1

Writing Your First AppleScript Program

There's no point in wasting time. You are reading this book because you want to learn how to write programs in AppleScript. So instead of wading through theory and terminology, I want you to begin by typing an AppleScript program, running it, and looking at the results. I introduce terminology as it is necessary along the way. If you begin this way, you'll have more fun and you won't get overwhelmed.

Starting with Script Editor

Your Mac OS X system comes with some preinstalled tools that you'll be using in this book. One of the most useful ones (as far as AppleScript programming goes) is an application called Script Editor. This application allows you to enter, compile, debug, and run your AppleScript programs in an interactive environment. It is a simple and effective tool.

The Script Editor application is stored in your `Applications` folder inside a subfolder called `AppleScript`. Its icon looks like the one shown in Figure 1-1.

Figure 1-1

Locate the Script Editor application and double-click it to start it. You should get a window on your screen that looks similar to the one shown in Figure 1-2. This window is where you type your AppleScript program. I show you how in the following Try It Out.

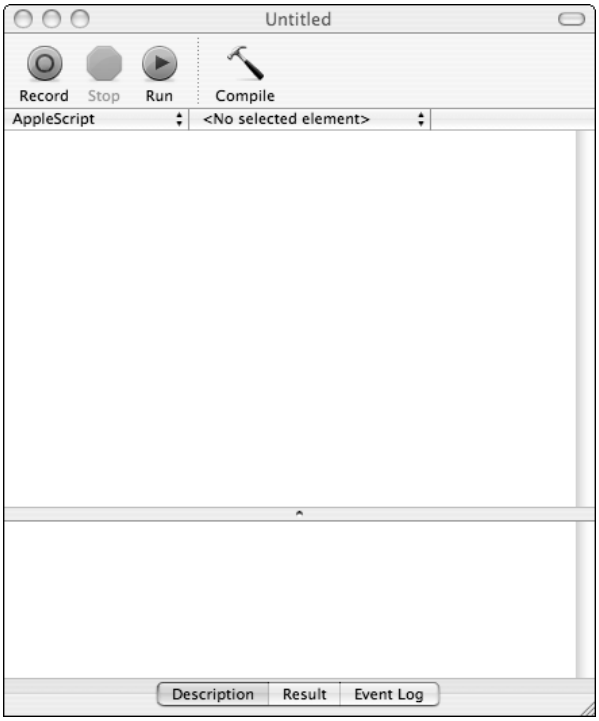

Figure 1-2

Try It Out Typing Your First Program

Click inside the Script Editor window and type the following:

```
100 * pi
```

Your window should look like Figure 1-3.

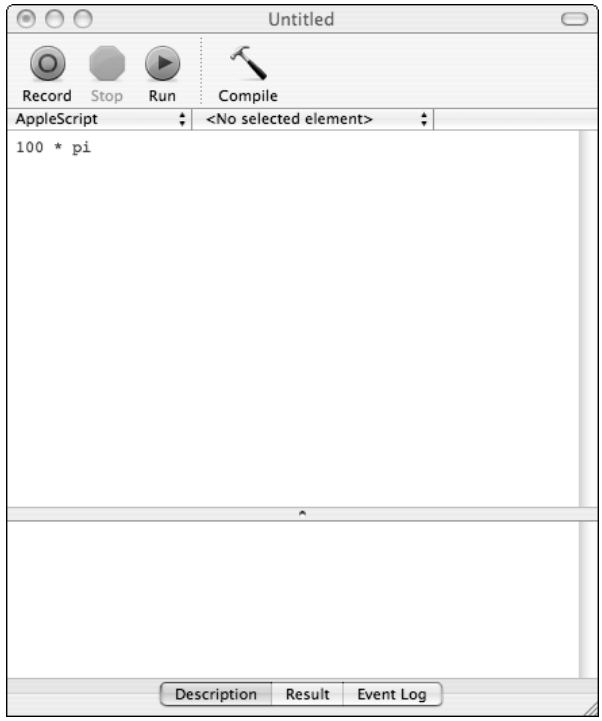

Figure 1-3

Compiling and Running Your Program

Try It Out Running Your First Program

After typing the indicated line in the window, move your pointer to the top of the window and click the button labeled Run, as depicted in Figure 1-4.

Figure 1-4

You'll notice that two things change in your Script Editor window, as depicted in Figure 1-5.

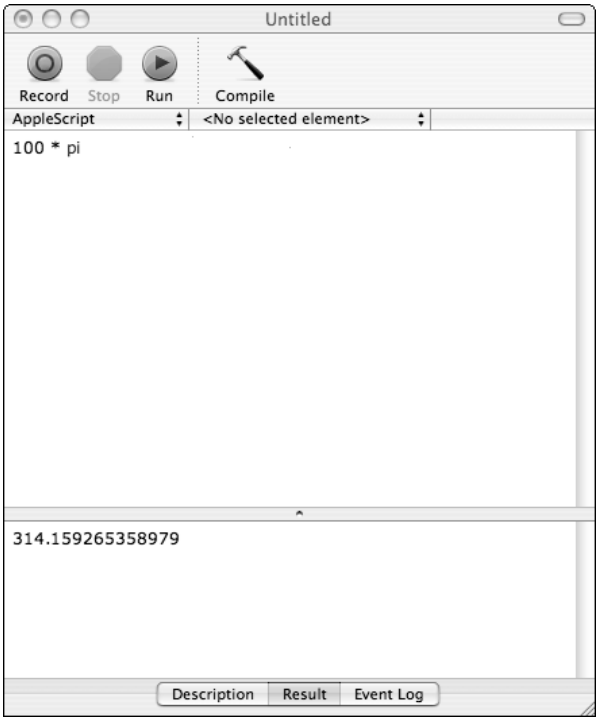

Figure 1-5

The first thing you may have noticed is that the color of your text changed in the window. Previously, when you typed the line

```
100 * pi
```

the text appeared in purple. After you clicked the Run button, the number `100` and the asterisk character `*` changed to black, and the word `pi` changed to blue. You may have noticed that the font changed as well, from Courier to Verdana.

> *This book is printed in black and white, so you obviously can't see the colors in Figure 1-5 or in any screen shots in this text. But if you're following along, you should see these color and font changes on your screen.*

Script Editor automatically displays any special words, known as *keywords*, in blue by default (you can select other colors to use if you like). This is done to visually aid you when you look at your program. As you'll see later, other colors and font styles are used to indicate program elements such as comments, variables, and operators.

At the bottom of the window, you should see the number `314.159265358979` (or something very close to it). Congratulations, this number represents the output (or the result) from running your first AppleScript program!

Make sure that the Result tab button is highlighted at the bottom of the window. If it isn't, click it so that the result of the operation can be seen.

How It Works

When you start up Script Editor, a window labeled Untitled is automatically created for you. In this window you can enter your AppleScript program. In the simplest case, a program can simply be an expression to evaluate, as in

```
100 * pi
```

This expression uses the multiplication operator, which is the asterisk (as it is in every other programming language I know of) to multiply the number 100 by something called pi. Recalling your high school mathematics, pi or π (as it's written using mathematical notation) is a number frequently used when performing calculations with circles. It's roughly equal to the value 3.14159. As you learn in greater detail in Chapter 2, the keyword `pi` is used in AppleScript as an approximation of the value of π.

So the expression

```
100 * pi
```

is an AppleScript expression that multiplies 100 by the value specified by the keyword `pi`, which explains the result you obtained.

After you type your program into Script Editor, you click the Run button. Clicking that button initiates the following actions:

- Analyzes your AppleScript code to make sure it conforms to the rules for writing commands and expressions. (This process is described in detail throughout this book and actually involves more than just a conformance check.) The result of this *compilation* process is the translation of your AppleScript program into another format that is more optimal for execution
- Indents your AppleScript code according to certain stylistic rules
- Changes the color of your words, operators, and expressions according to certain rules
- Runs your program and displays the result in the lower portion of your window, provided no errors are detected during the compilation phase.

Now that you've run your first program, you should save it. The following Try It Out walks you through doing just that.

Try It Out — Saving Your First Program

Even though this was an extremely simple first program, you should save it anyway.

1. Go to the menu bar at the top of your screen and choose File ➪ Save. A dialog box like the one shown in Figure 1-6 appears.

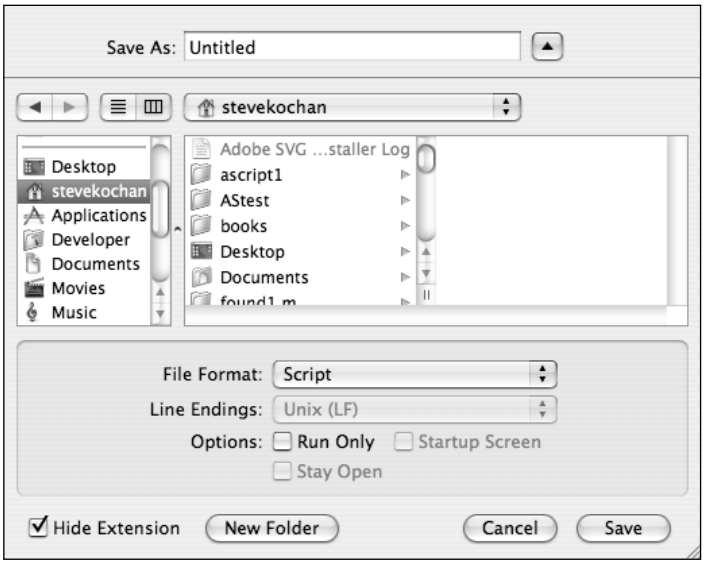

Figure 1-6

2. Your dialog box may look a little different based on the files stored on your system and which version of OS X you're running. In any case, in the box labeled Save As, replace the name `Untitled` with a more meaningful name such as `My First AppleScript Program`.

3. The other settings in the dialog box should remain as shown in Figure 1-6. After entering the file name, click the Save button. This causes your program to be saved to the specified file name.

When you go to save your program, Script Editor tries to compile it before saving. In that respect, it forces you to correct any errors before you can save your program to a file.

Later, if you want to make changes to your program, you can double-click the file name in the Finder to launch Script Editor with the specified file open. Alternatively, if Script Editor is already running, you can open the file for editing by choosing File➪Open.

You now know how to start the Script Editor application, type an expression (which represents a complete AppleScript program), compile and run the program, view the results, and save the program to a file.

Extending Your First Program

You're not quite finished with your first program. In this section, you see how Script Editor reports errors. Then you learn how to add a description to your program and how to use a command that opens a simple dialog box.

Reporting Errors in Your Programs

First, let's see what happens if you make a mistake. Erase the contents of your window (one way is by highlighting all the text using your mouse and pressing the Delete key).

Now type the following into an empty window:

```
100 * pie
```

Click Run to compile and run your program, as you did previously. A dialog box like the one shown in Figure 1-7 should appear.

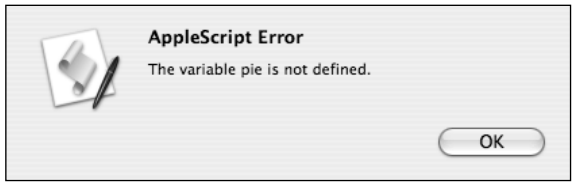

Figure 1-7

The Script Editor application is telling you that it doesn't know about a variable called `pie`; that is, `pie` is not defined. You learn about variables shortly; but essentially, a *variable* is a place in memory that you assign a name to and use for storing data. For now, just click OK to dismiss the dialog box. Your window should now look like the one shown in Figure 1-8.

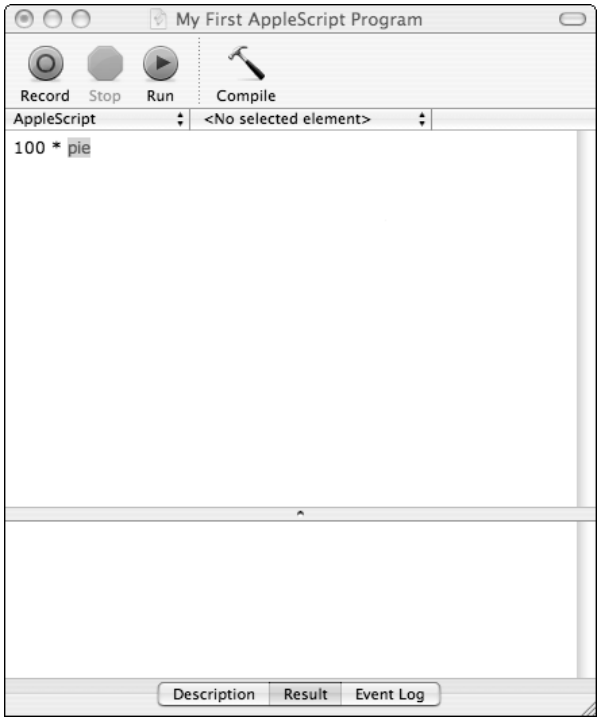

Figure 1-8

Notice that `pie` is highlighted. That's where the error occurred. Unlike the keyword `pi`, the word (or variable) `pie` does not have a special meaning in AppleScript, thus explaining the error message. In this case, you can simply delete the trailing `e` in `pie` and run the program again to remove the error.

Sometimes it's easy to decipher the error messages reported by Script Editor. At other times, the messages can seem quite cryptic. With experience, you will be able to readily understand the messages and the types of errors that cause them.

In the following Try It Out, you practice adding a description to your program.

Try It Out Adding a Description to Your Program

You may be wondering about the purpose of the two other tabs that abut the Result tab at the bottom of your window. One is labeled Description, and the other is labeled Event Log, as you can see in Figure 1-8. I describe the Event Log tab in Chapter 2. You can use the Description tab to enter a description about what your program does, that is, to add notes about its operation.

1. Click the Description tab and then click in the lower pane of your window. Script Editor waits for you to enter your description.
2. Type the following line into the pane:

```
This is my first AppleScript program to multiply two numbers.
```

Your window should now resemble the one shown in Figure 1-9.

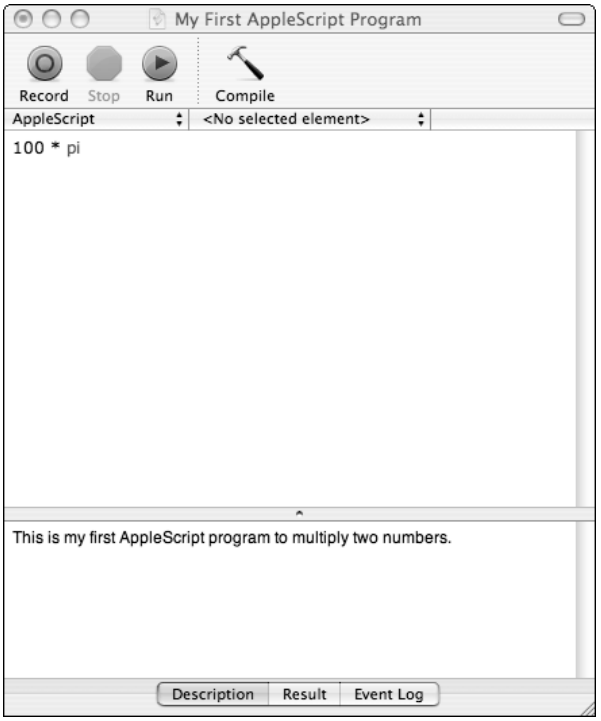

Figure 1-9

3. Now save your program by choosing File ➪ Save, and the description you entered is saved along with it.

   You've already given your program a name (you called it My First AppleScript Program when you initially saved it), so you can simply choose File ➪ Save to subsequently save any changes you make under the same name.

In Chapter 3, you see how the description you entered can be automatically displayed when you run your program as an application.

An AppleScript Command to Display a Dialog

Your first program shows the result of the expression evaluation in the Result pane. You use this Result pane (and the Event Log pane) to examine results throughout this book. However, to create an AppleScript program to talk to the user or to interactively display information, you need to do more. The simplest way to do this is to use an AppleScript command called `display dialog`.

For your second program, you put up a dialog box that says `Programming in AppleScript is fun.` If you still have your first program open in Script Editor, close the window by selecting File ➪ Close or by simply clicking the red dot in the top-left corner of the window. If you've made any changes to the program since the last time you saved it, Script Editor gives you a chance to save the changes before closing the window.

Now open a new window by choosing File ➪ New. You get a new empty window. Inside that window, type the following text:

```
Display Dialog "Programming in AppleScript is fun."
```

Your window should look like the one shown in Figure 1-10.

Now, instead of clicking the Run button, click the Compile button. This has the effect of analyzing your program and reporting any errors to you without actually running your program. Notice that the colors and formatting are changed by Script Editor, as depicted in Figure 1-11.

The `display dialog` command changes to blue and also to all lowercase letters. The blue color indicates special keywords, just as the keyword `pi` was displayed in blue earlier. And the conversion to all lowercase is done because, in general, AppleScript does not distinguish between upper- and lowercase letters. In later examples you see how, after compiling your program, Script Editor not only performs this case conversion, but it also may change and even rearrange some words in your commands! Finally, as noted before, the font of your small program changes from Courier to Verdana.

No error messages resulted from compiling your program, so you can now run the program by clicking the Run button.

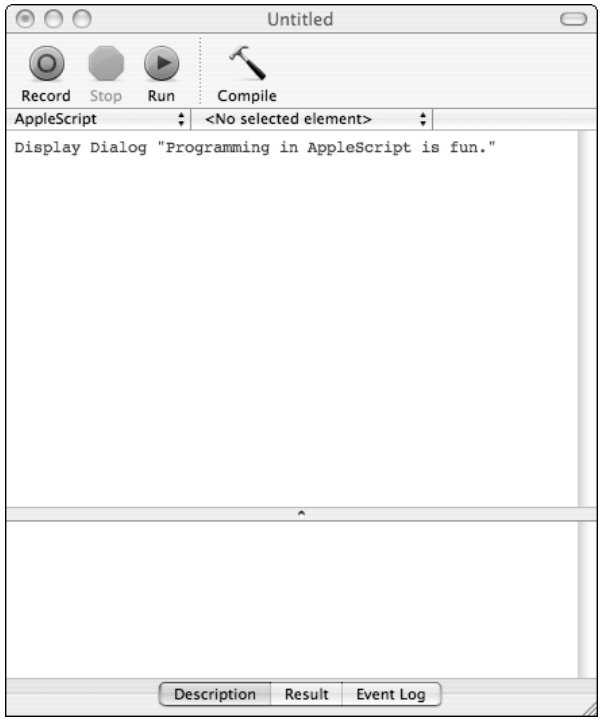

Figure 1-10

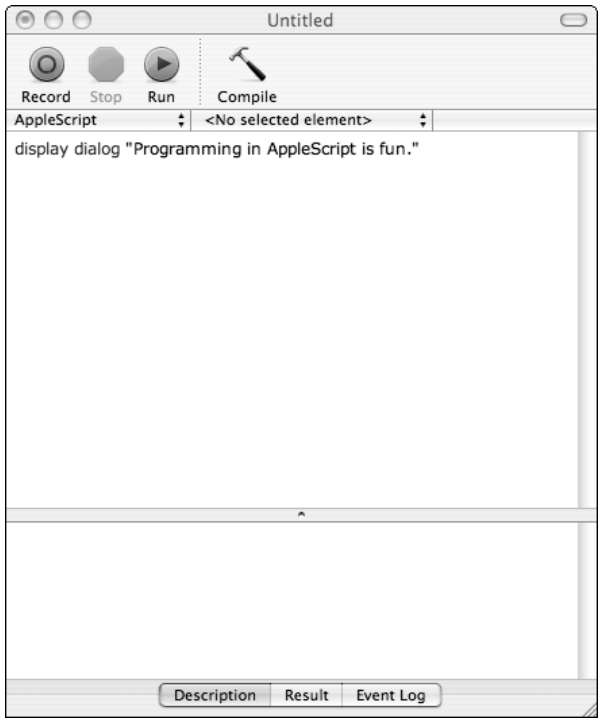

Figure 1-11

So that you don't get confused, remember the Compile button compiles your program but does not run it. The Run button also compiles your program if it has changed since the last time it was compiled. Then, if no errors are detected, it runs your program.

When you run your program, the dialog box shown in Figure 1-12 appears on your screen.

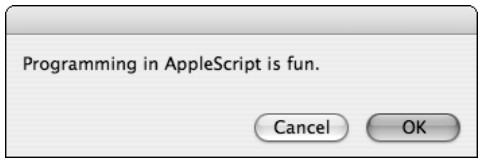

Figure 1-12

As you can see, the phrase `Programming in AppleScript is fun.` is displayed in the dialog box, without the quotes you typed around those characters. Those double quotes are used to define the beginning and end of your *character string*. Whatever is contained inside those quotes gets displayed by the `display dialog` command. You are also shown two buttons, one labeled Cancel, and the other OK. The OK button is highlighted, indicating that the default action that will be taken if you press the Return key. Click the OK button (or press the Return key) to return control back to Script Editor. Notice that the following appears in the Result pane:

```
{button returned:"OK"}
```

You see that this window doesn't just display the results of performing arithmetic operations. This time the window tells you that the OK button was clicked. This may not seem important now, but later you learn how to test which button is clicked in your program and how to take appropriate action based on that test. As a quick note, if you click the Cancel button instead of OK, no result is displayed in the Result pane. That's because, unless you take special action, the program is immediately terminated when you cancel it.

Understanding the Buttons Parameter in the display dialog Command

In the example shown in the preceding section, the Cancel button served no purpose. You can get rid of the Cancel button by using a `buttons` parameter with the `display dialog` command. The following Try It Out shows you how.

Try It Out Using the display dialog's Buttons Parameter

Follow these steps to eliminate the Cancel button from the previous dialog box:

1. Go back to the program you saved in the preceding Try It Out and add the following characters to the end of the `display dialog` command.

```
buttons {"OK"}
```

Your complete command line should look like this:

```
display dialog "Programming in AppleScript is fun." buttons { "OK" }
```

2. Compile and run the program. You see the dialog box shown in Figure 1-13.

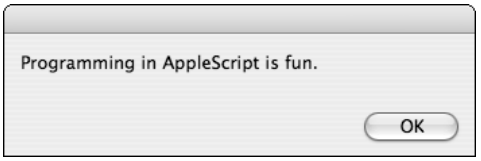

Figure 1-13

Because of the changes you made to the code, the dialog box no longer contains a Cancel button but just the single button labeled OK.

How It Works

The word `buttons` that you added to your program is an optional *parameter* to the `display dialog` command. AppleScript commands often accept parameters that enable you to extend the functionality of a command.

> *You can see that AppleScript, unlike any other programming language you probably have used, employs English words in its vocabulary. This allows you to write commands that are easy to read. However, it's also a trap that may mislead you into thinking you can write just about any English sentence in your program. That's not the case; the AppleScript language has a fairly rigid structure and, generally, likes its words to be in the expected places. Through practice, you can learn this syntax and begin to understand where words are acceptable and where they're not.*

Following the keyword `buttons` is the string `"OK"` enclosed inside a pair of curly braces `{` and `}`. These curly braces have special meaning in AppleScript: They define a *list*. Lists are discussed in full in Chapter 6. Between now and then, however, you learn some basic concepts for working with lists in your programs.

The list in this program contains just one *item*, which is the character string `"OK"`. You can include additional items in a list by placing them inside the curly braces and separating one item from the next with a comma. For example, here is a list containing seven strings:

```
{ "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" }
```

And here is a list containing three numbers:

```
{ 1, 2, 3 }
```

Finally, because lists can contain just about anything, here's one that contains a string and three numbers:

```
{ "DOB", 8, 8, 1986 }
```

Changing the Labels on a Button

Your program does not have to have a button labeled OK. In fact, you can choose any labels you want for your buttons. Simply specify the label in the list after the `buttons` parameter like this:

```
display dialog "Programming in AppleScript is fun." buttons { "You're so right!" }
```

When you run a program containing this command, the dialog displays a button with the label `You're so right!`.

The following Try It Out shows you how to set a default button.

Try It Out — Setting the Default Button in a Dialog

Follow these steps to set a default button in a dialog:

1. Go back to your program and change the command to the following:

```
display dialog "Pick a color" buttons {"Red", "Yellow", "Blue"}
```

   This change puts up a dialog box with the message `Pick a color` and buttons labeled Red, Yellow, and Blue, respectively.

2. Run your program containing this command, and you should get a dialog that looks like the one shown in Figure 1-14. The actual button you select should appear in the Result pane, as previously described.

Figure 1-14

   You may have noticed that most standard dialogs on the Mac have a default button. A *default button* is one that is highlighted to indicate what will be selected if you press the Return key.

3. Make the default button the third button in the list, which is labeled Blue. You can write either

```
display dialog "Pick a color" buttons { "Red", "Yellow", "Blue" } default button 3
```

or

```
display dialog "Pick a color" buttons { "Red", "Yellow", "Blue" } default button
     "Blue"
```

4. Compile and run your program. You should get the dialog shown in Figure 1-15.

Figure 1-15

How It Works

By using the `default button` parameter, you can specify the default. The value that follows this keyword is either the number of the button in the list (that is, 1, 2, or 3) or the button's label. It doesn't matter which you choose. The label may be a better choice because it makes the command easier to read.

> *If you type a long line into Script Editor, the editor automatically wraps your line at an appropriate point. This is fine. Script Editor knows that this continuation is part of the previous line when compiling your program. This is important because you cannot extend an AppleScript command over multiple lines simply by pressing the Return key to break the line. Instead, if you want to manually break a line (perhaps at a place you find more meaningful than where Script Editor breaks it for you), you have to press Option-Return to continue the command on the following line. Pressing Option-Return shows up as the character ¬ on your screen, as in the following example:*

```
display dialog "Pick a color" buttons {"Red", "Yellow", "Blue"} ¬
     default button "Blue"
```

As you can see in Figure 1-15, shown earlier, the button labeled Blue is highlighted in the dialog. If you simply press the Return key, notice that Blue is shown in the Result pane as the button returned.

> *You can have up to three buttons in your* `display dialog` *command. In Chapter 6, you learn about the* `choose from list` *command, which you can use to present any number of choices to the user.*

Adding an Icon to Your Dialog Box

As the last extension to your dialog, add a special symbol, known as an *icon*. You can add any icon you want, but three of them are predefined, as shown in the following table.

Icon Name	Symbol
`stop`	
`note`	
`caution`	

The parameter you use to add an icon is called `with icon`. It is followed by a name, a string, or a number. The following Try It Out shows you how to add an icon to a dialog.

Try It Out Adding a Note Icon to Your Dialog

Follow these steps to add a note icon to the dialog you generated earlier:

1. Go back to the program you created and revised earlier and add a note icon to the dialog so that your `display dialog` command looks like this:

```
display dialog "Pick a color" buttons {"Red", "Yellow", "Blue"}
    default button "Blue" with icon note
```

2. Run this program, and you should see the dialog shown in Figure 1-16.

Figure 1-16

Multiple AppleScript Commands

So far, your programs have done only a single thing: either perform a multiplication or display a dialog. It's obvious that you can do a lot more in your programs. Each line can contain another AppleScript command, which is executed sequentially, in turn, when the program is run.

The following Try It Out shows you how to add a sound effect to your program.

Try It Out Add a "Beep"

Suppose you're running a program and you detect that an error has occurred. For example, suppose the program can't locate a file it wants to use called `myFavorites`. It would be reasonable to put up a dialog box telling the user that the error occurred. Adding a beep can help get the user's attention.

Follow these steps to add a sound effect with AppleScript's `beep` command:

1. Create a new AppleScript program with the following code:

```
beep
display dialog "Couldn't find myFavorites" buttons { "OK" } default button "OK"
    with icon stop
```

2. Run the program. A beep will sound, and then a dialog is displayed with a stop icon and a single default button labeled OK. This is depicted in Figure 1-17.

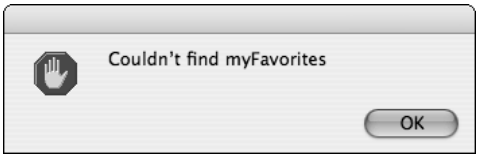

Figure 1-17

Adding Comments to Your Programs

As you start writing larger AppleScript programs, it is a good habit to document your program. You can do this directly in the program by inserting *comments*. A comment describes to the reader of the program the intentions of a particular sequence of code. Not only is this helpful when you go back months later to look at code that you wrote, but it can also aid the person who has to maintain your code.

You can insert a comment into an AppleScript program in one of two ways:

- By typing two consecutive hyphen characters (--) followed by any characters you want. Any characters that appear on the line after the two hyphens are ignored when the code is compiled.
- By typing an open parenthesis and an asterisk [`(*`] followed by your comments spread out over as many lines as you like. You terminate this comment style by typing an asterisk and close parenthesis [`*)`].

Here's an example of the first type of comment:

```
-- This program will resize all selected images to a specified size
```

And here's an example of the second type of comment:

```
(*
    This program creates an alphabetized list
    of all of the songs in someone's iTunes library

    Version 1.1 created 2/22/05
    Stephen Kochan
*)
```

You can mix and match your comment styles. And the second form of commenting can itself include comments. That is, comments can be nested.

The double hyphen is often used to add a single-line comment or a comment at the end of a line. In addition to its use for multiline comments, the paren-star comment is often used to "comment-out" a section of your program that is not yet finished or for debugging purposes.

Figure 1-18 shows how the previous program example appears in Script Editor after adding a comment and compiling it. Notice that Script Editor italicizes both types of comments.

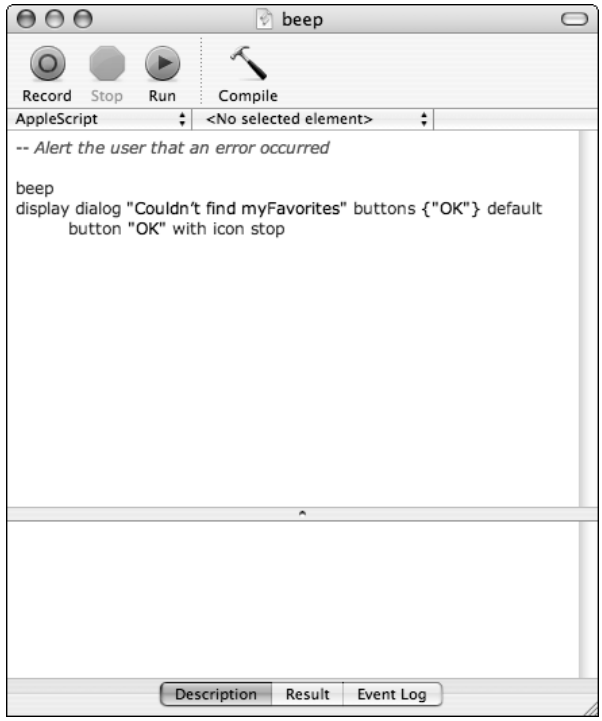

Figure 1-18

Summary

In this chapter, you learned how to do the following:

- Start up Script Editor
- Type a simple AppleScript program, compile it, and run it
- Look at the results of the program in the Result pane
- Save your AppleScript program to a file
- Add a description to your program
- Use the `display dialog` command to display a message to the user
- Define a character string
- Set up a simple list
- Use optional command parameters; in particular, use some of the `display dialog` command's parameters to change the labels on buttons, specify a default button, and display an icon
- Write an AppleScript program containing more than one command
- Add comments to your program

In the next chapter, you learn more about writing expressions, the different types of data that AppleScript supports, and how to work with variables in your programs. Before proceeding, however, try the exercises that follow to test your understanding of the material covered in this chapter. You can find the solutions to these exercises in Appendix A.

Exercises

1. Modify the expression from your first program so that it calculates the circumference of a circle with a radius of 1.48. As a hint, recall that the circumference of a circle with radius *r* is expressed by the formula $2\pi r$. Have the program perform the entire expression evaluation (that is, don't calculate 2*r* yourself!).
2. Write a program that poses the following question: "Do you think this is a great book?" Have two buttons labeled Yes and No, with Yes as the default.
3. Add a comment to the beginning of the program you created in Exercise 2 that describes the purpose of the program.
4. Add the following description to the program created in Exercise 2: "A program to ask a profound question!"
5. Modify the `display dialog` example (used earlier in this chapter) that presented the three color choices to make "Red" the default button.
6. Can you alter the order of the display dialog parameters? Experiment and see.
7. The `beep` command takes an optional numeric parameter that specifies the number of times to beep your system. For example, `beep 2` causes two beeps. Modify the last program in this chapter to beep 3 times before the dialog is displayed.
8. The AppleScript `say` command can be used to speak text. The command is followed by the character string to be spoken. Write an AppleScript program that speaks the phrase "Programming in AppleScript is fun."