

AppleScript

You can think of AppleScript programs (called *scripts*) as software robots. A simple AppleScript might perform some daily task for you: backing up your Documents folder, for example.

A more complex script can be pages long. In professional printing and publishing, where AppleScript enjoys its greatest popularity, a script might connect to a photographer's hard drive elsewhere on the Internet, download a photo from a predetermined folder, color-correct it in Photoshop, import it into a specified page-layout document, print a proof copy, and send a notification email to the editor—automatically.

Even if you're not aware of it, you use technology that underlies AppleScript all the time. Behind the scenes, numerous components of your Mac communicate with each other by sending *Apple Events*, which are messages bearing instructions or data that your programs send to each other. When you use the Show Original command for an alias, or the Get Info command for a file or folder, an Apple Event tells the Finder how to respond. Some of the files in the Speakable Items folder (page 471), furthermore, are AppleScripts that quit your programs, open AppleWorks, switch a window into list view, and so on.

AppleScript veterans will find a lot of enhancements in Mac OS X 10.3—and there's a nice surprise for first-timers, too.

Running Ready-Made AppleScripts

You don't have to *create* AppleScripts to get mileage out of this technology. Mac OS X comes with several dozen prewritten scripts that are genuinely useful—and all

you have to do is choose their names from a menu. “Playing back” an AppleScript in this way requires about as much technical skill as pressing an elevator button.

Tip: Similarly, you can download all sorts of useful scripts for iTunes, iPhoto, iDVD, AppleWorks, and other programs from the Apple Web site (www.apple.com/applescript/imovie; substitute the program name for that final part of the address).

To sample some of these cool starter scripts, you must first add the Script menu to your menu bar (see Figure 7-1).

Now open the newly installed Script menulet (which Apple calls simply the Script *menu*), whose icon looks like a scroll, to see the list of prewritten scripts.

Tip: The Script menu reflects the contents of two different Scripts folders: the one in your Home→Library Scripts folder, and the one in your main Library folder. (The ones in your Home folder are listed below the dotted line in the Script menu.)

These scripts aren't just for *running*. They're also ideal for opening up in Script Editor (just by double-clicking) and analyzing line by line, to learn how they work. (Script Editor is a program in your Applications→AppleScript folder, which you can use to type up your own scripts.) Once you understand the syntax, you can then copy bits of the code to modify and use in your own scripts.

POWER USERS' CLINIC

What's New in Panther's AppleScript

Dear AppleScript veterans:

A lot of the great old scripting tools (like scripting additions), Web sites, and Internet resources apply only to AppleScript in Mac OS 9 and earlier. And thousands of scripts that require the old Finder flounder in the Mac OS X Finder, which works somewhat differently.

Still, after several incarnations, the AppleScript of Mac OS X is finally becoming a solid tool with lots of features. Finder recordability has returned to the Mac in Panther—a great boost for first-timers.

Some of the other new features in Mac OS X 10.3 include:

User-interface Scripting. In Panther, AppleScript can choose menu commands, click radio buttons, turn on checkboxes, and type into text boxes in the course of running a script—a very useful enhancement in situations where the program you want to script isn't very AppleScript-savvy.

Script Editor 2. In version 2, Script Editor offers find-and-

replace commands, a spelling checker, and a contextual menu (available by Control-clicking the main panel of the Script Editor window) that offers quick access to a library of pre-built, error-free code snippets.

Better yet, Script Editor is now itself scriptable; in theory, you could even write scripts that write other scripts.

Folder Actions Setup. The new Folder Actions Setup panel lets you manage all of your Folder Actions (page 212) from a single panel. To open this panel, choose Configure Folder Actions from your Script menu (page 193), or open Applications→AppleScript→Folder Actions Setup.

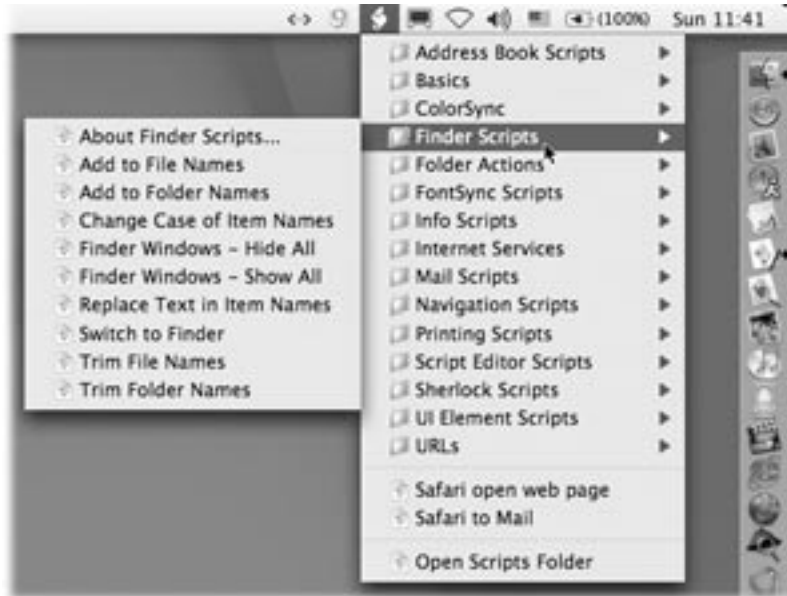
Image Events. The new Image Events application lets your scripts flip, convert, and otherwise manipulate graphics files without having to fire up Photoshop.

Help. AppleScript Help isn't actually new in Mac OS X 10.3, but it's been expanded so much, it merits a mention. Anyone new to AppleScript should explore the AppleScript or Script Editor Help as a first resort in times of confusion.

Some of the scripts operate on familiar components of the Mac OS, like the Finder; others show off applications or features that are new in Mac OS X. Here are the categories as they appear in the Script menu:

Figure 7-1:

Mac OS X comes with an assortment of useful scripts to try. The Script menu (which you install from your Applications→AppleScript folder) lets you launch Perl, Shell, and AppleScript scripts, just by choosing their names, from within any application. To install the Script menu, open your Applications→AppleScript folder and double-click Install Script Menu.



Address Book Scripts

In this folder, you'll find a single **Import Addresses** script that's designed to move addresses to the Address Book from Palm Desktop, Entourage, Eudora, or Outlook Express.

Basics

In this folder, you'll find a few simple scripts like **AppleScript Help** (which opens the Help Viewer and searches for the word *AppleScript*) and **AppleScript Website** (which opens the AppleScript Web page in your Web browser).

ColorSync

In this folder, you'll find seventeen ColorSync script *droplets* (scripts that run when you drop something on their icons) primarily designed for graphic artists, Web site designers, publishers, and so on.

If you choose a script's name from the menu, you'll get a terse help message and then an Open dialog box for choosing the graphics file you want to process. Alternatively, you can drag the graphics files onto the script's icon in the Library→Scripts→ColorSync folder.

Finder Scripts

All of these scripts have to do with the Finder—manipulating files and windows, for example. A few of the most useful:

- **Add to File Names, Add to Folder Names.** These scripts tack on a prefix or suffix to the name of every file or folder in the frontmost Finder window (or, if no windows are open, on the desktop). You could use this script to add the word *draft* or *final* or *old* to all of the files in a certain folder.
- **Finder Windows - Hide All** minimizes all open Finder windows to the Dock, as though you'd pressed Option-⌘-M. **Finder Windows - Show All**, of course, brings them back from the Dock.
- **Replace Text in Item Names** lets you do a search-and-replace of text bits inside file names, folder names, or both. When one publisher rejects your 45-chapter book proposal, you could use this script to change all 45 chapter files from, for example, "A History of Mouse Pads—A Proposal for Random House, Chapter 1" to "A History of Mouse Pads—A Proposal for Simon & Schuster, Chapter 1."
- **Trim File Names, Trim Folder Names.** If you made a mistake in using the Add to File Names script, you can always use Trim File Names script to undo the damage. This one *removes* file extensions, suffixes, or prefixes of your choosing.

For example, suppose you've just made a lot of new folders at once. Mac OS X calls them "untitled folder," "untitled folder 2," and so on. But what if you'd rather have them just called "folder 1," "folder 2," and so on? Run the Trim Folder Names script; when the dialog box asks you what you want trimmed, type *untitled* and click OK.

Folder Actions

This folder contains scripts that were required, in Mac OS X 10.2, to make *folder actions* work (page 212). In Panther, you probably won't use them much, since Control-clicking a folder (or inside its window) offers the same functions.

Folder Action Scripts

The scripts in this folder don't actually show up in the Script menu. They are, however, in your Library→Scripts folder. You need to attach them to folders to use them.

They're sample *folder action* scripts—designed to make folders behave the way you wish they would. These scripts are useful just as they are, but even more useful as examples to help you learn the syntax for writing more folder action scripts.

- **add – new item alert.** When you drop icons into a folder (to which you've attached this script), the Mac will tell you how many icons were added and offer you the chance to inspect the contents of the folder.
- **close – close sub-folders.** When you close the folder to which this script is attached, all folders *inside* it also close.

- **open – show comments in dialog.** Whenever you open the folder to which this script is attached, the Mac will show you the contents of its Comments box (that is, whatever you typed into the Comments area of its Get Info window). You'll be given the chance to open the Get Info window, delete the comments, or do nothing further.
- **convert - PostScript to PDF.** When you drop PostScript files into a folder to which you've attached this script, the Mac converts them into PDF files. (Pay attention, you graphic designers and print-shop staff who don't have Adobe's Acrobat Distiller on hand.)

The remaining nine scripts (**Image - Add Icon**, **Image - Duplicate as JPEG**, **Image - Duplicate as PNG**, **Image - Duplicate as TIFF**, **Image - Flip Horizontal**, **Image - Flip Vertical**, **Image - Info to Comment**, **Image - Rotate Left**, and **Image - Rotate Right**) all perform the specified flipping or conversion operations on graphics files. For example, if you dump a bunch of TIFF files into a folder to which you've attached the Duplicate as JPEG script, Mac OS X converts them into JPEG format for you (and preserves the originals).

These scripts are brought to you by Panther's new Image Events Scripting application (see page 194).

FontSync Scripts

FontSync is a noble Apple attempt to solve an old problem for desktop publishers. You finish designing some beautiful newsletter, take it to the local printing shop for printing on a high-quality press, and then have to throw out the entire batch—all because the fonts didn't come out right. The printing shop didn't have exactly the same fonts you had when you prepared the document. Or, worse, it did have the same font—but from a different font company, with the same name but slightly different type characteristics.

FontSync is designed to give you early warning for such disasters. When you run the Create FontSync Profile script, several minutes elapse—and then the Mac generates a FontSync Profile document. This file contains staggering amounts of information about the design, spacing, and curlicues of every font installed in your system. When you hand that profile over to your print shop, they can drop it onto the accompanying script, called Match FontSync Profile. It will tell them precisely what fonts are different on their Macs and yours.

The wishful-thinking aspect of this technology is, of course, that it assumes a lot: that your print shop uses Mac OS 9 or Mac OS X, that the print shop knows how to use FontSync, and that you remember to create the profile and submit it.

Info Scripts

These two scripts offer minor usefulness: **Current Date & Time** displays the current date and time in a dialog box, complete with a Clipboard button that copies the information, ready for pasting. **Font Sampler** was designed, in an era before Font Book (page 436), to show you what all your fonts look like.

Internet Services

These scripts are designed to show off the power of *XML-RPC* and *SOAP*, two Internet-query technologies that debuted in Mac OS X 10.1. For example, the **Stock Quote** and **Current Temperature by Zipcode** fetch those respective bits of information, popping them into a dialog box without having to use your Web browser, thanks to the power of SOAP (Simple Object Access Protocol).

Mail Scripts

This collection of scripts, expanded in Mac OS X 10.3, communicates with Mail (see Chapter 19). Some highlights:

- **Count Messages in All Mailboxes** counts all messages and unread messages in all of your mailboxes and displays the result.
- **Crazy Message Text** is Apple at its wackiest. When you run it, a dialog box asks you what message you want to send (“Happy Birthday,” for example). Mail then creates a colorful, zany, outgoing formatted message in which each letter has a random typeface, style, color, and size. It’s ideal for making people think you spent a long time with your Format menu for their entertainment.
- **Display All Accounts And Preferences** prepares a new mail message that includes information about every account you have configured and the preference settings you have set in Mail, all ready to send to some troubleshooting expert who needs the details in order to help you.
- **Import Addresses** grabs names from the address book from Entourage, Eudora, Outlook Express, or Palm Desktop. It’s not actually a Mail script—it brings the addresses into Mac OS X’s Address Book program—but it’s still handy if you’ve decided to switch from one of those programs to Apple’s Mail program.
- **Quick Mail** prompts you for an address and a subject line, launches the Mail application, and sets up a new message for you with those attributes. With a little analysis of this script, you should be able to see how it could save you time in generating canned, regularly scheduled outgoing mail messages. (A clean installation of Panther doesn’t include this one, but you may have it if you upgraded from Mac OS X 10.2.)

The **Mail Scripts**→**Rule Action** folder contains three scripts, two of which you’re supposed to trigger from inside Mail, using a *message rule* (page 593):

- **Help With Rule Actions** brings up the appropriate help screen so you can read about how to use Rule Actions.
- **Open Apple Website** brings up the Apple Web site. It’s intended as an example of how you can use Mail’s message rules to trigger AppleScripts automatically when certain conditions are met.
- **Sample Action Rule Script** is also intended as an example; it doesn’t do anything but display a dialog box indicating the name of the rule that summoned it, along with the subject of the message.

The **Mail Scripts**→**Scripts Menu** folder contains several scripts that illustrate how Mail's own Scripts menu can trigger AppleScripts. For example:

- **Create New Mailing List Mailbox**___ctl-m searches selected email messages for those it thinks came from a mailing list. If it strikes gold, it creates a new mailbox, stashes the mailing-list messages in it, and creates a rule that puts future messages from that list into the new mailbox automatically.

Tip: That “___ Control-m” business is supposed to illustrate how you can create keyboard shortcuts for your AppleScripts (at least those in the Script menu). At the end of the script's name, you just type three underline characters, followed by the key combination you want—Control-M, in this case. (Note that this keystroke works only in Mail, and refers to Mail's own private Script menu.)

- **Get Source of Selected Message** copies the raw message contents of the highlighted message into a new email message. Here, *source* means the raw, underlying code of the message: invisible headers, the message body, HTML formatting code, any attached files (in encoded form), and so on.
- **Remove Messages From Sender or Thread**___ctl-r. Mail maintains a blacklist of people you'd rather not hear from, and a list of *threads* (subject lines from an ongoing discussion) whose messages you no longer want to read. This script can assist you in updating that list.

Suppose, for example, that you select a message from *steveapple.com* with the subject line, “New Product.” When you choose this command, Mail displays a set of dialog boxes for creating a message rule that (a) automatically deletes all email from *steveapple.com* or (b) automatically deletes all email in the same thread (having either “New Product” or “Re: New Product” as its subject).

- **Speak Sender and Subject** makes the Mac say out loud the number of messages that are currently selected (if more than one), and then announce the sender's name and subject of each.
- **Summarize Message** creates a short summary of the selected message—and then reads it out loud. (Great for long-winded correspondents.)

Navigation Scripts

Four of these scripts—**New Applications Window**, **New Home Window**, and so on—all open the corresponding windows, just as though you'd used the Finder's Go menu to choose their names. (As for the word “New”—it's a red herring. These scripts do not, in fact, open a new window if one is already open.)

The **Open Special Folder** script presents a list of Mac OS X's special folders: Applications, Favorites, Movies, Sites, Utilities, and so on. Double-click the one you want to open—a straight shot to some of Mac OS X's most important folders.

Tip: If you're game to edit this script in Script Editor, you can modify it to let you choose and open more than one folder simultaneously (by **⌘**-clicking them, for example). Just type *multiple selections allowed true* right after the text *Choose folder to open:* (which appears at the end of a line about a third of the way down the script). Save your changes.

Script Editor Scripts

As the **About these scripts** command tells you, these 48 canned scripts can help you write faster and more accurate scripts, because the code chunks are free of typos and syntax errors. As you progress, you can add your *own* code-building scripts, customized for the kind of scripts you like to build, to make you even more productive.

In any case, to insert one of these code chunks into a script you're editing in Panther's Script Editor 2, just Control-click your fledgling script. When you choose from the contextual menu, the code block magically appears in the Script Editor window, right where you had clicked. (Alternatively, choose the chunk's name from the Script menu; the code winds up on your Clipboard, ready to paste.)

Sherlock Scripts

The sole script here, **Search Internet**, prompts you for a *search string* (the words you want to search for on the Web). When you click Search, the script opens Sherlock and proceeds to execute your search. All of this saves you a few mouse clicks, but this script was likely designed to serve primarily as an example for study by scripting hopefuls.

UI Element Scripts

Much of the time, scripts perform their magic quietly in the background, out of sight. But if you're trying to automate a program that doesn't respond to the usual AppleScript commands, your scripts can now "operate" them manually by choosing menu commands, clicking buttons, and so on.

Note: This feature, called UI (user-interface) scripting, doesn't work until you first open the Universal Access panel of System Preferences and make sure that "Enable access for assistive devices" is turned on.

You wouldn't want to run the scripts in the UI Element Scripts folder just as they are; they're simply samples that show you the correct syntax.

Here, for example, is a sample script that illustrates how you might use UI scripting. This script brings the frontmost TextEdit document to the front, highlights all of its contents, and then reads it out loud.

```
tell application "System Events"
  get properties
  tell process "TextEdit"
    set frontmost to true
    delay 1
    keystroke "a" using {command down}
```

```

click menu item "Start Speaking Text" of menu 1 of menu
item "Speech" of menu 1 of menu item "Services" of menu
"TextEdit" of menu bar item "TextEdit" of menu bar 1 of ap-
plication process "TextEdit" of application "System Events"
end tell
end tell

```

See how the text is selected by “pressing” ⌘-A, and the speech is triggered by “choosing” the TextEdit→Speech→Start Speaking Text menu command? (Of course, TextEdit *is* scriptable, and you could perform the same stunt using a much shorter script without using the UI scripts—but you get the point.)

Here are the demonstration scripts that you’ll find in the UI Element Scripts folder:

- **Get User Name.applescript** opens the System Preferences→Accounts→Password tab and extracts your name from the name field. (To try this script out, open it in Script Editor and run it. You’ll see System Preferences open, and in a moment, your account name will appear in Script Editor’s Result window.)
- **Key Down-Up.applescript**. Using TextEdit, this script demonstrates a number of ways to get AppleScript to do some typing for you. Use the examples in this script as a template for your own, more useful scripts.
- **Probe Menu Bar.applescript** and **Probe Window.applescript** are valuable demo scripts for “probing” menus and window elements (buttons, tabs, text boxes, and so on), in an effort to learn the correct syntax for making UI scripting work. For example, open the Menu Bar script and run it in Script Editor, and then observe the long list in the Result window. It reveals the correct ways to refer to all active (that is, not dimmed) Finder menu commands. (Feel free to change the word “Finder” in the script to another open program’s name.)
- **Set Network Location.applescript** changes your Network Location settings to Automatic—not an especially useful script on its own, but a useful demonstration of how you can trigger commands in your  menu. By modifying this script, you could, for example, make it change the location of the Dock, turn its magnification off, and so on.
- **Set Output Volume.applescript**. This script shows you the correct syntax for adjusting your Mac’s speaker volume. You might incorporate it, for example, into a larger script that reads the mail, launches iTunes, and turns up the volume.

URLs

All of the scripts in this folder simply open your browser, connect to the Internet if necessary, and then open the specified Web pages (the stock quote for Apple on Yahoo, the Apple Store, CNN, and so on). **Download Weather Map** is much cooler; in a flash, it downloads the current U.S. weather map image and then opens the file in the Preview program for viewing.

The AppleScript Related Sites scripts in this folder open the most popular and useful AppleScript sites: Bill Cheeseman’s AppleScript Sourcebook, MacScripter.net, and

Apple's own AppleScript Web site. Each offers a wealth of information and links to even more great AppleScript sites.

Creating Your Own AppleScripts

If you ask a crowd of Mac users how many of them write AppleScripts, few hands are likely to go up. That's too bad, because as programming languages go, AppleScript is easy to understand. It takes only a few weeks, not years, to become comfortable with AppleScript. And the power AppleScript places in your hands is well worth the effort you'll expend learning it.

For example, here's a fragment of actual AppleScript code:

```
open folder "AppleScript" of folder "Applications" of startup
disk
```

You probably don't need a manual to tell you what this line from an AppleScript program does. It opens the Applications→AppleScript folder on your hard drive. (That's the folder that contains Script Editor, the Mac OS X program that lets you write your own AppleScripts.)

If you have no interest in learning to program, you're not alone. But almost every Mac fan can benefit by understanding what AppleScript can do, why it's important in certain industries, and how it may be useful in everyday situations.

Recording Scripts in "Watch Me" Mode

You can, if you wish, create a script by typing out the computer commands one at a time, just as computer programmers do the world over. Details on this process come later in this chapter.

But if the task you want it to handle isn't especially complex, you'll be pleased to discover that in Mac OS X 10.3—surprise!—the Finder itself is once again *recordable*, much as it was in the Mac OS 9 days. That is, you can create a script just by doing the job manually—using menu commands, opening windows, changing window

GEM IN THE ROUGH

Text to Audio Files

Your Mac can perform an astonishing feat: It can convert a text file into a *spoken digital recording* in AIFF audio format.

This is a twist with huge implications. If you transfer these files to your iPod, you can listen to your documents—email, Web pages, reports, eBooks, or anything else you can type or download—as you commute, work out, or work outside.

Of course, commuters and joggers have been listening to Books on Tape for years, and companies like Audible.com create what you might call Books on MP3. But those products are expensive, and you can't listen to your *own* stuff.

To pull this off, just download the two sample scripts at www.apple.com/applescript/macosx/text2audio.html.

views, and so on—as Script Editor watches and writes out the necessary lines of code automatically. Recording an action and watching the Mac turn your movements into lines of AppleScript code is a fantastic way to learn how AppleScript works.

A Simple Auto-Recorded Script

The script you'll build in this experiment creates a squeaky-clean new folder, into which you can stuff your newly created documents each day for backup. In the following section, you'll find a line-by-line analysis of the result.

1. Open Script Editor.

Script Editor is in your Applications→AppleScript folder. It looks like Figure 7-2, except that the window is empty when it first opens.

Figure 7-2:

The Script Editor in action. Type a short description into the bottom pane of the window, if you like. This script appears already formatted with colored keywords (which aren't evident in the grayscale illustration here) and indents because it was created using "watch me" mode. (If you don't have an empty window before you begin recording, choose File→New.)



2. Click Record.

Script Editor is about to write out the code that describes your every mouse movement, click, and menu command from now until you stop recording. In this experiment, you'll create a new folder, name it, open it, change the window to list view, resize it, and then move it to a convenient position on the screen.

3. Click the desktop. Choose File→New Folder. Type *Today's Backup* and then press Return.

You've just made a new folder on your desktop. If you sneak a peek at your Script Editor window, you'll see that it has begun to notate the computer commands that represent what you've done so far.

4. Choose File→Open (or press ⌘-O), and then choose View→as List (⌘-2).

You've just taught your script to open the folder into List view so you can see what's in it. Next, you'll resize the window to make it fit in a corner of your screen.

5. Drag the resize handle (the lower-right corner of the window) to make the window smaller.

Finally, you'll move this backup folder window off to a corner, so that you'll have more room to drag your documents into it.

6. Drag the Today's Backup folder window to a new position on the screen.

That's all this modest AppleScript will do: Create a new folder, name it, open it, change the view to list view, and then resize and reposition it.

7. Click in the Script Editor window, and then click Stop.

Your newly created script is complete, as shown in Figure 7-2.

Click your desktop and then throw away the Today's Backup folder, so that you can start fresh. Once again, return to Script Editor—but this time, click Run.

In rapid, ghost-driven sequence, you'll see AppleScript create *another* Today's Backup folder, open it, change it to List view, then resize it and drag it to precisely the spot on the screen where you dragged the first one by hand—all in a fraction of a second. The script you've created isn't the world's most useful, but it illustrates how powerful and fast AppleScript can be.

AppleScript Commands

Scripts you create in "watch me" mode are actually fairly limited. If you want to write scripts for programs that aren't recordable—or more complex scripts for the Finder—you'll eventually have to learn to type out AppleScript code for yourself.

As a first step in understanding the AppleScript language, study the code written by Script Editor in the previous example. As it turns out, much of this script consists of standard AppleScript jargon that appears in almost every script. Here's a line-by-line analysis of the little folder-creating script you just made.

```
tell application "Finder"
  activate
  make new folder at folder "Desktop" of folder "chris" -
  of folder "Users" of startup disk with properties
  {name:"untitled folder"}
  set name of folder "untitled folder" of folder "Desktop" -
  of folder "chris" of folder "Users" of startup disk to "To-
  day's Backup"
  make new Finder window to folder "Today's Backup" -
  of folder "Desktop" of folder "chris" of folder "Users" of
  startup disk
  set current view of Finder window 1 to list view
```

```

set bounds of Finder window 1 to {10, 93, 417, 482}
set position of Finder window 1 to {11, 453}
end tell

```

Note: The `—` character means, “This command continues on the next line. Treat it all as one giant clump.” (It appears on these limited-width pages for clarity, but in Script Editor, you don’t actually need them. If you don’t insert this character manually [by pressing Option-Return], Script Editor wraps lines automatically.)

tell application “Finder”

Almost every script begins with a line like this. It specifies which Mac program or window this section of your script will control. If you were writing a script manually, you’d begin by typing this line.

Complex scripts can involve *several* programs—grabbing information from FileMaker and pasting it into QuarkXPress, for example. In those longer scripts, you’d see *tell application “FileMaker Pro”* at the beginning of the steps that involve the first program, and then *tell application “QuarkXPress”* later in the script, where the steps pertain to QuarkXPress.

activate

This command means, “Bring the abovementioned program to the front.” Technically, you don’t have to make the Finder the active program to perform the simple folder-creation steps in this script—AppleScript could create and manipulate your Today’s Backup folder even with the Finder in the background. But Script Editor inserted this step automatically because, when recording your actions, *you* made the Finder the active program (by clicking the desktop). You’re welcome to delete this step from your Script Editor window.

This is only the first of many examples you’ll find in which Script Editor records wordier scripts than necessary. Hand-typed scripts, like those described later in this chapter, are typically more compact and faster to run.

make new folder at folder “Desktop” of folder “chris” of folder “Users” of startup disk with properties {name:“untitled folder”}

This command tells the Finder to create a new folder on your desktop. (Why does it say “of folder chris”? Because remember that in Mac OS X, every account holder has a separate desktop.)

If you’re writing out scripts by hand, it’s worth knowing that you don’t really need a command this verbose. The line *make new folder at desktop* would do just as well.

set name of folder “untitled folder” of folder “Desktop” of folder “chris” of folder “Users” of startup disk to “Today’s Backup”

This step renames the new folder. (You could also type *set name of folder “untitled folder” to “Today’s Backup”* for a more concise version.)

AppleScript processes the expression in parentheses first, just as in mathematics, and uses its result to evaluate that part outside of the parentheses. Either way, the result is a renamed folder.

make new Finder window to folder “Today’s Backup” of folder “Desktop” of folder “chris” of folder “Users” of startup disk

This step opens the new folder to a Finder window. (Talk about wordy! The hand-scrip-ter’s more elegant alternative: *open folder “untitled folder”*.)

set current view of Finder window 1 to list view

The foremost window is always number 1 in the Finder’s eyes. This command changes its view to list view.

set bounds of Finder window 1 to {10, 93, 417, 482}

Here’s how AppleScript thinks when it resizes your Finder window. The first two numbers in the list represent the vertical and horizontal coordinates of the window’s upper-left corner, as measured in pixels from the upper-left corner of your screen. The second pair of numbers represents the position of the lower-*right* corner of the Finder window.

set position of Finder window 1 to {11, 453}

This script step moves your window to the specified position, once again measured in pixels from the upper-left corner of the screen to the upper-left corner of the window.

end tell

You’re finally telling the Script Editor that it can stop paying attention to Finder. *End tell* always accompanies the *tell application* command that begins a script. These

FREQUENTLY ASKED QUESTION

Colorful Words in Scripts

Why does Script Editor put some words in color and some in black?

Imagine that your word processor could color-code the words you type, according to their function in the sentence, or their part of speech. If you were asked to analyze the sentences for grammatical correctness, having the color code would be a big help; you could almost see the sentence structure.

AppleScript has rules of syntax as well. Script Editor checks your syntax for you, and displays the words you type in different colors according to whether it thinks that they’re

language keywords, application keywords, variables, operators, and so on. The color coding makes a script much easier to read. It can also sometimes help you find typos.

In Panther, AppleScript has a new standard color scheme. It doesn’t color-code everything, but it’s better than the all-black coloring approach of the old AppleScript.

To adjust the color settings, choose Script Editor→Preferences and click the Formatting icon. Double-click a color swatch to change it, or a “part of speech” name to change its font.

two commands form the bookends that delineate the instructions to the program in question. (AppleScripters call the entire chunk, beginning with *tell* and ending with *end tell*, a *tell block*.)

Tip: You can change the fonts and formatting Script Editor uses to write out your scripts. Choose Script Editor→Preferences, click the Formatting toolbar button, and then use Script Editor’s Font panel to choose new type specs. Experienced scripters can even change the color-coding of their scripts to make them easier to read and debug.

Saving a Script

Before you save a script, let Script Editor check its *syntax* by clicking the Compile button. (If you don’t perform a syntax check manually before trying to save, Script Editor will automatically do it for you as soon as you try to save.)

Note: The Compile button won’t find any errors if you created a script using the “watch me” system. After all, Script Editor itself wrote the script, so of *course* it’s perfect. But when you write scripts by hand, as described later in this chapter, you’ll find the Compile button a useful tool for cleaning stray errors out of your scripts.

If Script Editor finds your script to be correctly written, you’ll get no reaction from Script Editor except to see your script formatted with colors and fonts, as shown in Figure 7-3. If it does find a problem with the syntax, you’ll be limited to saving your script file in only one format: plain text.

Figure 7-3:

If you type an AppleScript manually, it appears just as it would in a word processor (top left): in Courier with no special formatting. When you click the Check Syntax button, however, Script Editor indents the tell block and changes all type to Verdana (bottom right), with AppleScript’s reserved words (language keywords and application keywords) in colored type and comments in smaller, italic type. Congratulations: You’ve got the Script Editor seal of approval.



At this point, you’re ready to save your script. Choose File→Save. Name your script and choose a location for it, by all means—but the important step here is to choose a *format* for your completed script (see Figure 7-4). Your choices depend on whether

you want your scripts to work in Mac OS X, Mac OS 9, or both. They also depend on how you intend to use your scripts.

- **Script.** Saving as a Script really means as a *compiled* script. You can't double-click a compiled script to run it. Nonetheless, compiled scripts are required by a number of *programs* that launch AppleScripts, including your Mac's Script menu (see Figure 7-1) and any programs that have their own Script menus. These so-called *attachable* programs include FileMaker Pro, Entourage, and Eudora, among others. Compiled scripts can run in both Mac OS X and Mac OS 9.

When you choose this option, you're offered an additional checkbox. If you choose Run Only, neither you nor anyone else can open your script for editing later. You may want to use Run Only scripts for distribution to people who shouldn't be modifying your scripts, or if you are being secretive about your code.

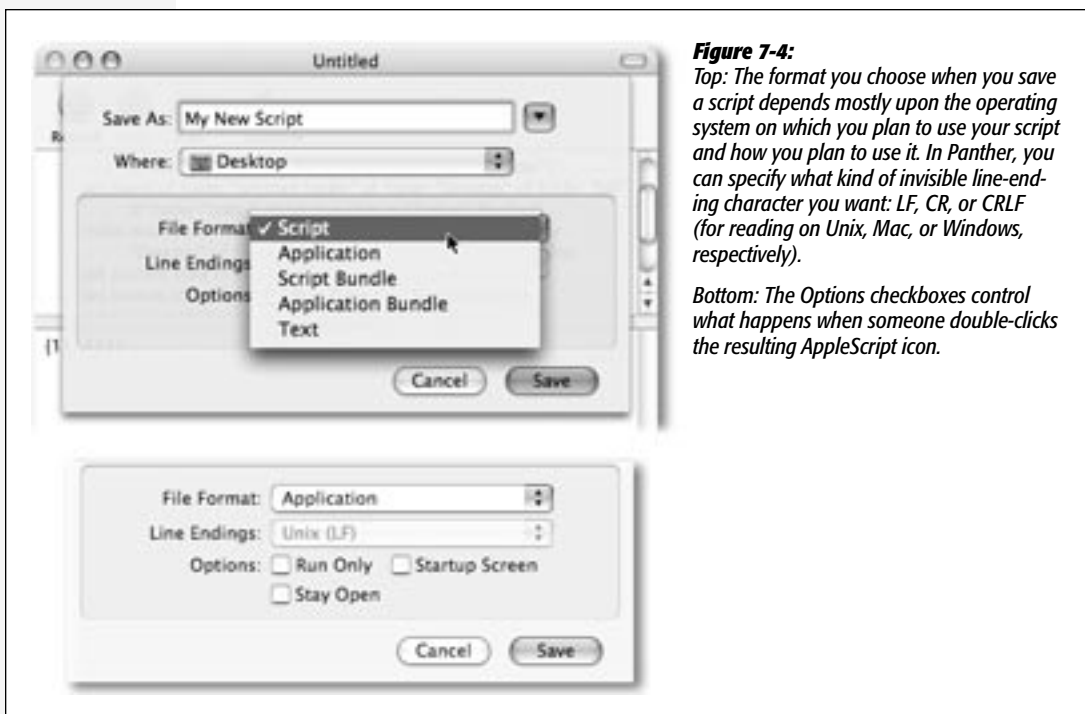


Figure 7-4:

Top: The format you choose when you save a script depends mostly upon the operating system on which you plan to use your script and how you plan to use it. In Panther, you can specify what kind of invisible line-ending character you want: LF, CR, or CRLF (for reading on Unix, Mac, or Windows, respectively).

Bottom: The Options checkboxes control what happens when someone double-clicks the resulting AppleScript icon.

- **Application.** Choose this option if you want to create a standalone, double-clickable application script suitable for use in Mac OS X or OS 9. You can put the resulting script into the Dock, for example, so that you can trigger it whenever you like. Or you can put the finished script into, say, your Startup Items list to run automatically at startup (to search the Web for news of your industry and save the Web pages in a new folder, for example).

When you choose this option, you're offered three additional checkboxes (Figure 7-4, bottom). **Run Only** works just as it does for a compiled Script, as explained

on the previous page. If you choose **Stay Open**, the applet you've created remains running after it's launched—just what you'd want for scripts that are meant to monitor some activity on your computer continuously.

In general, you'll want to leave **Startup Screen** turned off, so the startup message never appears. If you turn it on, however, then whatever description you provided for your script appears whenever the script runs. You, or whoever is using your script, must then click either Run or Quit after the script launches. This can become tiresome very quickly.

- **Script Bundle.** Choose this option if your script relies on external files like a video clip, a graphic, or even other scripts. Script Editor will bundle them together with your script into a single icon.
- **Application Bundle.** Like the Application option, this format creates a double-clickable, self-running script application—but one that can run only on Mac OS X 10.3 and later. Here again, the point is to store resources used by the script in a single icon (a package), making it possible to deploy more complicated scripts without worrying that some supporting files might go astray.
- **Text.** You can't actually run an AppleScript that's saved as a text file, but saving it this way provides a good way to exchange scripts with other people or to save an unfinished script you want to work on later.

Writing Commands by Hand

Using the “watch me” mode, you can create only very simple scripts. If you want to create anything more elaborate, you must type out the script steps one by one, testing your work, debugging it, reworking it, and so on.

Scriptable Programs

Most introductory articles about AppleScript discuss scripts that perform useful tasks *in the Finder*—that is, scripts that manipulate your files, folders, disks, and so on. That's because AppleScript can control almost every element of the Finder, making it an extraordinarily *scriptable* program.

But AppleScript can also control and communicate with almost every popular Mac program: FileMaker, AppleWorks, Adobe and Microsoft applications, and so on. Sherlock, iTunes, iPhoto, iMovie, QuickTime Player, Terminal, TextEdit, Mail, Internet Connect, and Image Capture are among the built-in Mac OS X programs that you can control with AppleScripts, and the list is always growing.

Before you can write a script that manipulates, say, FileMaker, you need to learn which commands FileMaker can understand. Most Mac programs understand at least the most basic four AppleScript commands—Open, Print, Quit, and Run—but some offer a much larger AppleScript vocabulary. To find out, you need to look at the application's *dictionary*—the list of AppleScript commands it understands.

To do so, open Script Editor; choose File→Open Dictionary. You're offered a list of all scriptable applications on your Mac. You can jump to the program you want by typing the first few letters of its name (a new Panther feature). Once it's highlighted, press Enter, or double-click the name of the program you want to check out (see Figure 7-5). If the application whose dictionary you seek is not in the list, click the Browse button and hunt for it.

Tip: You can also open a dictionary by dragging a program's icon onto the Script Editor icon—a good argument for parking Script Editor in the Dock.

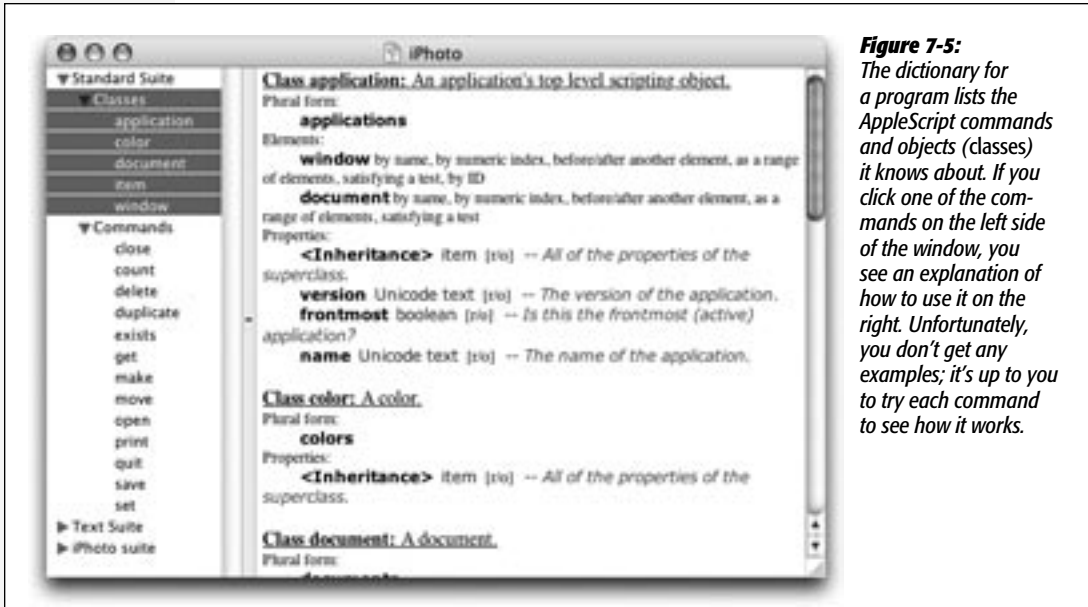


Figure 7-5: The dictionary for a program lists the AppleScript commands and objects (classes) it knows about. If you click one of the commands on the left side of the window, you see an explanation of how to use it on the right. Unfortunately, you don't get any examples; it's up to you to try each command to see how it works.

That's not to say, of course, that these commands make much sense to someone who's never written an AppleScript before. These commands, and the scripts that incorporate them, still require study and experimentation. But a glance at a program's AppleScript dictionary is a good way to assess its scriptability—and therefore how much the software company has embraced the Macintosh Way.

Two Sample Scripts

In the following hand-typed examples, you'll encounter new kinds of *tell blocks*, scripts that control more than one program at a time, and scripts that do things that you can't even do manually. If you look over these examples carefully—and type them up for yourself in Script Editor—you'll begin to see how similar to English AppleScript can be. You may also wind up with some useful scripts that can make your Macintosh life easier.

Example 1: Auto-backup before shutting down

Suppose you'd like to use your new iPod as a backup disk. You want to be able to back up your Home→Documents folder each day without having to burrow through folders, drag and drop, click OK, and so on. AppleScript can do the job automatically.

In Script Editor, choose File→New Script, and then type this:

```
tell application "Finder"
duplicate folder "Documents" of folder "chris" of folder "Users"
of startup disk to disk "Ye Olde iPod" with replacing
end tell
```

Here's how the key command breaks down:

- *duplicate*. AppleScript also offers commands called *move* and *copy*. So why *duplicate*?

Use *move* when you want to “drag” an icon around on the same disk, changing its location; use *duplicate* when you want to leave a copy behind. (In this example, it doesn't matter which command you choose. “Dragging” an icon to a different disk *always* makes a copy and *never* moves the original.)

Copy, to AppleScript, means “copy to the Clipboard,” which the Finder can't do by AppleScript control. (The Finder's dictionary lists the copy command as “not available yet.”)

As for the phrase *with replacing*: You'll probably want to run this auto-backup script every day, so you'll want it to wipe out yesterday's Documents backup each time you run it. That's what *with replacing* accomplishes.

- *folder “Documents” of folder “chris” of folder “Users” of startup disk*. All of this business simply reflects the location of your Documents folder. It's in your Home folder, of course, which bears your name. In this example, substitute your actual Home folder's name for “chris.”

After you've run your script a couple times from within Script Editor (by clicking the Run button) and enjoyed the mad, brain-shocking power, add a little finesse by adding the new commands shown here in italics:

```
tell application "Finder"
duplicate folder "Documents" of folder "chris" of folder "Users"
of startup disk to disk "Ye Olde iPod" with replacing
activate me
beep
display dialog "Backup Successful!" buttons "Cool!"
end tell
```

The additional commands make the Mac beep and report, in a little dialog box, “Backup Successful!” after it makes the copy. It even offers a button (“Cool!”) that you can click to dismiss the dialog box. (Of course, you can make the button say anything you like—“OK,” “Get on with it,” “Good boy!” or whatever.)

Once you've created this little script, save it as an application and put it in the Dock, on your Finder toolbar, or in your Sidebar. Or save it as a compiled script in your Home folder→Library→Scripts folder, so that you'll be able to trigger your backup from the Script menu.

As you get better at AppleScript, you'll realize that this script is only the beginning. You could, for example, add a date stamp in the folder name, label the folder, make it always keep the most recent backup (a *rolling backup* strategy), back the files up to your iDisk, "eject" the iPod disk after the backup is complete, and even speak out loud, "All done, O master!" after the copying is complete.

Example 2: Universal Shutdown

Every now and then, you might find it useful to quit all running programs except the Finder. Unfortunately, there's no one-step command that quits all of your open programs—but you can create one for yourself.

Nearly every Mac program on earth understands the Quit command when sent by AppleScript. All you have to do, then, is to send that command to each of the programs you're likely to have open. If you spend most of your time in AppleWorks, Mail, and Safari, for example, your script might look like this:

```
tell application "AppleWorks"
quit
end tell
tell application "Mail"
quit
end tell
tell application "Safari"
quit
end tell
```

In time, after hanging out on AppleScript mailing lists and studying a few other scripts, you'd realize that you could shorten those tell blocks like this:

```
tell application "AppleWorks" to quit
tell application "Mail" to quit
tell application "Safari" to quit
```

Even so, this script is doomed. If some of these programs aren't actually running, the script will dutifully *launch* each application and *then* quit it, which won't exactly make you the envy of your household. Fortunately, there's a better way.

What you really want is a script that briskly exits every running program, but is smart enough *not* to disturb the important system processes that run in the background. (Note that *process*, in computer-speak, refers to any running program, including both the applications you're using and the secret background ones that the Mac runs all the time.) So you'd want this:

```
tell application "Finder"
    set quitList to the name of every application process whose
```

```

file type is "APPL"
end tell
repeat with i from 1 to count of quitList
    tell application (item i of quitList) to quit
end repeat

```

Here's how some of these unfamiliar lines break down:

- *set quitList to the name of every application process whose file type is "APPL."* You're defining a new variable, a stunt double, called *quitList*. It's a list of every program whose file type (page 133) is APPL—that is, every double-clickable program. By limiting it to processes whose file type is APPL, you avoid quitting most of Mac OS X's important background programs.
- *repeat with i from 1 to count of quitList.* This is a loop. You're saying, "run through the quitList list, exiting each program until you're at the end of the list."

Unfortunately, if you actually try this script (hint: save it first), you'll discover that it's the neutron bomb of scripts: It quits *everything in sight*, including the Dock, invisible graphic-interface programs like SystemUIServer, and even Script Editor itself! Oops.

By adding a few more lines, however, you can specify the names of processes that you want untouched:

```

set excludedOnes to {"Script Editor", "loginwindow", "Dock",
"SystemUIServer", "System Events", "ClassicAuxInput"}
tell application "Finder"
    set quitList to the name of every application process whose
file type is "APPL"
end tell
repeat with i from 1 to count of quitList
set aProcess to item i of quitList
if aProcess is not in excludedOnes then
    tell application aProcess to quit
end if
end repeat

```

This script gives you an even greater degree of control, because you can add to the list of processes that aren't to be disturbed. Just add them to the bracketed list in the first line, always in quotes and followed by a comma. Now you're free to quit all of your running applications—even applications running in Classic—with one click of the mouse.

Folder Actions

A *folder action* is a script that the Mac triggers automatically whenever you do something specific to a particular folder: open it, close it, move it, resize it, or change its contents.

Attaching and Removing Folder Actions

If you've never before used folder actions, first run the Enable Folder Actions script from the Script menu, which turns on the feature. (You can also turn on Enable Folder Actions in the new Folder Actions Setup program, which is in your Applications→AppleScript folder.)

Then, to attach a script to a folder, see Figure 7-6.

Tip: A folder shows no indication that an action is attached. You can, however, add a little indicator badge to it manually. The badged icon has vanished from Apple's AppleScript site, but you can still download it from the "Missing CD" page at www.missingmanuals.com.

Keep your eye on www.apple.com/applescript/folderactions, too, for updated versions of the canned folder action scripts that come with your system, plus extra sample script.



Figure 7-6:

To attach a script to a folder, Control-click the folder's icon. From the contextual menu, choose Attach a Folder Action. Then, in the dialog box, select the folder action script you want to run whenever somebody interacts with the specified folder. (It's perfectly OK to attach more than one script to a folder.)

To remove a Folder Action script from its folder, use any of these approaches:

- Control-click the closed folder icon (or, if its window is open, Control-click inside the window and choose Remove a Folder Action from the contextual menu). From the submenu, choose the script to remove.
- In the Folder Actions Setup panel, click the name of the folder in the left-side list; in the right pane, click the script you want to remove and then click the – button beneath it.
- From the Script menu, choose Folder Actions→Remove Folder Actions. You're shown a list of folders that have folder actions attached; select the one you want to cleanse. Highlight the name of the script you want to remove, and click OK.

What They're Good For

For a simple example of what folder actions can do, here's one that notifies you whenever somebody has put new files into a particular folder. Type this text into Script Editor, save it to your hard drive, and then attach it to a folder as described on the previous page:

```
on adding folder items to this_folder
    tell application "Finder"
        set the folder_name to the name of this_folder
        display dialog "Someone has put new files into ↵
the folder called " & the folder_name giving up after 30
        end tell
    end adding folder items to
```

This script incorporates several useful AppleScript techniques you haven't yet seen in this chapter. For example:

- **on adding folder items to.** This phrase tells the Mac that the script should run when an icon is added to the folder.
- **this_folder.** This term isn't a standard AppleScript term; it's a *variable*. You can substitute almost any word you like (it doesn't have to be *this_folder*). It's no secret to the Mac which folder "this folder" refers to, because you *told* it which folder when you attached the folder action script.
- **folder_name.** This term is another variable. In order not to have to type *the name of this_folder* over and over again, the skilled scripiter would use a new variable "folder_name" to hold the value that longer phrase represents—that is, the folder's name.
- **&.** The & is the *concatenation operator*. It allows you to splice together, or *concatenate*, separate text chunks. When you're composing the message that you want to appear in a dialog box (as indicated by the "display dialog" command above), you can string together text you've written between quotes and text in any variables in your script. In this example, the fourth line of the script creates a dialog box that reads, "Someone has put new files into the folder called Fish Heads" (or whatever the folder is called).

After you've attached this script to a folder, nothing happens until you drop a new file into the folder, at which time a message appears, saying, "Someone has put new files into the folder called 'Fish Heads.'" This way, you can keep tabs on what your network buddies are submitting while you work on other things. (And if you don't click OK, the dialog box goes away by itself after 30 seconds.)

Tip: You can find an assortment of ready-made folder action scripts in your Library→Scripts→Folder Action Scripts folder. If you have a copy of Mac OS 9 on your Mac, a second set awaits in the System Folder→Scripts→Folder Action Scripts folder. Some don't work in Mac OS X, but they're a very good starting point for experimentation.

Keep an eye on Apple's AppleScript Web site for any updates that may include new folder action scripts. Useful things appear there frequently.

Advanced AppleScript

No single chapter—in fact, no entire book—can make you a master AppleScript programmer. Gaining that kind of skill requires weeks of experimentation and study, during which you'll gain a lot of appreciation for what full-time software programmers endure every day.

By far the best way to learn AppleScript is to study existing scripts (like those in the Library→Scripts folder) and to take the free online training courses listed at the end of this chapter. And there are thousands of examples available all over the Web. Trying to figure out these scripts—running them after making small changes here and there, and emailing the authors when you get stuck—is one of the best ways to understand AppleScript.

On the “Missing CD” page at www.missingmanuals.com, for example, you'll find a document called *Advanced AppleScript.pdf*. It introduces several more complex AppleScript concepts, including variables, loops, nested “if” statements, and so on.

When you get really serious about creating AppleScripts, you'll also want to check out AppleScript Studio (ASS for short—how did that one get past Marketing?). Technically, it's an integrated development environment that combines Project Builder and Interface Builder, making AppleScript a peer language of Java and Objective C. In plain language, ASS lets you put a real Aqua user interface on your scripts, complete with dialog boxes, text boxes, buttons, slide controls, and much more. It also lets you combine multiple scripts into a single program.

AppleScript Studio is free. It's part of the Developer Tools kit described on page 329.

Where to Learn More

Once you've exhausted the built-in AppleScript help screens (or *become* exhausted reading them), begin your quest for more information at Apple's AppleScript Web site, www.apple.com/AppleScript. There you'll find the link to an excellent, step-by-step tutorial in hand-coding scripts, as well as links to these outstanding online AppleScript guides:

- **Apple's own Web site.** The place to start your AppleScript quest is at the mother ship—Apple's own AppleScript web site (www.apple.com/applescript/). There you'll find lots of up-to-date information emanating from Apple, and links to all of the significant AppleScript sites on the Web.

Tip: At www.apple.com/applescript/guidebook/sbrt/index.html, you'll find on-line AppleScript Guidebook modules with a lot of useful routines that you can use, borrow from, or dissect for the purposes of learning.

- **MacScripter.net.** This site is a beehive of AppleScript activity (*www.macscripter.net*), run by a group of AppleScript enthusiasts. It hosts the latest in AppleScript news from all quarters, links to everything AppleScript under the sun, free scripts for downloading, discussion boards, and much, much more.
- **AppleScriptSourcebook.com.** Run by Bill Cheeseman, one of AppleScript's long-time enthusiasts, this site contains a wealth of technical details about every release of AppleScript, including an expansion on many of the terse Apple release notes. There's a lot here for scripters of all levels, so keep it in your browser's AppleScript bookmarks.

Each of those three Web sites offer links to commercial AppleScript training course offerings, technical encyclopedias that describe every AppleScript command in detail, AppleScript news sites, scripting additions, freeware scripts, code samples, multimedia training magazines, and so on.

Other resources:

- **AppleScript mailing lists.** Sign up for one of these free, email-based discussion lists whose members are all AppleScript fans. Apple runs the AppleScript Users' list; the MacScript list is independent. Given the rapidly changing face of AppleScript in Mac OS X, these lists are possibly the best source of up-to-date solutions for the new scripter. (Sign up at <http://lists.apple.com/mailman/listinfo>.)
- **AppleScript books.** If you want to buy a book about AppleScript, be sure that it's been updated for Mac OS X 10.3—like the latest edition of *AppleScript: The Definitive Guide*.

POWER USERS' CLINIC

Scripting Additions

Much of AppleScript's power comes in the form of add-on files called *scripting additions*. You can think of them as plug-ins, each of which adds a particular new feature to AppleScript's repertoire.

In Mac OS X, you may find scripting additions in any of three different places. The standard additions are in the System→Library→ScriptingAdditions folder. As with fonts, sounds, or other settings, you can also install scripting additions that only *you* can access by making a Scripting Additions folder in your Home→Library folder. Similarly, if you're an administrator, you can make a Scripting Additions folder in the main Library folder (in the hard drive window); everyone

with an account will be able to use the scripting additions you place there.

Like a scriptable application, each of these scripting additions has its own *dictionary*—its own bunch of specialized AppleScript commands that you can use in your scripts. You view these new commands just as you would when studying the vocabulary of a program: by opening Script Editor, choosing File→Open Dictionary, and then navigating to and opening the scripting addition you want.

As more power is built into AppleScript itself, there's less demand for the functions provided by scripting additions. In the meantime, now you know where to find them.

- **Bill Briggs' AppleScript Primers.** These articulate, thoughtful beginners' tutorials were written in the days of Mac OS 9, though most of the code will run in Mac OS X. Still the largest unified collection on the Web (www.maccentral.com/columns/briggs.shtml).