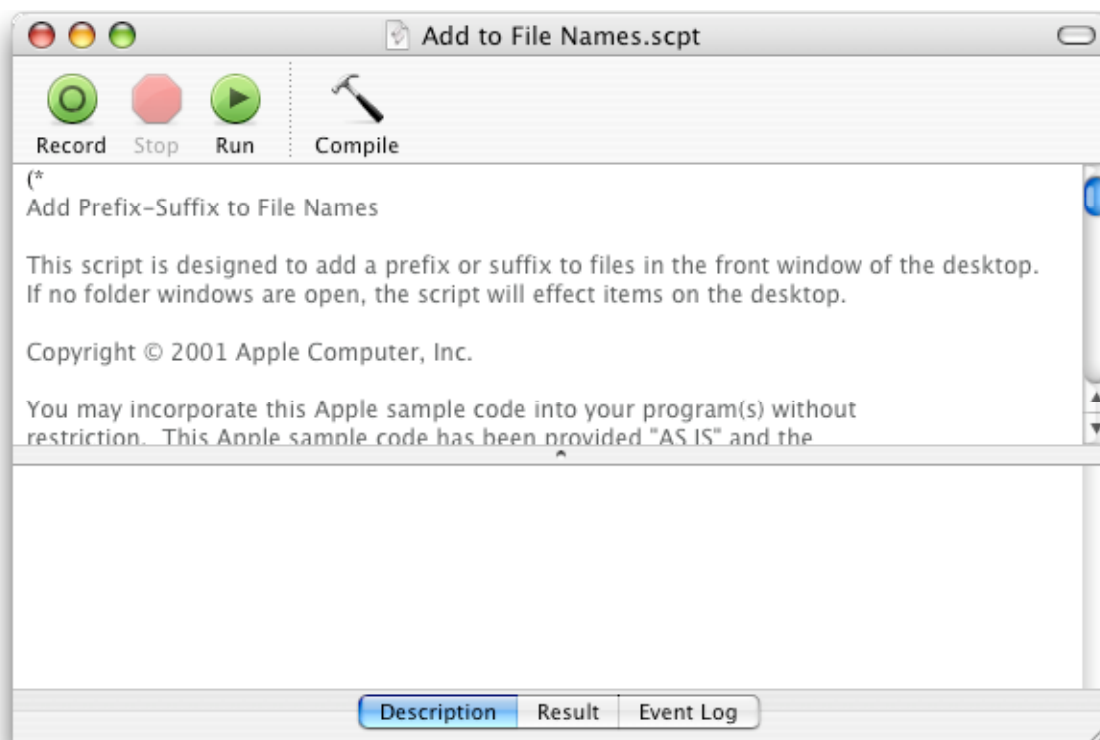


# Advanced AppleScript—Bonus Writeup

*Welcome to this bonus material, which is designed to accompany Chapter 7 of "Mac OS X: The Missing Manual," Panther Edition.*

*Imagine that you're deep in Chapter 7, reading about AppleScript, when suddenly:*

The script called Add to File Names.scpt is one of the professional sample scripts that come with Mac OS X. To open it, navigate to your Library -> Scripts -> Finder Scripts folder and double-click Add to File Names.scpt. A quick glance can tell you a lot about the tricks of professional scripters.



The Add to File Names script is a classic AppleScript example. It contains variables, subroutines, nested "if" statements, and many other common elements of professionally written scripts. (After typing a description into the box at the top of the window, you can click the flippy triangle to the left of the Description box to hide the entire thing, thus maximizing your script-writing space.)

- **Description.** Careful script writers *document* their work. They add lots of notes and explanatory comments for the benefit of whoever might want to study or amend the script later (which is often themselves).

(Whoever wrote this particular description didn't know the difference between *effect* and *affect*, but it's the thought that counts.)

- **Variables.** A *variable*, in programming terms, is a placeholder for some information that may change from time to time. The President of the United States is a variable: there always *is* one, but his name changes every few years.

Variables serve as stand-ins for more complicated ideas, which help to simplify the script, clarify its purpose, and save typing. In the one line of the Add to File Names script, for example, you can see the command *set this\_item to item i of the item\_list*. That common command tells AppleScript that you're defining your own variable called *this\_item*, which will henceforth mean "a reference to the *i*th item in the list of items called *item\_list*." In this case, "*i*" is a counter that refers to the item's position in the list.

Note: For some reason, AppleScripters tend to use variable names where the first letter is lowercase, but subsequent words appear with no spaces and capital letters, *somethingLikeThis*, or with underlines, *like\_this*.

- **Subroutines.** About a third of the way into the script shown in Figure 7-8, you'll find a line that says, *my set\_item\_name(this\_item, the\_new\_item\_name)*. That step tells the script *not* to proceed down the list of commands in sequence, but instead to jump to a *subroutine* (also called a *handler*)—a group of commands that's been separated from the first batch. Subroutines allow groups of commands to be more easily reused in different parts of the script. In this case, the subroutine begins four lines later (*my set\_item\_name(this\_item, new\_item\_name)*).
- **Nested "if" statements.** You might tell an underling, "If you have time this afternoon, would you please run to the store? If the steaks are still on sale, buy three pounds." That, believe it or not, is a *nested "if" statement*. In other words, the steak-buying will take place only if the underling has time this afternoon, that is, if some specified *conditions* are met.

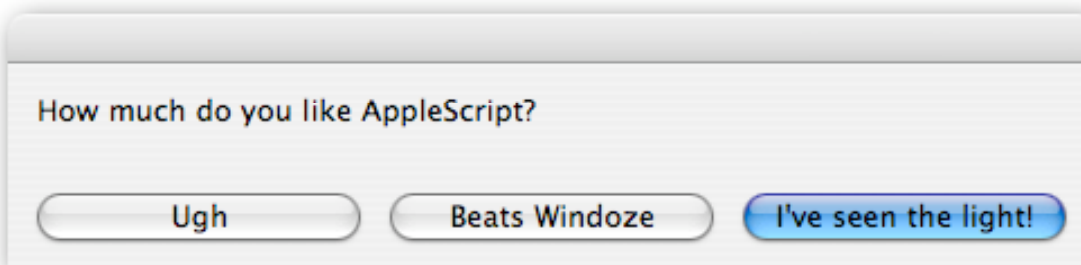
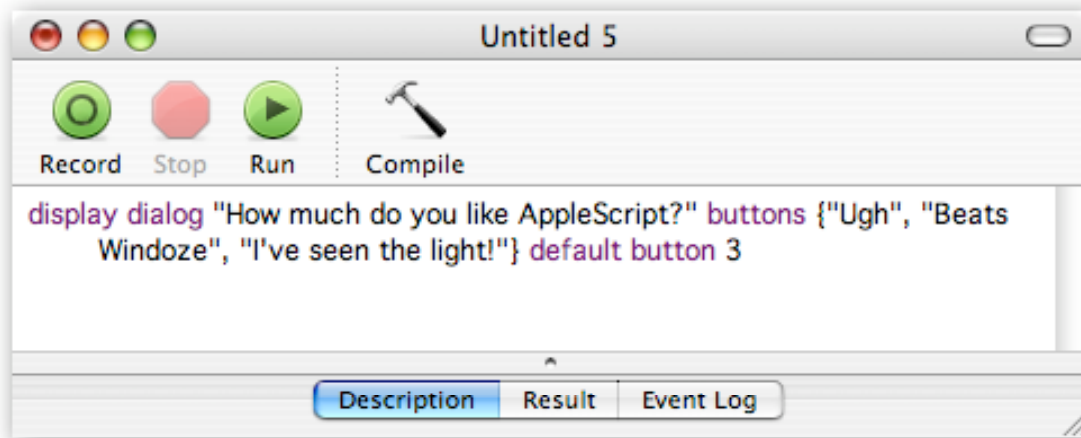
The Add to File Names script has a few nested "if" statements. For example, the sixth line says, in essence, "If this thing is neither a folder nor an alias"; the next line says, "...and if whoever's operating this thing clicks the Prefix button." The script proceeds only if both of those conditions are true.

- **Comments.** Do you see the italicized comment, "*--the suggested name is too long*"? That's the programmer's note to anyone who studies the script. It's a *comment*, an annotation to make clear what the code is supposed to be doing. When the Mac runs AppleScript programs, it ignores the comments. You create one by typing two hyphens (--); when you click the Check Syntax button, Script Editor automatically sets what follows in italics.

Tip: If you want to type a longer comment—a whole paragraph between commands in your script, for example—just precede and follow it with a parenthesis and asterisk (\*like this\*).

- **Looping.** The paired commands *repeat* and *end repeat* create a loop—a set of commands that AppleScript repeats over and over again until something (which you’ve specified) interrupts it—or not. In the case of the Add to File Names script, the script loops until it’s added a prefix or suffix to all of the files in the specified folder—at which point it’s supposed to beep twice. (In OS X versions before 10.3 it actually seems as though it beeps only once, thanks to a quirk of Mac OS X’s sound software, but in 10.3 you get two distinct beeps again.)
- **Dialog boxes and buttons.** If you ever want to play programmer for a day, open Script Editor and try creating dialog boxes and buttons of your own. The script command *display dialog* (followed, in quotes, by whatever message you want to appear on the screen) is all you need, as shown in the figure below.

Tip: Instead of a dialog box, you can also interact with the person using the computer by making the Mac speak. (See Chapter 14 for more on speech.) Just use the *say* command. If your AppleScript contains the line, *say “Hey! Pay some attention to me!” using “Zarvox,”* then the Mac speaks that line using the voice called Zarvox. In addition to being fun, audio feedback can be useful when debugging scripts.



The script shown here at top produces the dialog box shown at bottom. The buttons command lets you create your own buttons (up to three, punctuated exactly as shown here). The default button command tells the script which button should be the default, the pulsing or outlined one that you can “click” just by pressing the Return key. Default button 3 means that the third button is the default button.

- **Line breaks.** When a line of AppleScript code gets too long, it’s hard to read—especially if it’s wider than the Script Editor window itself. AppleScript pros, therefore, insert a special *continuation symbol* that makes the text wrap to the next line, which causes AppleScript to treat both lines as a single command. You create this line-break symbol by pressing Option-Return or Option-L.

Note: Don’t use a line-break character in the middle of text that appears inside quotation marks. You can, however, put normal carriage returns in a text string. Some commands and application keywords are more than one word long, and your script won't work if you split them across two lines with a line-break character.

- **Try...On error...End try.** This suite of three commands appears frequently in polished AppleScript scripts—including the one shown in Figure 7-9. In essence, it tells the Macintosh: “Try to do this. If it doesn’t work out, show this error message.” Here’s an example:

```
tell application "Finder"  
  try
```

```
    make new folder at startup disk with properties
{name:"Backup Folder"}
    on error
    display dialog "A folder already has that name."
    end try
end tell
```

This script tries to make a new folder named Backup Folder on the startup disk. But if there's already a folder there by that name, then AppleScript generates an error—and shows the error message you specify.