# AppleScript Finder Guide

**English Dialect**

# Contents

# Tables and Listings

# About This Guide

The *AppleScript Finder Guide: English Dialect* describes the commands and object classes defined by the Finder for use with the English dialect of the AppleScript language. The Finder scripting software allows you to write, record, or run scripts that control actions in the Finder such as opening and closing folders and manipulating files.

## Audience

This guide is for anyone who wants to record or write new scripts or modify existing scripts that control actions in the Finder.

Before using this guide, you should have installed AppleScript with the Installer program. You should also be familiar with the *AppleScript Language Guide: English Dialect*, which describes the English dialect of the AppleScript scripting language, the *AppleScript Scripting Additions Guide*, and *Getting Started With AppleScript*.

Macintosh software developers who want to make their programs scriptable or recordable should also refer to *Inside Macintosh: Interapplication Communication*.

## Organization of This Guide

This guide contains three chapters:

■ Chapter 1, "Introduction to Finder Scripting," introduces Finder scripting and describes how to record and modify simple scripts.

■ Chapter 2, "Finder Objects," provides definitions for all Finder object classes.

■ Chapter 3, "Finder Commands," provides definitions for all Finder commands.

At the end of the guide are two appendixes and an index.

■ Appendix A, "Finder Commands at a Glance," summarizes the commands defined by the Finder.

■ Appendix B, "Finder Errors," lists the error messages returned by the Finder.

# For More Information

## Getting Started

See the book *Getting Started With AppleScript* to learn what hardware and software you need to use AppleScript; how to install AppleScript; and how to run, record, and edit scripts.

## AppleScript Language

See the *AppleScript Language Guide: English Dialect* (referred to in this book as the *AppleScript Language Guide*) for complete information about the commands and other terms provided by the English dialect of the AppleScript scripting language and by the Scriptable Text Editor application.

## Scripting Additions

Scripting additions are files that provide additional commands or coercions you can use in scripts. A standard set of scripting additions comes with AppleScript. Scripting additions are also sold commercially, included with applications, and distributed through electronic bulletin boards and user groups.

For more information about the scripting additions that come with AppleScript, see the *AppleScript Scripting Additions Guide: English Dialect* (referred to in this book as the *AppleScript Scripting Additions Guide*).

## Other AppleScript Dialects

A *dialect* is a version of the AppleScript language that resembles a particular human language or a programming language. This guide describes the terms defined by the Finder for use with the AppleScript English dialect. Versions of the Finder for use with other dialects work the same way but define terms and syntax appropriate for those dialects.

# Conventions Used in This Guide

Words and sample scripts in `monospaced font` are AppleScript language elements that must be typed exactly as shown.

Here are some additional conventions used in syntax descriptions:

`language element`

Plain computer font indicates an element that you must type exactly as shown. If there are special symbols (for example, + or &), you must also type them exactly as shown.

*placeholder*

Italic text indicates a placeholder that you must replace with an appropriate value. (In some programming languages, placeholders are called nonterminals.)

`[optional]`

Brackets indicate that the enclosed language element or elements are optional.

`(a group)`

Parentheses group together elements. If parentheses are part of the syntax, they are shown in bold.

`(a group)...`

Three ellipsis points (. . .) after a group defined by parentheses indicate that you can repeat the group of elements within parentheses one or more times.

`[optional]...`

Three ellipsis points (. . .) after a group defined by brackets indicate that you can repeat the group of elements within brackets 0 or more times.

`a | b | c`

Vertical bars separate elements in a group from which you must choose a single element. The elements are often grouped within parentheses or brackets.

CHAPTER 1

# Introduction to Finder Scripting

*Finder scripting* refers to the use of AppleScript and an application such as the Script Editor to write, record, or run scripts that trigger actions in the Finder. This chapter introduces Finder scripting. It also describes how to

■ record your actions in the Finder in the form of a script

■ modify recorded Finder scripts

■ begin writing Finder scripts

This chapter assumes that you know how to use the Script Editor application to run a script. It also assumes that you understand how commands and object names can be combined in Tell statements to describe actions performed by an application. If these concepts are not familiar, read Part I of the *AppleScript Language Guide* before reading this chapter.

## Installation

If you obtained the Finder scripting software with the AppleScript Scripter's Toolkit, you can install it by running the Installer program on the AppleScript Setup disk. After the Installer opens, click the Customize button, then select the item named Finder Scripting Software and click Install. If you obtained the Finder scripting software on a disk that updates an older version of the AppleScript Scripter's Toolkit, run the Installer program that's included on the update disk and click the Install button after the Installer opens.

Finder scripting requires System 7.1 or later versions of system software. Versions later than System 7 Pro (System 7.1.1) include the Finder scripting software, which is installed automatically when you install the system software.

**IMPORTANT**

Finder scripts that return a lot of information may need as much as several hundred kilobytes (K) of free memory to work correctly. To see how much memory is currently free, activate the Finder, choose About This Macintosh from the Apple menu, and check the number labeled Largest Unused Block in the About This Macintosh window. If a script you've used successfully stops working, try quitting one or more applications to make more memory available. ▲

# What Is Finder Scripting?

The Finder is a specialized application that controls the Macintosh desktop: the working area of your screen where you can use the mouse and the keyboard to open and close folders, manipulate files, and inspect or alter various aspects of the computer's operation. Finder scripting involves performing the same kinds of tasks, except that instead of triggering actions with the aid of the mouse and the keyboard, you trigger actions from scripts.

The Finder is both scriptable and recordable. A *scriptable* application is one that can respond to commands sent to it when another application, such as the Script Editor, runs a script. A *recordable* application is one that uses Apple events to report user actions for recording purposes. When recording is turned on, the Script Editor creates statements corresponding to any significant actions you perform in recordable applications, including actions you perform in the Finder. By recording or writing scripts that control the Finder, you can automate many file management and networking tasks that you would otherwise have to perform manually.

For example, the script that follows copies the items from a folder on the startup disk to a folder on a different disk. Instead of opening all the folders by double-clicking them then dragging the contents of the AppleScript folder to a storage disk, you can run the script.

```
tell application "Finder"
    copy items of folder "AppleScript" of folder "Projects" of ¬
        startup disk to folder "Backups" of disk "Storage"
end tell
```

This script consists of a three-line Tell statement that names the Finder application running on the local computer as the target application. (The ¬ symbol is a "soft" return; it tells AppleScript to treat the line it's on and the line that follows as if they are a single line.) As far as AppleScript is concerned, the Finder is just like any other scriptable application.

Like other scriptable applications, the Finder defines commands that you can use in scripts to describe actions. Some of these are standard application commands, such as Copy, Open, Print, and Move, that can be used with most scriptable applications. Others are commands unique to the Finder, such as Eject, Put Away, and Restart. The Finder also defines classes of objects on which actions can be performed, such as files, folders, and disks. Most of these object classes are unique to the Finder.

The `copy` command in the preceding example acts on the objects described by the reference `items of folder "AppleScript" of folder "Projects" of startup disk`. The folder AppleScript is located at the top level of the folder Projects, which located at the top level of the startup disk. The term `items` refers to all objects in the AppleScript folder, including files, other folders, suitcases, and so on. Any object in the Finder can be identified from within a script as long as its container and if necessary the container's containers are also identified.

By using AppleScript features such as Repeat statements and other control statements, you can create more complex scripts that take different actions depending on the outcome of one or more tests. For example, the script that follows checks all the disks of a specified computer for the presence of a folder called Back Me Up. Whenever a disk contains such a folder, the script creates a folder on a storage volume and copies the contents of Back Me Up to that folder. (The storage volume must be mounted on the remote computer for this script to work correctly.)

```
tell application "Finder" of machine "Macintosh IIci"
   repeat with i in every disk in desktop
      if folder "Back Me Up" of i exists then
         set folderName to name of i & " " & ¬
            day of (current date) & " " & time of (current date)
         set newFolder to make folder at ¬
            disk "Storage" with properties {name:folderName}
         duplicate (items of folder "Back Me Up" of i) to newFolder
      end if
   end repeat
end tell
```

The Repeat statement in this script applies the statements it contains to every disk on the desktop of the computer named Macintosh IIci. The If statement within the Repeat statement checks each disk for the presence of a folder called Back Me Up. If the folder exists on a disk, the second line of the If statement sets the variable `folderName` to a string that consists of the disk's name, the day of the month, and the time in seconds since the beginning of that day, thus ensuring that any other backup folders created from the same disk at a different time will have slightly different names. The rest of the If statement creates a new folder with the name assigned to `folderName` on a storage disk and asks the Finder to duplicate the items in the Back Me Up folder to the new folder.

As you can see, scripts that control the Finder use familiar terms like `folder`, `disk`, and `desktop` as well as standard AppleScript terms like `of`, `repeat`, and `tell`. Chapter 2, "Finder Objects," describes all the terms the Finder defines for objects, and Chapter 3, "Finder Commands," describes the terms it defines for commands. You can use these chapters as references when you need detailed information about specific objects or commands. However, the easiest way to learn how to use the scripting terminology defined by the Finder is to record your actions in the Finder and examine the resulting script.

The next section describes how to use record and edit scripts that control the Finder. When you see how the Finder uses its own terminology in recorded scripts, you can begin writing your own scripts from scratch.

# Recording Actions in the Finder

You can record almost any actions in the Finder that involve manipulating Finder windows or icons: for example, opening folders, copying files, or changing View settings. To record actions in the Finder, click the Record button in the Script Editor application, then activate the Finder (by clicking on the desktop or choosing Finder from the Applications menu) and perform the actions that you want to record.

For example, suppose you like to have certain Finder windows arranged next to each other on the desktop when you're working on financial matters, so that related files, alias files, or applications are easily accessible. To record the arrangement you want, follow these steps:

1. **Open the Script Editor application.**

2. **Click the Record button.**

3. **Activate the Finder by clicking on the desktop or choosing Finder from the Applications menu.**

4. **If any Finder windows are currently open, close them all by holding down the Option key and clicking the close box of the active window.**

5. **Open, resize, and arrange the windows whose positions you want to record.**

6. **When you're satisfied with the arrangement of the desktop, activate the Script Editor again and click the Stop button.**

The recorded script should look something like the one in Listing 1-1 on page 6.

If you save your recorded script as a script application, you can arrange the windows the same way at any time by double-clicking the script's icon in the Finder. The Finder responds by closing any windows that are currently open and opening and arranging windows as specified in the script.

**Listing 1-1**      A sample recorded script

```
tell application "Finder"
   activate
   close every window
   select startup disk
   open selection
   set position of window of startup disk to {4, 43}
   set size of window of startup disk to {369, 464}
   select folder "Financial" of startup disk
   open selection
   set position of window of folder "Financial" of startup disk to ¬
      {378, 43}
   set size of window of folder "Financial" of startup disk to ¬
      {200, 155}
   select folder "Letters" of startup disk
   open selection
   set size of window of folder "Letters" of startup disk to ¬
      {257, 385}
   set position of window of folder "Letters" of startup disk to ¬
      {379, 222}
   set size of window of folder "Letters" of startup disk to ¬
      {200, 286}
end tell
```

## Modifying Recorded Scripts

You can often modify a recorded script to suit new circumstances. For example, you could adapt the script shown in Listing 1-1 for use with an external hard disk rather than the startup disk by changing the term `startup disk` to `disk` "*nameOfDisk*" throughout the script, where *nameOfDisk* is the name of the hard disk on which the folders Financial and Letters are located.

Or suppose you want to make sure the Finder opens each window specified in Listing 1-1 with a specific view selected in the Views menu. One way to do this might be to place the insertion point after the last line of the script in the script

window, turn on recording, choose the settings you want for each window from the View menu, and turn off recording. Any significant changes you make to the windows' views will be recorded; however, if some of the View settings for some of the windows are already set the way you want them, the Finder won't record the action because nothing changes as a result.

A more reliable approach is to add lines to the recorded script that set the View settings. To do so, you must know how to tell the Finder to set the view of a window.

The easiest way to learn how to describe an action in the Finder is to try recording the action in a sample script. To learn how the Finder sets a window's view, follow these steps:

1. **Open a new script window in the Script Editor by choosing New Script from the File menu.**

2. **Click the Record button.**

3. **Activate the Finder by clicking on the desktop or choosing Finder from the Applications menu.**

4. **Change the Views setting for one of the Finder's windows.**

5. **Activate the Script Editor again and click the Stop button.**

The recorded script looks something like this:

```
tell application "Finder"
    activate
    set view of window of folder "Financial" of startup disk to name
end tell
```

The third line of the recorded script sets the View property of the window for the folder Financial to `name`. You can now use this line as a template for setting the views of folder windows in a script. For example, to modify Listing 1-1 so that the windows will always open in specific views, you could add these lines just before the last line (that is, before `end tell`):

```
set view of window of startup disk to icon
set view of window of folder "Financial" of startup disk to name
set view of window of folder "Letters" of startup disk to size
```

After you run the new script, the windows have the views specified in the script even if you've previously changed them.

To modify recorded scripts or create new scripts that control the Finder, you may need to look up the definitions of some of Finder terms. Like any scriptable application, the Finder contains a dictionary of the AppleScript terms that you can use to control the application. To open the dictionary, drag the Finder's icon over the icon of the Script Editor, the script-editing application that comes with AppleScript. If you run across an unfamiliar word in a recorded script, you can look it up quickly in the dictionary. If you need a comprehensive definition with examples, look up the term's entry in this book.

## Simplifying Recorded Scripts

As is often true of scriptable applications, the Finder allows you to express the same action in several different ways. Recorded scripts are precisely worded so as to avoid ambiguity in variety of circumstances. In the script shown in Listing 1-1 on page 6, however, there is no need to specify the folder and disk to which a window belongs. If you wish, you can specify a window by name only:

```
set view of window "Letters" to size
```

To work correctly, the preceding statement must be contained in a Tell statement that names the Finder as the target application, and the window named Letters must be open. The container for a Finder window is always the Finder itself. If more than one window named Letters is open, the Finder changes the View property for all windows with that name to `size`.

The previous section described how to add three lines at the end of the script shown in Listing 1-1 that set the window's views to different values. If instead you want all the windows opened by that script to have the same view, you could add just one line:

```
set view of windows to name
```

The plural form `windows` identifies all open Finder windows.

After opening a window, the recorded script in Listing 1-1 adjusts the window's Position and Size properties separately. The terms Position and Size name two different properties of a window. Each property consists of a list of two

integers. For the Position property, the two integers specify the coordinates of the upper-left corner of the content region of the window (the portion of the window that displays its contents; the title bar and scroll bars are not part of the content region). For the Size property, the two integers specify the vertical and horizontal dimensions of the window's content region.

Instead of adjusting these properties separately, you can adjust them at the same time by setting the Bounds property. The value of the Bounds property is a list of four integers that specify the coordinates of the four corners of the window's content region. The first and second integers are identical to the value of the window's Position property. The third and fourth integers can be obtained by adding the first and second integers of the Position property to the first and second integers of the Size property, respectively.

You can easily obtain the bounds property of a window by asking the Finder for it. Just activate the window and run a script like this:

```
tell application "Finder"
   bounds of front window
end tell
```

```
--result: {4, 43, 373, 507}
```

If you wish, you can rewrite a script like the one in Listing 1-1 using the Bounds property instead of Position and Size:

```
tell application "Finder"
   activate
   close every window
   open startup disk
   set bounds of window of startup disk to {4, 43, 373, 507}
   open folder "Financial" of startup disk
   set bounds of window "Financial" to {378, 43, 578, 198}
   open folder "Letters" of startup disk
   set bounds of window "Letters" to {379, 222, 579, 508}
end tell
```

Although this simplified version of the script in Listing 1-1 won't run appreciably faster than the original, it is easier to read. In a longer script, using succinct statements can improve performance.

Recording Actions in the Finder **9**

# Writing Scripts That Control the Finder

You can use a variety of techniques in scripts you write yourself that you can't use in recorded scripts. For example, suppose you have several different window arrangements that you like to use, and they tend to evolve and change. Instead of recording a series of scripts, you can write a single script that allows you to perform one of two actions: take a snapshot of the current window arrangement or restore the most recently taken snapshot.

To write a script that performs these tasks, you need to know something about script properties, variables, the Display Dialog scripting addition, If statements, and Repeat statements. For more information about these AppleScript features, see the *AppleScript Language Guide* and the *AppleScript Scripting Additions Guide.*

Listing 1-2 shows an example of such a script. It begins by declaring two script properties, `boundsAll` and `refsAll`, that hold the windows' bounds and references to the objects the windows belong to when the script takes a snapshot. At first these properties consist of empty lists. When the script takes a snapshot of the current windows, these properties "remember" the windows' bounds and references until a new snapshot is taken or the script is recompiled.

After declaring its properties, the script uses the Display Dialog scripting addition command to display a dialog box. The dialog box asks the user to make a choice and displays two buttons: Restore Old Snapshot or Take New Snapshot. These buttons control which of two parts of the If statement that follows runs next.

If the user chooses Take New Snapshot, the second part of the If statement (beginning with `else`) takes control and stores the bounds and references for the open windows in the `boundsAll` and `refsAll` properties. If the user chooses Restore Old Snapshot, the first part of the If statement closes all the windows that are currently open and uses the values stored in the properties to identify the items whose windows are to be opened, open their windows, and set the windows' bounds. These actions are accomplished with a Repeat loop that acts on one item and its window at a time.

**Listing 1-2**    A script that either takes a snapshot of the current window arrangement or restores a previously stored snapshot

```
property boundsAll : {}
property refsAll : {}

set dialogResult to display dialog ¬
    "Restore old snapshot or take new one?" ¬
    buttons {"Restore Old Snapshot", "Take New Snapshot"}
if button returned of dialogResult is "Restore Old Snapshot" then
    tell application "Finder"
        close windows
        set increment to 0
        repeat (number of items in refsAll) times
            set increment to increment + 1
            set i to item increment of refsAll
            open i
            set x to item increment of boundsAll
            set bounds of window 1 to x
        end repeat
    end tell
else
    tell application "Finder"
        set boundsAll to the bounds of every window
        set refsAll to the item of every window
    end tell
end if
```

To keep this script handy, store it as a script application on the desktop. Whenever you want to record or restore a window arrangement, double-click the application's icon. If you want to be able to switch back and forth between several window arrangements, duplicate the script and use each copy to create and store a different one.

Listing 1-3 shows another script that uses the Display Dialog command. This script automates a couple of tasks that network administrators often have to perform: adding a new user and adding a drop folder for that user.

**Listing 1-3**    A script that adds a new user to the Users & Groups control panel and adds a drop folder for that user to the startup disk

```
tell application "Finder"
   set dialogResult to display dialog ¬
      "Name of new user:" default answer ""
   set newUser to text returned of dialogResult
   set cPanel to control panel "Users & Groups" of ¬
      control panels folder
   open cPanel
   make user at cPanel with properties {name:newUser}

   set newFolder to make folder at startup disk ¬
      with properties {name:newUser}
   set owner of newFolder to newUser
   set group of newFolder to "Department Group"

   set see folders of group privileges of newFolder to true
   set see files of group privileges of newFolder to true
   set make changes of group privileges of newFolder to true
   set see folders of owner privileges of newFolder to true
   set see files of owner privileges of newFolder to true
   set make changes of owner privileges of newFolder to true
   set see folders of guest privileges of newFolder to false
   set see files of guest privileges of newFolder to false
   set make changes of guest privileges of newFolder to true
end tell
```

Unlike the script in Listing 1-2, the script in Listing 1-3 consists of a single Tell statement, and the Display Dialog command is invoked from within the Tell statement. The dialog box asks for the name of the new user and stores it in the variable `newUser`. The three statements that follow open the Users & Groups control panel and use the Make command to create a new user object with the name stored in `newUser`.

The rest of the Tell statement consists of a series of Set commands that create a new folder on the startup disk with the new user's name and set the folder's owner and group and its sharing privileges. The owner and members of the Department Group have all sharing privileges set to true. The guest privileges See Folders, See Files, and Make Changes are set to `false`, `false`, and `true` respectively, so guests can drop items into the folder but can't open it. These privileges are signaled to guests by a belt around the folder and a small black arrow above it.

For the script in Listing 1-3 to work correctly, file sharing must be turned on and the computer on which the script is running must have a group named Department Group. You can modify that name or other settings in the script to suit your own purposes. You can also modify the script in a variety of other ways. For example, you can add an If statement that checks for the presence of a particular group and if necessary creates the group before creating the new user.

# Finder Objects

This chapter begins with a summary of the format used in object class definitions and an introduction to the Finder application object, the outermost container for most other Finder objects. The last section of the chapter provides complete descriptions of the object classes defined by the Finder.

## Using Object Class Definitions

Object class definitions describe what objects that belong to a particular class have in common. Each definition contains at least five kinds of information: properties, element classes, commands handled, default value class returned, and one or more examples. These categories are briefly described here, using examples from the definition of File that begins on page 61.

For a more detailed discussion of object classes and references to objects, see the *AppleScript Language Guide.*

### Properties

A *property* of an object is a characteristic that has a single value, such as the name of a window or the modification date of a file. Properties are contained by objects in much the same way that elements are (see "Element Classes" on page 16). The main difference is that each of an object's properties has only one value, whereas an object may have many different elements of a single class.

Properties of an object are distinguished from each other by their unique labels. For example, the script that follows sets the Position property of the file MyFile, causing the icon for that file to move to the specified position.

```
tell application "Finder"
    set position of file "MyFile" to {29, 235}
end tell
```

The file in this example must be located on the desktop. A file can have only one Position property, distinguished from the other properties of the file by its label, `position`.

The Properties section of an object class definition lists all the property labels defined for that object and describes how to use them, including the class used for each property's value and whether or not the property can be modified. Some objects inherit some or all their properties from other related objects. For example, the definition for File that begins on page 61 lists several properties that are inherited from the object class Item and several that are defined specifically for class File.

**IMPORTANT**

A property name has no plural form, even if you use it to refer to more than one property.  ▲

This script gets a list of the names of the files at the top level of the startup disk:

```
tell application "Finder"
    name of files in startup disk
end tell
```

If you replace `name` with `names`, the script won't compile.

## Element Classes

*Elements* are objects contained by an object. For example, a folder can contain files and other folders. The element classes listed in an object class definition show what kinds of elements objects of that class can contain. An object can contain many elements or none, and the number of elements of a particular class may change over time.

A file can be an element of other Finder objects in which files can be stored, such as folders, disks, or the desktop; so File is listed as one of their element classes. The definition for object class File that begins on page 61 indicates that a file can't contain any other elements.

## Commands Handled

Objects that belong to the same class can respond to the same commands. Object class definitions list the commands to which all objects of that class respond.

For example, the definition for File that begins on page 61 indicates that you can use any of these commands to perform actions on a file: Clean Up, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Make, Move, Open, Print, Put Away, Reveal, Select, Sort, Update.

## Default Value Class Returned

In most scriptable applications, each object has a value that you can obtain simply by referring to the object in a script. For example, the value of a paragraph object in a word-processing application is a string that includes style and font information.

Unlike other scriptable applications, the Finder can control only the representation of its objects within windows and the properties it defines for them; it can't retrieve or alter text in a text file, for example. Therefore, the value returned for most Finder objects is a reference to the object, not the object's data.

For example, this script returns a reference to the file MyFile, not the contents of the file:

```
tell application "Finder"
   get file "MyFile" of startup disk
end tell

--result: file "MyFile" of startup disk of application "Finder"
```

## Examples

Each object class definition in this book include one or more short examples demonstrating how to use the object name in a script. The first example for the File command definition that begins on page 61 is a script that returns a list of references to files whose modification dates are greater than a specified date, and the second example is a script that copies an icon to a group of files.

# The Finder Application Object

The Finder application object belongs to the object class Application. As is usually the case with scriptable applications, the application object functions as the outermost container for most of the objects the Finder defines.

To send commands to the Finder, use a Tell statement addressed to the Finder:

```
tell application "Finder"
   open startup disk
end tell
```

```
--result: startup disk of application "Finder"
```

The result shown appears in the Script Editor's Result window. The statement

```
startup disk of application "Finder"
```

is a complete reference to the startup disk. The Finder application itself is the startup disk's container. Disks, folders, and files are examples of objects that can be elements of the Finder application. Folders and files can in turn be elements of disks or of folders within disks.

You can identify nested Finder objects in a variety of ways. For example, if the startup disk is named My World, these statements all identify the same object:

```
folder "My Folder" of startup disk of application "Finder"
folder "My Folder" of disk "My World" of application "Finder"
item "My Folder" of container "My World" of application "Finder"
container "My World:My Folder" of application "Finder"
```

When recording is turned on, the Finder always records references the same way. Although the Finder can interpret the four statements in the previous example correctly, it records the reference as shown in the first statement.

To obtain a standard reference to an element of the Finder while you're writing a script, follow these steps:

1. **Select the object.**

2. **Choose Copy from the Edit menu.**

3. **Activate the Script Editor application.**

4. **Click at the point in your script where you want to paste the reference.**

5. **Choose Paste Reference from the Edit menu.**

Windows are also elements of the Finder application. You can address a Tell statement to a Finder window anywhere in a script:

```
tell front window of application "Finder"
   close
end tell
```

Because `window` is a standard AppleScript term, AppleScript can interpret this script correctly. However, when it compiles a script, AppleScript doesn't look up the Finder's terminology definitions until after it has compiled the first line of the first Tell statement that addresses the Finder. Therefore, a script that consists of a Tell statement addressed to any other Finder object won't compile:

```
tell startup disk of application "Finder"
   open
end tell
```

To address a Tell statement to a Finder object other than a window, enclose it within another Tell Statement, like this:

```
tell application "Finder"
   tell startup disk
      open
   end tell
end tell
```

## Properties and Elements of the Finder

The Finder application, like most other Finder objects, has many properties you can refer to in scripts. Some of these properties provide a shorthand method of referring to certain specialized objects.

For example, although the Finder application can contain any number of folders, only one Apple Menu Items folder can be located in the active System Folder for a given Macintosh computer. The Apple Menu Items folder can be described in a script as either an element of the System Folder or a property of the Finder application.

This script identifies the Apple Menu Items folder as an object contained by the System Folder:

```
tell application "Finder"
   open folder "Apple Menu Items" of ¬
      folder "System Folder" of startup disk
end tell
```

This script identifies the Apple Menu Items folder as a property of the Finder application:

```
tell application "Finder"
   open apple menu items folder
end tell
```

Other specialized containers, such as the startup disk, the System Folder, the Startup Items folder, and the Extensions folder, are also defined as Finder properties.

**Note**
You can use the Path To scripting addition command to obtain references to some of the objects that can be identified as properties of the Finder. However, Path To use its own constants for these references and returns a pathname as an AppleScript reference of the form `alias "`*Disk*`:`*Folder1*`:`*Folder2*`:...:`*Filename*`"`. ◆

The desktop is another container you can identify as a property of the Finder. Any items contained by the desktop—that is, items that are "loose" on the

working area of your screen and not contained by a folder, disk, or other object—can be considered elements of either the desktop or of the Finder application object, as shown in the following examples.

This script requests a list of the items contained by the desktop:

```
tell application "Finder"
    items of desktop
end tell

--result: {trash of application "Finder", disk
"Applications" of application "Finder", disk "Storage" of
application "Finder", startup disk of application "Finder"}
```

If you request the items of the Finder application, you get a similar list:

```
tell application "Finder"
    items
end tell

--result: {trash of application "Finder", disk
"Applications" of application "Finder", disk "Storage" of
application "Finder", startup disk of application "Finder",
desktop of application "Finder"}
```

The only difference is that the second list includes a reference to the desktop itself, which is also an item contained by the Finder application.

Finally, if you request the contents of the Finder application, you get a list that includes references to windows and to currently running processes as well as to the Finder's items:

```
tell application "Finder"
    contents
end tell

--result: {trash of application "Finder", disk
"Applications" of application "Finder", disk "Storage" of
application "Finder", startup disk of application "Finder",
```

```
window of startup disk of application "Finder", content
space of desktop of application "Finder", desktop of
application "Finder", application "Scriptable Text Editor",
application "Script Editor"}
```

For complete lists of the Finder application object's properties and element classes, see the definition for the Application object class, which begins on page 31.

## References to Finder Windows

One of the Finder elements you can refer to in scripts is a content space, which can be any kind of window or the desktop. If you request all the content spaces, the Finder returns a list of references to all kinds of windows (container windows, information windows, sharing windows, and status windows) and the desktop:

```
tell application "Finder"
    content spaces
end tell
```

```
--result: {information window of startup disk of
application "Finder", window of folder "Projects" of
startup disk of application "Finder", window of startup
disk of application "Finder", window of disk "Applications"
of application "Finder", content space of desktop of
application "Finder"}
```

The result shown in the preceding example includes one information window, three container windows, and the desktop content space. An information window is the window that opens when you select an object's icon in the Finder and choose Get Info from the File command. Containers are objects that can hold other objects, such as suitcases, folders, disks, and the desktop, and a container window is a window that belongs to a container. The only exception is the desktop window. Even though the desktop is a container, the desktop

content space is a special case and the Finder doesn't consider it a member of class Window (although you can refer to `window of desktop` in a script).

The desktop "window" and the desktop are actually the same object, whereas all other windows are distinct from the objects they belong to and may have different properties. For example, the Position property of a disk is different from the Position property of the disk's window.

The desktop doesn't have most of the characteristics of windows, so you can't refer to properties such as Bounds, Position, and so on. If you are performing operations on Finder windows that involve these properties, you may want to get a list of windows that doesn't include the desktop. Therefore, the Finder doesn't include the desktop when you request a list of windows:

```
tell application "Finder"
    windows
end tell

--result: {information window of startup disk of
application "Finder", window of folder "Projects" of
startup disk of application "Finder", window of startup
disk of application "Finder", window of disk "Applications"
of application "Finder"}
```

The list of window references returned doesn't include the desktop, but it does include information windows and any other windows that are open. If you want to get a list restricted to windows that belong to containers, request a list of container windows:

```
tell application "Finder"
    container windows
end tell

--result: {window of folder "Projects" of startup disk of
application "Finder", window of startup disk of application
"Finder", window of disk "Applications" of application
"Finder"}
```

You can refer to windows by name, by number, or as the window property of the object to which they belong. For example, these statements are equivalent:

```
window "My Folder"
window of folder "My Folder" of startup disk
```

These statements are also equivalent:

```
window 1
front window
```

The Finder's front window isn't necessarily the active content space, which can be the desktop as well as a window. To get a reference to the active content space, use this statement:

```
content space 1
```

## References Returned for the Insertion Location property

The Finder's Insertion Location property returns a reference to the object to which new information can currently be added. For example, if the New Folder command in the File menu is active, the insertion location is a reference to the container in whose content space the new untitled folder appears when you choose New Folder. This container can be a folder, a disk, or the desktop:

```
tell application "Finder"
    insertion location
end tell

--result: desktop of application "Finder"
```

If the active content space is a sharing window or an information window, the insertion location is a reference to the container to which the window belongs.

## References Returned for a Point

To get a reference to the content space that lies under a particular point on the desktop, run a script like this:

```
tell application "Finder"
    content space {100, 100}
end tell

--result: window of folder "Projects" of startup disk of
application "Finder"
```

The pair of integers in braces identifies the point in which you're interested relative to the entire screen (including the menu bar), and the result contains a reference either to a window or to the desktop. It is usually easiest to use the term Content Space to obtain this kind of reference, because the script returns a reference even if the point is located in the desktop window.

You can also specify more restricted window classes, such as Window, Container Window, Information Window, and so on, as shown in the next example.

```
tell application "Finder"
    container window {100, 100}
end tell
```

The script in this example returns a reference only if a container window contains the specified point. If the specified point is located in some other kind of content space, such as the desktop or an information window, the script returns an error.

You can use similar methods to find out whether an item or other object lies under a particular point. For example, this script returns a reference to an item if the item's bounds enclose the specified point:

```
tell application "Finder"
    item {100, 100}
end tell
```

# Object Class Definitions

This section defines the Finder's object classes.

## Accessory Process

An object of class Accessory Process is a process launched from a desk accessory file.

**PROPERTIES**

An accessory process has all the properties defined for object class Process on page 77: Creator Type, File, File Type, Frontmost, Name, Partition Size, Partition Space Used, Remote Events, Scriptable, and Visible.

An accessory process also has this property:

```
desk accessory file
```
> The desk accessory file from which this accessory process was launched.
> *Class:* Reference
> *Modifiable:* No

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Count, Data Size, Exists, Get, Sort

**DEFAULT VALUE CLASS RETURNED**

A reference or (if you use the plural form `accessory processes`) a list of references of the form

`application "`*AccessoryProcessName*`"`

where *AccessoryProcessName* is the name of an accessory process as it appears in the Applications menu.

**EXAMPLE**

This script returns a list of references to the accessory processes that are currently running:

```
tell application "Finder"
   every accessory process
end tell

--result: {application "Alarm Clock", application
"Calculator", application "Notepad"}
```

## Accessory Suitcase

An object of class Accessory Suitcase is a desk accessory suitcase.

**PROPERTIES**

An accessory suitcase has all the properties defined for object class Container on page 43: Completely Expanded, Container Window, Entire Contents, Expandable, Expanded, Previous List View, Selection, and View.

Like any other container, an accessory suitcase also has all the properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

**ELEMENT CLASSES**

The only elements that an accessory suitcase can contain are objects of class Item (see page 73 for the definition of Item). Items can be identified within an accessory suitcase by name or by number.

**COMMANDS HANDLED**

Clean Up, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Make, Move, Open, Put Away, Reveal, Select, Set, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

Reference to a suitcase or, if you use the plural form `accessory suitcases`, a list of references.

**EXAMPLE**

This script returns a list of references to the accessory suitcases at the top level of the startup disk:

```
tell application "Finder"
   accessory suitcases of startup disk
end tell

--result: {suitcase "DAs" of startup disk of application "Finder",
suitcase "More DAs" of startup disk of application "Finder"}
```

## Alias File

A object of class Alias File is an alias—that is, a file that represents another file or volume. Icons for alias files have italicized names.

**PROPERTIES**

An alias file has all the properties defined for object class File on page 61: Creator Type, File Type, Locked, Product Version, Stationery, and Version.

Like any other file, an alias file also has all the properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

Unlike other files, an alias file also has this property:

`original item`
> A reference to the original item associated with the alias file.
> *Class:* Reference
> *Modifiable:* No

**ELEMENT CLASSES**

> None

**COMMANDS HANDLED**

> Clean Up, Copy, Count, Data Size, Delete, Duplicate, Eject, Empty, Exists, Get, Make, Move, Open, Print, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

> Reference to a file or, if you use the plural form `alias files`, a list of references.

**EXAMPLE**

> If you run this script from the Script Editor, it reveals the original items for any alias files that are currently selected:

```
tell application "Finder"
   get selection
   repeat with moose in result
      if original item of moose exists then
         reveal original item of moose
      end if
   end repeat
end tell
```

The Get statement in this script returns a list of references to the selected items. The Repeat statement tests the Original Item property for each selected alias file. If the property's value is a valid reference, the If statement asks the Finder to reveal the original item.

For an example of a script application that performs a similar task, see the definition of the Reveal command on page 131. For examples of scripts that create alias files, see the definition of the Make command beginning on page 121.

**NOTES**

A reference to an object of class Alias File in a Tell statement addressed to the Finder is different from an AppleScript reference that uses an alias. You can use an AppleScript alias reference like this anywhere in a script to identify a file:

```
alias "Disk:Folder1:Folder2:...:Filename"
```

When you compile the script (for example, by clicking the Check Syntax button in the Script Editor), AppleScript creates an alias record that identifies the file even after you move the file from its original location—as long as you don't recompile the script.

Like an alias record, an alias file identifies another file or folder even after the original has been moved. However, a reference to an alias file is just like any other Finder reference: it must provide a complete description of the alias file's location in a form similar to this:

```
tell application "Finder"
   alias file "Filename" [ of container "ContainerX" ]... ¬
      of container "Container1" [ of disk "Disk" ]
end tell
```

If you move the alias file, the reference won't be accurate any longer. Also, if you attempt to use a Finder reference to an alias file outside of a Tell statement addressed to the Finder, the script won't run.

## Application

An object of class Application is the Finder application itself, the outermost container for most Finder objects.

Because of its role as the application that controls the Macintosh desktop, the Finder defines its own application object differently from the way most other applications define their application objects. For an introduction to the role of the application object in scripts that control the Finder, see "The Finder Application Object," which begins on page 18.

**PROPERTIES**

about this macintosh
> A list of references that identify the processes shown in the About This Macintosh window. This list includes a reference to system software but doesn't include a separate reference to the Finder.
> *Class:* List of references to processes
> *Modifiable:* No

apple menu items folder
> A reference to the Apple Menu Items folder in the System Folder. The contents of this folder appear in the Apple menu.
> *Class:* Reference
> *Modifiable:* No

clipboard    A reference to the Clipboard window, which can also be displayed by choosing Show Clipboard from the Edit menu.
> *Class:* Reference
> *Modifiable:* No

control panels folder
> A reference to the Control Panels folder in the System Folder. This folder is used to store control panels.
> *Class:* Reference
> *Modifiable:* No

desktop    A reference to the desktop.
> *Class:* Reference
> *Modifiable:* No

extensions folder

>A reference to the Extensions folder in the System Folder. This folder is used to store extension files.
>*Class:* Reference
>*Modifiable:* No

file sharing

>A Boolean value that indicates whether file sharing is on (`true`) or off (`false`).
>*Class:* Boolean
>*Modifiable:* Yes

fonts folder

>A reference to the Fonts folder in the System Folder. This folder contains the fonts currently available to the system software.
>*Class:* Folder
>*Modifiable:* No

frontmost    A Boolean value that indicates whether the Finder is the frontmost application (`true`) or not (`false`).
>*Class:* Boolean
>*Modifiable:* Yes

insertion location

>A reference to the object to which new information can currently be added (see page 24 for more information).
>*Class:* Container window
>*Modifiable:* No

largest free block

>An integer that indicates the size, in bytes, of the largest free block of process memory available. This value is approximately the same as the value labeled Largest Unused Block in the About This Macintosh window, except that the value in the window is given in kilobytes (K, where 1K = 1024 bytes).
>*Class:* Integer
>*Modifiable:* No

preferences folder

>A reference to the Preferences folder in the System Folder. This folder contains application preferences files.
>*Class:* Reference
>*Modifiable:* No

`product version`
> A string that describes the version of system software running on the same computer as the Finder.
> *Class:* String
> *Modifiable:* No

`selection`  A list of references to the object or objects that are currently selected. The objects in the selection are those that would be cut by a Cut command or copied by a Copy command. If no objects are selected, the value of the Selection property is an empty list.
> *Class:* List of references
> *Modifiable:* Yes

`sharing starting up`
> A Boolean value that indicates whether file sharing is in the process of starting up (`true`) or not (`false`).
> *Class:* Boolean
> *Modifiable:* No

`shortcuts`  A reference to the Finder Shortcuts window, which can also be displayed by choosing Finder Shortcuts from the Help menu.
> *Class:* Reference
> *Modifiable:* No

`shutdown items folder`
> A reference to the Shutdown Items folder in the System Folder. The contents of this folder are automatically opened when the computer is shut down.
> *Class:* Reference
> *Modifiable:* No

`startup items folder`
> A reference to the Startup Items folder in the System Folder. The contents of this folder are automatically opened when the computer starts up.
> *Class:* Reference
> *Modifiable:* No

`system folder`
> A reference to the System Folder.
> *Class:* Reference
> *Modifiable:* No

`temporary items folder`
> A reference to the Temporary Items folder in the System Folder.
> Applications use this folder to store temporary items.
> *Class:* Reference
> *Modifiable:* No

`version`      A string that describes the version of the Finder Scripting
> Extension or, for versions of Finder scripting software that
> don't require this extension, the version of the Finder file in
> the active System Folder.
> *Class:* String
> *Modifiable:* No

`view preferences`
> A reference to the Views control panel.
> *Class:* String
> *Modifiable:* No

`visible`      A Boolean value that indicates whether the Finder is currently
> visible (`true`) or hidden (`false`).
> *Class:* Boolean
> *Modifiable:* Yes

**ELEMENT CLASSES**

The Finder application can contain objects of any of the classes listed here. Page
numbers indicate the locations of corresponding definitions in this chapter.

Accessory Suitcase (page 27)

Alias File (page 28)

Application File (page 38)

Container (page 43)

Container Window (page 47)

Content Space (page 50)

Control Panel (page 51)

Desk Accessory File (page 55)

Desktop-Object (page 56)

Disk (page 57)

Document File (page 60)

File (page 61)

Folder (page 63)

Font File (page 65)

Font Suitcase (page 66)

Information Window (page 69)

Item (page 73)

Sharable Container (page 79)

Sharing Window (page 85)

Sound File (page 88)

Suitcase (page 90)

Trash-Object (page 91)

Window (page 95)

Elements of any of these classes can be identified by name or by number. With the exception of the desktop and window objects such as container windows and information windows, an element of the Finder can also be identified by an AppleScript alias record—that is, a representation of an object, much like an alias icon on the desktop, that identifies the object. For example, both these statements refer to the same file:

```
alias "Hard Disk:MyFile"
file "MyFile" of disk "Hard Disk"
```

In most cases the Finder can accept a reference like the first example, but you can't use a Finder reference like the second example outside of a Tell statement addressed to the Finder. This book describes how to use Finder references. For information about AppleScript references to alias records, files, applications, machines, and zones, see the *AppleScript Language Guide.*

Objects of class Application File, Disk, and Folder can also be identified by ID. The ID in each case consists of the application file's creator type (as a string), the disk's ID, and the folder's folder ID, respectively. For example, this statement identifies the startup disk:

```
disk ID -1
-—result: startup disk of application "Finder"
```

This statement identifies a folder:

```
folder ID 918 of startup disk
-—result: folder "My Folder" of startup disk of application
"Finder"
```

This statement identifies the TeachText application:

```
application file id "ttxt"
-—result: file "TeachText 7.1" of startup disk of
application "Finder"
```

You can identify the frontmost window that lies under a particular point on the desktop by running a script like this:

```
tell application "Finder"
   content space {100,100}
end tell

--result: window of folder "Projects" of startup disk of
application "Finder"
```

The pair of integers in braces in the preceding example identifies the point in which you're interested, and the result contains a reference to the window that contains that point. It is usually easiest to use the term Content Space to obtain this kind of reference, because the script returns a reference even if the point is located on the desktop or in some specialized window such as an Information Window.

You can also specify more restricted window classes, such as Window, Container Window, Information Window, and so on. For example, if you request a reference to a container window for a point that's located in some other kind of window, the Finder returns an error.

**COMMANDS HANDLED**

Clean Up, Computer, Count, Data Size, Exists, Get, Print, Quit, Restart, Select, Shut Down, Sleep, Sort

**DEFAULT VALUE CLASS RETURNED**

Reference.

**EXAMPLES**

This script turns on the Calculate Folder Sizes property of the Views control panel:

```
tell application "Finder"
    set calculate folder sizes of view preferences to true
end tell
```

This script turns on file sharing for the local computer.

```
tell application "Finder"
    set file sharing to true
end tell
```

The next example selects all the unselected items in the active window.

```
tell application "Finder"
    repeat with x in front window
        set selected of x to not selected of x
    end repeat
end tell
```

**NOTES**

Although Startup Disk and Trash are properties of the desktop object, you can also refer to them as if they were properties of the Finder. Thus, the Finder interprets both of these statements are references to whatever disk is currently the startup disk:

```
startup disk
startup disk of desktop
```

Similarly, the Finder interprets both of these statements as a reference to
the Trash:

```
trash
trash of desktop
```

## Application File

An object of class Application File is an application's file on a disk. References
to application files must always include a complete description of the file's
container. Such references are different from standard AppleScript references
to application objects, which don't always require a complete pathname.

**PROPERTIES**

An application file has all the properties defined for object class File on page 61:
Creator Type, File Type, Locked, Product Version, Stationery, and Version.

Like any other file, an application file also has all the properties defined for
object class Item on page 73: Bounds, Comment, Container, Content Space,
Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index,
Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

Unlike other files, an application file also has these properties:

```
minimum partition size
```
> An integer indicating the minimum amount of memory,
> in bytes, that the application requires to run. This value is
> equivalent to the value in the box labeled "Minimum size"
> under "Memory Requirements" in the application file's
> information window (see "Notes").
> *Class:* Integer
> *Modifiable:* Yes

```
partition size
```
> An integer indicating the amount of memory, in bytes, that the
> application is launched with if a block of this size is available.
> This value is equivalent to the value in the box labeled "Preferred
> size" under "Memory Requirements" in the application file's
> information window (see "Notes" on page 40).
> *Class:* Integer
> *Modifiable:* Yes

scriptable     A Boolean value that indicates whether the application is high-
               level event aware (`true`) or not (`false`)—that is, whether it can
               respond to the Open, Run, Print, and Quit commands.
               *Class:* Boolean
               *Modifiable:* No

suggested partition size
               An integer indicating the amount of memory, in bytes, suggested
               for running the application. This value is equivalent to the value
               in the box labeled "Suggested size" under "Memory Require-
               ments" in the application file's information window.
               *Class:* Integer
               *Modifiable:* No

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Clean Up, Copy, Count, Data Size, Duplicate, Exists, Get, Make, Move, Open,
Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a file or, if you use the plural form `application files`, a list
of references.

**EXAMPLES**

The script that follows checks the Scriptable Text Editor's partition size before
opening the application. If the amount of memory available is greater than
the partition size, the script opens the application. If the amount of memory
available is less than the partition size, the script displays a dialog box warning
the user that the Scriptable Text Editor prefers more memory. This might be
useful if a script performs actions requiring more memory than an applica-
tion's minimum partition size permits.

```
tell application "Finder"
   set x to largest free block
   set y to partition size of ¬
      application file "Scriptable Text Editor" of startup disk
   if x > y then
      open application file "Scriptable Text Editor" of startup disk
   else
      display dialog ¬
         "The Scriptable Text Editor prefers more memory. " & ¬
         "To increase free memory, quit one or more applications."
   end if
end tell
```

This script returns a list of all the application files in a folder:

```
tell application "Finder"
   application files in folder "MyApps" of startup disk
end

--result: {file "Scriptable Text Editor" of folder "MyApps" of startup
disk, file "Script Editor" of folder "MyApps" of startup disk}
```

**NOTES**

The Minimum Partition Size, Partition Size, and Suggested Partition Size are all given in bytes. These values are shown in information windows in kilobytes (K, where 1K = 1024 bytes).

The value of the Partition Size property must be at least as high as the value of the Minimum Partition Size property. If you set the Minimum Partition Size to a value higher than the value of the Partition Size, the Finder also adjusts the Partition Size to match the new Minimum Partition Size. Similarly, if you set the Partition Size to a value lower than the value of the Minimum Partition Size, the Finder also adjusts the Minimum Partition Size to match the new Partition Size.

For example, suppose that the Partition Size and the Minimum Partition Size of the Scriptable Text Editor are both set to 384K. Running this script sets the

Minimum Partition size to 460,800 bytes (450K) and at the same time increases the Partition Size to 450K:

```
tell application "Finder"
   set minimum partition size of ¬
      application file "Scriptable Text Editor" of ¬
      application "Finder" to 460800
end tell
```

If you then run this script, the Finder sets the Partition Size of the Scriptable Text Editor back to 393,216 bytes (384K) and at the same time decreases the Minimum Partition Size to 384K:

```
tell application "Finder"
   set partition size of ¬
      application file "Scriptable Text Editor" of ¬
      application "Finder" to 393216
end tell
```

## Application Process

An object of class Application Process is a process that has been launched from an application file.

**PROPERTIES**

An application process has all the properties defined for object class Process on page 77: Creator Type, File, File Type, Frontmost, Name, Partition Size, Partition Space Used, Remote Events, Scriptable, and Visible.

An application process also has this property:

application file
> The application file from which this process was launched.
> *Class:* Reference
> *Modifiable:* No

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Count, Data Size, Exists, Get, Sort

**DEFAULT VALUE CLASS RETURNED**

A reference or, if you use the plural form `application processes`, a list of references of the form

`application "`*ApplicationProcessName*`"`

where *ApplicationProcessName* is the name of an application process as it appears in the Applications menu.

**EXAMPLES**

You can use a script like this to get a list of all the application processes that are currently running:

```
tell application "Finder"
   application processes
end
```

```
--result: {application "HyperCard", application "Script
Editor", application "Scriptable Text Editor"}
```

The following script hides the TeachText application process:

```
tell application "Finder"
   set visible of application process "TeachText" to false
end tell
```

## Container

An object of class Container is any Finder container, such as a folder, a disk, or the Trash.

**PROPERTIES**

A container has all the properties defined for object class object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

Unlike other items, a container also has these properties:

`completely expanded`
> A Boolean value that indicates whether a container and any folders it contains are completed expanded in list view (`true`) or not (`false`). For a container to be completely expanded, the window that contains the container must be open in list view (that is, any item other than "by Small Icon" or "by Icon" is selected in the Views menu for the window), the container and all nested folders within the it must be fully expanded with that same list view, and the container's own window must be closed. Before you can set this property, the window that contains the container must already be open in a list view.
> *Class:* Boolean
> *Modifiable:* Yes

`container window`
> A reference to the container's window.
> *Class:*
> *Modifiable:* No

`entire contents`
> A list of references to the entire contents of the container, including the contents of any other containers within the container.
> *Class:* List of references
> *Modifiable:* No

expandable   A Boolean value that indicates whether the container can
be opened within its container in outline view (`true`) or
not (`false`).
*Class:* Boolean
*Modifiable:* No

expanded    A Boolean value that indicates whether the item is open in
outline view (`true`) or not (`false`).
*Class:* Boolean
*Modifiable:* Yes

previous list view
An integer between 2 and 8 or one of the corresponding constants
`name`, `modification date`, `size`, `kind`, `comment`, `label
index`, or `version`. If the View property of the container is
currently one of these list views, the value of this property is the
current list view. If the View property is `small icon` or `icon`,
the value of this property is the last list view selected. If no list
view has been selected since the container was first opened, the
value of this property is `name`. You can't get or set this property
unless the container is open, and the value of the property isn't
retained after the container has been closed.
*Class:* Integer
*Modifiable:* No

selection   A list of references to the container elements that are currently
selected. The elements in the selection are those that would
be cut by a Cut command or copied by a Copy command. If no
elements are selected, the value of the Selection property is
an empty list.
*Class:* List of references
*Modifiable:* Yes

view        An integer between 0 and 8 or one of the corresponding constants
`small icon`, `icon`, `name`, `modification date`, `size`, `kind`,
`comment`, `label index`, or `version`, indicating the current
selection in the View menu for the container's window. You can't
get or set this property unless the container is open.
*Class:* Integer
*Modifiable:* Yes

**ELEMENT CLASSES**

Objects of these classes can be identified at the top level of a container by name or by number. Page numbers indicate the location of corresponding definitions in this chapter.

Accessory Suitcase (page 27)

Alias File (page 28)

Application File (page 38)

Container (page 43)

Control Panel (page 51)

Desk Accessory File (page 55)

Document File (page 60)

File (page 61)

Folder (page 63)

Font File (page 65)

Font Suitcase (page 66)

Item (page 73)

Sharable Container (page 79)

Sound File (page 88)

Suitcase (page 90)

**COMMANDS HANDLED**

Clean Up, Close, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move, Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a container or, if you use the plural form `containers`, a list of references.

The first example opens all the containers on the desktop except for the Trash.

```
tell application "Finder"
   open (containers in desktop whose name is not "Trash")
end tell
```

The next example, if saved as a script application, toggles the value of the Completely Expanded property of all containers at the top level of the front window: that is, it alternately expands all the containers and their nested containers or collapses them. The window in which the script application is located must be open in list view for the script to work on the window's contents.

```
property expan : false

tell application "Finder"
   if expan = false then
      set completely expanded of containers in ¬
         container of front window to true
      set expan to true
   else
      set completely expanded of containers in ¬
         container of front window to false
      set expan to false
   end if
end tell
```

The script begins by declaring the value of the expan property to be false. This property is a Boolean value that changes after the script is run. The script sets expan to true whenever it sets the Completely Expanded property of the front window's containers to true, or to false whenever it sets their Completely Expanded property to false.

The value of the property container of front window is a reference to the container to which the window belongs. This script must specify the Container property of the window, rather than the window itself, or it may not work consistently. A window's container always has a fixed number of elements until new elements are added or deleted, whereas a window open in list view has a variable number of elements depending on which folders are expanded.

## Container Window

An object of class Container Window is a window for a container. For example, the windows for a disk, a folder, or a suitcase are all container windows, while a control panel window, the About This Macintosh window, and an information window are not.

**PROPERTIES**

A container window has all the properties defined for object class Window on page 95: Bounds, Closeable, Floating, Index, Modal, Position, Resizable, Titled, Visible, Zoomable, and Zoomed.

Unlike other windows, a container window also has these properties:

container    A reference to the container to which the container
             window belongs.
             *Class:* Reference
             *Modifiable:* No

disk         A reference to the disk on which the container to which the
             window belongs is stored.
             *Class:* Reference
             *Modifiable:* No

folder       A reference to the folder to which the container window belongs
             (available only for container windows of folders).
             *Class:* Reference
             *Modifiable:* No

item         A reference to the item to which the container window belongs.
             *Class:* Reference
             *Modifiable:* No

previous list view
             An integer between 2 and 8 or one of the corresponding constants
             `name`, `modification date`, `size`, `kind`, `comment`, `label
             index`, or `version`. If the View property of the container
             window is currently one of these list views, the value of this
             property is the current list view. If the View property is `small
             icon` or `icon`, the value of this property is the last list view
             selected. If no list view has been selected since the container
             window was first opened, the value of this property is `name`.

You can't get or set this property unless the container window is open, and the value of the property isn't retained after the container window has been closed.
*Class:* Integer
*Modifiable:* No

selection    A reference to the current selection.
*Class:* Reference or list of references
*Modifiable:* No

view    An integer between 0 and 8 or one of the corresponding constants `small icon`, `icon`, `name`, `modification date`, `size`, `kind`, `comment`, `label index`, or `version`, indicating the current selection in the View menu for the container window. You can't get or set this property unless the container window is open.
*Class:* Integer
*Modifiable:* Yes

**ELEMENT CLASSES**

The elements of a container window include all the items currently displayed inside the window. When folders in a window whose View property is name, date, size, or kind are expanded in outline view, the elements of the expanded folder become elements of the container window in which the folder is located.

Objects of the classes listed here can be identified at the top level of a container window by name or by number. Page numbers indicate the location of definitions in this chapter.

Accessory Suitcase (page 27)

Alias File (page 28)

Application File (page 38)

Container (page 43)

Control Panel (page 51)

Desk Accessory File (page 55)

Document File (page 60)

File (page 61)

Folder (page 63)

**COMMANDS HANDLED**

Clean Up, Close, Count, Data Size, Exists, Get, Open, Print, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

Reference to a file or, if you use the plural form `container windows`, a list of references.

**EXAMPLE**

The script that follows tiles all the open container windows in an overlapping pattern and sets each window's View property to 3 ("by Names").

```
set {a, b, c, d} to {4, 44, 267, 311}
tell application "Finder"
   repeat with n from 1 to (count of container windows)
      if n = 1 then
         set bounds of container window n to {a, b, c, d}
      else
         set {a, b, c, d} to {a + 20, b + 20, c + 20, d + 20}
         set tiledBounds to {a, b, c, d}
         set bounds of container window n to tiledBounds
      end if
      set view of container window n to 3
      open container window n
   end repeat
end tell
```

The phrase `count of container windows` at the beginning of the Repeat loop requests that the Finder count all the container windows. Windows that aren't container windows, such as information windows, are ignored. The Finder returns an integer, which the Repeat statement uses as the number of times to repeat.

The If statement within the Repeat statement resets the bounds of each successive container window so that it is shifted down and to the right from the previously tiled window. After the View for each container window is set, the last statement within the Repeat statement sends an Open command to that window so that it overlaps the previously tiled window.

## Content Space

An object of class Content Space can be any Finder content space, including a container window, the desktop, an information window, a sharing window, a status window, or a window.

**PROPERTIES**

None

**ELEMENT CLASSES**

Like any window, a content space can contain any of the elements that can be contained by the object to which it belongs.

**COMMANDS HANDLED**

Clean Up, Close, Count, Data Size, Exists, Get, Open, Print, Sort, Update

**DEFAULT VALUE RETURNED**

Reference to a content space or, if you use the plural form `content spaces`, a list of references.

This script returns a list of all Finder content spaces that are currently open, including the desktop:

```
tell application "Finder"
    content spaces
end tell

--result: {information window of startup disk of
application "Finder", window of folder "Projects" of
startup disk of application "Finder", window of startup
disk of application "Finder", window of disk "Applications"
of application "Finder", content space of desktop of
application "Finder"}
```

For more information about using content spaces in scripts, see "References to Finder Windows," which begins on page 22.

## Control Panel

An object of class Control Panel is a control panel file on a disk.

A control panel has all the properties defined for object class File on page 61: Creator Type, File Type, Locked, Product Version, Stationery, and Version.

Like any other file, a control panel also has all the properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

Unlike other kinds of files and unlike other control panels, the Views control panel may also have these properties:

calculate folder sizes
A Boolean value that indicates whether the checkbox labeled "Calculate folder sizes" in the Views control panel is selected (`true`) or not (`false`).
*Class:* Boolean
*Modifiable:* Yes

comment heading
A Boolean value that indicates whether the checkbox labeled "Show comments" in the Views control panel is selected (`true`) or not (`false`).
*Class:* Boolean
*Modifiable:* Yes

date heading
A Boolean value that indicates whether the checkbox labeled "Show date" in the Views control panel is selected (`true`) or not (`false`).
*Class:* Boolean
*Modifiable:* Yes

disk information heading
A Boolean value that indicates whether the checkbox labeled "Show disk info in header" in the Views control panel is selected (`true`) or not (`false`).
*Class:* Boolean
*Modifiable:* Yes

icon size     An integer indicating the icon size selected under List Views in the Views control panel: 0 for the largest size, 1 for the medium size, and 2 for the smallest size.
*Class:* Integer
*Modifiable:* Yes

kind heading
A Boolean value that indicates whether the checkbox labeled "Show kind" in the Views control panel is selected (`true`) or not (`false`).
*Class:* Boolean
*Modifiable:* Yes

`label heading`
> A Boolean value that indicates whether the checkbox labeled "Show label" in the Views control panel is selected (`true`) or not (`false`).
> *Class:* Boolean
> *Modifiable:* Yes

`size heading`
> A Boolean value that indicates whether the checkbox labeled "Show size" in the Views control panel is selected (`true`) or not (`false`).
> *Class:* Boolean
> *Modifiable:* Yes

`snap to grid`
> A Boolean value that indicates whether the checkbox labeled "Always snap to grid" in the Views control panel is selected (`true`) or not (`false`).
> *Class:* Boolean
> *Modifiable:* Yes

`staggered grid`
> A Boolean value that indicates whether the checkbox labeled "Staggered grid" in the Views control panel is selected (`true`) or not (`false`).
> *Class:* Boolean
> *Modifiable:* Yes

`version heading`
> A Boolean value that indicates whether the checkbox labeled "Show version" in the Views control panel is selected (`true`) or not (`false`).
> *Class:* Boolean
> *Modifiable:* Yes

`view font`    The ID of the font selected in the Views control panel.
> *Class:* Integer
> *Modifiable:* Yes

`view font size`
> The size of the font selected in the Views control panel.
> *Class:* Integer
> *Modifiable:* Yes

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Clean Up, Close, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move,
Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

Reference to a file or, if you use the plural form `control panels`, a list of
references.

**EXAMPLES**

This script turns on the Snap to Grid property of the Views control panel:

```
tell application "Finder"
   set snap to grid of control panel "Views" of ¬
      control panels folder to true
end tell
```

You can also use the View Preferences property of the Finder application object
as a reference to the Views control panel. For example, this script toggles the
Date Heading property of the Views control panel on and off:

```
tell application "Finder"
   set date heading of view preferences to ¬
      not date heading of view preferences
end tell
```

## Desk Accessory File

An object of class Desk Accessory File is a desk accessory file on a disk.

**PROPERTIES**

A desk accessory file has all the properties defined for object class File on page 61: Creator Type, File Type, Locked, Product Version, Stationery, and Version.

Like any other file, a desk accessory file also has all the properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Clean Up, Close, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move, Open, Put Away, Reveal, Select, Set, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

Reference to a file or, if you use the plural form `desk accessory files`, a list of references.

**EXAMPLE**

This script opens all the desk accessory files listed in the Apple menu:

```
tell application "Finder"
    open desk accessory files in apple menu items folder
end tell
```

## Desktop-Object

An object of class Desktop-Object is the desktop of a computer. The Desktop property of the Finder application is a reference to an object of this class.

**PROPERTIES**

A desktop-object has some (but not all) of the properties defined for object class Container on page 43: Container Window, Entire Contents, Previous List View, Selection, and View.

A desktop-object also has some of the properties defined for object class Item on page 73: Comment, Content Space, Creation Date, Disk, Kind, Modification Date, Name, Physical Size, Selected, Size, and Window.

Unlike other containers, a desktop-object also has these properties:

`startup disk`
                A reference to the startup disk from which the Finder
                application was launched.

`trash`         A reference to the Trash folder.

**ELEMENT CLASSES**

Objects of these classes can be identified at the top level of the desktop-object by name or by number. Page numbers indicate the location of corresponding definitions in this chapter.

Accessory Suitcase (page 27)

Alias File (page 28)

Application File (page 38)

Container (page 43)

Control Panel (page 51)

Desk Accessory File (page 55)

Document File (page 60)

File (page 61)

Folder (page 63)

Font File (page 65)

Font Suitcase (page 66)

Item (page 73)

Sharable Container (page 79)

Sound File (page 88)

Suitcase (page 90)

**COMMANDS HANDLED**

Clean Up, Count, Data Size, Exists, Get, Print, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

Reference.

**EXAMPLE**

To refer to a desktop-object, you must use the Desktop property of the Finder application:

```
tell application "Finder"
    clean up desktop
end tell
```

For more information about the Desktop property, see "Properties and Elements of the Finder," which begins on page 20.

## Disk

An object of class Disk is any mounted volume on the desktop. In addition to hard disks and floppy disks, objects of class Disk can include RAM disks, tape drives, CD-ROM disks, and other storage devices.

**PROPERTIES**

A disk has all the properties defined for object class Sharable Container on page 79: Exported, Group, Group Privileges, Guest Privileges, Inherited Privileges, Mounted, Owner, Owner Privileges, Protected, Shared, Sharing Window.

Like any other sharable container, a disk also has all the properties defined for object class Container on page 43: Completely Expanded, Container Window, Entire Contents, Expandable, Expanded, Previous List View, Selection, and View.

In addition, like any other container, a disk has all properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

Unlike other sharable containers or containers, a disk also has the following properties:

capacity    The total number of bytes that can fit on the disk.
            *Class:* Integer
            *Modifiable:* No

ejectable   A Boolean value that indicates whether the disk can be ejected
            or put away (`true`) or not (`false`).
            *Class:* Boolean
            *Modifiable:* No

free space  The total number of bytes of free space remaining on the disk.
            *Class:* Integer
            *Modifiable:* No

local volume
            A Boolean value that indicates whether the disk is a local (`true`)
            or remote (`false`) volume.
            *Class:* Boolean
            *Modifiable:* No

startup     A Boolean value that indicates whether a disk is the startup
            disk (`true`) or not (`false`).
            *Class:* Boolean
            *Modifiable:* No

**ELEMENT CLASSES**

Objects of the classes listed here can be identified at the top level of a disk by name or by number. Page numbers indicate the location of corresponding definitions in this chapter.

Accessory Suitcase (page 27)

Alias File (page 28)

Application File (page 38)

Container (page 43)

Control Panel (page 51)

Desk Accessory File (page 55)

Document File (page 60)

File (page 61)

Folder (page 63)

Font File (page 65)

Font Suitcase (page 66)

Item (page 73)

Sharable Container (page 79)

Sound File (page 88)

Suitcase (page 90)

**COMMANDS HANDLED**

Clean Up, Close, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

Reference to a file or, if you use the plural form `disks`, a list of references.

EXAMPLE

The script that follows tests the Ejectable property of each item in a list of the mounted disks and ejects every disk that's ejectable.

```
tell application "Finder"
        put away (every disk whose ejectable is true)
end tell
```

## Document File

An object of class Document File is a document file on a disk.

PROPERTIES

A document file has all the properties defined for object class File on page 61: Creator Type, File Type, Locked, Product Version, Stationery, and Version.

Like any other file, a document file also has all the properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

ELEMENT CLASSES

None

COMMANDS HANDLED

Clean Up, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move, Open, Print, Put Away, Reveal, Select, Sort, Update

DEFAULT VALUE CLASS RETURNED

Reference to a file or, if you use the plural form `document files`, a list of references.

This script opens all the document files at the top level of a disk:

```
tell application "Finder"
    open document files in window "My World"
end tell
```

Application files, desk accessories, suitcases, and other nondocument files
are ignored.

# File

An object of class File can be any file on a disk, including document files,
application files, and so on.

A file has all the properties defined for object class Item on page 73: Bounds,
Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID,
Information Window, Kind, Label Index, Modification Date, Name, Physical
Size, Position, Selected, Size, and Window.

Unlike other items, a file also has these properties:

creator type
> A four-character code that indicates the creator type of the
> application that created the file.
> *Class:* String
> *Modifiable:* Yes

file type    A four-character code that indicates the type of the file.
> *Class:* String
> *Modifiable:* Yes

locked       A Boolean value that indicates whether the file is locked (`true`)
> or not (`false`).
> *Class:* Boolean
> *Modifiable:* Yes

product version
        The version number shown at the top of the information
        window for the file.
        *Class:* String
        *Modifiable:* No

stationery  A Boolean value that indicates whether the file is a stationery
        pad (`true`) or not (`false`).
        *Class:* Boolean
        *Modifiable:* Yes

version     The version number shown near the middle of the information
        window for the file.
        *Class:* String
        *Modifiable:* No

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Clean Up, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Make, Move,
Open, Print, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a file or, if you use the plural form `files`, a list of references.

**EXAMPLES**

This script returns a list of references to the files whose modification dates are
greater than a specified date:

```
tell application "Finder"
    files of startup disk whose modification date > ¬
        date "November 18, 1993"
end tell
```

If you save this script as a script application, it copies a new icon to any files whose icons you drop on the script's icon. The script uses the scripting addition command Choose File to ask the user to specify which file's icon is to be copied:

```
on open x
   tell application "Finder"
      choose file with prompt ¬
         "Choose the file whose icon you want to copy:"
      set newIcon to icon of the result
      repeat with i in x
         set the icon of i to newIcon
      end repeat
   end tell
end open
```

## Folder

An object of class Folder is a folder on a disk.

### PROPERTIES

A folder has all the properties defined for object class Sharable Container on page 79: Exported, Group, Group Privileges, Guest Privileges, Inherited Privileges, Mounted, Owner, Owner Privileges, Protected, Shared, Sharing Window.

Like any other sharable container, a folder also has all the properties defined for object class Container on page 43: Completely Expanded, Container Window, Entire Contents, Expandable, Expanded, Previous List View, Selection, and View.

In addition, like any other container, a folder has all properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

**ELEMENT CLASSES**

Objects of these classes can be identified at the top level of the folder by name or by number. Page numbers indicate the location of corresponding definitions in this chapter.

Accessory Suitcase (page 27)

Alias File (page 28)

Application File (page 38)

Container (page 43)

Control Panel (page 51)

Desk Accessory File (page 55)

Document File (page 60)

File (page 61)

Folder (page 63)

Font File (page 65)

Font Suitcase (page 66)

Item (page 73)

Sharable Container (page 79)

Sound File (page 88)

Suitcase (page 90)

**COMMANDS HANDLED**

Clean Up, Close, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Make, Move, Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference or, if you use the plural form `folders`, a list of references.

**EXAMPLE**

This script returns references to all the folders at the top level of the startup disk that have been modified in the last 24 hours:

```
tell application "Finder"
   folders of startup disk where ¬
      modification date > ((current date) - 24 * hours)
end tell
```

## Font File

An object of class Font File is a file that contains font data.

**PROPERTIES**

A font file has all the properties defined for object class File on page 61: Creator Type, File Type, Locked, Product Version, Stationery, and Version.

Like any other file, a font file also has all the properties defined for object class Item on page 73: Bounds, Comment, Content Space, Container, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Clean Up, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move, Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a file or, if you use the plural form `font files`, a list of references.

**EXAMPLE**

If you save this script as a script application, it opens font suitcases or any other containers you drop on its icon and opens the windows for any font files they contain.

```
on open x
    repeat with i in x
        tell application "Finder"
            open i
            open font files in i
        end tell
    end repeat
end open
```

## Font Suitcase

An object of class Font Suitcase is a suitcase file that contains font information.

**PROPERTIES**

A font suitcase has all the properties defined for object class Container on page 43: Completely Expanded, Container Window, Entire Contents, Expandable, Expanded, Previous List View, Selection, and View.

Like any other container, a font suitcase also has all the properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

**ELEMENT CLASSES**

Items within a font suitcase can be identified by name or by number. For more information, see the definition of object class Item on page 73.

**COMMANDS HANDLED**

Clean Up, Close, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move, Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a suitcase or, if you use the plural form `font suitcases`, a list of references.

**EXAMPLE**

The script that follows moves a font suitcase called Janson Text into or out of the Fonts folder, first asking the user to confirm the move. (If you are using version 1.1 of the Finder scripting software, a second dialog box appears when the Finder is moving suitcases into the Fonts folder, warning the user that the suitcases won't be available to running applications right away.)

```
tell application "Finder"
   activate
   set x to name of items in fonts folder
   if "Janson Text" is in x then
      set response to display dialog ¬
         "Remove the Janson fonts from the System?"
      if button returned of response is "OK" then
         move font suitcase "Janson Text" of fonts folder to ¬
            folder "Font Storage" of startup disk
      end if
   else
      set response to display dialog ¬
         "Move the Janson fonts into the System?"
      if button returned of response is "OK" then
         move font suitcase "Janson Text" ¬
            of folder "Font Storage" of startup disk to fonts folder
      end if
   end if
end tell
```

## Group

An object of class Group is a group file in the Users & Groups control panel window. The Finder can't recognize or create a group unless the Users & Groups control panel window is open.

**PROPERTIES**

bounds
: The coordinates of the rectangle that bounds the content region of the item's icon.
*Class:* List of four integers (Bounding Rectangle data type). The first two integers specify the coordinates of the upper-left corner of the item's icon, and the last two integers specify the coordinates of the lower-right corner of the icon.
*Modifiable?* Yes

icon
: A bitmap of the item's icon.
*Class:* Icon family (data type defined by Finder)
*Modifiable:* Yes

label index
: The number of the label currently selected for the item in the Label menu (None = 0).
*Class:* Integer
*Modifiable:* Yes

name
: The group's name.
*Class:* String
*Modifiable:* Yes

position
: Two integers that specify the position of the upper-left corner of the group's icon.
*Class:* List of two integers (Point data type) that specify the coordinates of the upper-left corner of the group's icon
*Modifiable?* Yes

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Clean Up, Close, Count, Data Size, Exists, Get, Make, Open, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a group or, if you use the plural form `groups`, a list of references.

**EXAMPLE**

This script displays a dialog box requesting the name of the new group to create, then opens the control panel Users & Groups, creates the new group, and sets its name:

```
tell application "Finder"
   set response to display dialog ¬
      "Name of new group:" default answer ""
   set newGroup to text returned of response
   set cPanels to control panel "Users & Groups" of ¬
      control panels folder
   open cPanels
   make group at cPanels with properties {name: newGroup}
end tell
```

## Information Window

An object of class Information Window is the window that opens when you select an item and choose Get Info from the File menu.

**PROPERTIES**

An information window has all the properties defined for object class Window on page 95: Bounds, Closeable, Floating, Index, Modal, Position, Resizable, Titled, Visible, Zoomable, and Zoomed.

Unlike other windows, an information window also has these properties:

comment        The comment displayed in the information window.
               *Class:* String
               *Modifiable:* Yes

creation date
               The date on which the item to which the information window
               belongs was created.
               *Class:* Date
               *Modifiable:* No

icon           A bitmap of the icon for the item to which the information
               window belongs.
               *Class:* Icon family (data type defined by Finder)
               *Modifiable:* Yes

item           A reference to the item to which the information window
               belongs.
               *Class:* Reference
               *Modifiable:* No

locked         A Boolean value that indicates whether the file to which the
               information window belongs is locked (`true`) or not (`false`).
               *Class:* Boolean
               *Modifiable:* Yes

minimum partition size
               An integer indicating the minimum amount of memory, in
               bytes, that an application requires to run (available only for
               information windows that belong to application files). This
               value is equivalent to the value in the box labeled "Minimum
               size" under "Memory Requirements" in the information
               window (see "Notes").
               *Class:* Integer
               *Modifiable:* Yes

modification date
               The date on which the item to which the information window
               belongs was most recently modified.
               *Class:* Date
               *Modifiable:* No

`partition size`
> An integer indicating the amount of memory, in bytes, that an application is launched with if a block of this size is free (available only for information windows that belong to application files). This value is equivalent to the value in the box labeled "Preferred size" under "Memory Requirements" in the information window (see "Notes").
> *Class:* Integer
> *Modifiable:* Yes

`physical size`
> The physical size of the item on disk, in bytes.
> *Class:* Integer
> *Modifiable:* No

`product version`
> The version number shown at the top of the information window.
> *Class:* String
> *Modifiable:* No

`size`　The logical size of the item on disk, in bytes.
> *Class:* Integer
> *Modifiable:* No

`stationery`　A Boolean value that indicates whether a document file is a stationery pad (`true`) or not (`false`) (available only for information windows that belong to document files.)
> *Class:* Boolean
> *Modifiable:* Yes

`suggested partition size`
> An integer indicating the amount of memory, in bytes, suggested for running an application (available only for information windows that belong to application files). This value is equivalent to the value in the box labeled "Suggested size" under "Memory Requirements" in the information window (see "Notes").
> *Class:* Integer
> *Modifiable:* No

`version`　The version number shown near the middle of the information window.
> *Class:* String
> *Modifiable:* No

`warn before emptying`
> A Boolean value that indicates whether the Finder should warn the user before emptying the Trash (`true`) or not (`false`) (available only for the Trash).
> *Class:* String
> *Modifiable:* Yes

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Close, Count, Data Size, Exists, Get, Open, Sort

**DEFAULT VALUE CLASS RETURNED**

A reference to an information window or, if you use the plural form `information windows`, a list of references.

**EXAMPLE**

This script opens the information windows for all application files in a disk whose partition sizes are greater than 1000K:

```
tell application "Finder"
   open (information window of application files of ¬
      disk "Applications" whose partition size > 1024000)
end tell
```

**NOTES**

The Minimum Partition Size, Partition Size, and Suggested Partition Size are all given in bytes. These values are shown in information windows in kilobytes (K, where 1K = 1024 bytes). For more information about the relationships among these properties, which are identical to the properties of the same name defined for class Application File, see page 40.

## Item

An object of class Item is any item in a container.

**PROPERTIES**

bounds    The coordinates of the rectangle that bounds the content region
          of the item's icon.
          *Class:* List of four integers (Bounding Rectangle data type).
          The first two integers specify the coordinates of the upper-
          left corner of the item's icon, and the last two integers specify
          the coordinates of the lower-right corner of the icon.
          *Modifiable?* Yes

comment   The comment displayed in the item's information window.
          *Class:* String
          *Modifiable:* Yes

container A reference to the container in which the item is located.
          *Class:* Reference
          *Modifiable:* No

content space
          A reference to the content space that opens when the item
          is opened.
          *Class:* Content Space
          *Modifiable:* No

creation date
          The date on which the item was created.
          *Class:* Date
          *Modifiable:* No

disk      A reference to the disk on which the item is stored.
          *Class:* Reference
          *Modifiable:* No

folder    A reference to the folder, if any, in which the item is stored.
          *Class:* Reference
          *Modifiable:* No

icon      A bitmap of the item's icon.
          *Class:* Icon family (data type defined by Finder)
          *Modifiable:* Yes

id The item's hierarchical file system (HFS) ID number, an integer that identifies an application file, folder, or disk. (For examples of the use of this property, see page 35.)
*Class:* Integer
*Modifiable:* No

information window
A reference to the item's information window.
*Class:* Reference
*Modifiable:* No

kind The kind of item as listed under Kind in the item's container window.
*Class:* String
*Modifiable:* No

label index
The number of the label currently selected for the item in the Label menu (None = 0).
*Class:* Integer
*Modifiable:* Yes

modification date
The date on which the item was most recently modified.
*Class:* Date
*Modifiable:* No

name The item's name.
*Class:* String
*Modifiable:* Yes

physical size
The physical size of the item on disk, in bytes.
*Class:* Integer
*Modifiable:* No

position Two integers that specify the position of the upper-left corner of the item's icon.
*Class:* List of two integers (Point data type)
*Modifiable?* Yes

selected A Boolean value that indicates whether the item is selected (`true`) or not (`false`).
*Class:* Boolean
*Modifiable:* Yes

size         The logical size of the item on disk, in bytes.
             *Class:* Integer
             *Modifiable:* No

window       A reference to the window that opens when the item is opened.
             *Class:* Reference
             *Modifiable:* No

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Clean Up, Close, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move, Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a file, folder, disk, the Trash, or the desktop or, if you use the plural form items, a list of references.

**EXAMPLES**

The script that follows moves a group of items that start with an at (@) symbol back and forth between the Apple Menu Items folder and a folder at the top level of the startup disk called Apple Menu Extras.

To work correctly, this script must be stored as a script application at the top level of the startup disk; the startup disk must also contain a folder called Apple Menu Extras to store the items you want to move in and out of the Apple Menu Items folder; and the names of the items in the Apple Menu Items folder must all begin with an @ symbol. It is also important, as with any script, to have some free memory available in addition to the memory used by any currently running processes—preferably several hundred kilobytes.

```
property Extras : false

tell application "Finder"

   if (exists item " Add Extras" of apple menu items folder) ¬
      = false and ¬
      (exists item " Remove Extras" of apple menu items folder) ¬
         = false then
      make alias file with data (file "Toggle Apple Menu Extras" of ¬
         startup disk) at apple menu items folder ¬
         with properties {name:" Add Extras"}
   end if

   if Extras is false then
      move contents of folder "Apple Menu Extras" of startup disk to ¬
         apple menu items folder
      set the name of item " Add Extras" in apple menu items folder ¬
         to " Remove Extras"
      set Extras to true
   else
      move (every item of apple menu items folder ¬
         whose name begins with "@") to ¬
         folder "Apple Menu Extras" of startup disk
      set the name of item " Remove Extras" in ¬
         apple menu items folder to " Add Extras"
      set Extras to false
   end if

end tell
```

The script begins by declaring the value of the `Extras` property to be `false`. This value may change after the script is run.

If the script hasn't been run before, the first If statement in the script creates an alias file of the script application in the Apple Menu Items folder with the name Add Extras. This alias file appears in the Apple menu along with all the other

items in the Apple Menu Items folder, and its name changes to Remove Extras after the extra items have been added to that folder.

If the value of `Extras` is `false`, the first part of the second If statement moves the contents of the Apple Menu Extras folder into the Apple Menu Items folder, changes the name of the alias file to Remove Extras, and sets the value of `Extras` to `true`.

If the value of `Extras` is `true`, the second part of the second If statement moves all items in the Apple Menu Items folder that begin with an @ symbol into the Apple Menu Extras folder, changes the name of the alias file to Add Extras, and sets the value of `Extras` to `false`.

## Process

An object of object class Process is a running process.

**PROPERTIES**

`creator type`
 A four-character code that indicates the creator type of the application from which the process was launched.
 *Class:* String
 *Modifiable:* No

`file` A reference to the file from which the process was launched.
 *Class:* Reference
 *Modifiable:* No

`file type` A four-character code that indicates the file type of the file.
 *Class:* String
 *Modifiable:* No

`frontmost` A Boolean value that indicates whether the process is the frontmost process (`true`) or not (`false`).
 *Class:* Boolean
 *Modifiable:* Yes

`name` The name of the process.
 *Class:* Reference
 *Modifiable:* No

partition size

>An integer indicating the amount of memory, in bytes, that a process is launched with if a free block of this size exists. This value is equivalent to the value in the box labeled "Preferred size" under "Memory Requirements" in the information window for the application file from which the process was launched, except that the value in the information window is given in kilobytes (K, where 1K = 1024 bytes).
>*Class:* Integer
>*Modifiable:* No

partition space used

>The number of bytes currently used in the partition for this process.
>*Class:* Integer
>*Modifiable:* No

remote events

>A Boolean value that indicates whether this process accepts (`true`) or doesn't accept (`false`) remote events.
>*Class:* Boolean
>*Modifiable:* No

scriptable A Boolean value that indicates whether the process is high-level event aware (`true`) or not (`false`)—that is, whether it can respond to the Open, Run, Print, and Quit commands.
>*Class:* Boolean
>*Modifiable:* No

visible A Boolean value that indicates whether the process is currently hidden or not.
>*Class:* Boolean
>*Modifiable:* Yes

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Count, Data Size, Exists, Get, Sort

**DEFAULT VALUE CLASS RETURNED**

A reference or (if you use the plural form `processes`) a list of references of the form

```
application "ProcessName"
```

where *ProcessName* is the name of a process as it appears in the Applications menu.

**EXAMPLE**

In the first version of the Finder scripting software, this script returns a list of all processes except the Finder itself:

```
tell application "Finder"
    every process
end tell
```

In later versions, the list of processes returned includes the Finder. To exclude the Finder, use the statement `every process whose file type is not "FNDR"`.

## Sharable Container

An object of class Sharable Container is a container that can be shared.

**PROPERTIES**

A sharable container has all the properties defined for object class Container on page 43: Completely Expanded, Container Window, Entire Contents, Expandable, Expanded, Previous List View, Selection, and View.

Like any other container, a sharable container also has all the properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

Unlike other containers, a sharable container also has these properties:

exported          A Boolean value that indicates whether the sharable container is either shared or inside another container whose contents are being shared. If one of these conditions is true, the value is `true` and the sharable container can be mounted or accessed remotely; if neither of these conditions is true, the value is `false` and the container isn't available to remote computers.
                  *Class:* Boolean
                  *Modifiable:* No

group             The name of the user or group with special access to the sharable container, as indicated in the pop-up menu labeled User/Group in the sharing window for the container.
                  *Class:* String
                  *Modifiable:* Yes

group privileges
                  The group privileges shown in the Sharing window for the container. For example, if the See Folders checkbox is not selected and the See Files and Make Changes checkboxes are selected for a user or group, the value of this property is `"See Files, Make Changes"` (see "Notes" on page 83).
                  *Class:* Sharing Privileges
                  *Modifiable:* Yes

guest privileges
                  The guest privileges shown in the sharing window for the container (that is, the settings labeled Everybody). For example, if the See Folders checkbox is not selected and the See Files and Make Changes checkboxes are selected for Everybody, the value of this property is `"See Files, Make Changes"` (see "Notes" on page 83).
                  *Class:* Sharing Privileges
                  *Modifiable:* Yes

inherited privileges
                  A Boolean value that indicates whether the privileges for the sharable container are the same as the privileges for the container in which it is stored (`true`) or not (`false`). This property corresponds to the checkbox in the sharing window labeled "Same as enclosing folder". The value of this property is

set to `false` if you set any of the group, guest, or owner privileges individually.
*Class:* Boolean
*Modifiable:* Yes

mounted          A Boolean value that indicates whether the sharable container is mounted on another computer's desktop (`true`) or not (`false`).
*Class:* Boolean
*Modifiable:* No

owner            The name of the sharable container's owner as indicated in the sharing window for the container.
*Class:* String
*Modifiable:* Yes

owner privileges
                 The owner privileges shown in the sharing window for the sharable container. For example, if the See Folders checkbox is not selected and the See Files and Make Changes checkboxes are selected for the container's owner, the value of this property is `"See Files, Make Changes"` (see "Notes" on page 83).
*Class:* Sharing Privileges
*Modifiable:* Yes

protected        A Boolean value that indicates whether the sharable container is protected (`true`) or not (`false`). This property corresponds to the checkbox in the sharable container's sharing window labeled "Can't be moved, renamed, or deleted".
*Class:* Boolean
*Modifiable:* Yes

shared           A Boolean value that indicates whether the sharable container and its contents are shared (`true`) or not (`false`). This property corresponds to the checkbox in the sharable container's sharing window labeled "Share this item and its contents". If the value is `true`, the sharable container can be mounted on the desktop of a remote computer.
*Class:* Boolean
*Modifiable:* Yes

sharing window
                 A reference to the sharing window for the sharable container.
*Class:* Reference
*Modifiable:* No

**ELEMENT CLASSES**

Objects of these classes can be identified at the top level of a sharable container by name or by number. Page numbers indicate the location of corresponding definitions in this chapter.

Accessory Suitcase (page 27)

Alias File (page 28)

Application File (page 38)

Container (page 43)

Control Panel (page 51)

Desk Accessory File (page 55)

Document File (page 60)

File (page 61)

Folder (page 63)

Font File (page 65)

Font Suitcase (page 66)

Item (page 73)

Sharable Container (page 79)

Sound File (page 88)

Suitcase (page 90)

**COMMANDS HANDLED**

Clean Up, Close, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move, Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a disk or a folder or, if you use the plural form `sharable containers`, a list of references.

This script changes the Shared property to `false` for all sharable containers that belong to a specified owner:

```
tell application "Finder"
   set shared of sharable containers whose owner is "Greg" to false
end tell
```

**NOTES**

The properties Group Privileges, Guest Privileges, and Owner Privileges each return an object of class Sharing Privileges. You can test for a specific privilege by checking the values of the Sharing Privileges properties as described in the next section.

## Sharing Privileges

An object of class Sharing Privileges corresponds to the group privileges, guest privileges, or owner privileges selected in the sharing window for a sharable container.

**PROPERTIES**

make changes
> A Boolean value that corresponds to the Make Changes checkbox for a specified user or group is selected in a sharable container's sharing window.
> *Class:* Boolean
> *Modifiable:* Yes

see files   A Boolean value that corresponds to the See Files checkbox for a specified user or group is selected in a sharable container's sharing window.
> *Class:* Boolean
> *Modifiable:* Yes

see folders

> A Boolean value that corresponds to the See Folders checkbox
> for a specified user or group is selected in a sharable container's
> sharing window.
> *Class:* Boolean
> *Modifiable:* Yes

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

None

**DEFAULT VALUE CLASS RETURNED**

None

**EXAMPLE**

This script returns a Boolean value that indicates whether the checkbox labeled
"See files" for the current User/Group in the startup disk's sharing window is
selected or not:

```
tell application "Finder"
   see files of group privileges of startup disk
end tell

--result: true
```

## Sharing Window

An object of class Sharing Window is a sharing window that appears when you choose Sharing from the File menu while a sharable container is selected.

**PROPERTIES**

A sharing window has all the properties defined for object class Window on page 95: Bounds, Closeable, Floating, Index, Modal, Position, Resizable, Titled, Visible, Zoomable, and Zoomed.

Unlike other windows, a sharing window also has these properties:

container    A reference to the container to which the sharing
             window belongs.
             *Class:* Reference
             *Modifiable:* No

exported     A Boolean value that indicates whether the container to which
             the sharing window belongs is either shared or inside another
             container whose contents are being shared. If one of these
             conditions is true, the value is `true` and the container can be
             mounted or accessed remotely; if neither of these conditions is
             true, the value is `false` and the container isn't available to
             remote computers.
             *Class:* Boolean
             *Modifiable:* No

folder       A reference to the folder, if any, to which the sharing
             window belongs.
             *Class:* Reference
             *Modifiable:* No

group        The name of the user or group selected in the pop-up menu
             labeled User/Group.
             *Class:* String
             *Modifiable:* Yes

group privileges

>   The group privileges selected in the sharing window. For
>   example, if the See Folders checkbox is not selected and the
>   See Files and Make Changes checkboxes are selected for a user
>   or group, the value of this property is `"See Files, Make
>   Changes"` (see page 83 for the definition of Sharing Privileges).
>   *Class:* Sharing Privileges
>   *Modifiable:* Yes

guest privileges

>   The guest privileges selected in the sharing window (that is,
>   the settings labeled Everybody). For example, if the See Folders
>   checkbox is not selected and the See Files and Make Changes
>   checkboxes are selected for Everybody, the value of this property
>   is `"See Files, Make Changes"` (see page 83 for the defini-
>   tion of Sharing Privileges).
>   *Class:* Sharing Privileges
>   *Modifiable:* Yes

inherited privileges

>   A Boolean value that indicates whether the privileges for the
>   object to which the sharing window belongs are the same as the
>   privileges for the container in which the object is stored (`true`)
>   or not (`false`). This property corresponds to the checkbox in
>   the sharing window labeled "Same as enclosing folder". The
>   value of this property is set to `false` if you set any of the group,
>   guest, or owner privileges individually.
>   *Class:* Boolean
>   *Modifiable:* Yes

item

>   The item to which the sharing window belongs.
>   *Class:* Reference
>   *Modifiable:* No

mounted

>   A Boolean value that indicates whether the sharable container to
>   which the sharing window belongs is mounted on another
>   computer's desktop (`true`) or not (`false`).
>   *Class:* Boolean
>   *Modifiable:* No

owner

>   The name of the sharable container's owner.
>   *Class:* String
>   *Modifiable:* Yes

owner privileges
The owner privileges selected in the sharing window. For example, if the See Folders checkbox is not selected and the See Files and Make Changes checkboxes are selected for the container's owner, the value of this property is `"See Files, Make Changes"` (see page 83 for the definition of Sharing Privileges).
*Class:* Sharing Privileges
*Modifiable:* Yes

protected    A Boolean value that indicates whether the sharable container is protected (`true`) or not (`false`). A protected container can't be moved, renamed, or deleted. This property corresponds to the checkbox labeled "Can't be moved, renamed, or deleted" and is not available for disks.
*Class:* Boolean
*Modifiable:* Yes

sharable container
The sharable container to which the sharing window belongs.
*Class:* Reference
*Modifiable:* No

shared    A Boolean value that indicates whether the sharable container to which the sharing window belongs is shared (`true`) or not (`false`). This property corresponds to the checkbox in the sharing window labeled "Share this item and its contents". If the value is `true`, the sharable container to which the sharing window belongs can be mounted on the desktop of a remote computer.
*Class:* Boolean
*Modifiable:* Yes

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Close, Count, Data Size, Exists, Get, Open, Sort

**DEFAULT VALUE CLASS RETURNED**

A reference to a sharing window or, if you use the plural form `sharing windows`, a list of references.

**EXAMPLE**

This script opens the sharing windows for sharable containers on the desktop that belong to a specified owner:

```
tell application "Finder"
   open (sharing window of sharable containers whose owner is "John")
end tell
```

## Sound File

An object of class Sound File is a sound file.

**PROPERTIES**

A sound file has all the properties defined for object class File on page 61: Creator Type, File Type, Locked, Product Version, Stationery, and Version.

Like any other file, a sound file also has all the properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Clean Up, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move, Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a file or, if you use the plural form `sound files`, a list of references.

**EXAMPLE**

This script plays the sound file MySound:

```
tell application "Finder"
    open file "MySound" of folder "Sounds" of startup disk
end tell
```

## Status Window

An object of class Status Window is a window indicating the progress of an operation in the Finder—for example, the windows that appear when items are being copied or when the Trash is being emptied.

**PROPERTIES**

A status window has all the properties defined for object class Window on page 95: Bounds, Closeable, Floating, Index, Modal, Position, Resizable, Titled, Visible, Zoomable, and Zoomed.

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Count, Data Size, Exists, Get

**DEFAULT VALUE CLASS RETURNED**

A reference to a status window or, if you use the plural `status windows`, a list of references.

**EXAMPLE**

If a status window is open, this script returns its name:

```
tell application "Finder"
    front status window
end tell


--result: status window "Copy" of application "Finder"
```

## Suitcase

An object of class Suitcase can be either an accessory suitcase or a font suitcase.

**PROPERTIES**

A suitcase has all the properties defined for object class Container on page 43: Completely Expanded, Container Window, Entire Contents, Expandable, Expanded, Previous List View, Selection, and View.

Like any other container, a suitcase also has all the properties defined for object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

**ELEMENT CLASSES**

The only elements that a suitcase can contain are objects of class Item (see page 73 for the definition of Item). Items can be identified within a suitcase by name or by number.

**COMMANDS HANDLED**

Clean Up, Close, Copy, Count, Data Size, Delete, Duplicate, Exists, Get, Move, Open, Put Away, Reveal, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

Reference to a suitcase or, if you use the plural form `suitcases`, a list of references.

**EXAMPLE**

This script returns a list of references to the suitcases at the top level of the startup disk:

```
tell application "Finder"
    suitcases of startup disk
end tell

--result: {suitcase "DAs" of startup disk of application
"Finder", suitcase "Extra Fonts" of startup disk of
application "Finder"}
```

## Trash-Object

An object of class Trash-Object is the Trash for a specific Finder application. The Trash property of the Finder application is an object of this class.

**PROPERTIES**

A trash-object has all the properties defined for object class Container on page 43: Completely Expanded, Container Window, Entire Contents, Expandable, Expanded, Previous List View, Selection, and View.

Like any other container, a trash-object also has all the properties defined for object class object class Item on page 73: Bounds, Comment, Container, Content Space, Creation Date, Disk, Folder, Icon, ID, Information Window, Kind, Label Index, Modification Date, Name, Physical Size, Position, Selected, Size, and Window.

Unlike other containers, a trash-object also has these properties:

```
warn before emptying
```
> A Boolean value that indicates whether the Warn Before
> Emptying checkbox in the Trash's Information Window is
> selected (`true`) or not (`false`).
> *Class:* Boolean
> *Modifiable:* Yes

**ELEMENT CLASSES**

Objects of these classes can be identified at the top level of a trash-object by name or by number. Page numbers indicate the location of corresponding definitions in this chapter.

Accessory Suitcase (page 27)

Alias File (page 28)

Application File (page 38)

Container (page 43)

Control Panel (page 51)

Desk Accessory File (page 55)

Document File (page 60)

File (page 61)

Folder (page 63)

Font File (page 65)

Font Suitcase (page 66)

Item (page 73)

Sharable Container (page 79)

Sound File (page 88)

Suitcase (page 90)

**COMMANDS HANDLED**

Count, Exists, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

Reference.

To refer to a trash-object, you must use the Trash property of the desktop:

```
tell application "Finder"
   clean up trash
end tell
```

Note that it's not necessary to refer to the `trash of desktop`; the Finder interprets `trash` as the desktop's Trash property.

## User

An object of class User is a user file in the Users & Groups control panel window. The Finder can't recognize or create a user unless the Users & Groups control panel window is open.

PROPERTIES

bounds      The coordinates of the rectangle that bounds the content region of the user's icon.
            *Class:* List of four integers (Bounding Rectangle data type).
            The first two integers specify the coordinates of the upper-left corner of the user's icon, and the last two integers specify the coordinates of the lower-right corner of the icon.
            *Modifiable:* Yes

icon        A bitmap of the user's icon.
            *Class:* Icon family (data type defined by Finder)
            *Modifiable:* Yes

label index
            The number of the label currently selected for the user in the Label menu (None = 0).
            *Class:* Integer
            *Modifiable:* Yes

name        The user's name.
            *Class:* String
            *Modifiable:* Yes

position    Two integers that specify the position of the upper-left corner
            of the user's icon.
            *Class:* List of two integers (Point data type)
            *Modifiable?* Yes

**ELEMENT CLASSES**

None

**COMMANDS HANDLED**

Clean Up, Close, Count, Data Size, Exists, Get, Make, Open, Select, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

A reference to a user or, if you use the plural form `users`, a list of references.

**EXAMPLE**

The script that follows displays a dialog box requesting the name of the new
user file to create, then opens the control panel Users & Groups, creates the
new user file, and sets the user's name. Listing 1-3 on page 12 incorporates this
script into a longer script that also creates a drop folder for the new user.

```
tell application "Finder"
   set response to display dialog ¬
      "Name of new user:" default answer ""
   set newUser to text returned of response
   set cPanels to control panel "Users & Groups" of ¬
      control panels folder
   open cPanels
   make user at cPanels with properties {name: newUser}
end tell
```

# Window

An object of class window is any Finder window.

**PROPERTIES**

bounds        The rectangle that bounds the content region of the window.
              (The "window frame"—the title bar and scroll bars—are not
              part of the content region.)
              *Class:* List of four integers (Bounding Rectangle data type). The
              first two integers specify the coordinates of the upper-left corner
              of the window, and the last two integers specify the coordinates
              of the lower-right corner of the window.
              *Modifiable?* Yes

closeable     A Boolean value that indicates whether the window has a close
              box (true) or not (false).
              *Class:* Boolean
              *Modifiable?* No

floating      A Boolean value that indicates whether the window is a floating
              window (true) or not (false). (A floating window is a window
              that appears in front of all other windows.) No Finder windows
              are floating windows.
              *Class:* Boolean
              *Modifiable?* No

index         The number of the window (window 1 is the frontmost window,
              window 2 is the window immediately behind window 1, and
              so on).
              *Class:* Integer
              *Modifiable?* Yes

modal         A Boolean value that indicates whether the window is modal
              (true) or not (false). (A modal window is one that requires
              a response from the user before the user can perform any
              other tasks).
              *Class:* Boolean
              *Modifiable?* No

position    Two integers that specify the upper-left corner of the content
            region of the window. (The "window frame"—the title bar and
            scroll bars—is not part of the content region.)
            *Class:* List of two integers (Point data type)
            *Modifiable?* Yes

resizable   A Boolean parameter that indicates whether the window can be
            resized (`true`) or not (`false`).
            *Class:* Boolean
            *Modifiable?* No

titled      A Boolean parameter that indicates whether the window has a
            title bar (`true`) or not (`false`).
            *Class:* Boolean
            *Modifiable?* No

visible     A Boolean parameter that indicates whether the window is
            visible (`true`) or not (`false`). The value of this property is
            always `false` for Finder windows.
            *Class:* Boolean
            *Modifiable?* No

zoomable    A Boolean parameter that indicates whether the window can be
            zoomed (`true`) or not (`false`).
            *Class:* Boolean
            *Modifiable?* No

zoomed      A Boolean parameter that specifies whether the window is full
            size (`true`) or not (`false`).
            *Class:* Boolean
            *Modifiable?* Yes

**ELEMENT CLASSES**

If the window is a container window, its elements include all the items currently
displayed inside the window. When folders in a window whose View property
is name, date, size, or kind are expanded in outline view, the elements of the
expanded folder become elements of the container window in which the folder
is located.

Information windows, sharing windows, and status windows have no elements.

**COMMANDS HANDLED**

Clean Up, Close, Count, Data Size, Exists, Get, Open, Print, Sort, Update

**DEFAULT VALUE CLASS RETURNED**

Reference.

**EXAMPLE**

This script closes all Finder windows. It has the same effect as holding down the Option key and clicking the close box of the front window.

```
tell application "Finder"
   close windows
end tell
```

For an example of a script that either takes a snapshot of the current window arrangement or restores a previously stored snapshot, see Listing 1-2 on page 11.

**NOTES**

In addition to the properties listed here, a Finder window can inherit certain properties, such as Name, from the object to which it belongs.

```
tell application "Finder"
   name of front window
end tell

--result: "My World"
```

# Finder Commands

This chapter begins with a summary of the format used in command defini-
tions. The rest of the chapter provides complete definitions of the Finder
commands.

## Using Command Definitions

Command definitions provide information about what commands do and how
to use them in scripts. Each definition contains at least four kinds of information:
syntax, parameters, results, and examples. These categories are briefly described
here, using examples from the definition of the Move command that begins
on page 124.

For a more detailed introduction to application commands, see the *AppleScript
Language Guide.*

### Syntax

Each command definition begins with a "Syntax" section that describes how
to use the command in a statement. Syntax descriptions use the typographic
conventions summarized on page xi: plain computer font indicates a language
element you must type exactly as shown; italic text indicates a placeholder you
must replace with an appropriate value; brackets indicate that the enclosed
language element or elements are optional; and vertical bars separate elements
from which you must choose a single element.

For example, here's the syntax statement for the Move command from page 124:

```
move referenceToObject to referenceToContainer                    ¬
    [ replacing ( conflicts | existing items ) ]
```

To use the Move command, you must replace the placeholders *referenceToObject* and *referenceToContainer* with a reference to the object you want to move and a reference to the container to which you want to move it, respectively. The optional parameter in brackets indicates that you want the moved object to replace any object in *referenceToContainer* that has the same name.

## Parameters

Parameters are values that are included with a command. The "Parameters" section of a command definition lists that command's parameters and the information you need to use them correctly.

Many commands include a direct parameter that specifies the object of the action. The direct parameter immediately follows the command. In the Move command syntax described in the previous section, the *referenceToObject* parameter is the direct parameter.

If a command includes parameters other than the direct parameter, they are identified by labels. Parameters that are identified by labels are called *labeled parameters.* In the Move command syntax, the to *referenceToContainer* parameter is a labeled parameter with the label to, and the replacing (conflicts | existing items) parameter is a labeled parameter with the label replacing.

Each parameter value you supply must belong to a particular class, which is listed in its description in the command definition. For example, the *referenceToObject* and *referenceToContainer* parameters for the Move command must both be references.

Parameters can be required or optional. *Required parameters* must be included with the command; *optional parameters* need not be. Optional parameters are enclosed in brackets in syntax descriptions. For optional parameters, the description in the "Parameters" section specifies a default value that is used if you don't include the parameter.

## Result

Many (but not all) commands return results. The *result* of a command is the
value generated when the command is executed. The "Result" section of a
command definition tells whether a result is returned, and if so, lists its class.
For example, the result of a Move command is a reference to the object that
was moved.

## Examples

Each command definition includes one or more short examples demonstrating
how to use the command. The example for the Move command definition
on page 124 is a script that moves a file into a folder.

## Command Definitions

This section defines the commands that are understood by the Finder and
its objects.

The Finder supports most of the standard application commands described
in the *AppleScript Language Guide.* As is true for most scriptable applications,
the Finder's definitions for some of these commands differ slightly from the
standard definitions. Table 3-1 summarizes the differences between the defined
behavior of the standard commands and the Finder behavior.

**Table 3-1**    Variations from standard behavior in Finder versions of standard
application commands

| Command | Finder version |
| --- | --- |
| Close | Closes one or more containers. Behaves like the standard version, except that the Finder needn't save data when it closes containers. |
| Copy | Copies an object or objects. Behaves like the standard version, except that the Finder version must include parameters. |

*continued*

**Table 3-1** Variations from standard behavior in Finder versions of standard application commands (continued)

| Command | Finder version |
|---------|----------------|
| Count | Counts elements of a particular class in a container. Behaves like the standard version, except that the Finder always returns a single integer, never a list of integers. |
| Data Size | Returns the size, in bytes, of the value returned by a Get command on the same object or objects. Identical to the standard version. |
| Delete | Deletes one or more objects. Identical to the standard version. |
| Duplicate | Copies an object or objects. Behaves like the standard version, and also allows you to specify whether or not to replace items in the destination container. |
| Exists | Determines whether an object exists. Identical to the standard version. |
| Get | Returns the value of an object. Identical to the standard version, except that it returns references for most Finder objects rather than values. |
| Make | Creates a new object. Behaves like the standard version, except that the Finder can't set the value of the data for most new objects, and you can't use new when you make a new file. |
| Move | Moves an object or objects. Behaves like the standard version, and also allows you to specify whether or not to replace items in the destination container. |
| Open | Opens an object or objects. Behaves like the standard version, except that it can open objects such as folders and suitcases as well as files. |
| Print | Prints one or more objects. Identical to the standard version. |
| Quit | Terminates the Finder process. Behaves like the standard version, except that the Finder need not save data before terminating. |
| Set | Sets value of one or more objects. Identical to standard version. |

Table 3-2 lists the commands defined by the Finder Suite—that is, the commands unique to the Finder.

**Table 3-2**     Commands defined by the Finder Suite

| Command | Summary |
| --- | --- |
| Clean Up | Aligns the icons in a window or on the desktop along the grid pattern selected in the Views control panel. |
| Computer | Returns information about the local computer's operating environment. |
| Eject | Ejects a disk. |
| Empty | Empties the trash. |
| Erase | Erases a disk. |
| Put Away | Returns specified objects on the desktop or in the Trash to the folders or disks to which they belong. |
| Restart | Restarts the computer. |
| Reveal | Makes an object visible by opening its container and selecting it. |
| Select | Selects specified objects. |
| Shut Down | Shuts down the computer. |
| Sleep | Puts a PowerBook to sleep. |
| Sort | Sorts a list of references to Finder objects. |
| Update | Updates an object and its elements, if any, to match their representation on disk. |

The rest of this chapter provides detailed definitions of the commands summarized in Table 3-1 and Table 3-2.

## Clean Up

A Clean Up command is a request to align objects in a window or on the desktop along the grid pattern selected in the Views control panel. The View property for the window must be 0 ("by Small Icon") or 1 ("by Icon"). Depending on the parameters used with it, the Clean Up command can have the same effect as choosing Clean Up Window, Clean Up Desktop, or Clean Up All from the Special menu.

**SYNTAX**

clean up [ all | ( *referenceToObject* [ by *propertyLabel* ] ) ]

**PARAMETERS**

*referenceToObject*

A reference to the desktop or to one or more container windows, or a reference to one or more elements of the desktop or a container window. If the reference is to the desktop or a container window, the command acts on the contents of the object. If the reference is to elements of the desktop or of a container window, the Finder aligns the specified objects to the nearest grid points and ignores the *propertyLabel* parameter.
*Class:* Reference or list of references
*Default value:* A reference to the desktop

*propertyLabel*    One of these property labels, specifying the order in which to clean up the objects: name, size, modification date, version, or comment. If you include the *referenceToObject* parameter and omit the *propertyLabel* parameter, the Finder aligns the objects to the nearest grid points without sorting them into a new arrangement. The command clean up all is equivalent to the command clean up desktop by name.
*Class:* Class identifier
*Default value:* None

**RESULT**

Reference to the specified object or list of references.

This script cleans up all of the Finder's open windows (not including the desktop) for which "by Icon" or "by Small Icon" is selected in the Views menu. The `by name` parameter specifies that the cleaned up icons are to be arranged in the windows by name.

```
tell application "Finder"
   clean up windows by name
end tell
```

This script aligns the icons in the window for the disk My World to the nearest grid points without sorting them into a new arrangement:

```
tell application "Finder"
   clean up window "My World"
end tell
```

The command `clean up all` is equivalent to the command `clean up desktop by name` and to the Clean Up All menu command (available when you select an item on the desktop, hold down the Option key, and pull down the Special menu). All these commands rearrange items on the desktop by name and place the Trash in its original position relative to the lower-right corner of the screen. However, if you use the Clean Up command without any parameters, the Finder aligns objects on the desktop to the nearest grid points without rearranging them.

If the *referenceToObject* parameter of the Clean Up command is a reference to one or more items in an open window or on the desktop, the command aligns their icons to the grid but ignores the *propertyLabel* parameter, if any. For example, this script aligns only the icon for the folder My Folder (not the contents of My Folder) to the grid:

```
tell application "Finder"
   clean up folder "My Folder" of window "My World"
end tell
```

However, if the window for a folder is open, you can use a script like this to clean up the contents of the window:

```
tell application "Finder"
    clean up contents of folder "My Folder" of window of startup disk
end tell
```

## Close

A Close command is a request to close one or more windows. The Finder version of the Close command behaves like the standard version described in the *AppleScript Language Guide,* except that the Finder ignores the `saving` and `saving in` parameters recognized by AppleScript and most other applications.

**SYNTAX**

close *referenceToObject*

**PARAMETERS**

*referenceToObject*

A reference to the object to close or a list of references. The reference can be to either a window or a container to which a window belongs.
*Class:* Reference

**RESULT**

Reference to the closed window or a list of references.

**EXAMPLES**

This statement closes all Finder windows:

```
tell application "Finder" to close windows
```

This statement closes all Finder windows whose name includes the word "messages":

```
tell application "Finder"
   close windows whose name contains "messages"
end
```

You can also specify a container to which a window belongs as the parameter of the Close command:

```
tell application "Finder"
   close startup disk
end tell
```

## Computer

The Computer command is a request for information about the local computer's operating environment.

This command is intended for use by programmers only. To use it, you should be familiar with the `Gestalt` function as described in *Inside Macintosh: Operating System Utilities.*

**SYNTAX**

```
computer gestaltSelector [ has bitsToTest ]
```

**PARAMETERS**

*gestaltSelector*   The Gestalt selector for the feature you want to test. The Finder converts these constants to the Gestalt codes for the equivalent information about the operating environment:

```
CPU
FPU
MMU
hardware
operating system
sound system
memory available (including virtual memory)
memory installed
```

For other selectors, use the appropriate code as described in *Inside Macintosh: Operating System Utilities.*
*Class:* Four-character code recognized by the `Gestalt` function. If you use a code instead of one of the constants listed above, it must have four characters and it must be given either in the form of a string or within double angle brackets after the word `class`:

```
"xxxx"
«class xxxx»
```

*bitsToTest*   The value of the bits to test.
*Class:* Integer
*Default value:* None

**RESULT**

Integer if the *bitsToTest* parameter is not included; Boolean if it is.

**EXAMPLES**

This script returns `true` if the sound input device bit returned by Gestalt is set, and `false` if it isn't:

```
tell application "Finder"
   set soundInputDevice to 32
   computer sound system has soundInputDevice
end tell
```

This script returns an integer that indicates the total amount of memory available (including virtual memory), in bytes:

```
tell application "Finder"
    computer memory available
end tell
```

## Copy

The Copy command is a request to copy an object or objects to a new location. The Finder version is similar to the standard application command described in the *AppleScript Language Guide,* except that the Finder's Copy command must include parameters.

As shown in the syntax definition, put and into are synonyms for copy and to. When you compile a script, put and into are automatically changed to copy and to.

**SYNTAX**

( copy | put ) *expression* ( to | into ) *referenceToObject*

**PARAMETERS**

*expression*    The expression whose value is to be copied. If *expression* is a reference or a list of references, the Finder copies the objects specified by the references.
*Class:* Any class

*referenceToObject*
A reference to the object to which to copy *expression*.
*Class:* Reference

**RESULT**

A reference to the copied object or a list of references.

**EXAMPLE**

This script copies two files and places them in a folder on another disk:

```
tell application "Finder"
   copy {file "Release Notes" of disk "My World", ¬
      file "Report" of disk "My World"} to folder "Backups" of ¬
      startup disk
end tell
```

## Count

The Count command counts the number of elements of a particular class in a container. The Finder version of the Count command is similar to the standard application command described in the *AppleScript Language Guide*, except that the Finder's Count command always returns a single integer.

**SYNTAX**

count [ each | every ] *className* [ ( in | of ) *referenceToContainer* ]

number of *className* [ ( in | of ) *referenceToContainer* ]

**PARAMETERS**

*className*    The class name of the elements to be counted. If you use the term each or every, you can use only the singular form of the class name. The kinds of elements that Finder objects can contain are listed in the object class definitions for each class in Chapter 2, "Finder Objects." You must specify a class name or the command won't work.
*Class:* Class identifier

*referenceToContainer*

A reference to the container whose elements are to be counted. If you do not specify this parameter, the Finder counts the elements in the default target of the Tell statement (for example, the Finder application).
*Class:* Reference
*Default value:* Reference to default target of Tell statement

**RESULT**

Integer.

**EXAMPLES**

In the following example, `every item of window "My World"` returns a list of items. The Finder counts the items in the list.

```
tell application "Finder"
   count every item of window "My World"
end tell

--result: 12
```

The following statement is equivalent to the previous example:

```
tell application "Finder"
   count items of window "My World"
end tell
```

## Data Size

A Data Size command is a request for the size, in bytes, of the data of one or more objects. The value returned is the size of the data (a value) that would result from a Get command on the same object or objects. This command has

little meaning for the Finder, because the result of a Get command for most Finder objects is a reference. In the case of icons, Data Size returns the data size of the icon family.

**SYNTAX**

data size of *referenceToObject* [ as *className* ]

**PARAMETERS**

*referenceToObject*
    A reference to the object or objects whose data size is to be returned.
    *Class:* Reference

*className*     The class of data for which to determine the size.
    *Class:* Reference
    *Default value:* The default value class for the object

**RESULT**

An integer that indicates the size, in bytes, of the data that would be returned by a Get command on the specified object or objects. For most Finder objects, this command returns the size of the reference, not the size of the object referred to.

If the *referenceToObject* parameter specifies a single object only (such as icon of file "My File" of startup disk or window 1), the result is a single integer that specifies the data size in bytes. If the specified object doesn't exist, the Finder returns an error.

If the *referenceToObject* parameter refers to more than one object (such as items of startup disk), the result is a list of integers. The first item in the list is the data size of the first item, the second item is the data size of the second item, and so on. If the specified objects don't exist, the result is an empty list.

**EXAMPLE**

This script returns a list of integers that specify the size, in bytes, of the icons for the files My File and My Other File, both located on the startup disk:

```
tell application "Finder"
   data size of {icon of file "My File" of startup disk, ¬
      icon of file "My Other File" of startup disk}
end tell

--result: {1016, 2320}
```

## Delete

A Delete command is a request to delete one or more objects. The Finder version of the Delete command is identical to the standard version described in the *AppleScript Language Guide*.

**SYNTAX**

```
delete referenceToObject
```

**PARAMETER**

*referenceToObject*
> A reference to the object or objects to be deleted.
> *Class:* Reference

**RESULT**

None

**EXAMPLE**

```
tell application "Finder"
   delete file "OldFile" of startup disk
end tell
```

## Duplicate

A Duplicate command is a request to make a copy of an object or objects and insert the new copy either at a specified location or in the same container as the object that was copied. The Finder version of the Duplicate command is similar to the standard version described in the *AppleScript Language Guide*, except that the Finder's Duplicate command allows you to specify whether or not to replace items of the same name in the destination container.

**SYNTAX**

```
duplicate referenceToObject [ to referenceToLocation ] ¬
   [ replacing ( conflicts | existing items ) ]
```

**PARAMETERS**

*referenceToObject*
> A reference to the object or objects to be duplicated.
> *Class:* Reference

*referenceToLocation*
> A reference to the location for the duplicated object or objects.
> *Class:* Reference
> *Default value:* If you don't specify a new location, the object is inserted in the same container as the original object and the word "copy" is appended to the new object's name.

**RESULT**

A reference to the new object or a list of references.

**EXAMPLE**

This script duplicates a file to a disk, replacing any items at the top level of the disk that have the same name:

```
tell application "Finder"
   duplicate file "My File" of startup disk ¬
      to disk "Backup" replacing conflicts
end tell
```

You can use `replacing existing items`, `with replacing`, or `replacing true` instead of `replacing conflicts` without changing the meaning of the script.

**NOTES**

The statement `duplicate items in` *referenceToContainer* (where *referenceToContainer* is a reference to any container) won't compile. Instead, use the statement `duplicate every item of` *referenceToContainer.*

## Eject

An Eject command is a request to eject a disk.

**SYNTAX**

`eject [` *referenceToDisk* `]`

**PARAMETERS**

*referenceToDisk*
> A reference to the disk to be ejected or a list of references.
> *Class:* Reference or list of references
> *Default value:* List of references to all mounted disks whose Ejectable property is `true`

**RESULT**

A reference to the ejected disk or a list of references.

**EXAMPLE**

This script ejects the disk named Untitled:

```
tell application "Finder" to eject disk "Untitled"
```

**NOTES**

If you use the Eject command to eject a disk, the disk's dimmed icon remains on the desktop after the disk ejects. To eject a disk without retaining its dimmed icon, use the Put Away command, which is defined on page 127.

## Empty

An Empty command is a request to empty the trash. It has the same effect as choosing Empty Trash from the Special menu, except that it never warns the user before emptying—even if the Warn Before Emptying property of the Trash is set to true.

**SYNTAX**

```
empty [ referenceToTrash ]
```

**PARAMETERS**

*referenceToTrash*

A reference to a trash container.
*Class:* Reference
*Default value:* Reference to the local computer's Trash

**RESULT**

A reference to the trash container that was emptied.

**EXAMPLE**

```
tell application "Finder" to empty the trash
```

## Erase

The Erase command is a request to erase one or more disks. It is equivalent to selecting one or more disk icons on the desktop and choosing Erase Disk from the Special menu. Before the specified disk is erased, a dialog box appears asking the user to confirm that formatting should proceed.

**SYNTAX**

```
erase referenceToDisk
```

**PARAMETERS**

*referenceToDisk*
                        A reference to the disk to be erased or a list of references.
                        *Class:* Reference or list of references.

**RESULT**

A reference to the erased disk or a list of references.

**EXAMPLE**

```
tell application "Finder" to erase disk "Untitled"
```

## Exists

An Exists command is a request to determine whether the object specified by a reference exists. The Finder version of the Exists command is identical to the standard version described in the *AppleScript Language Guide*.

**SYNTAX**

```
exists referenceToObject
```

*referenceToObject* `exists`

**PARAMETER**

*referenceToObject*
A reference to the object to find or a list of references.
*Class:* Reference or list of references

**RESULT**

A Boolean value. If `true`, all of the objects referred to by *referenceToObject* exist. If `false`, one or more of the objects referred to by *referenceToObject* do not exist.

**EXAMPLE**

This script checks whether a disk named Storage is mounted and, if it is, copies all files from the folder Correspondence whose modification dates are more than 30 days old to disk Storage and deletes the originals.

```
tell application "Finder"
   if exists disk "Storage" then
      set aMonthAgo to (current date) – (30 * days)
      copy (every item in folder "Correspondence" of startup disk ¬
         whose modification date < aMonthAgo) ¬
         to folder "Old Letters" of disk "Storage"
      delete (every item in folder "Correspondence" of startup disk ¬
         whose modification date < aMonthAgo)
```

```
   else
      display dialog "The Storage cartridge isn't mounted."
   end if
end tell
```

Note that you can't use the Move command instead of the Copy and Delete commands in this script, because the Move command changes to Copy when moving items between volumes.

## Get

The standard application command Get is a request to return the value of an object or objects. The Finder version of the Get command is similar to the standard application command, except that in most cases the Finder simply returns a reference to objects or objects requested. For properties that consist of a value (such as the Name property) rather than a reference (such as the Desktop property of the Finder application), the Finder returns the value. In all cases, the Finder assigns the value returned to the predefined variable `result`.

**SYNTAX**

[ `get` ] *referenceToObject* [ `as` *className* ]

**PARAMETERS**

*referenceToObject*
> A reference to an object whose value is to be returned in the `result` variable or a list of references.
> *Class:* Reference or list of references

*className*  A class identifier that specifies the desired value class for the returned data.
> *Class:* Class identifier
> *Default value:* The default value class for the object or objects

**RESULT**

A reference or list of references to the requested objects or, in the case of properties that consist of a value, a value of either the object's default value class or the class specified by the *className* parameter, or a list of values of that class. If the Finder can't return data in the value class specified by the *className* parameter, it generates an error.

If the *referenceToObject* parameter specifies a single object only (such as `name of window 1`), the result is a single value or reference. If the *referenceToObject* parameter refers to more than one object (such as `items in startup disk`), the result is a list of values or references. If the specified object doesn't exist, for example, if the reference is `folder "My Folder" in startup disk` and there is no folder of that name in the startup disk, the Finder generates an error.

**EXAMPLES**

This script returns the sizes, in bytes, of the folders at the top level of the startup disk:

```
tell application "Finder"
    get size of folders in startup disk
end tell
```

```
--result: {27636, 1438359, 1774813}
```

This script returns a list of references to the folders at the top level of the startup disk:

```
tell application "Finder"
    get folders in startup disk
end tell
```

```
--result: folder "Projects" of startup disk of
application "Finder", folder "Financial" of startup disk of
application "Finder", folder "Applications" of startup disk
of application "Finder"
```

**NOTES**

The word get in the Get command is optional. For example, these statements are equivalent:

```
name of startup disk
get name of startup disk
```

## Make

The standard application command Make is a request to create a new object that can include values for properties of the object. The Finder version of the Make command is similar to the version described in the *AppleScript Language Guide*, except that the Finder's Make command can't set the value of the new object's data unless the new object is an alias file.

**SYNTAX**

```
make [ new ] className [ at referenceToContainer ]              ¬
   [ with properties                                           ¬
      { propertyLabel:propertyValue [ , propertyLabel:propertyValue ]...}]¬
   [ ( to | with data ) referenceToObject ]
```

**PARAMETERS**

*className*     The class of the object to be created. (Note that if you are
                creating a new file, you cannot use the optional term new
                before *className*.)
                *Class:* Class identifier

*referenceToContainer*
                The container in which to create the new object.
                *Class:* Reference
                *Default value:* The Finder's current insertion location

*propertyLabel*  The name of a property whose value is to be set.
                *Class:* String

*propertyValue*   The value to assign to the property.
*Class:* The value class of the property as specified in the object class definition in Chapter 2, "Finder Objects," or a value that can be coerced to the class of the property
*Default value:* If you create a file or folder without specifying its name, the Finder creates a an item named Untitled File or Untitled Folder. Other property values, such as the creation date, icon, and so on, are provided by the Finder on the basis of information in the operating environment.

*referenceToObject*
A reference to the object or objects to which you want to make an alias. This parameter is meaningful only if you are making a new alias.
*Class:* Reference or list of references to the object or objects for which aliases are to be made
*Default value:* None

**RESULT**

A reference to the newly created object.

**EXAMPLES**

The script that follows creates a new folder named My Folder at the top level of the startup disk.

```
tell application "Finder"
   make new folder at startup disk with properties ¬
      {name: "My Folder"}
end tell
```

If you save this script as an application, it makes an alias file on the desktop for any objects whose icon you drop onto the script's icon.

```
on open x
   repeat with i in x
      tell application "Finder"
         make alias file to i
      end tell
   end repeat
end open
```

If you save this script as an application and drag a folder onto the application's icon, the script searches the entire startup disk for any items whose names include the name of the folder and creates alias files in that folder for all matching items:

```
on open x
    tell application "Finder"
        repeat with i in x
            set n to name of i
            open i
            make alias file to every item of entire contents ¬
                of startup disk whose name contains n at i
        end repeat
    end tell
end open
```

This script may take a minute or more to run if the startup disk contains a large number of items.

**NOTES**

When you use the Make command to create a new file, you can't use the term new before the term file:

```
tell application "Finder"
    make file at startup disk with properties ¬
        {name: "My File"}
end tell
```

For all other object classes, the term new is optional.

## Move

A Move command is a request to move an object or objects. The Finder version of the Move command is similar to the version described in the *AppleScript Language Guide*, except that the Finder's Move command allows you to specify whether or not to replace items in the destination container.

**SYNTAX**

```
move referenceToObject to referenceToContainer                  ¬
    [ replacing ( conflicts | existing items ) ]
```

**PARAMETERS**

*referenceToObject*
> A reference to the object to move or a list of references.
> *Class:* Reference or list of references

*referenceToContainer*
> A reference to the location to which to move the object
> (see "Notes").
> *Class:* Reference

**RESULT**

A reference to the object that was moved or a list of references.

**EXAMPLE**

This script moves a file into a folder, replacing any files of the same name that exist in that folder:

```
tell application "Finder"
    move file "My File" of startup disk ¬
        to folder "My Folder" of startup disk ¬
        replacing conflicts
end tell
```

You can use `replacing existing items`, `with replacing`, or `replacing true` instead of `replacing conflicts` without changing the meaning of this script.

**NOTES**

If an object specified by the *referenceToObject* parameter is located on a different disk from the container specified by the *referenceToContainer* parameter, the Move command copies the object instead of moving it, but doesn't provide any warning that it has failed to remove the original object. To move any object from one disk to another, copy the object to the new disk (using either the Move or the Copy command), then delete the original object using the Delete command. See page 118 for an example of a script that uses this technique.

## Open

An Open command is a request to open an object. The Finder version of the Open command is similar to the version described in the *AppleScript Language Guide*, except that the Finder's Open command can open objects such as application files, folders, suitcases, and disks as well as files, and in the case of document files allows you to specify the application that opens them.

**SYNTAX**

`open` *referenceToObject* [ `using` *referenceToApplication* ]

**PARAMETER**

*referenceToObject*
> A reference to the object to be opened or a list of references.
> *Class:* Reference or list of references

*referenceToApplication*
> A reference to the application file with which you want to open the document files specified by *referenceToObject*.
> *Class:* Reference or list of references
> *Default value:* None

**RESULT**

Reference to the opened object or a list of references.

**EXAMPLES**

This script opens a file and a folder:

```
tell application "Finder"
   open {file "My File" of startup disk, ¬
      folder "My Folder" of startup disk}
end tell
```

This script opens a text file using the Scriptable Text Editor:

```
tell application "Finder"
   open file "Text File" of startup disk using ¬
      application file "Scriptable Text Editor" ¬
      of startup disk
end tell
```

**NOTES**

If one of the objects specified by *referenceToObject* is already open, it remains open and its window becomes the frontmost window.

The statement `open items in` *referenceToContainer* (where *referenceToContainer* is a reference to any container) won't compile. Instead, use the statement `open every item of` *referenceToContainer.*

## Print

A Print command is a request to print one or more objects. The Finder version of the Print command is identical to the standard version described in the *AppleScript Language Guide.*

**SYNTAX**

`print` *referenceToObject*

**PARAMETER**

*referenceToObject*

A reference to the object or objects to print—typically files or containers—or a list of references.
*Class:* Reference or list of references

**RESULT**

Reference to the printed object or a list of references.

**EXAMPLE**

`tell application "Finder" to print file "Addresses" of startup disk`

**NOTES**

The statement `print items in` *referenceToContainer* (where *referenceToContainer* is a reference to any container) won't compile. Instead, use the statement `print every item of` *referenceToContainer.*

## Put Away

The Put Away command is a request to return specified objects on the desktop or in the Trash to the folders or disks from which they were last moved. It may also be used to unmount volumes. The Put Away command has the same effect as selecting one or more icons on the desktop or in the Trash and choosing Put Away from the File menu.

**SYNTAX**

`put away` *referenceToObject*

**PARAMETER**

*referenceToObject*
A reference to the object to put away or a list of references.
*Class:* Reference or list of references

**RESULT**

Reference to the object put away or a list of references.

**EXAMPLES**

This script puts away the specified objects on the desktop:

```
tell application "Finder"
   put away {file "My File", folder "Projects"}
end tell
```

You don't have to specify the desktop explicitly, but to put away items in the
Trash you must specify that they are in the Trash:

```
tell application "Finder"
   put away {file "My File" of trash, ¬
      folder "Projects" of trash}
end tell
```

**NOTES**

The statement `put away items in` *referenceToContainer* (where
*referenceToContainer* is a reference to any container) won't compile. Instead, use
the statement `put away every item of` *referenceToContainer.*

## Quit

A Quit command is a request to quit the Finder application. The Finder's Quit command is similar to the one described in the *AppleScript Language Guide,* except that the Finder ignores the parameters recognized by most other applications. You can't tell other processes to quit from within a Tell statement addressed to the Finder.

When you quit the Finder, any icons on the desktop and any Finder windows that are currently visible disappear.

▲ **WARNING**
If you are using version 1.1 of AppleScript, you can launch the Finder again only by quitting all running processes or by restarting the computer. ▲

**SYNTAX**

```
quit
```

**PARAMETERS**

None

**RESULT**

None

**EXAMPLE**

This statement quits the Finder:

```
tell application "Finder" to quit
```

If you are using version 1.1 of AppleScript, you can relaunch the Finder only by quitting all running applications or by restarting the computer. Other versions

allow you to relaunch the Finder with the Launch command (or with any Tell statement addressed to the Finder):

```
tell application "Finder" to launch
```

## Restart

The Restart command is a request to restart the computer. It is equivalent to choosing Restart from the Special menu.

**SYNTAX**

```
restart
```

**PARAMETERS**

None

**RESULT**

None

**EXAMPLE**

```
tell application "Finder" to restart
```

## Reveal

The Reveal command is a request to make an object visible by opening its container and selecting it. If you specify objects in several containers, only the last specified object remains selected.

**SYNTAX**

`reveal` *referenceToObject*

**PARAMETER**

*referenceToObject*
> A reference to the object or objects to be revealed.
> *Class:* Reference or list of references

**RESULT**

Reference to the revealed object or a list of references.

**EXAMPLE**

If you save the script that follows as an application, it reveals the original item for any alias file whose icon you drag over the script application's icon.

```
on open x
   repeat with i in x
      tell application "Finder"
         if i exists then
            reveal i
         end if
      end tell
   end repeat
end open
```

Each original item revealed by this script is selected only momentarily. After the script runs and quits, the Finder selects the script's icon again. This is standard Finder behavior whenever an application (in this case, a script application) quits.

## Select

The Select command is a request to select one or more objects. If you specify objects in several containers, only the objects in the last specified container will remain selected. The container must be open for the specified object to be selected.

**SYNTAX**

select *referenceToObject*

**PARAMETER**

*referenceToObject*
> A reference to the object or objects to be selected.
> *Class:* Reference or list of references

**RESULT**

Reference to the selected object or a list of references.

**EXAMPLE**

This script selects two files in a single folder:

```
tell application "Finder"
   select {file "To Tom" in folder "Letters" of startup disk, ¬
      file "To Lorraine" in folder "Letters" of startup disk}
end tell
```

The folder must be open for the script to work correctly.

## Set

The Set command is a request to set the values of one or more objects. The Finder version of the Set command is identical to the standard version described in the *AppleScript Language Guide.*

**SYNTAX**

```
set referenceToObject to expression
```

**PARAMETERS**

*referenceToObject*

A reference to the object whose value is to be set or a list of references.
*Class:* Reference or list of references

*expression*    The expression whose value or values are to be assigned.
If *expression* is a reference or a list of references, they must be references for which the Finder can obtain a value.
*Class:* Any appropriate class for the objects whose values are to be set.

**RESULT**

The value assigned or a list of values.

**EXAMPLE**

This script sets the name and position of the selection:

```
tell application "Finder"
   set name of selection to "My Folder"
   set position of selection to {140, 160}
end tell
```

## Shut Down

The Shut Down command is a request to shut down the computer. It is equivalent to choosing Shut Down from the Special menu.

**SYNTAX**

```
shut down
```

**PARAMETERS**

None

**RESULT**

None

**EXAMPLE**

```
tell application "Finder" to shut down
```

## Sleep

The Sleep command is a request to put a PowerBook to sleep. It is equivalent to choosing Sleep from the Special menu. Before putting the computer to sleep, the Finder displays a dialog box asking the user to confirm the decision.

**SYNTAX**

```
sleep
```

**PARAMETERS**

None

**RESULT**

None

**EXAMPLE**

```
tell application "Finder" to sleep
```

## Sort

The Sort command is a request for a sorted list of references.

**SYNTAX**

```
sort referenceList by propertyLabel
```

**PARAMETERS**

*referenceList*    A list of references to the objects that are to be sorted. This may take the form of a single reference to several objects in the same container or a list of references to objects in several containers. *Class:* Reference or list of references

*propertyLabel*    The name of the property by which the objects are to be sorted. *Class:* Class identifier

**RESULT**

A list of references sorted according to the property specified in the *propertyLabel* parameter.

**EXAMPLES**

This script returns a list of references to the folders in the startup disk, sorted by creation date:

```
tell application "Finder"
    sort folders in startup disk by creation date
end tell

--result: {folder "Letters" of startup disk of
application "Finder", folder "Projects" of startup disk
of application "Finder", folder "Current Correspondence" of
startup disk of application "Finder"}
```

This script sorts files from several different folders by size.

```
tell application "Finder"
    sort {files in startup disk, files in disk "Data"} ¬
        by size
end tell

--result: {file "George" of startup disk of application
"Finder", file "Notes" of disk "Data" of application
"Finder", file "Expenses" of startup disk of application
"Finder"}
```

**NOTES**

The Sort command usually sorts in ascending order; for example, if you sort by name, the resulting references are sorted alphabetically by name. When you sort by creation date or modification date, the Sort command returns a list of references sorted in descending order from the newest item to the oldest.

To sort the actual items that are displayed in a window, rather than a list of references to those items, either set the View property of the window or, if the items are displayed by icon or by small icon, use the Clean Up command as described on page 104.

## Update

An Update command is a request to update an object and its elements, if any, to match their representation on disk. This involves making new items show up in the correct position and updating the desktop database. The Finder updates containers automatically when processing time is available; the Update command just causes it to happen immediately.

**SYNTAX**

update *referenceToObject*

**PARAMETERS**

*referenceToObject*
A reference to the container or containers to update.
*Class:* Reference or list of references

**RESULT**

Reference to the updated container or a list of references.

**EXAMPLE**

```
tell application "Finder"
   update {startup disk, disk "Data"}
end tell
```

# Appendixes

# Finder Commands at a Glance

This appendix summarizes the commands described in this guide and the placeholders used in syntax descriptions. For more detailed information about these commands, see Chapter 3, "Finder Commands."

The placeholder descriptions in the last section of this appendix define the placeholders used in the syntax summaries.

## Commands

Table A-1 summarizes the Finder commands descrfibed in this guide and their syntax.

*continued*

**Table A-1**    Finder command syntax

| Command | Syntax | | Result |
|---|---|---|---|
| clean up | clean up | | Reference or list of references |
| | clean up all | | |
| | clean up *referenceToObject* | | |
| | clean up *referenceToObject* by *propertyLabel* | | |
| close | close *referenceToObject* | | Reference or list of references |
| computer | computer *gestaltSelector* | | Integer |
| | computer *gestaltSelector* has *bitsToTest* | | Boolean |
| copy | ( copy \| put ) *expression* ( to \| into ) *referenceToObject* | ⌐ | Reference or list of references |
| count | count [ each \| every ] *className* | | Integer |
| | count [ each \| every ] *className* ( in \| of ) *referenceToContainer* | ⌐ | |
| | number of *className* | | |
| | number of *className* ( in \| of ) *referenceToContainer* | | Integer or list of integers |
| data size | data size of *referenceToObject* | | |
| | data size of *referenceToObject* as *className* | | |
| delete | delete *referenceToObject* | | None |
| duplicate | duplicate *referenceToObject* | | Reference or list of references |
| | duplicate *referenceToObject* to *referenceToLocation* | | |
| | duplicate *referenceToObject* to *referenceToLocation* replacing ( conflicts \| existing items ) | ⌐ | |

**Table A-1** Finder command syntax (continued)

| Command | Syntax | Result |
|---|---|---|
| eject | eject | Reference or list of references |
| | eject *referenceToDisk* | |
| empty | empty | Reference |
| | empty *referenceToTrash* | |
| erase | erase *referenceToDisk* | Reference |
| exists | exists *referenceToObject* | Boolean |
| | *referenceToObject* exists | |
| get | [ get ] *referenceToObject* | Reference, value, or list of references or values |
| | [ get ] *referenceToObject* as *className* | |
| make | make [ new ] *className* | Reference to the new object |
| | make [ new ] *className* at *referenceToContainer* | |
| | make [ new ] *className* at *referenceToContainer* with properties { *propertyLabel:propertyValue* [, *propertyLabel:propertyValue* ]...} | |
| | make [ new ] *className* at *referenceToContainer* ( to &#124; with data ) *referenceToObject* | |
| | make [ new ] *className* at *referenceToContainer* with properties { *propertyLabel:propertyValue* [, *propertyLabel:propertyValue* ]...} ( to &#124; with data ) *referenceToObject* | |
| move | move *referenceToObject* to *referenceToContainer* | Reference or list of references |
| | move *referenceToObject* to *referenceToContainer* replacing ( conflicts &#124; existing items ) | |

*continued*

Commands **143**

**Table A-1** Finder command syntax (continued)

| Command | Syntax | Result |
|---------|--------|--------|
| open | open *referenceToObject* | Reference or list of references |
| | open *referenceToObject* using *referenceToApplication* | |
| print | print *referenceToObject* | Reference or list of references |
| put away | put away *referenceToObject* | Reference or list of references |
| quit | quit | None |
| restart | restart | None |
| reveal | reveal *referenceToObject* | Reference or list of references |
| select | select *referenceToObject* | Reference or list of references |
| set | set *referenceToObject* to *expression* | Value or list of values |
| shut down | shut down | None |
| sleep | sleep | None |
| sort | sort *referenceList* by *propertyLabel* | List of references |
| update | update *referenceToObject* | Reference or list of references |

# Placeholders

Table A-2 explains the placeholders used in the syntax descriptions in this appendix.

**Table A-2**      Placeholders used in syntax descriptions

| Placeholder | Explanation |
|---|---|
| *bitsToTest* | The value of the bits to test with the Computer command. |
| *className* | A class identifier or an expression that evaluates to a class identifier. |
| *expression* | A series of terms whose value is a Boolean, class identifier, constant, data, date, integer, list, real, record, reference, or string. |
| *gestaltSelector* | The Gestalt selector for the information about the operating environment to be tested with the Computer command. |
| *propertyLabel* | The name of a property. |
| *propertyValue* | An expression that evaluates to a value of the appropriate class for the property being defined. |
| *referenceList* | A list of references. |
| *referenceToApplication* | A reference to an application. |
| *referenceToContainer* | A reference to a container. |
| *referenceToDisk* | A reference to a disk or a list of references. |
| *referenceToLocation* | A reference to the new location for an object copied with the Duplicate command. |
| *referenceToObject* | A reference to an object or a list of references. |
| *referenceToTrash* | A reference to the Trash. |

# Finder Errors

This appendix lists error numbers and error messages for errors returned by the Finder. For information about handling errors, see the *AppleScript Language Guide.*

| Error number | Error message |
| --- | --- |
| –15260 | Finder is busy |
| –15261 | Window must be open to use this command |
| –15262 | Cannot put item away because it is not in the trash or on the desktop, or it was created on the desktop |
| –15263 | Window must be open in icon or small icon view to use this command |
| –15264 | Window must be open in list view to use this command |
| –15265 | Destination is not a container |
| –15266 | Source object cannot be moved to specified destination |
| –15267 | An item with the same name already exists in the destination |
| –15268 | Can't move folder into one of its subfolders |
| –15269 | Can't replace a folder with one of its subfolders |
| –15270 | Command cannot be used on items in the trash |
| –15271 | Item is already in specified destination folder |
| –15272 | No user or group of that name exists |
| –15273 | Share points cannot inherit the privileges of their parent folder |
| –15278 | Invalid size for object |
| –15279 | Value out of range |

# Index

## T

## U

## V

## W, X, Y

## Z