AppleScript Overview

Scripting & Automation > AppleScript

2007-10-31

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Introduction to AppleScript Overview 7

Who Should Read This Document 7
Organization of This Document 7
See Also 8

About AppleScript 9

When to Use AppleScript 10
Limitations of AppleScript 10

Open Scripting Architecture 13

The Parts of the Open Scripting Architecture 13
Apple Events 14
Apple Events Sent by the Mac OS 14
Script Execution in the Open Scripting Architecture 15
Extending AppleScript with Coercions, Scripting Additions, and Faceless Background Applications 16
16
- Coercions 16
- Scripting Additions 16
- Faceless Background Applications 17

Scripting with AppleScript 19

Script Editor and AppleScript Scripts 19
- A Simple AppleScript Script 19
- Script Editor 20
- Interacting with the User in Scripts 21

What You Can Control with Scripts 22
Using AppleScript with Web Services 22
Using AppleScript with Other Scripting Systems 23
- Executing Shell Commands From AppleScript Scripts 23
- Executing AppleScript Scripts as Shell Commands 23
- Scripting the Terminal Application 23
- Using Other Scripting Languages 24

Scriptable Applications 25

Specifying Scripting Terminology 25
Determining What to Make Scriptable 26
Registering to Receive Apple Events 26

Resolving Objects in the Application 26
Recording 27
Creating and Sending Apple Events 27
Executing Scripts 28
Summary of Operations in a Scriptable Application 28
Mac OS X Support for Creating Scriptable Applications 29
Support for Carbon Applications 29
Support for Cocoa Applications 30

Scripting Bridge 31

Automator 33

AppleScript Studio 35

AppleScript Utilities and Applications 37

AppleScript Utility 37
Folder Actions Setup 37
System Events and GUI Scripting 38
Image Events 39
Database Events 40

Document Revision History 41

Figures and Listings

Open Scripting Architecture 13

Figure 1 How parts of the OSA work together in executing scripts 15

Scripting with AppleScript 19

Figure 1 The Finder dictionary in Script Editor (in Mac OS X v10.5) 21
Listing 1 A script that counts the files in the Applications folder 19

Introduction to AppleScript Overview

AppleScript Overview provides a high-level overview of AppleScript and its related technologies to help you determine where you can use them in your work.

Note: For information on the universe of scripting technologies available on Mac OS X, see *Getting Started With Scripting & Automation*.

AppleScript is a scripting language that makes possible direct control of scriptable applications and of many parts of the Mac OS. A scriptable application is one that makes its operations and data available in response to AppleScript messages, called Apple events.

With scriptable applications, users can write scripts to automate operations, while developers can use AppleScript as an aid to rapid prototyping and automated testing. Developers can also use technologies including Apple events, AppleScript, Automator, and Scripting Bridge, to take advantage of services provided by other applications and by the Mac OS.

AppleScript and Apple events are based on the Open Scripting Architecture, which is implemented by several Mac OS X frameworks. Apple provides a number of additional applications and technologies that enhance AppleScript or take advantage of its features.

Who Should Read This Document

You should read *AppleScript Overview* to get a broad understanding of AppleScript and related automation technologies, and to determine where they fit into your development process.

This document may also be of interest if you write AppleScript scripts and would like to know more about the technology behind them.

AppleScript Overview is intended for a general developer audience, but experience with some kind of scripting language is helpful. If you are starting from scratch, see *Getting Started with AppleScript*.

Organization of This Document

This document contains the following:

- "About AppleScript" (page 9) introduces AppleScript, describes when you might use it, and notes some limitations.
- "Open Scripting Architecture" (page 13) describes the underlying technology used to implement AppleScript and Apple events. It also describes how to extend AppleScript.

- "Scripting with AppleScript" (page 19) provides a brief description of how you work with AppleScript scripts. It also describes options for combining AppleScript scripting with other kinds of scripting.
- "Scriptable Applications" (page 25) explains how scriptable applications work, including how they specify their scripting terminology, and describes the programming resources available for creating scriptable applications.
- "Scripting Bridge" (page 31) describes a technology available starting in Mac OS X version 10.5 that generates Objective-C API for accessing scriptable applications.
- "Automator" (page 33) describes Apple's graphical automation program and how developers can take advantage of it.
- "AppleScript Studio" (page 35) describes a technology for creating Mac OS X applications that can provide sophisticated user interfaces and be driven by AppleScript scripts.
- "AppleScript Utilities and Applications" (page 37) describes utilities and applications that work with AppleScript or provide additional features you can use in AppleScript scripts.

See Also

You can find additional introductory information on AppleScript and related technologies in *Getting Started with AppleScript*.

There are also links to related documentation throughout *AppleScript Overview*.

About AppleScript

AppleScript is a scripting language that provides direct control of scriptable applications and scriptable parts of the Mac OS. A **scriptable application** is one that can respond to a variety of Apple events by performing operations or supplying data. An **Apple event** is a type of interprocess message that can encapsulate commands and data of arbitrary complexity. By providing an API that supports these mechanisms, the "Open Scripting Architecture" (page 13) makes possible one of the most powerful features in Mac OS X—the ability to write scripts that automate operations with multiple applications.

You can use AppleScript scripts to perform repetitive tasks, automate complex workflows, control applications on local or remote computers, and access web services. Because script writers (or scripters) can access features in any scriptable application, they can combine features from many applications. For example, a script might make remote procedure calls to a web service to get stock quotes, add the current stock prices to a database, then graph information from the database in a spreadsheet application. From controlling an image-processing workflow to performing quality assurance testing for a suite of applications, AppleScript makes automation possible.

While the AppleScript scripting language (described in AppleScript Language Guide, and in a number of detailed third-party books) uses an English-like terminology which may appear simple, it is a rich, object-oriented language, capable of performing complicated programming tasks. However, its real strength comes from providing access to the features available in scriptable applications. If you make your application scriptable, it will help scripters get their work done, and quite likely become indispensable to their work process.

The "Automator" (page 33) application, available starting in Mac OS X version 10.4, lets users work in a graphical interface to put together complex, automated workflows. Workflows consist of one or more actions, which are provided by Apple, by developers, and by scripters, and can be written in AppleScript and in other languages, including Objective-C. Starting in Mac OS X v10.5, developers can incorporate workflows directly in their applications, providing another mechanism for accessing features of other applications and the Mac OS.

"Scripting Bridge" (page 31), available starting in Mac OS X version 10.5, provides an automated process for creating an Objective-C interface to scriptable applications. This allows Cocoa applications and other Objective-C code to efficiently access features of scriptable applications, using native Objective-C syntax. Some other scripting languages, such as Ruby and Python, can use Scripting Bridge, but also have their own software bridges to access features of scriptable applications—for more information, see *Getting Started With Scripting & Automation*.

AppleScript has several other new or improved features in Mac OS X v10.5, including full support for Unicode text, additional support for identifying and working with application objects in scripts, 64-bit support, more accurate and useful error messages, and additional scriptability in Apple technologies such as iChat and the Dock. For more information, see AppleScript Features.

When to Use AppleScript

The following are common scenarios in which you might use AppleScript or related technologies in your development work.

- You're creating or updating an application for Mac OS X and you want it to be scriptable. As a scriptable application, users can invoke it in their AppleScript scripts and you can write scripts to perform automated testing during development. You can also make the application accessible to Automator users, or access it through Apple's Scripting Bridge, or through open source bridge technologies, using languages such as Objective-C, Ruby, and Python.

  For information on these technologies, see "Scriptable Applications" (page 25), "Scripting Bridge" (page 31), and "Automator" (page 33) in this document and the related learning paths in *Getting Started With Scripting & Automation* and *Getting Started with AppleScript*.

  "Framework and Language Support," in About Apple Events in *Apple Events Programming Guide*, describes trade-offs involved in developing scriptable applications in procedural and object-oriented languages, and in using support provided by the Carbon APIs or the Cocoa application framework.

- You're interested in automating repetitive operations, whether in development or in other work, using scripts or "Automator" (page 33) workflows.

  For information on working with scripts, see "Scripting with AppleScript" (page 19) in this document, as well the learning paths in *Getting Started with AppleScript*.

  To learn about applications and technologies that extend AppleScript and help it work with graphic images, XML, property lists, databases, and other technologies, see "AppleScript Utilities and Applications" (page 37).

- You're looking for a way to do rapid application development. For example, you may want to quickly create a user interface to test your code, or add an interface to a faceless development tool.

  "AppleScript Studio" (page 35) allows you to create applications with complex user interfaces that are driven by AppleScript scripts or by code written in other languages (or by a combination of the two). Developing a Studio test application with Xcode and Interface Builder, you can be up and running in a minimum of time.

  Again, you'll find a starting point for working with AppleScript Studio in the learning paths in *Getting Started with AppleScript*.

Limitations of AppleScript

The AppleScript scripting language excels in its ability to call on multiple applications, but was not designed to perform task-specific functions itself. So, for example, you cannot use AppleScript to efficiently perform intensive math operations or lengthy text processing. However, you can use AppleScript in combination with shell scripts, Perl scripts, and other scripting languages. This allows you to work with the most efficient language for the task at hand. For related information, see "Using AppleScript with Other Scripting Systems" (page 23).

While you can save AppleScript scripts as applications, AppleScript provides a minimum of user interface options. However, you can use "AppleScript Studio" (page 35) when you need to provide a more complex interface.

AppleScript relies on developers to build scriptability into their applications. However, a mechanism called GUI scripting, introduced with Mac OS X version 10.3, does allow some scripting of applications that do not contain code for responding to Apple events. For more information, see "System Events and GUI Scripting" (page 38).

Open Scripting Architecture

The **Open Scripting Architecture (OSA)** provides a standard and extensible mechanism for interapplication communication in Mac OS X. Communication takes place through the exchange of Apple events, a type of message designed to encapsulate commands and data of any complexity.

Apple events provide an event dispatching and data transport mechanism that can be used within a single application, between applications on the same computer, and between applications on different computers. The OSA defines data structures, a set of common terms, and a library of functions, so that applications can more easily create and send Apple events, as well as receive them and extract data from them.

Note: Apple events are not always the most efficient or appropriate mechanism for communicating between processes. Mac OS X offers other mechanisms, including distributed objects, notifications, sockets, ports, streams, shared memory, and Mach messaging. These mechanisms are described in "IPC and Notification Mechanisms" in System-Level Technologies in *Mac OS X Technology Overview*.

The OSA supports several powerful features in Mac OS X:

- the ability to create scriptable applications (described in "Scriptable Applications" (page 25))
- the ability for users to write scripts that combine operations from multiple scriptable applications
- the ability to communicate between applications with Apple events

The Parts of the Open Scripting Architecture

Applications that need full access to the Open Scripting Architecture can get it by linking with the Carbon framework. Some applications that work with Apple events (especially those with minimal user interface requirements) may be able to obtain all the services they need by linking to the Core Services framework.

Note: A **framework** is a type of bundle (or directory in the file system) that packages software with the resources that software requires, including the headers that define its interface. Frameworks are typically located in `/System/Library/Frameworks`, though they may be nested inside other frameworks.

The Open Scripting Architecture is made up of the following parts:

- The **Apple Event Manager** provides an API for sending and receiving Apple events and working with the information they contain. It supplies the underlying support for creating scriptable applications. It is implemented in `AE.framework`, a subframework of `CoreServices.framework`. (Prior to Mac OS X version 10.5, the `AE.framework` was a subframework of `ApplicationServices.framework`.)

  This framework also defines constants that developers can use to support a standard vocabulary for Apple events among different applications.

For API documentation, see *Apple Event Manager Reference*. For conceptual documentation and code samples, see *Apple Events Programming Guide*.

- The **Carbon umbrella framework** includes the HIToolbox framework, which in turn defines certain functions used in processing and sending Apple events (for example, in the header file `Interaction.h`).
- The **Open Scripting framework** defines standard data structures, routines, and resources for creating scripting components, which support scripting languages. Because of its standard interface, applications can interact with any scripting component, regardless of its language. This framework provides API for compiling, executing, loading, and storing scripts. It is implemented in `OpenScripting.framework`, a subframework of `Carbon.framework`.

  For documentation, see *Open Scripting Architecture Reference*.
- The AppleScript component (in `System/Library/Components`) implements the AppleScript language, which provides a way for scripts to control scriptable applications.

  The AppleScript language is described in *AppleScript Language Guide*, as well as in a number of third-party books.

Apple Events

The Apple event is the basic message for interprocess communication in the Open Scripting Architecture. With Apple events, you can gather all the data necessary to accomplish a high level task into a single package that can be passed across process boundaries, evaluated, and returned with results.

An Apple event consists of a series of nested data structures, each identified by one or more four-character codes (also referred to as Apple event codes). These data structures, as well as the codes and the header files in which they are defined, are described in *Apple Events Programming Guide*. That document also provides conceptual information about Apple events and programming examples that work with them. For a list of four-character codes and their related terminology used by Apple, see *AppleScript Terminology and Apple Event Codes Reference*. Your application can reuse these terms and codes whenever it performs an equivalent function.

Apple Events Sent by the Mac OS

The Mac OS takes advantage of Apple events to communicate with applications, such as to notify an application that it has been launched or should open or print a list of documents. Applications that present a graphical user interface must be able to respond to whichever of these events make sense for the application. For example, all such applications can be launched and quit, but some may not be able to open or print documents.

For detailed information on the events sent by the Mac OS and how to respond to them, see:

- For Carbon applications: "Handling Events Sent by the Mac OS" in Responding to Apple Events in *Apple Events Programming Guide*.
- For Cocoa applications: How Cocoa Applications Handle Apple Events in *Cocoa Scripting Guide*.

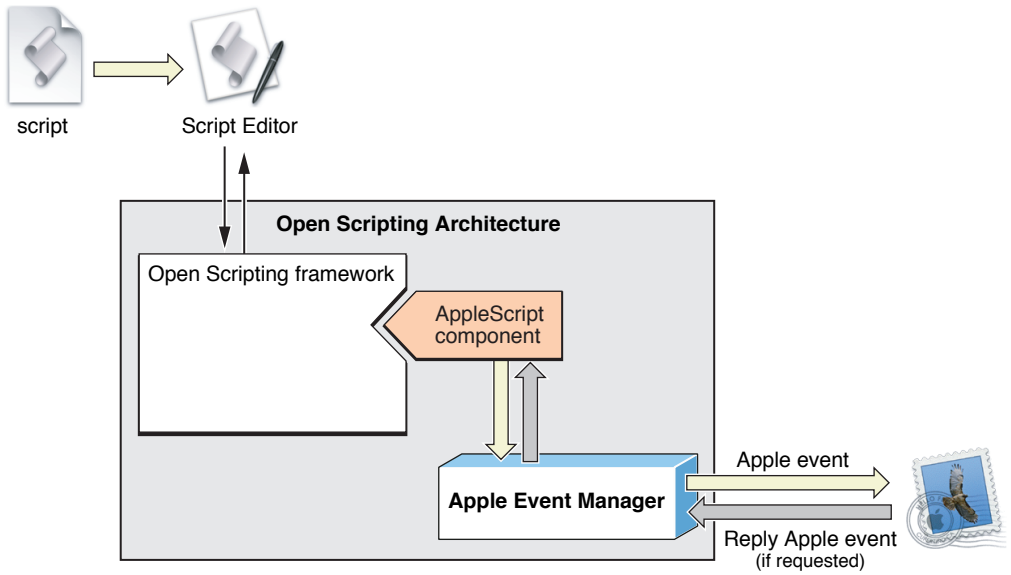
Script Execution in the Open Scripting Architecture

The Open Scripting Architecture allows users to control multiple applications with scripts written in a variety of scripting languages. Each scripting language has a corresponding scripting component. The AppleScript component supports the AppleScript language. When a scripting component executes a script, statements in the script may result in Apple events being sent to applications.

Although AppleScript is the most widely used language (and the only one provided by Apple), developers are free to use the Open Scripting Architecture to create scripting components for other scripting languages. Depending on the implementation, scripts written in these languages may be able to communicate with scriptable applications.

Figure 1 shows what happens when the "Script Editor" (page 20) application executes an AppleScript script that targets the Mail application. Script Editor calls functions in the Open Scripting framework. The Open Scripting framework communicates through the AppleScript component, which in turn uses the Apple Event Manager to send any required Apple events to the Mail application. If a reply is requested, the Mail application returns information in a reply Apple event.

Figure 1 How parts of the OSA work together in executing scripts



Applications can also call Apple Event Manager functions directly to send Apple events to other applications and get replies from them (not shown in Figure 1).

Extending AppleScript with Coercions, Scripting Additions, and Faceless Background Applications

Developers can extend AppleScript by creating bundles that provide command handlers and coercion handlers. The bundles can be applications or scripting additions. However, in many cases the best solution for extending AppleScript is to provide features through a faceless background application—that is, a sort of invisible server application.

Coercions

Coercion is the process of converting the information in an Apple event from one type to another. A **coercion handler** is a function that provides coercion between two (or possibly more) data types. Mac OS X provides default coercion between many standard types. For a complete listing, see Default Coercion Handlers in *Apple Events Programming Guide*.

Coercion is available to both scripts and applications. In a script, for example, the following statement coerces the numeric value 1234 to the string value "1234".

```
set myString to 1234 as text
```

A scriptable application can specify a type when it uses an Apple Event Manager function to extract data from an Apple event. If the data is not already in the specified type, the Apple Event Manager will attempt to coerce it to that type. An application can provide coercion handlers for its own data types, as described in Writing and Installing Coercion Handlers in *Apple Events Programming Guide*.

Scripting Additions

A **scripting addition** is a file or bundle that provides AppleScript commands or coercions. A single scripting addition can contain multiple handlers. For example, the Standard Additions scripting addition in Mac OS X (filename `StandardAdditions.osax`), includes commands for using the Clipboard, obtaining the path to a file, speaking text, executing shell scripts, and more. The commands provided by the Standard Additions are available to all scripts. To see what terminology a scripting addition provides, you can examine its dictionary, as described in "Displaying Scripting Dictionaries" (page 20).

Terms introduced by scripting additions exist in the same name space as AppleScript terms and application-defined terms. While this has the advantage of making a service globally available, it also means that terms from scripting additions can conflict with other terms, polluting the global name space. Debugging scripting additions can also be difficult. Because you can not simply set breakpoints in your own code, you may need to use sampling, profiling, and various other tools to determine where a problem lies. (See "Faceless Background Applications" (page 17) for an approach that avoids these problems.)

A scripting addition provides its services by installing event handlers (for commands) or coercion handlers (for coercions) in an application's system dispatch tables. The handlers for the Standard Additions (and for any other scripting additions installed by the Mac OS in `/System/Library/ScriptingAdditions`) get installed if the application calls API in the Open Scripting framework, or if the application specifically loads a scripting addition. An application can also specifically load other scripting additions from other locations.

For information on writing scripting additions, see Technical Note TN1164, Native Scripting Additions. For information on loading scripting additions, see Technical Q&A QA1070, Loading Scripting Additions Without Initializing Apple Script in Mac OS X.

Faceless Background Applications

A faceless background application (now more commonly referred to as an agent), is one that, as its name implies, runs in the background and has no visible user interface. By creating a scriptable agent, you can provide services without some of the disadvantages of scripting additions. For example, you can develop and debug in a standard application environment, and any terminology you provide does not pollute the global name space—it is available only within a script's `tell` statement that targets the agent.

You can install your agent directly, but if it is intended for use with another application, you can put it in the `Resources` folder of the application it supports. That promotes ease of use by allowing a drag-and-drop installation process, and will minimize users stumbling across the agent and asking "What is this for?" The agent will be launched whenever it is referenced in a `tell` statement. It can be told to quit, and you can also set it up to time out, so it can get unloaded when it is no longer in use.

Apple provides a number of scriptable services through agents, as described in "System Events and GUI Scripting" (page 38), "Image Events" (page 39), and "Database Events" (page 40). For example, scripts can use the System Events application to perform operations on property list files.

Scripting with AppleScript

The following is a brief introduction to AppleScript scripts, tools for working with them, and information on using AppleScript scripts together with other scripting systems. For related documents, see the learning paths in *Getting Started with AppleScript*.

Script Editor and AppleScript Scripts

An AppleScript script consists of one or more statements, written in a syntax described in AppleScript Language Guide (and in a number of third-party books). AppleScript defines some scripting terms, while scriptable applications and parts of the Mac OS specify additional terms for scriptable features they support. Scripting terminologies generally use common English words, resulting in scripts that are easier to read. For example, the following is a valid script statement:

```
display dialog "Welcome to AppleScript."
```

Users can compile and execute scripts with the "Script Editor" (page 20) application and can save them in various executable formats, including as stand-alone applications.

A Simple AppleScript Script

Listing 1 shows an AppleScript script that simply returns the number of files in the `Applications` folder on the current system disk (denoted by `startup disk`, a term understood by the Finder). If the folder cannot be found, the script returns a count of zero. This script counts just files in the specified folder, not folders or the files they might contain.

Listing 1 A script that counts the files in the Applications folder

```
tell application "Finder"
    if folder "Applications" of startup disk exists then
        return count files in folder "Applications" of startup disk
    else
        return 0
    end if
end tell
```

When a script is compiled and executed, some statements perform basic operations, such as assigning a variable or returning a value. A statement that targets a scriptable application results in an Apple event being sent to that application. The application can return information to the script in a reply Apple event.

The script in Listing 1 causes an Apple event to be sent to the Finder, which locates the Applications folder on the startup disk, counts the files in it, and returns that value. The `if...then...else` structure is one of several standard programming language features that AppleScript supports.

Script Editor

The Script Editor application is located in `/Applications/AppleScript`. It provides the ability to edit, compile, and execute scripts, display application scripting terminologies, and save scripts in a variety of formats, such as compiled scripts, applications, bundled applications, and plain text.

Script Editor can display the result of executing an AppleScript script and can display a log of the Apple events that are sent during execution of a script. In the Script Editor Preferences, you can also choose to keep a history of recent results or event logs.

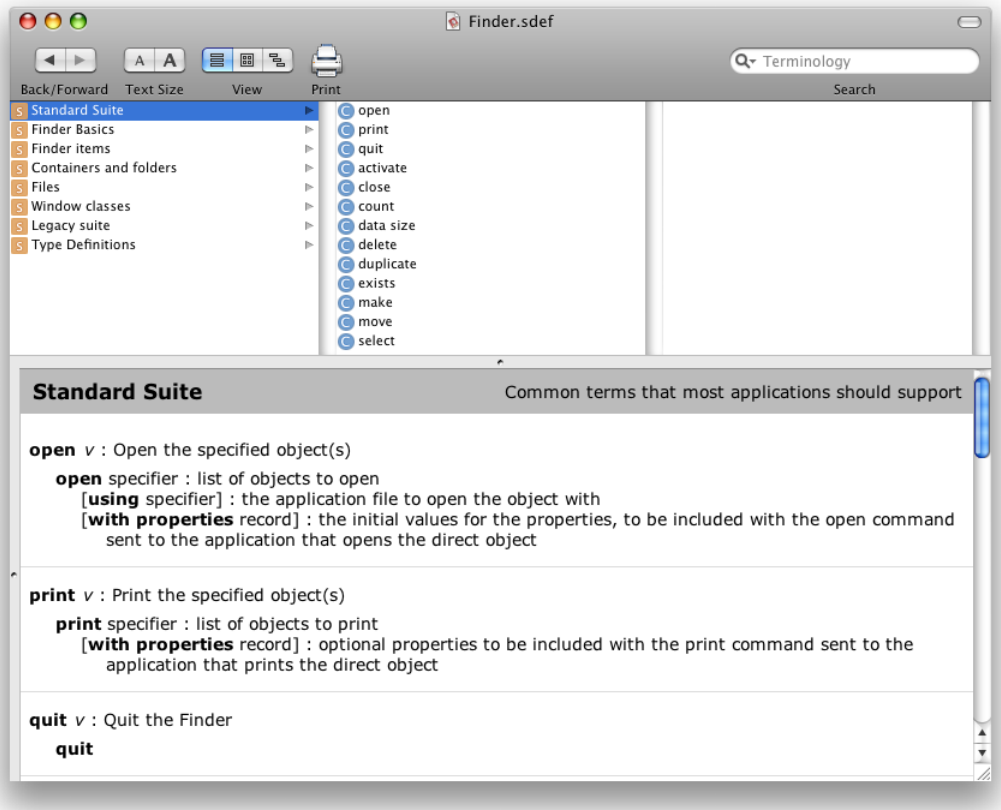
Script Editor has text formatting preferences for various types of script text, such as language keywords, comments, and so on. You can also turn on or off the Script Assistant, a code completion tool that can suggest and fill in scripting terms as you type. In addition, Script Editor provides a contextual menu to insert many types of boilerplate script statements, such as conditionals, comments, and error handlers.

Displaying Scripting Dictionaries

You can choose File > Open Dictionary in Script Editor to examine the scripting dictionary of a scriptable application or scripting addition on your computer. Or you can drag an application icon to the Script Editor icon to display its dictionary (if it has one). You can also open scripting dictionaries in Xcode.

To display a list that includes just the scriptable applications and scripting additions provided by the Mac OS, choose Window > Library. Double-click an item in the list to display its dictionary. Figure 1 shows the dictionary for the Finder application in Mac OS X version 10.5. The dictionary is labeled as "Finder.sdef". The sdef format, along with other terminology formats, is described in "Specifying Scripting Terminology" (page 25).

Figure 1 The Finder dictionary in Script Editor (in Mac OS X v10.5)



Debugging and Third Party Products

Script Editor supports only simple debugging strategies, such as logging event output and inserting `speak` or `display dialog` statements within scripts. However, there are a number of third-party products for working with AppleScript, some of them quite powerful. For example, there are script editors and tools for monitoring and debugging scripts, Apple events, and scriptable applications. Some of these third-party products are listed at the AppleScript Resources web page.

For "AppleScript Studio" (page 35) applications, you can use Xcode's debugging to set breakpoints in scripts, evaluate variables, step through scripts, and so on.

For information on debugging scriptable applications and Apple events, see the documents *Cocoa Scripting Guide* and *Apple Events Programming Guide*.

Interacting with the User in Scripts

AppleScript provides little direct support for interacting with the user in scripts. However, the Standard Additions scripting addition provides terminology for obtaining various choices from the user. For example, it includes commands for letting the user choose an application, a color, a file, a filename, and so on. It also

provides the `display dialog` command, which allows you to display a dialog with various options for text labels, buttons, and text input. Scripting Additions are described in "Extending AppleScript with Coercions, Scripting Additions, and Faceless Background Applications" (page 16).

In cases where you need a more complex interface, you can use "AppleScript Studio" (page 35), which supports a broad range of user-interface items.

What You Can Control with Scripts

Many applications from Apple are scriptable and you can also script some parts of the Mac OS. For example, the Finder, iTunes, QuickTime Player, and Mail are highly scriptable. For a complete list, see the Scriptable Applications web page at the AppleScript website. For more information on scriptability provided by Apple, see "AppleScript Utilities and Applications" (page 37).

Note: You can also control many Apple technologies and applications with "Automator" (page 33), which is available starting in Mac OS X version 10.4.

Many third-party applications are scriptable—their advertising and packaging usually mention if they are scriptable. The documentation for a scriptable application typically lists the AppleScript terminology that the application understands. You can also determine if an application is scriptable by attempting to examine its dictionary with the Script Editor application, as described in "Displaying Scripting Dictionaries" (page 20).

Note: For a list of scriptable applications from all parties, see "Scriptable Applications" at the MacScripter website.

Using AppleScript with Web Services

XML-RPC and SOAP are remote procedure call protocols that support exchanging commands and information over the Internet. Starting with Mac OS X version 10.1, AppleScript and the Apple Event Manager provide XML-RPC and SOAP support such that:

- Scripters can make XML-RPC calls and SOAP requests from scripts.
- Developers can make XML-RPC calls and SOAP requests from applications or other code by sending Apple events.

For documentation on using AppleScript with web services, see *XML-RPC and SOAP Programming Guide* (some examples may be out of date). For additional sources and examples, see Web Services. For information on developing web content and applications for the web in Mac OS X, see *Getting Started with Internet and Web*.

Using AppleScript with Other Scripting Systems

Mac OS X supports a UNIX-like shell environment that is familiar to many developers. That support includes the Terminal application, located in `/Applications/Utilities`, which you can use to open shell windows and execute shell scripts. AppleScript provides two convenient mechanisms to interact with a shell environment: you can execute shell commands from within AppleScript scripts and you can execute AppleScript scripts as shell commands.

Executing Shell Commands From AppleScript Scripts

AppleScript provides the `do shell script` command to support executing a shell command as part of an AppleScript script. For example, the following script statement uses a `do shell script` command to change the directory to the current user's home directory and obtain a list of the files found there. The list information is stored in the AppleScript variable `fileInfo`:

```
set fileInfo to do shell script "cd ~; ls"
```

The `do shell script` command is primarily of use to scripters. Although applications can execute AppleScript scripts that use the `do shell script` command, they have more efficient options for executing shell commands, as described in "Support for Carbon Applications" (page 29) and "Support for Cocoa Applications" (page 30). For more information on the `do shell script` command, see Technical Note TN2065, do shell script in AppleScript.

Executing AppleScript Scripts as Shell Commands

To execute AppleScript scripts as shell commands in a Terminal window or shell script file, you can use the `osacompile` command and the `osascript` command (located in `/usr/bin`). The former compiles an AppleScript script, while the latter executes a plain text or a compiled AppleScript script. Man pages provide documentation for these commands. For example, type `man osascript` in a Terminal window to get information on the `osascript` command.

Starting in Mac OS X version 10.5, there is a command-line tool to display compiled scripts as text, `osadecompile`. Again, see the man page for details.

Also starting in Mac OX X v10.5, AppleScript allows use of the # symbol as a comment-to-end-of-line token (the traditional double hyphen (--) is also still supported). This means that you can make a plain AppleScript script into a Unix executable by beginning it with the following line and giving it execute permission.

```
#!/usr/bin/osascript
```

Scripting the Terminal Application

The Terminal application is itself scriptable. For example, you can use the `do script` command to execute text as a shell script or command. To see the operations Terminal supports, you can examine its scripting dictionary with Script Editor.

Using Other Scripting Languages

For those who have experience with various scripting languages and environments, the previous sections have probably already provided an urge to start experimenting. And you do have a lot of options for combining features from the scripting tools, languages, and environments that are most appropriate for specific kinds of tasks. For example, the following one-line shell script statement combines Perl, AppleScript, and various tools to find duplicate entries in the Address Book application.

```
osascript -e 'tell app "Address Book" to get the name of every person' | perl
-pe 's/, /\n/g' | sort | uniq -d
```

This statement uses `osascript` to execute an inline AppleScript script (`'tell app "Address Book" to get the name of every person'`) that returns the names of every address entry from the Address Book application. It pipes the output of this script through the `perl` tool, and with a series of other commands and pipes, obtains and formats a (possibly empty) list of duplicate names.

For additional information about working with AppleScript from languages such as Ruby and Python, see "Scripting Bridge" (page 31).

Scriptable Applications

A **scriptable application** is one that goes beyond the basics of responding to Apple events sent by the Mac OS to make its most important data and operations available to AppleScript scripts or to other applications. To do this, the application must provide both a terminology for scripters to use and the underlying Apple event code to support it. Both Carbon and Cocoa applications can be scriptable, and the Cocoa framework contains built-in support that minimizes the amount of code you have to write.

Specifying Scripting Terminology

Scriptable applications describe the scripting terminology they support by supplying a scripting dictionary. A dictionary specifies the commands and objects an application supports, as well as other information that is used by AppleScript or the application itself, and possibly by other applications or scripts that want to take advantage of the application's scriptability. For information on designing a scripting terminology, see Technical Note TN2106, Scripting Interface Guidelines.

Users typically examine a dictionary for information on how to control an application in their scripts. You can display the dictionary for a scriptable application or scripting addition with Script Editor, as described in "Displaying Scripting Dictionaries" (page 20).

There are currently three dictionary formats:

- **sdef:** "sdef" is short for "scripting definition." This XML-based format is a superset of the two formats described next and supports new and improved features. Although prior to Mac OS X version 10.4, you could not use an sdef directly in your application, you could convert an sdef into either of the other formats with the `sdp` tool. Starting in Mac OS X v10.4, Cocoa applications can work natively with the sdef format, as described in Creating a Scripting Definition File and other chapters in *Cocoa Scripting Guide*.

  In Mac OS X v10.5 (Leopard), it's possible to create applications that provide dictionary information solely in sdef format, both for Carbon and Cocoa applications. You can read about additional refinements to sdef usage in Cocoa applications for Leopard in the Scripting section of *Foundation Release Notes*.

  For documentation on the sdef format, including a change history, see the `sdef`(5) man page. Scripting Interface Guidelines also includes information on working with sdefs. For documentation on the `sdp` tool, see the man page for `sdp`(1), as well as Evolution of Cocoa Scriptability Information in *Cocoa Scripting Guide*. For an example of how to use an sdef file, see the Sketch sample application. For other examples, see the sample code projects listed in "Support for Cocoa Applications" (page 30).

- **script suite:** This is the original format used by Cocoa applications and it is still supported for backward compatibility. A script suite contains a pair of information property list (plist) files that provide both AppleScript information and information used by the application. An application can contain multiple script suites.

  For documentation, see Specifying Application Scripting Support in *Cocoa Scripting Guide*.

- **aete:** This is the original dictionary format, and is still used in Carbon applications. The name comes from the Resource Manager resource type in which the information is stored (`'aete'`). An aete is useful in 10.4 and earlier, in both Carbon and Cocoa applications, to provide a dictionary that scripting languages can use without launching the application.

Determining What to Make Scriptable

In designing a scriptable application, it's a good idea to provide access to all of the application's main features, though it may make sense to start with just a key subset. You don't typically make your application's user interface directly scriptable. A good design allows users to script your application's model objects (which represent data and basic behaviors) rather than its user interface (which presents information to the user).

For example, the scripting support for a drawing application might allow a script to rotate an image, but not to perform the user interface operation of clicking a Rotate button. Some applications provide additional capabilities through their scripting interface that aren't otherwise available.

For design information, see "Learning How to Make an Application Scriptable" in *Getting Started with AppleScript* and Technical Note TN2106, Scripting Interface Guidelines.

For information on how to support printing in a scriptable application, see *The Enhanced Print Apple Event*.

Registering to Receive Apple Events

A scriptable application typically responds to a set of common commands, such as `get data`, `set data`, `delete`, and `save`, as well as to other commands that support operations specific to the application. Commands are represented in Apple events by constants defined in framework or application headers. To support a command, an application registers an event handler routine with the Apple Event Manager to handle Apple events it receives that specify that command. The Apple Event Manager dispatches received events to the handlers registered for them.

Note: For Cocoa applications, commands are registered automatically, so that developers rarely need to register apple event handlers directly.

For more information on creating and registering event handlers, see Apple Event Dispatching and Responding to Apple Events in *Apple Events Programming Guide*.

Resolving Objects in the Application

Apple events often specify items in the application. For example, a `get data` event might ask for the text of a paragraph in an open document. A distinct item in an application that can be specified in an Apple event is known as an **Apple event object**. (The term object does not imply that the items must be represented internally as objects in an object-oriented programming language.) All such objects are considered to be contained in other objects, with the application itself serving as the ultimate container. For a given application,

the **AppleScript object model** (also called the Apple event object model) specifies the classes of objects a scripter can work with in scripts, the accessible properties of those objects, and the inheritance and containment relationships for those objects.

The structures within an Apple event that identify objects are referred to as **object specifiers**. Finding the Apple event objects they specify is known as resolving the object specifiers. To resolve object specifiers, an application must include functions that are able to find objects within their containers. The application registers these functions with the Apple Event Manager, and works with the Apple Event Manager to call them at the appropriate time to obtain the objects they specify.

For Cocoa applications, Cocoa scripting support does much of the work of resolving object specifiers, but a scriptable application must still supply methods that can locate an object within its object model containment hierarchy.

For an example of an AppleScript object model, see About Scriptable Cocoa Applications; for information on how Cocoa applications resolve objects, see Object Specifiers; both are in *Cocoa Scripting Guide*.

Recording

A recordable application is one that sends Apple events to itself when a user performs actions with the application. If the user has turned on recording in the Script Editor application (with Script > Record), actions that generate Apple events are recorded into an AppleScript script.

Applications that support recording typically:

- Factor code that implements the user interface from code that actually performs operations (a standard approach for applications that follow the model-view-controller design paradigm).
- Send Apple events within the application to connect those two parts of the application. The Apple Event Manager provides a mechanism for doing this with a minimum of overhead, described in "Addressing an Apple Event for Direct Dispatching" in Creating and Sending Apple Events in *Apple Events Programming Guide*.
- Make sure that any significant action or series of related actions within the application generates an Apple event.

The Finder application in Mac OS X is recordable. Starting in Mac OS X version 10.5, the "Automator" (page 33) application has a separate Record mechanism that lets users record actions into an Automator workflow.

Creating and Sending Apple Events

An application can create and send Apple events directly. This is usually done either to send internal Apple events, as described in "Recording" (page 27), to obtain services from a scriptable application, or to communicate directly with another application. The Open Scripting Architecture provides various mechanisms for creating and sending Apple events.

Starting in Mac OS X version 10.5, applications can use "Scripting Bridge" (page 31) to obtain services from scriptable applications. Scripting Bridge lets you work efficiently in a high-level language (Objective-C) without having to handle the details of sending and receiving Apple events. (See also "Support for Cocoa Applications" (page 30) for related information.)

When you really do need to send an Apple event directly, see Building an Apple Event and Creating and Sending Apple Events in *Apple Events Programming Guide*.

Executing Scripts

To execute scripts, an application establishes a connection with the AppleScript scripting component. It can then:

- Use the standard scripting component routines to manipulate scripts associated with any part of the application or its documents.
- Let users record and edit scripts.
- Compile and execute scripts.

Note: Starting in Mac OS X version 10.5, applications can use "Scripting Bridge" (page 31) to obtain services from scriptable applications. This can be much more efficient that manipulating scripts.

An application can store and execute scripts regardless of whether it is scriptable or recordable. If an application is scriptable, however, it can execute scripts that control its own behavior, thus acting as both the client application and the server application for the corresponding Apple events. For more information, see *Open Scripting Architecture Reference*.

In Cocoa, the `NSAppleScript` class, described in *NSAppleScript Class Reference*, provides a high-level wrapper for executing AppleScript scripts from applications. For more information, see "Support for Cocoa Applications" (page 30).

Summary of Operations in a Scriptable Application

The following list summarizes how scriptable applications interact with the Open Scripting Architecture to make their features available to scripters.

- The Apple Event Manager defines data structures that are used to construct Apple events.
- The Open Scripting Architecture (OSA) provides a data transport and event dispatching mechanism for Apple events, built on top of lower level protocols.
- AppleScript defines a scripting language, described in AppleScript Language Guide (and third-party books) and implemented by the AppleScript component in Mac OS X.
- There is a small set of Apple events sent by the Mac OS, such as `open application`, `quit`, and `open documents` that all applications should be able to respond to. A scriptable application responds to additional common events, such as `get data` and `set data`, as well as to its own specific commands.
- A scriptable application provides a scripting terminology (or dictionary) for the operations it supports. The application can reuse some event constants defined by the OSA or use its own for custom events. (Constants defined by Apple, many of which you can reuse in your applications, are described in *AppleScript Terminology and Apple Event Codes Reference*.)

The sdef file format provides a mechanism for creating one terminology definition that can be converted for use in different environments.

- Developers design their applications so that key operations can be invoked in response to received Apple events.
- A scriptable application works with the Apple Event Manager to:
  - Register handlers for Apple events it can process.
  - Extract information from received Apple events, then perform requested operations or return requested data.
  - Construct Apple events for replies or other purposes.

  Scriptable Carbon applications work with the Apple Event Manager directly, but for scriptable Cocoa applications, much of this work is handled automatically.
- Scripters write AppleScript scripts that specify scriptable applications and the operations to perform.
- When a script is executed, script statements that target applications are translated by the AppleScript component into Apple events that are sent to those applications.

  Applications can also send Apple events directly to other applications.
- An application responds to the Apple events it receives by performing operations, returning data, or both.

Mac OS X Support for Creating Scriptable Applications

Mac OS X supplies a number of resources that applications can use to work with Apple events and to support scriptability, including the API provided in the following frameworks:

- The underlying support in Mac OS X for creating scriptable applications and working with Apple events is provided by the Open Scripting Architecture, and is described in "The Parts of the Open Scripting Architecture" (page 13).
- The **Cocoa framework** (`Cocoa.framework`) includes the Application Kit and Foundation frameworks, which together provide the building blocks for sophisticated Mac OS X applications. The Cocoa framework includes a great deal of support for creating scriptable applications.

  For specific Cocoa scripting documentation, see *Cocoa Scripting Guide*.
- The **AppleScriptKit** framework (`AppleScriptKit.framework`) provides capabilities for AppleScript Studio applications, which are a type of Cocoa application. For more information on AppleScript Studio, see *AppleScript Studio Terminology Reference*.
- Java applications are not typically scriptable, though they can be made AppleScript-aware using the mechanisms described in Mac OS X Integration for Java in *Java Development Guide for Mac OS X*.

Support for Carbon Applications

Carbon applications have traditionally worked directly with the Apple Event Manager to create, send, receive, and interpret Apple events. These topics are described in detail in *Apple Events Programming Guide*.

For information on making your Carbon application scriptable, see previous sections in this chapter, as well as the learning paths in *Getting Started with AppleScript*.

Carbon applications can use functions such as `OSACompile` and `OSAExecute` from `OpenScripting.framework` to compile and execute scripts. Keep in mind, however, that if you are executing a script merely to send a simple command to another application, it is more efficient to create and send an Apple event directly.

If the purpose for executing a script is just to perform a `do shell script` command, Carbon applications can do so more efficiently using one of the BSD calls `system(3)`, `popen(3)`, or `exec(3)`, which you can read about at their respective man pages.

Support for Cocoa Applications

The Foundation and Application Kit frameworks provide Cocoa applications with automated handling for certain Apple events. This includes events that may be sent by the Mac OS, such as the `open application`, `open documents`, `print documents`, and `quit` Apple events.

In addition, Cocoa provides substantial support for creating scriptable applications. To take advantage of it, applications provide scriptability information in one of the formats described in "Specifying Scripting Terminology" (page 25). They also create KVC-compliant accessors for scriptable properties in their scriptable classes. (Key-value coding, or KVC, is described in *Key-Value Coding Programming Guide*.) Though creating a fully scriptable application is a non-trivial task, an application can support many standard AppleScript commands, such as those for getting and setting properties of application objects, with a relatively small number of additional steps.

Cocoa applications can also use any of the Open Scripting Architecture APIs available to Carbon applications, and in fact, Cocoa links with the Carbon framework. For example, a Cocoa Application might call an Apple Event Manager function to send an Apple event directly (there currently is no Cocoa API to do that).

Starting in Mac OS X version 10.5, the "Scripting Bridge" (page 31) technology provides an efficient way for Cocoa applications to interact with scriptable applications at a high level—that is, without having to construct or parse individual Apple events.

Cocoa provides the `NSAppleScript` class for tasks such as compiling and executing scripts. This gives applications another mechanism to control scriptable applications and take advantage of services they provide. However, you should not use `NSAppleScript` to execute a script merely to result in sending an Apple event, because it is far more expensive than using Scripting Bridge or creating and sending an Apple event directly. And if the purpose for executing a script is to perform a `do shell script` command, Cocoa applications can execute shell commands more efficiently using `NSTask`.

The Cocoa framework also includes classes such as `NSAppleEventDescriptor`, for working with underlying Apple event data structures, and `NSAppleEventManager`, for accessing certain Apple Event Manager functions.

Cocoa support for handling Apple events and creating scriptable applications is documented in *Cocoa Scripting Guide*. For related information, see "Framework and Language Support" in About Apple Events in *Apple Events Programming Guide*. For introductory sample code, see *SimpleScripting*, *SimpleScriptingProperties*, *SimpleScriptingObjects*, and *SimpleScriptingVerbs*. For a more complex example, see the Sketch sample application.

Scripting Bridge

Scripting Bridge, introduced in Mac OS X version 10.5, provides an automated process for creating an Objective-C interface to scriptable applications. This allows Cocoa applications and other Objective-C code to efficiently access features of scriptable applications, using native Objective-C syntax. Some other scripting languages, such as Ruby and Python, can use also Scripting Bridge (they also have open-source software bridges to scriptable applications—RubyOSA and py-appscript). For more information, see *Ruby and Python Programming Topics for Mac OS X*.

To use Scripting Bridge, you add the Scripting Bridge framework to your application project and use command-line tools to generate the interface files for the scriptable application you want to target. Then in your application code, you obtain a reference to an application object for the targeted scriptable application and send Objective-C messages to it.

For details, see *Scripting Bridge Programming Guide for Cocoa* and *Scripting Bridge Framework Reference*. For related sample code, see *ScriptingBridgeFinder*.

Automator

Automator is a workflow automation application, first available in Mac OS X version 10.4. Automator, which is located in `/Applications`, lets you create complex workflows using a graphical interface that does not require any knowledge of scripting languages. A workflow consists of one or more actions, executed sequentially, with each action typically taking the output of the previous action as its input. An action performs a distinct operation, such as copying a file, cropping a photo, or sending an email message. You can run a workflow in Automator or save it as a standalone application.

Starting in Mac OS X version 10.5, you can also embed and execute Automator workflows in your application.

Automator includes actions for many Apple applications, including Finder, Mail, Safari, Xcode, iPhoto, iTunes, and QuickTime Player, and you can write actions that make features of your applications available in Automator. You use Xcode and Interface Builder to put together actions, using the Action project template (also available starting in Mac OS X v10.4). Actions are implemented as plug-ins and you can write them using AppleScript (for scriptable applications) or Objective-C.

Note: If your application is scriptable, other developers and users can write actions for it. However, by supplying actions yourself, you can be sure to make your application look its best in Automator.

For information on using the Automator application, choose Help in Automator or Help > Mac Help in the Finder and search for "Automator". For information on creating actions, see *Automator Programming Guide* and *Automator Framework Reference*. For information on using workflows in your application, see *Automator Release Notes*, as well as the class descriptions for `AMWorkflow`, `AMWorkflowView`, and `AMWorkflowController` in *Automator Framework Reference*

AppleScript Studio

AppleScript Studio combines several Apple technologies to let developers and sophisticated users create native Mac OS X applications that can be driven by AppleScript scripts. You work in the full-featured Xcode development environment and use Interface Builder to create complex user interfaces that support the *Apple Human Interface Guidelines*. Studio applications make use of user-interface objects from the Cocoa application framework, such as windows, buttons, text fields, tabs, tables, and progress bars.

AppleScript Studio can be very useful in rapid prototyping and is ideal for adding a user interface to an existing command-line tool. You can write a Studio application using AppleScript alone or a mix of AppleScript and systems programming languages, such as C, Objective-C, and Java. For example, from an AppleScript script in a Studio application, you can call a Cocoa method, written in Objective-C, to execute a performance-critical operation.

You can find a learning path for AppleScript Studio in *Getting Started with AppleScript*. Or go directly to the documents *AppleScript Studio Programming Guide* and *AppleScript Studio Terminology Reference*. Many sample Studio applications are available in `<Xcode>/Examples/AppleScript Studio`.

AppleScript Studio applications require Mac OS X version 10.1.2 or later.

AppleScript Utilities and Applications

Apple provides a number of utilities and applications in Mac OS X to enhance the features of AppleScript and your scripts. You can get additional information on some items described in this section by searching in Mac Help in the Finder or by going to the AppleScript website.

AppleScript Utility

AppleScript Utility, located in `/Applications/AppleScript`, is an application that first became available in Mac OS X version 10.4. Starting in Mac OS X version 10.5, this utility is itself scriptable.

AppleScript Utility helps you manage several AppleScript-related features in Mac OS X that were formerly available separately. For example, AppleScript Utility provides an interface to:

- Select a default script editor (to be launched when you double-click a script file.
- Enable or disable GUI scripting (described in "System Events and GUI Scripting" (page 38)).

  Prior to Mac OS X v10.4, GUI scripting was enabled through the "Enable access for assistive devices" checkbox in the Universal Access preference pane in System Preferences.
- Launch the Folder Actions Setup application (described in "Folder Actions Setup" (page 37)).
- Specify settings for the Script menu.

  The Script menu provides access to scripts for performing tasks such as the following:

  - Opening AppleScript related folders.
  - Working with Apple applications such as Address Book, Mail, and Script Editor.
  - Working with parts of the OS, such as ColorSync, Finder, and Folder Actions.
  - Working with features such as internet services, printing, and URLs.

  In Mac OS X version 10.3, you install and remove the Script menu with utility applications located in `/Applications/AppleScript`.

Folder Actions Setup

Folder Actions is a feature that lets you associate scripts with folders. A script is executed when the folder to which it is attached is opened or closed, moved or resized, or has items added or removed.

Folder Actions Setup, located in `/Applications/AppleScript`, is an application that first became available in Mac OS X version 10.3. Starting in Mac OS X version 10.5, Folder Actions Setup is itself scriptable.

This utility helps you perform tasks related to Folder Actions, including the following:

- Enable or disable Folder Actions.
- View the folders that currently have scripts attached and view the attached scripts.
- Add folders to or remove folders from the list of folders.
- Attach one or more scripts to a folder.
- Remove attached scripts from a folder.
- Enable or disable scripts attached to a folder.

System Events and GUI Scripting

System Events is an agent (or faceless background application) that supplies the terminology for using a number of features in AppleScript scripts. Among these features is GUI scripting, which allows your scripts to perform some actions in applications that have no built-in scripting support. System Events, which is located in `/System/Library/CoreServices`, has been part of Mac OS X since version 10.1 (Puma), though its features have evolved since that release.

The following are some of the terminology suites supplied by System Events in Mac OS X version 10.4 (and where noted, in version 10.5). For more information, display the application dictionary, as described in "Displaying Scripting Dictionaries" (page 20). You can also get information on many of the features supported by System Events in Mac Help (from the Help menu in Mac OS X) and at the AppleScript GUI Scripting web page at the AppleScript in Mac OS X website.

- Accounts suite and Login Items suite

  System Events supports scripting of the System Preferences Accounts pane through the terminology in these two suites.

- Audio File suite and Movie File suite

  Available starting in Mac OS X version 10.5, these suites provide terminology for accessing audio files and movie files, and the data they contain.

- Desktop suite

  Available starting in Mac OS X version 10.5, this suite provides access to Desktop preferences, such as the current desktop picture or pictures folder, and the interval for changing the desktop picture.

- Disk-Folder-File suite

  Provides terminology to access disks, files, and folders without targeting the Finder. This can be more efficient than using the Finder, and can prevent certain kinds of conflicts.

- Dock Preferences suite and Expose Preferences suite

  Available starting in Mac OS X version 10.5, these suites provide terminology for accessing Dock preferences, as well as Exposé (including Spaces) and Dashboard mouse and key preferences.

- Folder Actions suite

  Starting with AppleScript 1.9.0 in Mac OS version 10.2, System Events supports the Folder Actions feature, described in "Folder Actions Setup" (page 37).

- Network Preferences suite

Available starting in Mac OS X version 10.5, this suite provides terminology for working with items such as connections and disconnections, network locations, and network services.

- Power suite

  Provides commands for sleeping, logging out, shutting down, or restarting your computer.

- Property List suite

  Provides terminology for reading and writing the information in property list files.

- Processes suite

  Provides classes and commands for GUI Scripting, a feature available starting in Mac OS X version 10.3 that allows scripters to control applications that are either not scriptable or only partially scriptable. With GUI Scripting, AppleScript scripts can select menu items, push buttons, and perform other tasks to control the interfaces of most non-Classic applications. However, as the name implies, GUI scripting works by scripting the user interface, and so tends to result in fragile scripts. For example, items in an application's user interface may change in various ways between releases, or even between launches of the application, depending on preference settings and other factors.

  This suite is called the Processes suite because in GUI scripting, the root for any script must be a process (represented by an instance of the `application process class`). The GUI Scripting architecture is based on the accessibility support in Mac OS X, which must be enabled, in Mac OS X v10.4, through the AppleScript Utility. Prior to Mac OS X v10.4, GUI scripting was enabled through the "Enable access for assistive devices" checkbox in the Universal Access preference pane in System Preferences.

  For more information, see the GUI Scripting web page.

- QuickTime File suite

  Available starting in Mac OS X version 10.5, this suite provides terminology for working with QuickTime files, including data, annotations, and tracks.

- Security suite

  Available starting in Mac OS X version 10.5, this suite provides access to Security preferences, such as automatic login and password requirements.

- System Events suite

  This suite provides a great deal of terminology for working with parts of the OS. That includes properties for accessing certain folders (preferences folder, favorites folder, desktop pictures folder, and so on), the startup disk, and other useful items.

- XML suite

  Provides terminology for working with information in XML files.

Image Events

Like System Events, Image Events is an agent (or faceless background application) that is located in `/System/Library/CoreServices`. Image Events supports the manipulation, from scripts, of images and image-related data through operations such as cropping, embedding, matching, padding, profiling, rotating, and scaling. These operations are typically used in print, web, and media publishing.

Image Events has been part of Mac OS X since version 10.4 and provides access to the built-in service called SIPS (Scriptable Image Processing Server) that became available in that OS release.

You can find more information on Image Events in Mac Help (from the Mac OS X Help menu) or at the Image Events web page. You can also examine the Image Events dictionary with Script Editor, as described in "Displaying Scripting Dictionaries" (page 20). SIPS is described in Technical Note TN2035 ColorSync on Mac OS X. You can also get some information about SIPS by typing `sips --help` in a Terminal window.

Database Events

Database Events is a simple, scratchpad database implementation for use in AppleScript scripts. It allows any script applet to create and edit its own database.

You can use Database Events to create a new database with a file associated with it or to open a database file to access its database. Databases contain records, records contain fields, and fields contain a name and a value. The assignment of names and values is free form, as the scripter defines it. Databases persist in the file system, across executions of Database Events.

Database Events has been part of Mac OS X since version 10.4. Like System Events, Database Events is an agent (or faceless background application) that is located in `/System/Library/CoreServices`.

You can examine the Database Events dictionary with Script Editor, as described in "Displaying Scripting Dictionaries" (page 20).

Document Revision History

This table describes the changes to *AppleScript Overview*.

Date	Notes
2007-10-31	Updated to reflect AppleScript changes for Mac OS X version 10.5.
	Added a chapter to describe the "Scripting Bridge" (page 31) technology that was introduced in Mac OS X v10.5. In several places, noted that using Scripting Bridge can be easier and more efficient than using native Apple events to use the services of scriptable applications.
	In "Open Scripting Architecture" (page 13), added a section "Faceless Background Applications" (page 17) that describes this mechanism for extending AppleScript.
	In the renamed chapter "AppleScript Utilities and Applications" (page 37), added a section to describe the AppleScript helper application "Database Events" (page 40), which is available beginning in Mac OS X v10.4. In the same chapter, noted that the AppleScript Utility and Folder Actions Setup applications are themselves scriptable. Also added information on new terminology supported by the System Events application, including the Desktop, Dock Preferences, Network Preferences, Security, and other new suites.
	Moved the descriptions for "AppleScript Studio" (page 35) and "Automator" (page 33) to separate chapters and, in the latter, added information on new features available in Mac OS X v10.5.
	Combined information that was previously in several places into the renamed chapter "Scriptable Applications" (page 25). The section "Support for Cocoa Applications" (page 30) now provides links to sample code projects.
	Added various information to the section "Specifying Scripting Terminology" (page 25), including a note that changes to sdef usage in Cocoa applications for Mac OS X v10.5 are described in the Scripting section of *Foundation Release Notes*.
	In the section "Executing AppleScript Scripts as Shell Commands" (page 23), noted that in Mac OS X v10.5, you can use the # symbol as a comment-to-end-of-line token, providing a way to make a plain AppleScript script into a Unix executable. Also noted that in Mac OS X v10.5 there is a command-line tool to display compiled scripts as text, `osadecompile`.
	In the section "The Parts of the Open Scripting Architecture" (page 13), noted that in Mac OS X v10.5, the Apple event framework, `AE.framework`, is now part of `CoreServices.framework`.
	Updated Figure 1 (page 21) showing a Finder dictionary.

Date	Notes
2006-03-08	Made minor editorial corrections and updated links.
	In several places that refer to how AppleScript works with Cocoa applications, added or revised links to *Cocoa Scripting Guide*.
	In the section "Resolving Objects in the Application" (page 26), added information about the AppleScript object model and about resolving objects in Cocoa applications.
2005-04-29	Added information on Open Scripting Architecture (OSA) and the Automator, Script Utility, System Events, and Image Events applications. Changed title from "AppleScript for Mac OS X."
2004-05-27	Added sections "Determining What to Make Scriptable" (page 26) and "AppleScript Studio" (page 35).
	Moved some existing material into a new section, "Scriptable Applications" (page 25).
	Made minor text revisions to avoid duplication with *Getting Started with AppleScript*.
	Made some document references into links.
2003-09-17	Updated some links to other documentation.
2003-05-15	Moved this material into a separate document.
	Reorganized some sections, added documentation links, and made minor corrections.