1 Introduction to AppleScript Programming

AppleScript is an Open Scripting Architecture (OSA)–compliant command language that can communicate with and control scriptable applications and scriptable features of Macintosh Operating System (Mac OS) X.

The OSA is a mechanism within the Mac OS that provides a library of functions and allows inter-application communication and control by sending and receiving messages called Apple Events.

An *Apple Event* is a basic message exchange system of the OSA that is used to send instructions to an application and optionally, send back a result. Apple Events can be used to control inter-process communication within an application, between applications on a single computer, and between applications on a remote computer. Figure 1.1 illustrates the path of an Apple Event message.

Since 1994, when System 7.5 was released, most Mac users have been unaware that they send Apple Events every day. For example, each time they double-click a document to open it, an Apple Event is responsible for instructing the appropriate application to launch and to open the file, as shown in Figure 1.2.

AppleScript provides an easy to learn, English-like language that enables users to write scripts that send and receive Apple Events. Each script acts like a new feature of the OS or an application. Scripts can integrate third-party applications, creating custom solutions that perform very specific tasks.

Because of its English-like syntax, even novice programmers can build scripts to perform virtually any function. With pervasive support of AppleScript throughout the Mac OS and many third-party applications, it is the ideal platform to create efficiency-rich, workflow automation solutions.

This amazing and award-winning technology provides a simple and affordable way to automate repetitive computing tasks and leave users free to focus their attention on more creative tasks.

1 In This Chapter

- An introduction to AppleScript
- Locating AppleScript applications and other resources
- Looking at AppleScript's resources and unique characteristics
- Who uses AppleScript and what they automate
- Looking at AppleScript's influence

Figure 1.1

The path of an Apple Event message sending a command to an application

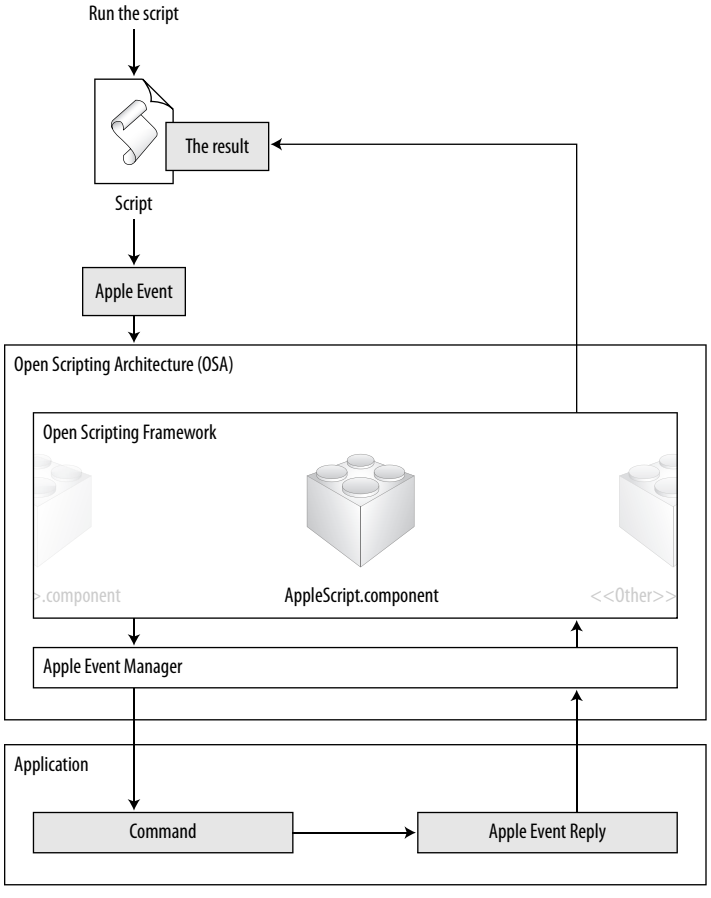
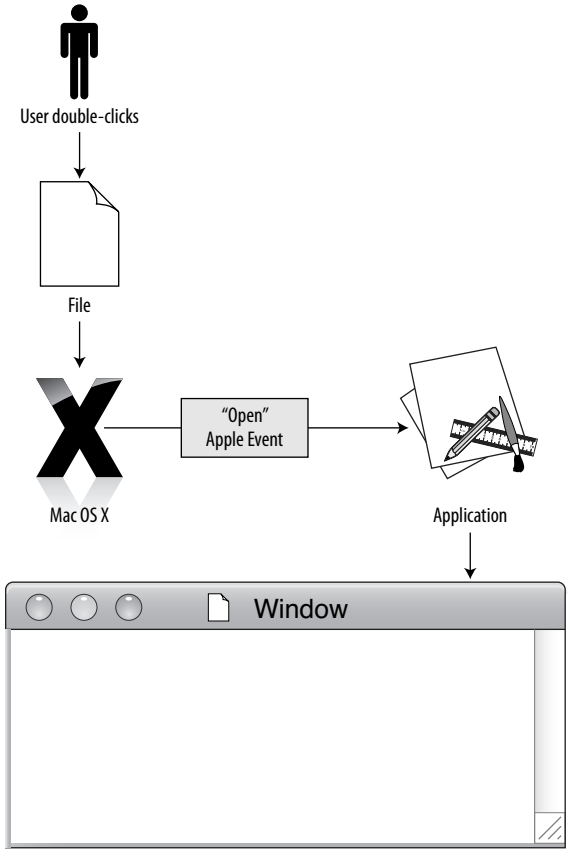


Figure 1.2

The underpinnings of an Apple Event opening a document

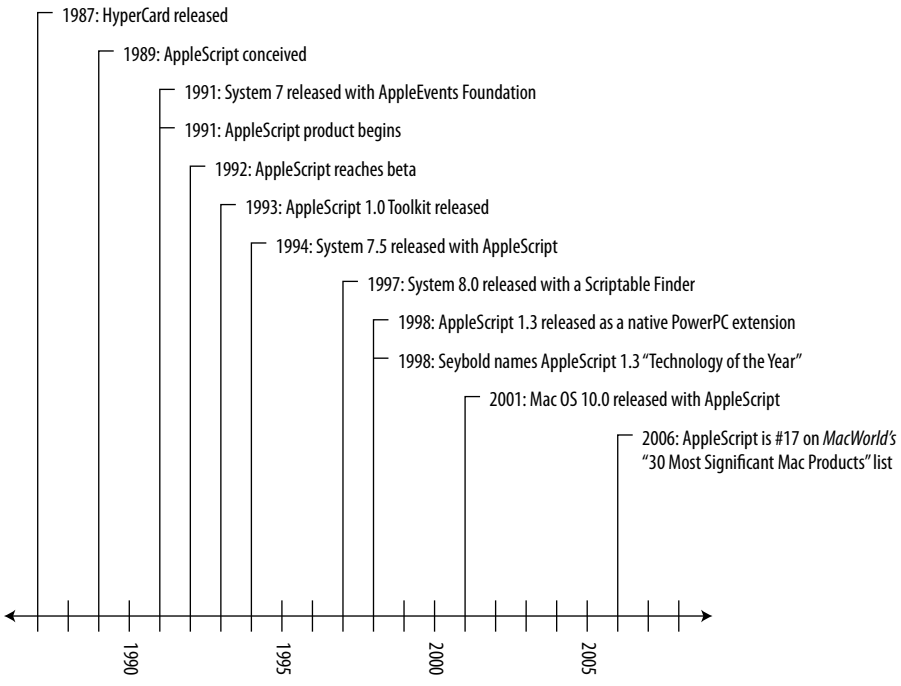


A Brief History of AppleScript

AppleScript and its associated tools were conceived, designed, and implemented between 1989 and 1993. It was a long-term investment in fundamental infrastructure that matured over a span of several years (see Figure 1.3).

Figure 1.3

A timeline of the history of AppleScript



Often considered a precursor to and inspiration for AppleScript, HyperCard was released in 1987. This software enabled novice programmers to rapidly create custom tools that would carry out a set of specific processes. Featuring an easy-to-learn, English-like scripting language called HyperTalk, it was easier to learn and use than other programming languages available at that time.

AppleScript was officially conceived in 1989 as a research project by the Advanced Technology Group (ATG) at Apple Computer and was code-named "Family Farm." The research team was led by Larry Tesler and included Mike Farr, Mitchell Gass, Mike Gough, Jed Harris, Al Hoffman, Ruben Kleiman, Edmund Lai, and Frank Ludolph. Their goal was to create a new system-level development environment for the Mac OS that would allow for inter-application communication and control and provide a user-level language. The original group was disbanded in mid-1990 and new teams were assembled to design and implement the ideas first conceived.

The first step was the development of Apple Events, which is the inter-application communication foundation of AppleScript in the OS. Written in Pascal, like much of the Mac OS at the time, this foundation needed to be in place before the development of AppleScript could begin. The AppleScript project officially began in April 1991, just months before Mac OS 7, when the new Apple Events foundation was released.

NOTE

The AppleScript project was code named "Gustav" after a team member's dog.

In September 1992, AppleScript reached beta. However, in January 1993, the original team was disbanded when several leaders left the project. It wasn't until April of that year that the AppleScript 1.0 Developer's Toolkit shipped as a stand-alone product that could be installed on any Mac running System 7. In September, AppleScript version 1.1 was included as part of System 7.1.1 (System 7 Pro). In December, the first "end user" release — AppleScript 1.1 Developer's Toolkit and Scripting Kit — was released. Finally, in 1994, AppleScript was ready to revolutionize how people use computers when it took its place as an official part of Macintosh System 7.5.

Since that time, AppleScript has slowly evolved into the invaluable tool that we know today. In 1997, the Macintosh Finder finally became scriptable, eliminating the need to use the Finder scripting extension. When Macintosh OS 8.0 was released in July 1997, it included AppleScript version 1.1.2 with many minor improvements.

NOTE

In 1997, Apple had plans to eliminate AppleScript in order to cut expenses but, thankfully, this plan was thwarted by a campaign by loyal users of the technology.

In October 1998, AppleScript 1.3 was released, recompiled as a native PowerPC extension and included Unicode support. In that year, Steve Jobs demonstrated AppleScript at Seybold, and *Macworld* magazine named AppleScript 1.3 the "Technology of the Year." In 2006, AppleScript held position #17 on *Macworld*'s list of the 30 most significant Mac products ever.

NOTE

Read the entire history of AppleScript at `www.cs.utexas.edu/~wcook/Drafts/2006/ashopl.pdf`.

A 1999 technology study by research firm GISTICS estimated that AppleScript produced more than $100 million in annual savings for North American media firms. Today, Google returns more than two million results when searching for the word "AppleScript."

In Mac OS 10.6, released in 2009, AppleScript, Standard Additions, and all AppleScript-related system applications, such as System Events, are now 64-bit capable.

The technology has flourished and now boasts a thriving and happily efficient user base.

Finding AppleScript Resources

AppleScript is made up of various elements located on each Mac computer. These elements include applications, scripting additions, and components.

Applications

AppleScript developers use two applications: the AppleScript Editor and the Folder Actions Setup application.

NOTE

Mac OS 10.5 included a folder called "AppleScript" inside the `/Applications/` folder that contained three applications: Script Editor, AppleScript Utility, and Folder Action Setup. Mac OS 10.6 doesn't include this folder. The Script Editor is now the "AppleScript Editor" and is in the `/Utilities/` folder; the options accessible from the AppleScript Utility are now in the Editor's preference panel; and Folder Action Setup is now in the `/System/Library/CoreServices` folder.

AppleScript Editor

Probably the most important application in the AppleScript toolbox is the AppleScript Editor, which is located in the `/Applications/Utilities/` folder. This application is used to create, write, edit, compile, run, and save scripts. It contains many features that assist a developer in learning the language, writing scripts, and exploring the command library of scriptable third-party applications.

CROSS-REF

See Chapter 6 for more information about using the AppleScript Editor.

Folder Actions Setup

The Folder Actions Setup application, located in `/System/Library/CoreServices/`, is used to assign script actions to folders. This enables a script to respond to various folder actions, such as the arrival or removal of a file or folder, and perform a sequence of automated tasks on it.

NOTE

You can access the Folder Actions Setup application by clicking a folder while pressing the Ctrl key or by clicking the right button on your mouse and selecting the Folder Actions Setup option from the contextual menu.

The Folder Actions Setup window, shown in Figure 1.4, lets you enable and disable folder actions globally as well as add, show, and remove folders on a computer. Once you have added a folder, you can attach one or more scripts to it.

CROSS-REF

See Chapter 16 for more information about using Folder Actions.

Figure 1.4

The Folder Actions Setup window

Scripting additions

A *scripting addition* is used to extend the AppleScript language by providing a set of additional commands. Scripting additions can be stored in several locations on a computer. Apple includes several scripting additions in the OS and you can find additional third-party scripting additions on the Internet.

CROSS-REF

See Chapter 16 for more information about installing and using scripting additions.

TIP

A set of sample scripts provided by Apple and installed as part of the Mac OS 10.6 installation is located at `/System/Library/Scripts/`.

Components

Components are files that provide basic functionality for AppleScript, Apple Events, and other OSA–related languages. While the process of using or developing scripts does not require you to be concerned with these components, they are provided in this book for informational purposes only. Except when adding or removing additional language components, such as JavaScript, you should never attempt to remove, modify, or be concerned with the whereabouts of any of these components.

The Apple Event Manager provides an application programming interface (API) for sending and receiving Apple Events, thereby providing support for the creation of scriptable applications.

It exists as part of the `CoreSErvices.framework` and is called the `AE.framework`. This is important for those creating scriptable applications but not important for those writing scripts with AppleScript.

Likewise, the `OpenScripting.framework` is a part of the `Carbon.framework` and is not something AppleScript users and developers need to worry about. It defines data structures, routines, and resources that support scripting components regardless of the language. It also compiles, executes, loads, and stores scripts.

The `AppleScript.component` file, the default OSA scripting language component provided by Apple, enables a computer to use the AppleScript language. It is located at `/System/Library/Components`.

Other OSA component files, such as the `JavaScript.component`, can be installed in `~/Library/Components` for each user account that will use it. If your computer is connected to an office network, you may need to contact your network administrator before installing additional OSA components.

Understanding the Unique Characteristics of AppleScript

While old-fashioned macro recording utilities were quite useful in their time — they could simulate a series of literal keystrokes and mouse clicks, respectively — it was difficult to use them in a dynamic and practical manner. With AppleScript you can not only automate a sequence of literal actions, but also you can create a dynamic script that includes logical branches, variable content, and options for different behavior depending on specific conditions. This gives AppleScript the power of a real programming language.

AppleScript possesses more unique characteristics that add to its appeal, such as its English-like syntax, the fact that it is universally open-ended, its deep level of access into the Mac OS framework and the frameworks of third-party applications, and its consistency between OS updates.

English-like syntax

One of the most unique characteristics of AppleScript is its English-like syntax. While some detractors might say it is not even close to "natural spoken English," most would agree that it is certainly more like a spoken language than most other scripting and programming languages. The difference in syntax can be illustrated with a few simple examples.

The following examples present a sort of Rosetta Stone of programming languages. The code in each example performs exactly the same function: It builds a text-based list of numbers within a range specified by two variables. At the end of each script, the resulting value will be a sequence of numbers from 25 to 30 with a carriage return after each number.

Listing 1.1 illustrates the English-like language used in AppleScript. Notice how setting the content of a variable doesn't require any code-like declarations or explicit identification of the initial value's data class. Also, there is no need to worry about line endings; just type a return and keep on typing.

Listing 1.1

AppleScript

```
set numLength to 5
set numStart to 25
set textResults to numStart as string
if numLength > 1 then
    repeat with a from 1 to numLength
        set numStart to numStart + 1
        set textResults to textResults & return & numStart
    end repeat
end if
```

The code script shown in Listing 1.2 performs the same functions with JavaScript. Putting a value into a variable is a little less English-like. Like many other languages, JavaScript requires a special line ending, which in this case is a semi-colon. Also, the repeat loop is more cryptically phrased and, therefore, less clear than the AppleScript example.

Listing 1.2

JavaScript

```
var numLength = 5;
var numStart = 25;
var textResults = numStart;
for (var a=numStart+1; a<=(numStart+numLength); a++) {
     textResults=textResults + '<br>' + a;
}
```

The code in Listing 1.3 performs the same functions using the REALBasic language. Notice that the variable declarations are more complicated and require the data class of each must be specifically stated prior to placing a value within them.

Listing 1.3

REALBasic

```
Dim numLength as Integer
Dim numStart as Integer
Dim textResults as String
Dim a as Integer
numLength = 5
numStart = 25
textResults= str(numStart)
if numLength > 1 then
  For a=1 to numLength
    numStart=numStart+1
    textResults = textResults + Chr(13) + str(numStart)
  Next
end if
```

Finally, the code shown in Listing 1.4 performs the same functions with PHP (Hypertext Preprocessor). Like JavaScript, it requires line endings and brackets enclosing a more cryptic combination if-then and repeat function.

Listing 1.4

PHP

```
$numLength = 5;
$numStart = 25;
$textResults = $numStart;  //no coercion is necessary
if ($numLength > 1) {
for ($a = $numStart+1; $a <= ($numStart + $numLength); $a++) {
   $textResults = $textResults . "<br>\n". $a;

}
}
```

Certainly, the development time required to build a script always depends on the complexity of the tasks to be automated. However, AppleScript's English-like language and automatic handling of many fundamental programming chores significantly reduce the time required to develop a solution.

NOTE

The code in Listings 1.1, 1.2, 1.3, and 1.4 are available for download from the book's Web site at `www.wiley devreference.com`.

Universally open-ended

Another important characteristic of AppleScript is its success as an open-ended, universal scripting language. Before AppleScript, the few applications that enabled users to automate tasks each had their own way of doing things. In today's modern Internet-driven world full of standard languages, it is hard to imagine a time when each application had its own set of rules. Imagine a nightmarish world in which every Web site had its own language and, therefore, its own browser. Perhaps that is a bit dramatic, but that is not totally unlike the way things were with automation.

In a world without AppleScript, each application would need its own macro language and features. To use them, you would have to learn and remember each of them from any number of applications. Creating a macro that works across multiple applications would be difficult to virtually impossible. Some software might try to bridge this gap, but it could break with each OS upgrade and generally would be more trouble than it's worth.

AppleScript addresses such nightmares by implementing a universal language with which any and all applications could communicate. An application possesses its own object model that can be automated by AppleScript, specifying the objects, properties, and commands that it can work with. In a way, each application's script dictionary becomes an extension of the AppleScript language, offering its unique services to a script in a familiar way.

Today, virtually every application developed for the Mac has some scriptable features. Widely used applications like Adobe's Creative Suite have extensive support for automating almost every conceivable feature. For those applications that don't support scripting, or to access a command not made accessible to AppleScript, Apple developed *System Events*. This amazing scripting addition enables AppleScript to simulate mouse clicks, keystrokes, and menu selections in many un-scriptable applications, thereby allowing you to script the un-scriptable.

CROSS-REF

See Chapter 20 for more information about System Events.

Deep level of access

AppleScript enables scripts to tap into many of the technologies of Mac OS X with its unique ease of use. For example, the `do shell script` command in the Standard Scripting Additions enables a script to run any shell script.

CROSS-REF
See Chapter 16 for more information about using AppleScript to perform shell scripts.

AppleScript also has access to many of the frameworks created by Apple for their applications and for third-party developers. These are made available to scripts with Scripting Additions. The Image Events scripting addition enables scripts to access the Core Image framework of the OS. Also, soon after the Core Data framework was developed, the Database Events scripting addition became available.

Consistency maintained between updates

Because AppleScript is built as a clean and easy-to-use layer on top of the OS that obscures the complex connection to the lower-level architecture, scripts typically continue to work from one OS update to the next. While major OS changes typically require bug fixes or complete overhauls of third-party applications, scripts are usually immune to such things. Even the transition from the "Classic" OS (System 9) to Mac OS X didn't require much more than a recompiling and resaving of older scripts.

When third-party applications require major rewrites or make changes to their object model and command libraries, it can break scripts. However, most applications eventually maintain a relatively consistent language for objects, their properties, and the commands they support.

CROSS-REF
See Chapter 17 to learn about the object model of third-party applications.

Exploring the Uses and Users of AppleScript

As a flexible, inter-application scripting language, AppleScript has a virtually unlimited number of uses and can be used by every user of a Mac computer. Even though many users will not take the time to learn the language and write their own scripts, they can still use the scripts written by others.

TIP
Always use caution when using scripts written by someone else. Be sure they are reputable or have a personal connection to you. A script could potentially perform malicious or virus-like behavior.

Scripts can react to various stimuli. Double-clicking a script application can begin an automated process while another application begins processing files a user drops on it. Scripts can be attached to the script menu, embedded into application menus and palettes, and on a Finder

window toolbar. They can be launched by an iCal alarm or whenever files or folders appear in a watched folder. Scripts can talk to other scripts on a user's computer or other computers across the office or the Internet. Scripts can run day and night, constantly watching, checking, processing, and reporting on anything that anyone takes the time to develop.

CROSS-REF

See Chapter 3 to learn more about AppleScript deployment options.

Uses for AppleScript

While some scripts may possess wider appeal and mass-market usefulness, typically most focus on a specific company's unique workflow. They can integrate third-party applications in ways that would not be practical or economical for mass production. Every manual task performed on a Mac computer can be automated with AppleScript, leaving its limitations to that of your imagination.

The following examples provide a glimpse of the limitless activities that can be automated with AppleScript.

File processing

There are numerous ways to streamline user interaction with files:

- **Image files can be opened, resized, flattened, modified with filters, and saved into one or more new files.** A script can automatically create preview images for the Web and high-resolution files for print. It can make "decisions" and vary the processing tasks based on a characteristic of the file, such as the file name, layer names, size, or page orientation. Individual files can be merged into a preview sheet or split into separate files. Any of these tasks can be automatically performed on a folder of 10 images or 10,000.
- **PDF files can be manipulated with AppleScript to suit the needs of even the most complex workflow.** Scripts can add or remove pages, stamps, comments, and more. Separate files can be merged into a single file while multiple page files can be split into individual files. A script can generate PDFs, print them, and place them in a folder for human review. The content of the files can be resumes, books, instruction manuals, fact sheets, and countless others. A script can detect and process each with a separate set of business logic.
- **Scripts can manipulate text-based data files of any type.** They can change delimiters to comma, tab, or fixed width text. They can extract specific rows or columns based on date ranges, product numbers, or customer name and then generate a new file with only the desired information. The files can be resorted, have data added or removed, and can be checked for common errors. Characters can be replaced with other characters to conform to importing regulations of databases, hypertext markup language (HTML) language rules or your company business rules.

E-mail processing

On an automated e-mail account or a user's computer, AppleScript can help manage a crowded inbox and assist in sending bulk e-mail:

- **Scripts can scan, read, and take predetermined actions on incoming messages in many popular e-mail applications.** They can also help you prepare, create, and customize messages to a batch of contacts.
- **Scripts can detect, read, parse, and route data from a Web form to a database.** On a user's computer, this can be done with a mail rule that runs an AppleScript when Mail detects a web form email. A server with a fully autonomous script and dedicated e-mail account can constantly monitor the inbox for various messages from different Web forms. Once an incoming message has been processed, an alert can be sent to key personnel notifying them of the new information.
- **Scripts can help send e-mail, especially in batches.** Sending a single e-mail message to a huge list can be less impersonal by a script that sends a separate e-mail to each person on a list with a personalized greeting. Scripts can even build a unique e-mail based on customer ordering patterns, available material, and encoded business rules that describe how the recipient and the personal message should be matched up.

Desktop publishing automation

Using scripts to automate laborious desktop publishing tasks could fill endless volumes. Scripts can build simple pages with a variety of styles or build complex pages with hundreds or thousands of specific business rules encoded. They can build catalogs, complex tables, and charts, and automatically import text and images from various sources. They do all of this while reducing errors, improving consistency, and flawlessly enforcing style guidelines. In addition:

- **Scripts can build catalog files of any type.** Even a catalog with uniform styling and formatting can be a burdensome task to build by hand. A user must copy, paste, position, and style text, then search for and import the appropriate artwork or images. When catalog pages vary by category or product data, scripts eliminate the need to remember numerous layout design rules. They can quickly and accurately locate the text and images necessary to build richly styled and dynamically spaced product information into a copy of the appropriate template. Watching for changes in category or other group fields, a script can diverge the styles and structure based on the appropriate rules. For example, each product category could require the insertion of a title page with a leading blank page or it might prompt the creation of an entire new document for each section.
- **Scripts can scan documents and build indexes or tables of contents.** They can cross-reference products across corresponding documents, such as a catalog that has a separate price book.

Some processes are too daunting to contemplate without scripts, such as modifying hundreds of price changes or adding or removing products. Instead of manually updating pricing and squeezing new products into spaces left by old ones, a script enables you to simply build an entirely new catalog.

Monitoring sales performance of a catalog or other advertisement can be difficult when simply looking at the numbers. The context of the sale — such as the page the customer was reading when ordering — can be just as important to determine which strategies are working and which are not. A script can open documents, color-code product information based on actual sales, expand the page size, and integrate key sales data right on the page. Imagine the advantage of not just knowing how well a product did but also why it did so well when placed in a certain location on a page.

Central processing and resource monitoring

Even as a script saves you time, it might be wasting it. Rather than sitting idle, mesmerized as you watch a script take control of the applications on your computer, you might consider setting up an automation server. Just as a file server, mail server, or Web server centrally processes information for its respective services, an *automation server* is a shared computer on a network that centrally processes all of your automated activity.

TIP

A Windows-based company can still benefit from AppleScript automation with a single Mac acting as an automation server. All of the scripts process incoming files and then push them back out to the users. This can even be a great way to help convince management to consider a full switch to Mac OS X.

Instructions can be sent to an automation server is many different ways. The most common is to simply move files for processing into folders that are being watched by a script. When a new file appears, the script "wakes up" and begins processing them. Likewise, users might indicate through a database or a small custom script which catalog they want to build. The script would receive these instructions and begin the process. With an automation server watching folders, multiple users can send files dozens or hundreds of files at the same time. They are queued up in the watched folder and are processed while users work on other tasks, go to lunch, or leave for the day. The scripts keep on working.

Scripts that monitor valuable resources and ensure that key systems are up and running are best suited when they are running on an automation server. For example:

- **Scripts can search, sort, and extract data from databases and then create and send periodic e-mail reports to key personnel.** A script can send a daily sales report, tabulated survey data, a summary of sales leads, and any other kind of report you can generate manually. This saves users the time it takes to locate, open, and create a report query, and a report can even be generated early in the morning before the staff arrives.
- **A script can scan a file server to ensure that folder structure rules are being followed.** If misfiled items are detected, it might send an e-mail alert listing the offending items and can even include a URL link so the reader can quickly navigate to the item to remedy the situation. It might send one e-mail to graphic artists informing them that images are supposed to be in a subfolder called Product Images and another to the sales team indicating that spreadsheets need to be in the Reports folder. Managers can be CC'd on these, notifying them of issues and empowering them when they need to enforce compliance.

Users of AppleScript

While the AppleScript language is designed to be accessible for casual programmers of varying skill level to write scripts, it can be used by anyone. Scripts built with AppleScript can dramatically increase a user's productivity while simultaneously reducing stress and errors and increasing adherence to business rules Scripts free up users to focus on details that cannot be automated and require human attention.

Individuals

Whether an employee works for a small, family-owned business or a Fortune 500 corporation, he can use AppleScript solutions to streamline any repetitive task. Any task, large or small, involving at least some repetition can typically benefit from automation. These custom tools can be built to help remind, assist, and notify this employee with his daily work.

Teams

As part of a team, individuals need to focus not only on their work but also keep in mind how their work integrates with their co-workers and the goals of the team. When each member of the group has access to a well-designed set of tools that helps her get her work done quickly while adhering to company policy, the team will benefit enormously. Scripts can help team members navigate project folders; store files in the correct location, with approved naming conventions; and notify others about their status. Scripts can remove much of the stressful and redundant tasks that can sap the creative effort a team needs to thrive. An overworked team might rejoice over more work as long as they have the tools necessary to get it done.

Companies

The breadth of a scripted solution has no limitations. As long as it is worth the development effort required, automation can provide solutions to individuals, teams, divisions, and more. Any and every department within a company can benefit by automation. Whether it is sales, marketing, design, inventory, shipping, or any other department, AppleScript provides a powerfully scalable technology that can be a solution to almost every repetitive problem you can imagine, regardless of the industry. Scripts enable companies to improve the quality and quantity of work while reducing costs, errors, and stress, leaving in their wake an environment conducive to creative thought.

Respecting the Power of AppleScript

In the past, especially the early years of its genesis, AppleScript was often given a bad rap. Some would say it was only a script language, not a programming language. Others referred to it as buggy, slow, or weak. Some even predicted its obsolescence once Apple's OpenDoc technology emerged. Ironically, it was OpenDoc that was scraped in 1997.

Sometimes the criticism would come from programmers who promoted costly development in other languages, while other times, it came from developers or users who were "Windows centric"

and looked down on the Mac and Apple. In this age of iPods, iPhones, and the witty "Mac versus PC" commercials, it may be difficult to believe this, but at the time, it was quite fashionable to predict doom and gloom for Apple and all of her wares, both hard and soft.

Admittedly, at least some of the criticism may have been earned, at least at first. There were bugs and speed issues in the early years, but the same was true of the Mac OS. It relied a lot on third-party scriptable applications, which were slow to embrace it. But slowly, over time, the bugs were fixed, the computers became faster, and the scriptable applications began appearing everywhere. The language made the transition to the new and modern OS X and continued to evolve and to embrace new technologies.

AppleScript survived and, more importantly, it flourished. It won awards, was the focus of "return on investment" studies, and finally silenced the critics. Today a skillful developer can automate a large portion of a workflow, creating an unprecedented boom in efficiency.

It is finally time that AppleScript gets the respect it is due as a truly unique, powerful, and revolutionary technology that remains hard pressed to find an adequate rival on any platform. It is easy to learn, easy to use, and can do almost anything. It has been used to build countless catalogs, brochures, and factsheets. It has processed images, text, numbers, and more. It has been the workhorse of the computer age, doing the heavy lifting between applications, processes, and users. All indications show it will continue doing so for the foreseeable future.

Summary

In this chapter, you were introduced to AppleScript. I discussed its historic design and gradual percolation into many corners of the Mac OS. You became acquainted with the location of key resources, including applications and scripting additions. I also discussed the unique characteristics of AppleScript and explored the uses for, and users of, this amazing technology.