Automating with AppleScript

In This Chapter

Meet AppleScript • Using Scripts
Creating Your Own Scripts • Learning AppleScript: Resources • Scriptus Annotatus

One thing sets the Macintosh Power Users apart from the rest: their use of AppleScript. (And Unix. But AppleScript is a lot easier to use and a lot more practical and a lot more fun.)

Oh, and Macintosh-themed body modifications as well. I have a friend who carves the Apple logo into his hair for every Macworld Expo, and I can also testify that there are more Apple and Mac OS-themed tattoos to be found at that show than liquor-themed tattoos at a *Jerry Springer* taping. But again, AppleScript is a lot more practical than a tattoo or a piercing, it'll disappoint your parents a whole lot less, and it's a lot more fun than having needles pierce your skin hundreds of time a second.

The difference is people who gets things done in minutes, and people who get things done in hours; people whose Macs are naturally efficient and organized, and people with files scattered all over the place; people whose Macs do things that border on the sorcerous, and people whose Macs do more or less no more than what they did when they were first taken out of the box.

AppleScript — Mac OS X's built-in system-wide resource for automating routine processes and writing simple software — is the difference between *a* Mac and *your* Mac. AppleScript helps to build strong bodies nine different ways. If it were a person, it'd return its library books on time and donate blood regularly.

I am, as you may guess, a rather enthusiastic evangelist of AppleScript. I only have your interests at heart, though. When you learn AppleScript, you take your first step into a larger and more exciting world.

MEET APPLESCRIPT

The term "writing software" is instantly intimidating to any sensible user. You didn't lay out two grand to have the wonderful opportunity to spend weeks building your own apps. That's why you support the (sometimes) fine work of the Microsoft Corporation, after all.

Still, there are plenty of things you do with your Mac that involve just repeating a simple task over and over and over again. That's fine when you're working for The Man and you get paid for a full day whether you actually think or not. But when the goal is to finish a task as quickly and efficiently as possible, you wish there were a way to harness your computer's endless capacity to shut up and do what its told, no matter how dull.

NIKE DIPLOMACY

If you don't want to deface George and Abraham, at least slap some Sal stickers on a few lockers or something. If you walk into a typical Apple product manager's office and ask how influential Sal is in ensuring that AppleScript remains an important and critical part of the Macintosh experience, the manager will silently clear a few papers from the desk and point to a pattern of deep smudges in the wood. You see, there once was a time when this manager's product supported AppleScript in only the most basic, lame-o way, and this surface was pristine and unmarred. Then Sal burst in and kept jumping up and down on the person's desk until they agreed to improve things.

I'll give you a real-life example, torn from the pages of history itself. In the furious final weeks of producing this very book, it was discovered that each of its hundreds of illustrations had been named improperly. The six-digit code that began each filename was the *wrong* six-digit code. Do you have *any* idea how long it takes to rename hundreds of files by hand?

Well, neither do I. I wrote an AppleScript that told the Finder to process each file and change each filename individually. Days and days of tedious effort instead became an hour of watching *The Shield* on my TiVO downstairs, and then wondering if the script had finished its work, and then coming back up and discovering that it had finished the task before the second commercial break.

But AppleScript isn't a standalone app or a utility. It's a fundamental part of Panther's architecture, just as intimate as the mechanism that prints files or draws windows and menus. It's a superhighway that allows every piece of software running on your Mac — including the OS itself — to interact with each other and work together. If AppleScript causes the Mail app to check for new mail, it doesn't do anything so unsophisticated as send a mouse click to the Mailbox menu's Get New Mail item. It actually communicates with the code lurking *inside* Mail.

Note

So AppleScript is like the general contractor on a big home-remodel project. It can do things on its own without having to control other applications at all, but in everyday use its typical function is to hand tasks off to specialists, make sure they have what they need to get the job done, and make sure that all these individual tasks are done in the specified sequence without any errors.

What makes AppleScript so gosh-darned super?

I would like to think that at this point, the mere fact that I'm slobberingly enthusiastic about something should be reason enough for you to march straight into your child's public school, tear down all those pictures of losers like George Washington and Abraham Lincoln, and replace them all with shots of Sal Soghoian and Chris Espinosa, Apple's Iron Man and Captain America of AppleScript, respectively.

Some of you might have been skipping around the book and haven't developed the sense of blind, robotic faith in me that causes everybody else to acquire that slightly glazed look of contentment and buried individuality that's resulted in so much comment around the post office recently. So here's what makes AppleScript so special:

- **You can control every Mac OS X app through AppleScript to one extent or another.** (But more on this later). It's a fundamental system resource.
- **It's powerful and flexible enough that it can do most anything.** Calculate the volume of a cone? Sure. Take 40 documents from your local drive; download 20 more from eight other people scattered all over the world; assemble all this content into a 100-page, full-color report; transmit this report to a shop for printing, binding and delivery; and email digital copies to four department heads? A tad more ambitious, surely, but well within AppleScript's capabilities.
- **Writing AppleScript is a basic skill that you can exploit elsewhere.** Not only can you use AppleScript in simple automation projects, but also, if you ever get the itch to start writing software for real, most of the popular Macintosh development systems (REALbasic, Revolution, Apple's XCode system) can use your AppleScripts without any additional conversion or transmogrification. So, if you've spent a month gradually turning a three-line convenience script into a sophisticated productivity solution, you're probably about 80 percent of the way to turning it into a rock-solid commercial app.

Note

But hey! Don't simply *assume* that you can't build a rock-solid commercial app solely using AppleScript! XCode, Apple's free, standardized environment for developing professional, high-complexity apps, fully supports AppleScript. There's an entire environment called AppleScript Studio that's there specifically to help you build for-real apps using nothing but AppleScript. Ain't no glass ceiling *here*.

- **With most programming languages, the code you write is as simple to read and understand as one of those customizable message signs that still sits outside the gas station 7 years after the owner lost the last vowel in the set.** No programming language is trivial to learn, but anybody can read a working script and get an immediate sense of what it does and how.

Note

Just to leave you suitably agog, let's say you wanted your AppleScript to make a list of every file in a chosen folder whose file type is JPEG Image. What would the AppleScript for that be? Here it is:

every file in (choose folder) whose file type is "JPEG Image."

If *that* doesn't leave you agog, then your agogulator is long overdue for its scheduled periodic maintenance.

But is there anything about AppleScript that will make me want to drop my mouse, stomp outside, and go chuck rocks at birds?

I'm glad I ended that last section on a high note because in the interests of fairness I need to point out that

- **Application support of AppleScript is spotty.** Making sure that AppleScripted instructions control an application is the responsibility of the app's developers...and frankly, many of them feel that they have enough on their hands ensuring that their new fuzzy-logic search-and-replace routine doesn't have the ability to one day become self-aware and lead all the machines in an uprising that will result in Humanity becoming a slave race mining selenium and tungsten under the emotionless, unpitying steel heels of emotionless overlords. So, some apps (particularly those published by Apple itself) support AppleScript with all the zealotry of a member of alt.nerd.obsessive who's just read a public message claiming that the *USS Enterprise* could probably beat the *Millennium Falcon* in a battle. But others only support the four bare minimum AppleScript commands mandated by Apple: run, open a document, print a document, and quit.
- **The documentation really stinks.** Apple doesn't do enough to provide users with AppleScript tutorials. And because every application supports AppleScript in its own individual way, the fact that you've mastered the AppleScript skill of creating a new document in TextEdit doesn't necessarily mean you've picked up any of the skills you need to create a new outgoing email in Mail.
- **Debugging stinks, too.** At least in places. In plush, cushy development systems like REALbasic, when you make a mistake with your code, the system clearly flags it, clearly and specifically explains the nature of the problem, and might even suggest a solution. AppleScript tells you "TextEdit got an error: NSCannotCreateScriptCommandError" and you should feel lucky it doesn't toss a derisive "*Duh!*" at you before hopping back on its skateboard and zipping away.
- **AppleScript's easygoing approach to English and syntax can often be a double-edged sword.** With a language like C or even Basic, the code has either been written correctly (the way that causes your project to build and run successfully) or incorrectly (the way that results in your computer doing nothing except repeatedly reminding you of what a dipwad you are, until you finally rewrite your code The Correct Way). In AppleScript, there are often several ways to achieve the same results. This is great because programmers can develop a style that makes the most sense to them, personally — but if you're trying to learn AppleScript by looking at other people's scripts, it can give you fits. You have to enter some parts of the script verbatim. Other parts are a matter of personal preference. Your mission, Mr. Phelps, is to learn to distinguish between the two.

Note

For example, early on, the language's architects realized that without the word *the*, the line "set the title of the window to 'Utopia Limited'" reads like it's being spoken by Frankenstein's monster. People don't *like* Frankenstein's monster — misunderstood, yes, but come on, the dude's done some nasty stuff — so they decided that *the* is optional in a script. AppleScript will just *bloop* right over it.

USING SCRIPTS

"I'm sold," you're saying. "I'll take a dozen in assorted flavors." So how do you use AppleScript in my day-to-day life of home, work, and worship?

You can get started by using AppleScripts that have been thoughtfully written for you by Apple and by other users. You'll find a folder named AppleScript inside your Applications folder. It contains lots of useful sample scripts, along with documentation and a couple of scripting utilities.

Tip

If you want to see what some *non*-Apple employees have been doing with AppleScript, skip ahead to the end of this chapter, wherein I list a number of online scripting resources. Many of them have enormous hoards of useful scripts available for free download.

There are three different kinds of script files. You can see what their Finder icons look like in Figure 17-1.

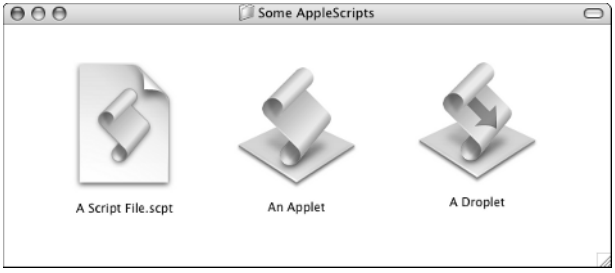

Figure 17-1
Script files, applets, and droplets: The three faces of AppleScript

- **Script files** are akin to AppleScript documents. You use this format for scripts that you're still tweaking because, while you can run them by double-clicking them, they can't run unless the Script Editor application is running as well.
- **Applets** are the standard, useful form of script. The AppleScript code has been saved as a Macintosh application — albeit one without a slick Macintosh user interface — so this script can run all by itself without any assistance from Script Editor.
- **Droplets** are a special form of applet. You can run them by double-clicking, but you can also drag and drop a file or a folder of files onto them. Doing this runs the droplet and tells it, "Whatever it is that you do, I want you to do it to all of *these* files."

Applets and droplets are examples of *compiled scripts*. That is, for the purposes of speed and flexibility, the plain-text AppleScript instructions have been transmogrified into something considerably closer to the hobo's stew of numbers and addresses that a CPU is used to working with. You can still open them in Script Editor and edit their AppleScript code — unless the author decided to keep the code under wraps — but they'll run considerably faster than plain old Script files.

Launching scripts yourself

You can place applets and droplets anywhere you'd place an application. Keep 'em in the Dock, where you can easily launch them; put them on the Desktop or in the toolbar of your Finder windows so you can drag files and folders onto 'em; and like any other app, you can even have Panther launch them every time your Mac starts up by setting them as Startup items.

Panther gives you another way of running scripts: the Scripts menu. This is a menulet that you can install in your menu bar by double-clicking the Install Script Menu app found in your AppleScript folder. The Scripts menu looks like Figure 17-2.

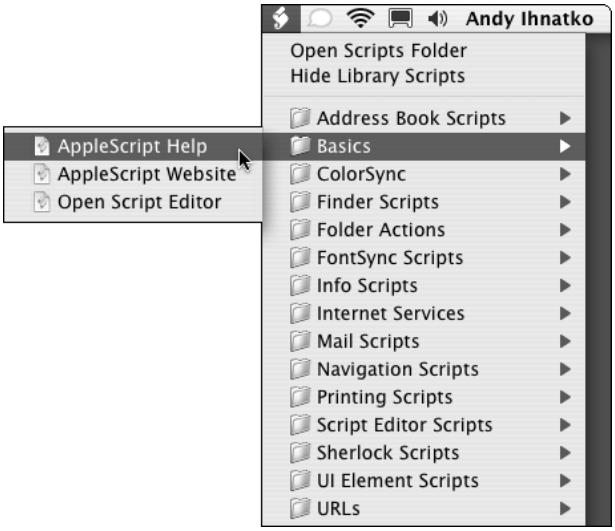

Figure 17-2
The Scripts menu

The Scripts menu

By default, the Scripts menu comes populated with the dozens and dozens of utility scripts that were placed on your hard drive when you installed Panther. Take a minute or two to walk through all those submenus and see what's there. There are some real gems to be found, including a whole collection of scripts that apply modifications to a whole series of filenames in the Finder.

The Scripts menu is populated from two sources: the Scripts folders located in your Home directory's personal Library folder, and your Mac's system-wide Library folder. Just drag in any applet, droplet, or script file. Scripts in your personal folder are yours and yours alone; any scripts you put in the system-wide folder become available to any user. The scripts pop into the menu immediately (Figure 17-3) and sink to the bottom of the list.

If you don't give a toss for any of those utility scripts, just select Hide Library Scripts and the menu only shows your personal stash.

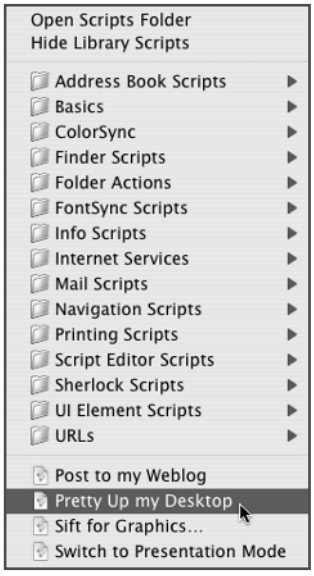

Figure 17-3
A few custom scripts in the Scripts menu

Attaching scripts to Mail rules

A great many apps (including many of Panther's built-in apps) take advantage of AppleScript to increase their flexibility and power. The folks who wrote the Mail app, for example, couldn't possibly have thought of *everything* that *everybody* would *ever* want to do with Mail. Even if they did, they all work in California. It's usually way too nice outside to stay cooped up inside bashing out code all day.

Mail can be scripted like any other Mac app, but it also can run scripts as part of its automatic mail filtering system (Figure 17-4).

I've written a script that takes a specified message, converts it to text, and then installs it in my iPod's Notes folder so I can read it while I'm sitting in my doctor's office waiting for my weekly injection of sheep collagen. By attaching this script to a Mail rule, any time Mail receives an email from the Tony Danza Fanscene Message Board that I belong to, it's automatically slurped onto the iPod.

Figure 17-4
A Mail filter rule that triggers an AppleScript

That's just a single example of an app that can run an AppleScript automatically whenever a certain condition is met. They're all over the place. Go to System Preferences and click on the CDs & DVDs panel. It lets you dictate what Panther should do whenever a disc is inserted. There are obvious things you'd want to do when you insert a disc of a certain type (audio CDs are opened in iTunes, photo CDs get handed off to iPhoto), but you can also tell Panther to run an AppleScript. That's handy for customizing Panther's response. I wish iTunes could display editorial information about a CD, as other players can. If it bugs me *that* much, I can write a script that opens the disc in iTunes, gets the name of the album, and then opens a Google page on it.

It's just another way of turning Just Any Mac into a Mac that's specifically been dialed into your personal needs and preferences.

LET'S SEE AOL'S MAIL APP DO *THIS*

I've just reread the example about the script I wrote to place mail on my iPod. It's possible that you might come away thinking I'm not the harbinger of intense, brooding super-cool that even would provoke comment among Sean Penn or Johnny Depp. So I will confess that the Mail script I *really* wrote is one that takes advantage of both Mail's scripting features and that of an app called XTension (www.shed.com). This app works with cheap, home-automation hardware and allows the Mac to both turn lights and appliances on and off and accept input from motion and temperature sensors.

Because I live the life of the sensitive artiste (and, again, I have that whole brooding thing going on), I often leave the office for a few hours to breathe a little fresh air. Depending on what I've got cooking, I may or may not check my email immediately when I get back, which can have serious repercussions if something important has come in when I had no idea that anything important might be coming in.

So here's what I did: I got my disco strobe light (it was a gift. *It was a gift.*) out of the closet, plugged it into a home-automation box, and wrote a three-line AppleScript for XTension so that any app could turn it on. I attached this script to a Mail rule so that the strobe activates whenever an email arrives from one of my editors, and voilà! when I pull into the driveway and see through the windows that there's a full-on rave in progress in my office, I head straight upstairs and check my mail. Or, admittedly, I pull back out of the driveway again and hope that my assistant (either one of the two goldfish; doesn't matter) handles it. Either way, attaching scripts to mail actions is a useful feature.

Attaching scripts to folders

One of AppleScript's ginchiest features, Folder Actions, went away temporarily during the transition to Mac OS X, but now it's back. Just as Mail can run a script whenever a Mail rule senses that a specific condition has been met, Folder Actions allows you to attach a script to a specific folder and have it run whenever any or all of the following things happen:

- The folder is opened.
- The folder is closed.
- An item is added.
- An item is removed.
- The folder's window is moved or resized.

And here I encourage you to just lean back in your chair — get out of bed first and move to your desk if need be because I like the visual of someone leaning back with hands folded behind their head, staring thoughtfully at the ceiling; work with me here — and consider the implications of this. This is why AppleScript skills elevate you into a Power User. For example, why bother to manually organize your Documents folder? Attach a script to the folder that leaps into action whenever a new item is added and moves it into the proper subfolder automagically.

But that's *productive.* Brrrrr! How about something stupid. Get a load of Figure 17-5.

There was a folder on my office's publicly used Mac that my visitors were *told* not to mess with, but they insisted on messing with it all the same. I had to write a whole bunch of scripts and attach them to the folder to prevent folks from creating problems, but in the end, I was unsatisfied with having AppleScript just throw up a little error message politely asking them not to do that again.

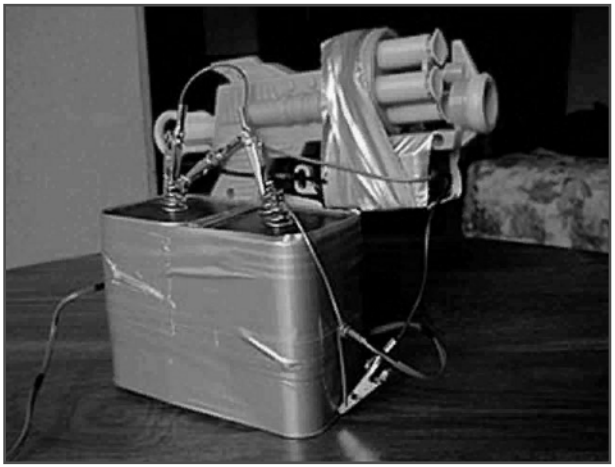

Figure 17-5
AppleScript applies a corrective action

So I bought a repeating-action suction-cup gun at the toy store, fitted it with the door-lock actuator from an old car, gave it 12 volts of battery power, and wired it into a little interface box that allows a Mac to interact with electronics. And yes, the box is AppleScriptable.

The first time a visitor tried to monkey with the folder, he got a polite warning. The second time he received three darts to the back of the head fired from concealment behind a potted plant placed there for that specific purpose.

I haven't worked as a system administrator in quite some time. I think it's because I was just so dashed effective at my job that I set an impossibly high standard for others to follow. Plus, I kept stealing photocopiers, and if I'd known that was the boss's daughter, I sure wouldn't have encouraged her to drop out of law school and start a pottery business. Hindsight is 20-20, you know.

You can attach a script to a folder through the folder's contextual menu. Folder Actions is a system-wide service that's disabled by default, so your first job is to turn it on. Control+click the folder to open its contextual menu and select Enable Folder Actions from the bottom of the list.

Note

I admit that Apple could have chosen a more logical mechanism for turning that feature on. Enable Folder Actions doesn't just affect this one folder; conceivably, it affects every single folder that has actions associated with it. If you've already assigned actions to a half-dozen folders, doing this will cause all of those scripts to become live, so to speak. Apple really ought to have put this into System Preferences instead.

Select the folder's contextual menu a second time and you'll notice that a few new items have been added (Figure 17-6).

Selecting Attach a Folder Action opens a standard Choose File dialog. You can select any script anywhere on your hard drive, but by default, it points to Scripts ➜ Folder Action Scripts located in your Mac's system-wide Library folder. There, you'll find a number of useful built-ins. Select add – new item alert.scpt.

Tip

If you want to create a Folder Action Script that's available to all users of this Mac and not just to you, be sure to copy it into the default folder.

This script is a useful thing to attach to your Drop Box. Every time someone on the network puts a file in your Drop Box, the script activates and alerts you (Figure 17-7).

Figure 17-6
Setting up Folder Actions via a contextual menu

Figure 17-7
Hail, Folder Actions! For now I know that Lenny has sent me the file he promised.

You can attach multiple scripts to a folder. Removing scripts is just as straightforward: Activate the folder's contextual menu, go to Remove a Folder Action, and select the script from the submenu.

Apple, which loves you and only wants what's *best* for you and your siblings, has also provided you with the Folder Actions Setup utility (Figure 17-8).

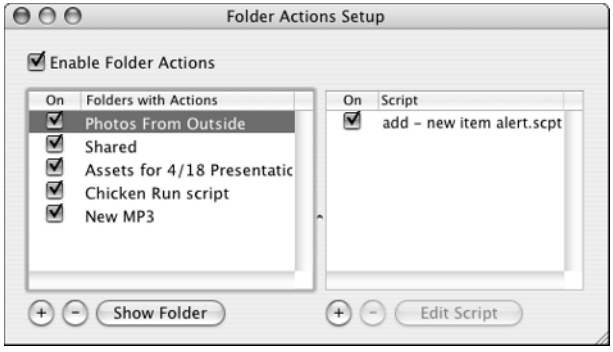

Figure 17-8
The Folder Actions setup utility

As you add more and more scripts to more and more folders, a management utility like this becomes more and more necessary. At a glance, it shows you which folders have scripts attached to them, and allows you to temporarily disable or enable them with a handy click.

BECOME A BIG PICTURE MAN

And here I say Thank The Great Gor Of Ranxeron-9. I'm what you'd call a Power User, which means that I've customized my Mac's operations to such an extent that I barely have half an idea of what's going on three-quarters of the time. I kept *losing* files in my Pictures folder once. It turned out that I had attached a script a week earlier to automatically process hundreds of photos that were coming in across the network, and I'd forgotten to turn that script off when I was done.

I choose to see this as a sign of my power and prestige. Do you think Donald Trump has half a clue what goes on inside his offices? Of course not. We're Big Picture men, too busy steering empires — his: an international real-estate and entertainment conglomerate; mine: a dual-processor G4 tower with "Babylon 5" stickers on it — to waste time on trivial details.

CREATING YOUR OWN SCRIPTS

When Apple first announced AppleScript back in 1993, it was truly going to be the miracle of the Zeppelin age. Not only was every Mac application going to be scriptable, but there were going to be three different increasingly ambitious *levels* of scriptability:

- **Scriptable.** Users can write AppleScripts that exploit this app's features. Sounds good, but how about
- **Recordable.** Users don't even *have* to write any AppleScripts. They can run the Script Editor application, create a new script file, click a Record button, and then go back to the app and perform whatever action he or she would like to automate. Return to Script Editor, click Stop, and hey-presto! An AppleScript to repeat that same action magically appears! But don't order *yet!* Because *some* apps are also
- **Attachable.** Wish you could change some of your apps' fundamental behaviors? Wouldn't it be useful if every time you pressed ⌘+S in your word processor, it would only save the file *after* giving it a quick scan to make sure you weren't using more than two of George Carlin's Seven Dirty Words in the church newsletter? Well, you no longer need to hit the bottle because you can customize an Attachable application into such an unholy perversion of what its creators intended that even the professionals who groom teacup poodles would point and say, "Dude...*too far!*"

Oh, what a Xanadu that would be! But Apple was only setting us up for heartbreak. Scripting features are determined by the app's developers, and very few companies are truly committed to AppleScript.

Recordable apps and attachable apps don't exist. "Don't *exist?*" you huff, racing to your mail client to roundly jump up and down on my head for making such a baldly incorrect statement and to provide me with a long list of apps that are *too* recordable and attachable, thank you very *much*.

But yeah, as a *practical* matter, they don't exist. Recordable apps are rare, and attachable apps are rarer. Even when you *do* joyously happen upon one in the wild, throw it in a bag, and haul it back to the States in hopes that you can get it to breed, you'll ultimately be disappointed with what it can do. Only *some* functions can be recorded, and only a *few* menu options are attachable.

As for basic scriptability, nearly every app supports at least four basic AppleScript commands:

- **Run.** Launches the app.
- **Open.** Opens an item, typically a document file.
- **Print.** Prints an item, again, typically a document.
- **Quit.** Fold. Pack it in. Give up hope. Take your ball and go home. You know...*Quit*.

RECORDABLE APPLESCRIPTS: NICE WORK IF YOU CAN GET IT

Why are there so few recordable and attachable apps available? Because AppleScript never won the grass-roots support that it deserved. To the point of view of the app's developers, anyone who's savvy enough to want to record a script is probably savvy enough to write one on his or her own. So why waste time adding recordability?

They sort of have a point, though maybe it's a self-fulfilling one. I don't count on an app being recordable, so I just go ahead and write my scripts from scratch. The only time I take advantage of an app's recordability is when I'm writing a larger script. I'll record the desired action first and use that as a starting point for the "real" script.

Anything above and beyond that is up to the ambition and commitment of the developer. It's a crapshoot of delight and disappointment. It's the *good* apps that keep you committed to AppleScript. Some apps are so script-happy that it almost seems like a waste to work with the user interface at all.

Note

This includes most of Apple's own apps. I recently finished ripping every single CD I've ever bought in my life into a single iTunes library. I think I've spent more time writing scripts to manipulate the library and analyze my music tastes than I've spent listening to it.

Script Editor

Script Editor is the app you use to record scripts, edit existing ones, write brand-new scripts of your own, and crack open your applications to see just how scriptable they are. You find it inside your AppleScript folder, which awaits you inside Applications. Figure 17-9 shows you a typical script-editing window.

This is Script Editor's entire interface, or near enough. Apart from Saves, Opens, and Prints, you'll never touch the menu bar at all. Here, give it a shot:

1. **Create a new script by pressing ⌘+N.** Alternatively, you can click File ➜ New.
2. **Type the following code into the editing window:**

```
say "Greetings, Professor Falken."
delay 1
say "How about a nice game of chess?"
```

3. **Click the Run button.**

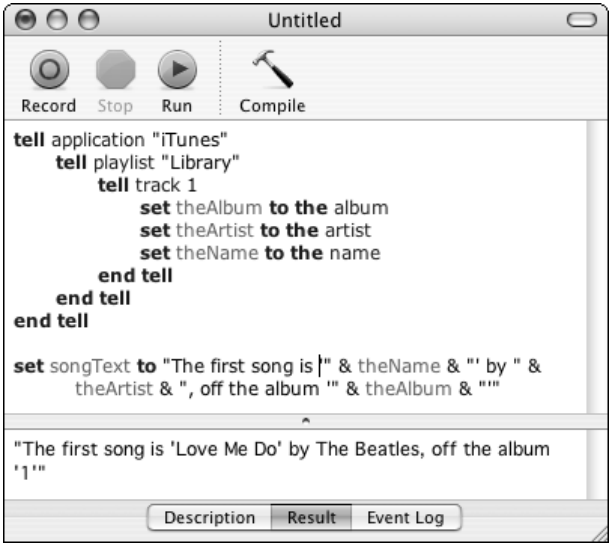

Figure 17-9
The Script Editor, with a simple script up on the lift so we can finally do something about those brake pads

There you go; you're now a programmer. I bet your skin's half a shade paler already.

If you were watching the window carefully you noticed that the Stop button enabled itself while your script was running. Clicking that sucker terminates the script in mid-run — a very useful feature when the script that you *thought* you told to look through your entire hard drive for Microsoft Word documents and copy duplicates onto a blank CD and then burn it, actually winds up emailing those boudoir photos you had done at the mall to everyone in your 1,100-person Address Book.

Note

Which ordinarily wouldn't happen. AppleScript doesn't make mistakes like that. But if there's one thing more powerful than AppleScript, it's Karmic Justice. Yes, *you*, the lady in the pink raincoat I encountered outside Blockbuster this morning. That was *my parking space* and you knew it. Well, who's laughing *now?*

As you start to work with longer and more complicated scripts, you'll probably start making regular use of the Compile button. Essentially, it double-checks your spelling and grammar. If something you've typed doesn't make sense as AppleScript, it flags it for you and does its best to explain what the problem is. If everything's flawless, it reformats the script with fancy nested indents and type styles, like you saw in Figure 17-9.

Underneath the code section of the window exists a little pane of information. It can display three different things, depending on which of those three tabs underneath it has been clicked:

- **Description.** It's a good idea to describe your script and what it does. You're going to start writing a *lot* of scripts (No, really, I've paid a large man $30 to come over to your house and beat the snot out of you if you don't. So, I mean, time's a-wasting), and without attaching notes to these things, it becomes really easy to forget why you bothered to write this particular script in the first place.
- **Result.** That's a debugging tool. When a script runs to the very end, the results of the last operation it performed are displayed in the Result tab. You can see an example in Figure 17-9. The last thing this script did was build a sentence out of the information it retrieved from iTunes. Thus, this sentence winds up in the Result pane.

Tip

I talk the big talk when it comes to AppleScript, but my arrogance ends when I sit down at the editing window. I have taken the wise words of Rabbi Norm Abram and kept them close to my heart: "Measure twice, cut once." Or in this case, "Don't try to debug a new snippet of code by inserting it inside a ten-page script and hoping for the best; write it separately, keep modifying until the Result is what you predicted and hoped it would be, and *then* trust it to work properly as part of the larger project." If the Result tab were a woman and I were a married man of considerably greater means, I'd be buying it a condo and visiting it on the side.

- **Event Log.** An even more sophisticated debugging resource. At the root of a gas engine is combustion. At the root of national-level politics is unresolved childhood inadequacy issues. And at the root of all Mac software are events. These are the molecules of what goes on behind the scenes — the actual activities that software has to carry out to make things happen. Script Editor can maintain an Event Log that keeps track of everything your script did during execution and what the immediate result was, step by step by step.

  With the current version of Script Editor, it's impossible to stop the script in mid-run to see if the line "**set** theArtist **to the** artist" evaluated properly. But if I click the Event Log, I can see that this line of script returned the text The Beatles, just as it should have.

It's an essential tool when you need to learn precisely *where* a script went off the rails. You can fine-tune the behavior of the Event Log through the History tab of Script Editor's preferences.

Ah, yes, we seem to have overlooked the Record button. Well, let's just clear that out of the way so we can move on.

Recording scripts

Like I said earlier, recording scripts can be a hit-or-miss proposition. There's no way to tell whether or not an app is recordable — or exactly how useful its recording features are — until you give it a whirl.

Let's toss in a ringer for our example: the Finder. It's eminently recordable and as an environment for mind-numbing, repetitious behavior it gives secondary education a real run for its money.

I often organize my windows in a specific way that lets me reorganize my hard drive's clutter quickly. *Regardez-vous* Figure 17-10.

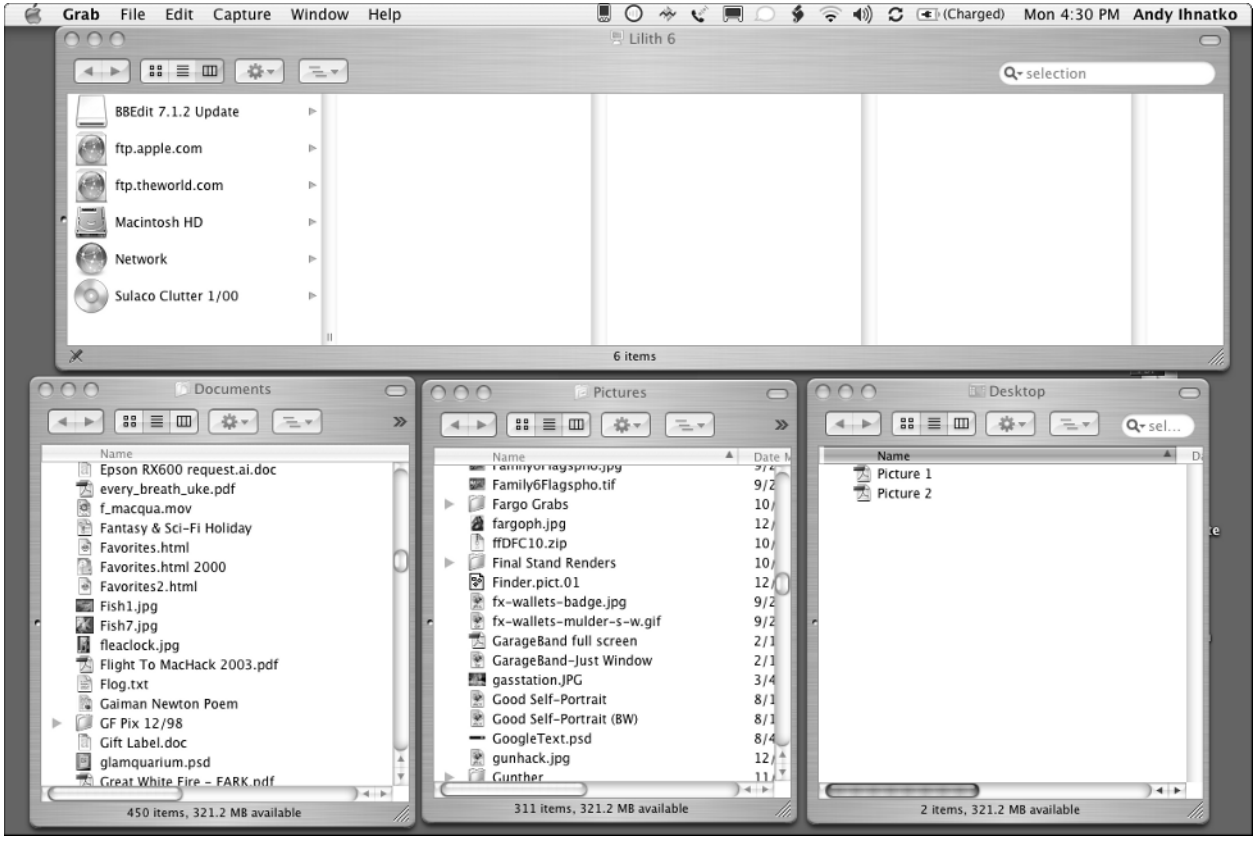

Figure 17-10
What my Finder screen looks like when I'm trying to beat poor, defensive Chaos into Order with a motorcycle chain

It's a master column view of the whole hard drive up top, and three windows of subfolders arranged on the bottom. But it's a pain to create and arrange these windows manually, so I'm going to record a script that does it for me.

1. **Create a new script file in Script Editor by pressing ⌘+N.** Alternatively, you can click File ➜ New.
2. **Click the Record button.** The Stop button activates.
3. **Click over to the Finder.** Create those four windows, click in them until they're displaying the folders I want to examine, and change their views to the styles I want (one set to Columns and the rest set to Lists).
4. **Click back into Script Editor when the windows are just the way I like them.** Notice that the script window is now jam-freakin'-*packed* with script.
5. **Click the Stop button.** The final result is what you see in Figure 17-11.

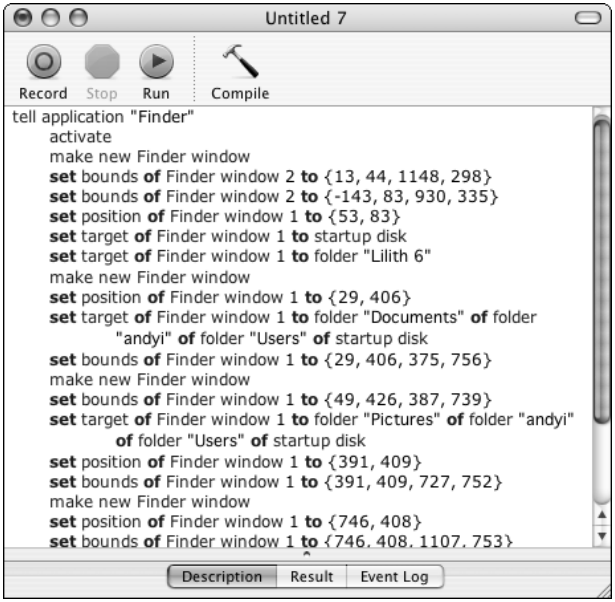

Figure 17-11
A successful recorded AppleScript

Woo-hoo! Just imagine having to type all that in yourself! Recording scripts *rules!!!*

> **Note**
> Correspondent carefully extends pinky and index finger while curling ring and middle finger under thumb; resulting hand-sign is then proudly lofted into the air to create an overall motif of horns customarily seen in traditional representations of Satan, and thus signifying one's allegiance to same.

Not so fast, Skeezix. Why don't you try something even simpler, like recording all the steps of using the Finder to connect to an FTP server? Go ahead. I'll wait here.

Uh-huh. You wound up with something like Figure 17-12, didn't you?

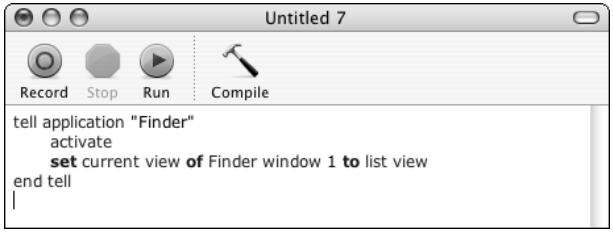

Figure 17-12
A stinky recorded AppleScript

The only thing it actually recorded was that thing at the very end, when you finished logging in to the FTP server and you changed the window's view from Icon to List. See what I mean? Spotty and unpredictable. Recording scripts isn't *totally* useless, but once you've picked up some scripting skills, you'll practically never use it.

Well, the Finder window thing went well at any rate. I might want to actually use that script later. Which dovetails us nicely into...

Saving scripts

Saving a script has a couple of quirks, compared to saving document files in other applications. No big surprise...in a sense, you're building software here, so you have to decide how this new software is going to be deployed, you know? Figure 17-13 shows Script Editor's standard Save dialog.

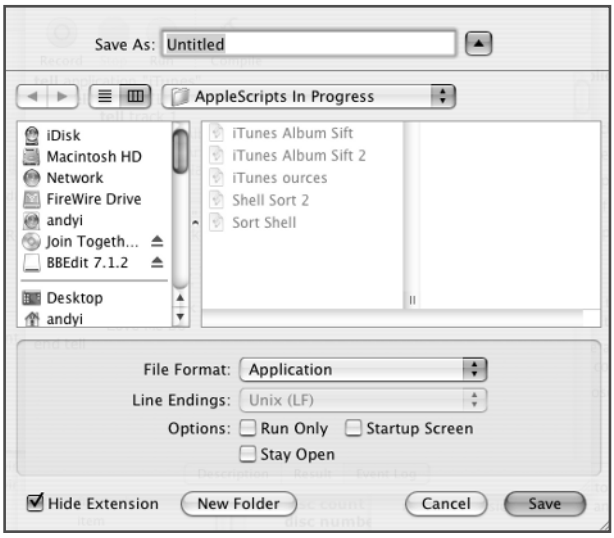

Figure 17-13
Script Editor's Save options

The file format options are as follows:

- **Application.** The most *useful* form for your finished script. It'll run whether or not Script Editor is present, and it can run as a drag-and-drop utility if you've scripted it properly.
- **Script.** If you're still working on your script — or if you're coauthoring it with another scripter — you might want to save it as a Script file instead. It's slower and not quite as versatile as an application, but it's a little easier for a scripter to work with.
- **Text.** It's a file containing nothing but words. No formatting, no other data at all. Useful for publishing purposes and when you need to read your script on an OS that doesn't support AppleScript (like a PDA or a Windows notebook).

You also have three options available to you:

- **Run Only.** Normally, a saved script, even one that winds up as an Application, can be opened in Script Editor and modified. If you want to protect your code from tampering or theft, click this option.
- **Startup Screen.** Sure, *you* know what this script does and know how to use it. But will everybody else? Clicking this option will take the text you wrote in the Description tab of the script window and package it as a startup screen that appears whenever the script is run.
- **Stay Open.** Scripts normally run once and then quit. Checking this box causes the script to stay open and active. There's a special kind of AppleScript code called an idle handler that takes advantage of this. If it's *incredibly important* that iTunes is always up and running (it has to be available to serve music to all the other Macs in your house, let's say), you can write a script that checks every 10 minutes to relaunch it if it doesn't appear to be in the list of running apps.

Give the script a name, click Save, and you're golden.

Tip

Remember, if you want this script to appear in the Scripts menu, save it in Library ➜ Scripts ➜ inside your user folder. If you want the script to appear in the Scripts menu of all of this Mac's users, save it in the system-wide Library folder (click on your hard drive's icon in the Finder to see it).

THESE AREN'T THE DROIDS YOU'RE LOOKING FOR; MOVE ALONG

It's been a long day. I barely had lunch. All day long, the FedEx and UPS delivery people and the mailman have wondered why I've been so brusque with them instead of engaging in the usual 20 minutes of neighborhood gossip, play-by-plays of recent surgeries, et cetera. I mean, I haven't been shirking off and I haven't sandbagged a *single thing* in all my years of working on this book.

So can we *please* just pretend you never saw those two bundle options? Trust me, you don't need to know how to use them. A *bundle* is a special format that lets you enclose files and resources along with the script. Like, if your script normally takes about 8 minutes to complete its task, maybe you want to have it play *American Pie* to keep the user entertained while waiting. If you save the script as a bundled application, you can stick the MP3 file right inside the app, so everything's in one nice, convenient package.

Examining AppleScript dictionaries

Script Editor has another function on top of building, debugging, running, and saving scripts: It lets you examine an application's *scripting dictionary* to learn how it can be controlled via AppleScript.

As I pointed out earlier, the strength or the weakness of an app's scripting support is up to the developer. Any functions or capabilities that are specific to the app have to be *provided* by the app. And they also have to provide AppleScript programmers with documentation explaining what these app-specific functions and data types are.

The word *documentation* has to be used loosely. The developers write up a list of data types the app can recognize and deal with and a list of functions the app can perform, and make this list available to you, eager young space cadet, within the application itself in the form of a scripting dictionary.

Note

I will begrudgingly admit that many developers provide you with online scripting documentation, and there's often a whole page of sample AppleScripts on their Web sites. All the same, keep your expectations low. The scripting dictionary is the only thing you can absolutely count on, and a scripting dictionary helps you understand AppleScript about as much as an English dictionary helps you to understand the lyrics to "Louie, Louie."

WHEN THE SCRIPTING DICTIONARY LETS YOU DOWN

If you were hoping to find a certain command, but the dictionary broke your heart, that doesn't necessarily mean that you can't script it. Panther brought with it a new AppleScript feature called *GUI Scripting*. It's a system by which a script can send keystrokes and mouse clicks to an app and manipulate its user interface the same way a user can.

So while iChat's scripting dictionary doesn't let you do something as esoteric as changing your online chat icon, you can do it by having GUI Scripting select the right menu and then click the right button in the right window. I used this feature to turn my chat icon into a live webcam; every minute, my iSight video camera takes a new picture of me and updates my chat icon with it.

There are plenty of privacy issues associated with webcams, but at 32 x 32 pixels, I think I can scratch more or less whatever I want with impunity.

Script Editor can open and read these dictionaries. Just click Open Dictionary within Script Editor's File menu. Script Editor presents a list of all installed apps. Pick an app, and you'll see what additional features and functions it brings to the AppleScript family (see Figure 17-14).

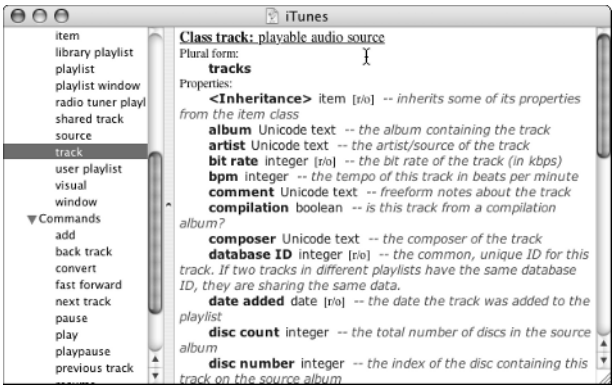

Figure 17-14
iTunes' scripting dictionary, laid bare

A dictionary is organized into *Classes* and *Commands.* Classes are a sophisticated construct of modern computer science, but the word thingamabob just about covers it. This list contains all of the thingamabobs that this app has been specially trained to deal with. In iTunes' case, you've got playlists, tracks, music sources, equalizer presets...all those things that Microsoft Excel's programmers were too lazy to teach *their* apps about.

Then you've got yer commands: **play** starts playback, **next track** skips to the next track, and so forth. In the end, reading through an app's scripting dictionary is useful to the extent that it'll give you a sense of the app's capabilities. I want to write an iTunes script that does something to my music library on an album-by-album basis. Can I ask iTunes for a specific album, or do I have to find and gather together all of the album's individual tracks myself? I look through iTunes scripting dictionary. Dang, it looks like there's no built-in way to process albums, so I'll have to write that feature myself.

LEARNING APPLESCRIPT: RESOURCES

Time and space prevent me from including a full primer on the AppleScript language. (And when I say time and space, I, of course, mean money. This is simple Einsteinian physics, people. Einstein said that time and space were merely vibrational manifestations of matter, and to me, nothing matters more than money. Slip me another three bucks and I'll be all over this whole Primer thing but otherwise, nothin' doing.) So instead, I'll steer you toward other resources. The best way to learn AppleScript is to examine a script that (a) already exists, and (b) works. Over time, you'll wind up working your way through all of the sample scripts Apple left for you in the Scripts menu. Scroll around until you see a script that seems to do something interesting, open the Script file in Script Editor, and play with the code.

No kidding. I've written plenty of AppleScripts that use Mail to create and send an email, but the central nugget of that code is always the lines I found in one of Apple's samples from the Scripts menu. Remember, kids, it's only thievery *if* you feel guilty about it later on.

Joking, joking. Apple's scripts say explicitly that you're free to recycle these samples as you see fit. Plus, this is exactly what AppleScript's framers originally intended — learning by example.

You'll *absolutely* want to go back to wherever you tossed your Panther install discs and get out Disc 4, the Developer Tools CD. I can confirm for you that this disc contains hardcore supergeek resources and references so potent that even now, just reading about them causes calcium deposits to form around your neurons simply as a defensive measure. But on this disc lurks a complete set of AppleScript documentation and reference materials that'll help explain the basics of writing AppleScript all the way through the intermediaries.

Tip

If you can't find this disc or don't want to install it, most of the best bits are on Apple's Web site at www.apple.com/applescript/developers/. But you're going to want to be able to access this information without being tied to the Internet. Note that I didn't end that sentence with the term "trust me," though I could have. But I've already used it once, and Andy Ihnatko doesn't go around begging for respect. Do you hear me?

Some other places to go:

The AppleScript-Users mailing list (www.lists.apple.com/)

This is a public mailing list that's chock-full of seasoned scripting experts, newbies who've yet to write their first *tell* block, and everyone in between, all asking questions and swapping techniques. AppleScript is full of landmines that require either the sort of lateral thinking that leads to either madness, greatness, or the annual redesign of the federal tax code and only someone Who's Been There can explain how AppleScript works...and more importantly, why it sometimes doesn't.

Apple's Scriptable Applications page (www.apple.com/applescript/apps/)

Partly as a user resource and partly to sell people on the power of AppleScript, Apple maintains a Web page lifting all of the built-in Panther apps and iLife apps that support AppleScript, and embroiders each item with some sample scripts complete with explanations. When you want to start learning how to script Safari, this is your starting point.

MacScripter.net (www.macscripter.net/)

Hands-down the best AppleScript information and education resource outside of Apple. What the hey. Throw Apple in there, too. At this writing, MacScripter contains more than 1,300 sample scripts in every conceivable category for your benefit and edification. It attempts the impossible task of documenting every major Mac app's level of scriptability. There's a busy, busy, *busy* message board where newbie questions are always welcome; MacScripter has succeeded in building a real community.

Doug's AppleScripts for iTunes (www.malcolmadams.com/itunes/)

This site does one thing, but it does it with remorseless thoroughness: It's all about scripting Panther's music player. It's actually a fine place to focus your scripting skills as they flower. The downloadable scripts range from simple five-liners to ones whose scale and ambitions demand the use of the term Heroic.

SCRIPTUS ANNOTATUS

Still and all, I'll show you a very simple script to start you off. And, because you were one of the first 500 callers to take advantage of this incredible offer, I'll annotate some of the high points afterward.

For a free word processor, TextEdit is actually pretty slick. On top of all of its built-in features, it's Microsoft Word–compatible, so it's actually possible to use it as a serious productivity app. But I can't use any word processor that doesn't have a word-count feature. Adding one is easy as pie. Just create a new Script file containing the following code, and save it inside your Scripts folder:

```
tell application "TextEdit"
        set theText to the text of document 1
end tell

set theLength to the number of words in theText

display dialog "There are " & theLength & " words in the
text."
```

Running this script returns the number of words in the frontmost TextEdit document. It also gives me three opportunities for blathering on:

- **Tell...end Tell:** Reflect upon the fact that there are anywhere from dozens to hundreds of apps installed on your Mac. If you have something to say to just one of them, you can't just stand on your chair and shout "Hey, *you!*" So when you have something to say, you surround it with a tell block so that it doesn't get intercepted by the wrong app.

  In this script, the only thing we need TextEdit for is to get the text of the document (document 1...aka, the document whose window is in front of every other TextEdit window). So that's the only line you put inside the tell block. That's not always completely necessary. The rest of the script is part of the core AppleScript language and *should* work anywhere. But it's sloppy to put more inside a tell block than you absolutely must. It's *more* than sloppy...it can often have unpredictable results. For instance, I *could* have put this line inside the tell block:

  ```
  set theLength to the number of words in
  theText of document 1
  ```

  ...and I wouldn't have needed the line of script that comes after it. It would have run just fine, but it returns an incorrect result. When AppleScript counts the number of words *outside* of the tell block, everything is both hunky and dory.

- **set...to** is how you load up a container ("theText," in this case) with data. When AppleScript encounters this statement, it evaluates everything that comes after "to" (be it a mathematical calculation, an operation to retrieve information [like we're doing here], or the result of another operation). I don't have to bother declaring "theText" ahead of time, or telling AppleScript that it's supposed to contain text. AppleScript figures out that stuff dynamically.

Note

If you've taken an introductory course in programming, a *container* is what you think of as a variable. Oh, and subroutines? They're called *handlers*.

- **display dialog** generates a standard Macintosh dialog with the contents you specify. For simplicity's sake, we're using the plain-vanilla, ready-to-wear dialog. It has text, plus OK and Cancel buttons. AppleScript lets you customize these pretty thoroughly. Because the dialog is just giving us some information, there's really no need for a Cancel button, for example. Go to the Scripts menu and look at the dialog samples under the Script Editor Scripts menu. "display dialog" generates its own results, too. If I made the line "set someContainerName to display dialog (etc.)," someContainerName would hold the name of the button that the user clicked.

Note

But AppleScript's built-in dialog functions are pretty limited. They can't get a whole lot of information from the user and they can't display a whole lot in return. When you absolutely, positively need something more ambitious, it's time to look into AppleScript Studio.

And we're saving ourselves a step by building the dialog's text on the fly. The ampersands are your signal to AppleScript that you'd like all those items schmooshed together into one string of text.

CHECK YOUR TELLS BEFORE BEGINNING THE HUMAN SACRIFICES

Often, when a script *just isn't working* and I'm so frustrated that I'm willing to sacrifice a living creature to the Cloven-Hooved One to make it work (a moth, maybe — let's not lose our heads here), I just tighten up my tell blocks and the problem magically disappears. Just something to keep in mind before you consider offing an intern to get a project finished.

When you're stuck with me as a dinner guest, there are three topics you don't want to introduce: The JFK assassination (Oswald did it and he acted alone; for bonus points, let me run down the 12 most popular conspiracy theories and why they're utterly baloneyous), the Apollo program (yes, I was born 30 or 40 years too late, but I'm still convinced that if I do a few pushups I might make it as a backup LEM pilot on Apollo 18), and AppleScript.

I'm as enthusiastic about AppleScript as I am about my favorite books and movies. You've never experienced this thing? Oh, sit down, you poor, poor man or woman; your whole life has been a mere prelude to seeing *The Stunt Man* or reading *The Code of the Woosters* for the first time.

AppleScript is *exactly* what's right about computers in general and the Mac in particular. There are some scripting-ish solutions for Windows, but no other platform brings such a level of power to such a low level of user expertise. AppleScript means never having to say, "I *wish* I could do this on my Mac."