you have just created. They can do far more than read the information displayed in the textboxes; they can also change that information. Textboxes, in fact, were designed for editing string information. If a user alters text from the currently displayed record within the textbox, that change can be made permanent. That is, the contents of the database file can be altered to reflect the change. The Data control is capable of sending changes in the contents of the record back to the database file for storage. The new data overwrites the old.

There are two ways to make changes permanent:

1. If a user changes information displayed in a textbox, then uses the Data control to move to another record, the altered record is written back to the database file. The changes are saved. You don't have to write any code for your program to have this behavior: the Data control gives it to you "for free."
   You could try this out now with Animal Species database. Even though the menu items have yet to be coded, you can run the program and edit a field's textbox. Advancing to the next record, or returning to a previous element, makes your change permanent. Try closing the file, reopening it, and checking for your change.

2. You can execute the **Update** method on the Data control by adding code to the event procedure of the appropriate menu command (Update, in the Records menu).
   The syntax for the **Update** method is:

   ```
   Data1.RecordSet.Update
   ```

RecordSet refers to the collection of records currently linked to the Data control. This collection can be an entire table from the database, a subset of a table, or even a set of records composed of fields from several tables. Using SQL, you can define a set of records chosen by specific criteria. These records, once chosen, are separated from the other records of the database and are treated as a unit. Examples of how to use SQL are presented later in the chapter.

The **Update** method forces the contents of the copy buffer to be copied back into the database file. This replaces the record in the database on disk with the edited contents of the Data control's current

record. The copy buffer is the area of memory used to hold the current record—the one being edited.

For the Animal Species application, you can think of the copy buffer as being the textboxes and check boxes in which the fields of the record are displayed. The real copy buffer may be larger. It is not required that every field of the record be displayed in a bound control. (A bound control is a data-aware control, like a textbox, connected to a database through its DataSource property.) Nevertheless, all the fields of the current record, even those not displayed, are present in the copy buffer.

The Records menu contains three commands in its submenu. See Figure 14-27. The first allows the user to force the database to be updated with the information contained in the bound controls, when that information has been edited. For instance, if there were a spelling error in one of the records, the user could use the Data control's arrows to make that record the current record, so that its fields appear in the textboxes. Once the record is displayed in the textboxes, the user could edit the text of its fields. The version of the record in the copy buffer will then differ from the record stored in the database.

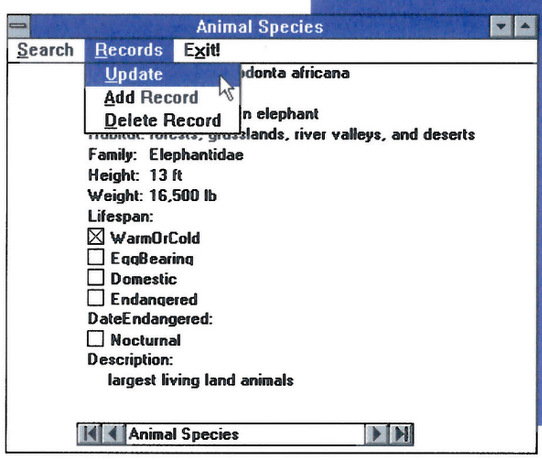


Figure 14-27
Records menu

After a record has been edited, the user can write the changes back to the database by choosing the Update command from the Records menu. This command forces an update of the current record in the database file with the contents of the copy buffer. The user can also update the database simply by moving to a different entry using the arrows on the data control.

Enter this line in the mnuUpdate_Click event:

```
Data1.RecordSet.Update
```

Adding a Record

Users may want to add information to the database, as well as editing or updating it. For this purpose, you included an Add Record command in the Records menu. Writing the code for this command involves adding a single line to its event procedure.

To add a record to the database, the copy buffer must be cleared. When this buffer is cleared, the user can enter information directly into the textboxes connected to the fields of the database. When the fields

are filled, the **Update** method is executed to write the contents of the new record to the database file.

You use the **AddNew** method to clear the copy buffer and the textboxes connected to the database file. The syntax of this method is the same as that of the **Update** method:

```
Data1.RecordSet.AddNew
```

Add this line of code to the mnuAdd_Click routine. Note that you could have used a command button (captioned Add) instead of a menu command for this purpose.

Deleting a Record

Another way in which users will want to interact with the database is to delete information from it. For this purpose, they will select the Delete Record command from the Records menu. Coding this menu command to delete database information is as easy as coding the Update and Add Record commands: you need to add a single line to the routine for the command.

A deleted record doesn't disappear from the textboxes immediately, but it is removed from the database file. Once a user moves to another record, the deleted record is no longer available.

The syntax for the Delete method is the same as that for the **Update** and the **AddNew** methods. Add this line to the mnuDelete_Click routine:

```
Data1.RecordSet.Delete
```

Writing the Code for the Search For Command

You have written only three lines of code so far for this program. Updating, deleting, and adding records are quite simple operations. Searching the database, however, is a more complex operation. It is one of the principal means by which data is transformed into information. We store data in databases to unburden ourselves from remembering vast amounts of detail. The ability to search a database justifies this approach: it lets us reliably and rapidly find the needle in the haystack. By searching, we can sift through tons of data to isolate the small amount of material we seek. Searching lets us discover patterns and structure in large amounts of data that we would not be able to detect without the aid of a computer.

For all but one of the menu items in the Search For submenu, a search string is collected from the user with an **InputBox** function. The overall structure of the routine is as follows.

If Select All Records is clicked **Then**
    choose all the records from the database
    and display them ordered by their CommonName field
Else
Collect search string, *Target*, from user
    If "Order" is chosen
        choose records whose "Ordr" matches the *Target*
    If "Family" is chosen
        choose records whose "Family" matches the *Target*
    If "Species" is chosen
        choose records whose "Species" matches the *Target*
    If "Common Name" is chosen
        choose records whose "CommonName" field matches
the *Target*
    If "Date Endangered" is chosen
        choose records whose "Date Endangered" field matches
the *Target*
Finally, refresh the record set from the database.

Last things first. The **Refresh** method goes to the database file and selects the records that match the search criteria expressed in the SQL query. This is always the last step in an SQL search.

Coding the SQL Search

To initiate an SQL search, load the RecordSource property of the Data control with an SQL query string. Take a look at such a string:

SELECT * FROM Animals ORDER BY CommonName

The first part of the phrase, *SELECT **, tells Visual Basic to select all the fields of all the records that match the search criteria.

The next part of the string designates the table from which the records are to come. In this case, the records come from the Animals table. Recall that this is the only table defined in the database. It is possible to fetch records containing fields from more than one table using the *FROM* clause of the *SELECT* statement.

The last part of the statement displays the records in *ORDER BY* the values in the *CommonName* field. This shows the Animal Species database in alphabetical order by the common name of the animal. Regardless of the order in which records have been entered into the database, this command lets the Data control move through the records in the order of their *CommonName* field.

In some ways this is not a typical SQL search string. There is no *WHERE* clause. The *WHERE* clause puts restrictions on what records are

chosen. The way this SQL string is written, all the records of the database are chosen. This string is used in the code that selects all the records from the database.

Now you will enter the code for the commands in the Search menu:

1. Select the form from the Project window.
2. Enter these lines in the mnuSearch_Click procedure.

```
Dim Target
'—Select All is different from the others.
If Index = 7 Then
   Data1.RecordSource = "SELECT * FROM Animals ORDER BY CommonName"
   Data1.Refresh
Else
```

From this code you can see how to use the SQL string. It is enclosed in quotes and assigned to the RecordSource property of the Data control. The **Refresh** method is applied to the Data control to reselect the appropriate records from the database. The next section of code collects the target string from the user. By this point of the procedure, the user has chosen one of the submenu choices that requires the entry of a search string.

3. Enter these lines:

```
Const Msg = "Enter search string"
Const Msg2 = "Searching For..."
'—Collect search information from user
Target = InputBox(Msg, Msg2)
```

4. The section of code that deals with finding records that have a certain date has an interesting twist. Unlike the other SQL statements in this section, the date string isn't really a string at all. The date is a special kind of value, neither string nor whole number nor decimal. To signal Visual Basic you are dealing with this special kind of data, enclose the date string within number signs: #.
Enter these lines:

```
Select Case Index
   Case 5
      Data1.RecordSource = "SELECT * FROM Animals WHERE DateEndangered=
#" & Target & "#"
```

The SQL string is joined together from standard pieces like "SELECT *" and "FROM Animals" along with the *WHERE* clause and the actual target string. Notice the number signs bracketing the date string collected through the **InputBox** function.

The *WHERE* clause puts a restriction on the records collected from the Animals table of the database. In this clause the DateEndangered field is restricted to those values that match the date in the *Target* string. No other records are chosen from the database file.

The remaining cases all assign to *Data1.RecordSource* an SQL string with a *WHERE* clause of the following form: WHERE *fieldname* = "Target". This clause will restrict the records fetched to those that match this selection criterion.

5 Enter these lines:

```
    Case 1
        Data1.RecordSource = "SELECT * FROM Animals WHERE
Ordr= '" & Target & "' ORDER BY Family"
    Case 2
        Data1.RecordSource = "SELECT * FROM Animals WHERE
Family= '" & Target & "' ORDER BY ScientificName"
    Case 3
        Data1.RecordSource = "SELECT * FROM Animals WHERE
ScientificName= '" & Target & "'"
    Case 4
        Data1.RecordSource = "SELECT * FROM Animals WHERE
CommonName= '" & Target & "'"
    End Select
    Data1.Refresh
End If
```

Look at the "Ordr" and "Family" branches. The *SELECT* statements in those branches have a *WHERE* clause and an *ORDER BY* clause at the end. These statements pick all the records whose Ordr field or Family field matches the target string. In addition, the records are displayed in order. If select by Ordr is chosen, the records are displayed in order by their Family. If select by Family is chosen, the records are displayed in order by their scientific names.

QUESTIONS AND ACTIVITIES

1. If you were redesigning the Animal Species database, what new fields would you include? Which, if any, would you drop? Justify your answers.

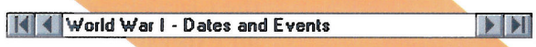
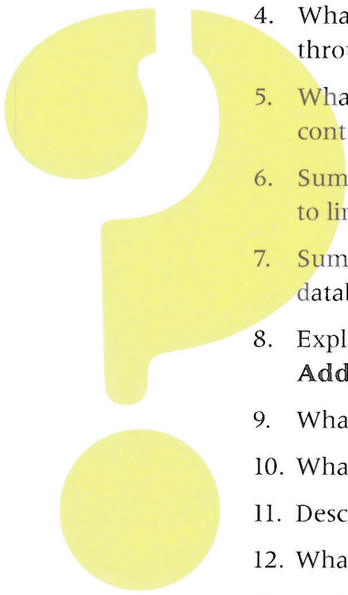


Figure 14-28
Functions of arrows on Data control

2. Explain the functions of the arrows on the Data control. See Figure 14-28.



3. Is the Data control more like a linked or an embedded object? Why?
4. What does it mean when we say the Data control is the "conduit through which the records of the database are accessed"?
5. What properties do data-aware controls have that nondata-aware controls do not have? Which controls are data-aware?
6. Summarize the steps necessary to set up a Data control on a form to link to a database file.
7. Summarize the steps necessary to link data-aware textboxes to the database through the Data control to display fields of the database.
8. Explain the syntax and action of the following methods: **Update**, **AddNew**, and **Delete**.
9. What is the RecordSet?
10. What does SQL stand for?
11. Describe the copy buffer.
12. What does the **Refresh** method do?
13. Explain and give an example of each of the following SQL terms:
    a) "SELECT *"
    b) "FROM ..."
    c) "WHERE ..."
    d) "ORDER BY ..."
14. What special action needs to be taken to indicate that a string in an SQL string represents a date or time?
15. Use the Help system to look up WHERE. Write an SQL string that will choose all the records in the Animal Species database where the Weight is more than 80 lb.

Summary

OLE, or object linking and embedding, and the Data control are two of the ways Visual Basic can manage data from other applications.

OLE permits sophisticated cutting and pasting of data from one place to another. The Class property of an OLE object determines the type of data that may be represented by the object. An example of an entry in the Class property is "ExcelWorksheet", indicating the object represents an Excel spreadsheet.

The SourceDoc property links the OLE control with a file that contains the data for the object. The SourceItem property links the OLE control with a subset of the information contained in the SourceDoc file.

The OLETypeAllowed property determines whether the object displayed on the form is linked or embedded. A linked object maintains a connection from the data displayed in the OLE object to the original

data in the original application. The displayed data is stored in and maintained by the original application that created the data. An embedded object contains the data displayed in the OLE object and maintains a link to the application that created the object.

If the AutoActivate property is set to **True**, double-clicking on the OLE object on the form starts up the application that created the data displayed in the OLE object. This allows the user to modify the information contained in the OLE object.

The SizeMode property of the OLE object determines the relationship between the size of the data represented by the object and the size of the window given to the object in which to display the data. The SourceItem property represents spreadsheet ranges with the Row, Column format. For instance, R1C1:R3C10 indicates the rectangular range in a spreadsheet from row 1, column 1, to row 3, column 10. The Action property of the OLE object can create the linked or embedded object at run-time once all the other properties just discussed are set.

The Microsoft Access database engine, which is the name for the routines that perform much of the work on Access database files, is a part of Visual Basic. As a result, Visual Basic has the ability to manipulate these database files.

The Data Manager program lets the user create database files in the format of an Access database. A database is made up of tables of related information. Each table in turn is made of fields. Each field has a data type and a name. The information of the database is stored in the fields. A record is a complete entry of information in a database.

The Data control, which is included in versions 3 and 4 of Visual Basic, gives a Visual Basic program access to the information contained in database files. Data-aware controls, such as the textbox or picture box, link to the Data control and display information from the file to which the Data control is linked. The DatabaseName property of the Data control links the control with a file containing a database.

The RecordSource property of the Data control links the control to the records of a particular table within the database. It is used to select particular records specified by an SQL search string. The DataSource property of a data-aware control links the control with a Data control. This is the source of the data displayed by the data-aware control. The DataField property of the data-aware control links the control with a particular field of the database. This field is then displayed and can be modified through the data-aware control.

The **Update**, **AddNew**, and **Delete** methods let you control the contents of the database through the copy buffer. The contents of the copy buffer are displayed through the data-aware controls such as the

text or picture boxes. When the **Update** method is executed, any changes made to the copy of the currently displayed record in the copy buffer are written back to the database file. The **AddNew** method clears the copy buffer and the textboxes of the form and allows the user to enter a new record into the database file. Using the **Delete** method, a user can remove the currently displayed record from the database file.

Visual Basic supports Structured Query Language, which is a language designed for searching in a database file. Some commonly used parts of the SQL language are the SELECT, FROM, WHERE, and ORDER BY clauses.

1. The Excel CD Program

Use Excel to create a simple list of CDs. Use categories such as the following:

- Title
- Artist
- Performance date
- Number of songs
- Total playing time
- Guest Artists

Once the spreadsheet has been created, write a Visual Basic program that uses an OLE control to display the spreadsheet. First, link the spreadsheet to the Visual Basic program. Then try embedding the spreadsheet. What differences do you find in the way the application behaves?

2. The Personal Inventory Program

Create a database of personal items. Use categories like those below, but add, adapt, or omit categories to meet your needs:

- Description
- Quantity
- Date last used
- Location
- Value

Create an Access format database using Access or the Data Manager program. Each of the categories will become field names in the *Inventory* table. Each record of the database will represent a different item. Once the database has been designed, write a Visual Basic program to access the fields of the database. Your program should let you display each record of the database. It should also let you edit, delete, and add new records.

3. The Word Processor Project

Create three word processing documents, using Microsoft Word, Write, or any other application that supports object linking and embedding. In the first document, include a short summary of events from your home life that have been significant; in the second, a list of interesting classroom events; and in the third, a summary of events you have enjoyed in your work or extracurricular activities.

After these documents are created, create a Visual Basic program using the Multiple Document Interface (or just four forms if you skipped that chapter). The first form should contain a menu to load each of the other three. The second form should contain an OLE control linked to your first document, displaying a paragraph of that document. The third form should be linked to the second document, and the fourth form should be linked to the third document. Size the windows and open the word processor and the Visual Basic program so you can see both at once. Make changes to the word processing document and see if the changes are reflected in the Visual Basic OLE control on the appropriate form.

4. The Database Project

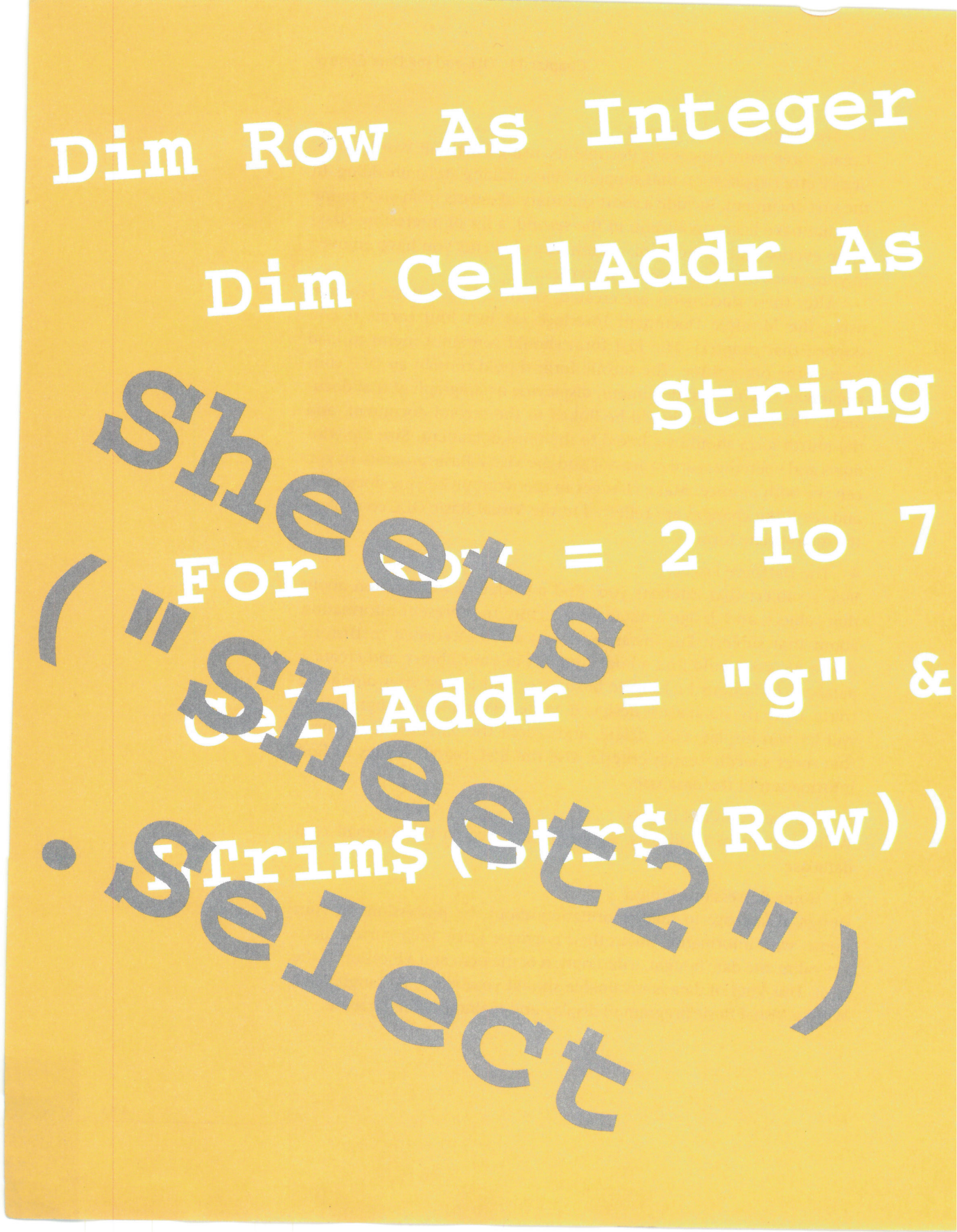
Pick a subject that interests you, find a source of information about that subject, and design a database structure to represent information about that subject. For instance, if you were interested in Historic Route 66, you could find a book about it in your library and create a database of famous or historic businesses and tourist sites along the route. Once the database is designed, write a Visual Basic program that will let you display, edit, delete, and search the database for records that meet specific search criteria. Use the SQL search strings to find information in the database.

5. The Baseball Card Project

Write a program to display, maintain, and search the baseball card database.

6. Home Possessions Project

Design a database to hold information about the possessions of your home, with information about their purchase price, their current market value, the date bought, a description of the item and a location of the item. This kind of data is invaluable should your home be burglarized. Write a Visual Basic program to display and maintain that database.


Dim Row As Integer
Dim CellAddr As
String
For
= 2 To 7
CellAddr = "g" &
Trim$(Str$(Row))
Sheets
("Sheet2")
.Select