Scripting the Finder From Your Application

The Finder has long been a black box to users and developers — extending the Finder or even examining its state has been nearly impossible. With System 7.5, Apple has shipped a Finder that supports the Object Support Library; this Scriptable Finder opens a new world to developers by allowing applications to interact with the Finder through Apple events.

The System 7 Finder has always accepted a number of simple events that provide services such as duplicating files, making aliases, and emptying the Trash. But the System 7.0 and 7.1 Finder events are very limited and have strict requirements for the order of parameters and for parameter data types. The Finder that shipped with System 7.5 greatly expands the set of available events: it uses the Object Support Library (OSL) to provide full compatibility with AppleScript, and it provides a new set of events to do things such as examine the Finder’s selection, change Finder preferences, and modify file sharing settings.

The term Scriptable Finder refers to any Finder that’s OSL compliant. In System 7.5, this support is implemented by the Finder Scripting Extension in the Extensions folder; however, future Finders will have scriptability built into their core code base. Developers can count on the presence of the Scriptable Finder in all future versions of system software.

The OSL and the Open Scripting Architecture are critical additions to the Macintosh Toolbox. They mark the end of black-box applications and system software and pave the way for configurable, component-based systems. A Scriptable Finder is only the first step in providing a more unified, open system, but it’s an important one.

This article shows you how to generate Finder events from your application. First we’ll look at event addressing and the Apple Event Manager, and then we’ll see how to specify Finder objects. Finally, the section “Making the Finder Do Tricks” provides a taste of the power and flexibility of the Scriptable Finder, showing some practical uses of this great new capability. On this issue’s CD, you’ll find the complete code for the article’s examples along with sample applications that show how to control the Finder with Apple events. The header file FinderRegistry.h on the CD declares all of the event message IDs, class IDs, and property IDs that the Finder defines.

GREG ANDERSON is currently the Technical Lead of the Finder Team at Apple and was the lead engineer on the Finder Scripting Extension. He’s known to engage in a number of activities of questionable sanity, including running straight up hills that are much too steep and much too long, working at Apple for four solid years, making chain mail by hand (with pliers, actually), and putting on armor and hitting people with sticks. Don’t worry, he never hits anyone in staff meetings or developer conferences.*
CREATING AND ADDRESSING FINDER EVENTS

Every feature of the Scriptable Finder is accessible via AppleScript. For example, the following script, if typed into the Script Editor and executed, would create a new folder on the desktop:

```
tell application "Finder"
    make folder at desktop
end tell
```

An application doesn’t need to compile and execute scripts, however, to use the features of the Scriptable Finder; every command that a script can instruct the Finder to do has a corresponding representation as an Apple event. An application that controls the Finder may bypass AppleScript entirely and send Apple events to the Finder directly. That’s the technique we’ll use in this article.

There are a number of ways to address an Apple event, but for sending an event to the Finder on the local machine, the simplest and most straightforward technique is to address the event by process serial number (PSN). To determine the Finder’s PSN, you walk the list of running processes and search for the Finder’s file type and creator, ‘FNDR’ and ‘MACS’.

Listing 1 shows one way to generate an address targeted at the Finder on the local machine. Notice that we’ve used TDescriptor, which is a C++ wrapper class that corresponds to the Apple Event Manager type AEDesc. (See “C++ Wrappers” for an explanation of wrappers used in this article.)

Should the Finder not be running, looking for processes with the signature ‘MACS’ will find other user interface shells, such as At Ease, and in some cases you might prefer your application to do that. However, no shells other than the Finder currently support the full Finder Event Suite, so the sample code provided here always requires the process type to be ‘FNDR’.

Earlier Finders were not only unaware of the OSL, but they also didn’t use the Apple Event Manager. That’s right, the System 7.0 and 7.1 Finders never call AEProcessAppleEvent — they interpret and process high-level events in their own special way, without ever informing the Apple Event Manager of what’s going on. This means that an application that sends any unrecognized high-level event to the System 7.0 or 7.1 Finder will never get a reply; the application will sit idle in AESend until the event times out (assuming that the send mode was kAEWaitReply).

C++ WRAPPERS

The sample code in this article makes extensive use of C++ wrappers. The file AppleEventUtilities.h, included on this issue’s CD, defines the wrapper classes TDescriptor and TAEEvent, which correspond to the Apple Event Manager types AEDesc and AppleEvent, respectively. The class TDescriptor contains methods for examining and extracting the contents of AEDesc, AEDescList, or AERecord structures. TAEEvent inherits from this class (since an Apple event really is an AERecord) and adds methods for getting and setting attributes and addressing and sending events.

The use of the C++ wrappers makes the code easier to read, but it would be a simple matter to translate the code back into straight C or Pascal functions that call the Apple Event Manager directly. If you do this, don’t forget that the C++ constructor of TDescriptor automatically initializes the fields of the AEDesc to a null descriptor (descriptor type = typeNull, data handle = nil). You must do this explicitly in your C or Pascal program, or you could cause problems for the OSL. For example, CreateObjSpecifier will crash if its second parameter is a pointer to an uninitialized AEDesc rather than a valid object specifier or a null descriptor.
To determine whether the Finder on the local machine supports the Finder Event Suite, an application can call Gestalt with the selector gestaltFinderAttr and check the gestaltOSLCompliantFinder bit of the result. Before System 7 Pro, gestaltFinderAttr didn’t exist, so Gestalt will return the error gestaltUndefSelectorErr on some machines.

Unfortunately, the only way to determine whether the Scriptable Finder is running on a remote machine is to send it an event and wait for it to time out. The best event to send is the Gestalt event from the Finder Event Suite (an event whose class is kAEFinderSuite and whose ID is kAEGestalt) with a direct parameter whose type is typeEnumeration and whose value is gestaltFinderAttr. If the Scriptable Finder is running, the result will have the gestaltOSLCompliantFinder bit set. Under System 7 Pro, the Finder will return an error (event not handled) if the Scriptable Finder isn’t running, but the System 7.0 and 7.1 Finders will never return a result.

The Gestalt event can be used to ask for the value of any Gestalt selector. It’s easier to call Gestalt directly on the local machine (and more reliable, since the Scriptable Finder might not be running), but some distributed computing applications may want to examine the result of Gestalt selectors on remote machines to determine which are suitable for use as remote hosts.
**SPECIFYING FINDER OBJECTS**

Most events operate on some Finder object, such as a file, a folder, or a window. These objects are always specified with an Apple event descriptor (AEDesc) placed in the direct object of the event. Some events require specification of more than one object; for example, the Copy event requires parameters for both the objects to be copied and the location to copy them to. In these cases, the direct object of the event is the object being operated on, and other parameters are defined for any other object it requires. The destination of the Copy event goes in the parameter keyAEDestination; other events may define other keywords for parameters they use.

Most scriptable applications require object specification parameters to be in a very specific format called an object specifier. The Finder is a little more flexible than that — it will accept a descriptor of type typeAlias (alias record) or typeFSS (FSSpec) in any parameter that requires an object specifier. All the same, understanding object specifiers is critical to sending events to the Finder, because many objects cannot be represented by an alias record or an FSSpec, and therefore must be referenced by object specifier.

Object specifiers are described in “Apple Event Objects and You” in develop Issue 10 and in “Better Apple Event Coding Through Objects” in develop Issue 12. See also Chapter 6, “Resolving and Creating Object Specifier Records,” in Inside Macintosh: Interapplication Communication.

An object specifier is a descriptor whose type is typeObjectSpecifier, but it's actually an Apple event record (AERecord), and can be accessed as such if coerced to type typeAERecord. To build an object specifier, it's most convenient to use the routine CreateObjSpecifier (MakeObjectSpecifier in AppleEventUtilities.cp), which takes four parameters: the desired class of the specified object, the key form, the key data, and the object container.

- The **desired class** indicates the kind of object. Some classes that the Finder recognizes are disks, windows, and folders. The desired class may also be set to typeWildCard to indicate any class of object.
- The **key form** specifies how the object is being addressed; the most common choices are by name and by index.
- The **key data** contains the specification for the object in a format compatible with the key form. For example, if the key form is formName, the key data will contain the name of the object being specified.
- The **object container** is either another object specifier or a null descriptor. Thus, object specifiers have a recursive definition that's always terminated with a null descriptor.

The null descriptor in the object specifier's container is a reference to a special container called the null container, which serves as the root container of every scriptable application. In most applications, the items accessible from the null container (called the elements of the container) include all the open documents and open windows. The Finder doesn't have any documents that it can open on its own; its null container contains all the open windows, plus all the objects on the desktop, including the mounted disks and the Trash.

You specify properties, such as the name of an object or the original item of an alias file, with an object specifier whose desired class is cProperty and whose key form is formProperty. The key data is always of type typeType, and it contains the four-
character code identifying the property. The container of the property’s object specifier is, as required, an object specifier or a null descriptor.

Usually, the property’s container specifies an object — for example, “name of disk 1” would be represented as a property specifier for pName with a container specifier to disk 1. It’s also possible to create property specifiers of property specifiers, such as “name of startup disk” (since the term “startup disk” is represented as a property specifier for pStartupDisk). Additionally, there are properties that refer to the Finder itself, or to the Macintosh that the Finder is running on — such as “file sharing,” the property that indicates whether file sharing is on or off. These are called properties of the null container, and the container of these property specifiers is always a null descriptor.

Any four-character code that’s recognized by FindFolder may be provided as a Finder property that refers to the folder returned by FindFolder. You’ll find this useful when moving, copying, or setting properties of special folders.

The Finder defines a special key form named formAlias. The key data of an object specifier whose key form is formAlias should be an alias record; the desired class should be typeWildCard; and the object container must always be a null descriptor. At first, the existence of formAlias may seem superfluous. The Finder will accept alias records in any object-specifier parameter, and there’s no functional difference between a descriptor of type typeAlias and an object specifier of form formAlias. However, formAlias object specifiers are very useful in one regard, and that’s to specify properties of files referenced by alias records. As mentioned earlier, the container parameter of an object specifier must be another object specifier. If an application already has an alias record, it may use it to build an object specifier of form formAlias for use in other object specifiers. Putting a descriptor of type typeAlias into the container parameter of an object specifier doesn’t work, and can even cause the OSL to crash.

MAKING THE FINDER DO TRICKS

A Macintosh running the Scriptable Finder is capable of a variety of tricks that other Finders only dream about. This section demonstrates a number of these features, including events that examine and change the state of the Finder, that notify the Finder of changes, and that manipulate files on disk. For a summary of events the Finder recognizes, see “Overview of Finder Events.” This issue’s CD includes the complete code for listings in this section.

A complete list of properties the Finder recognizes can be found in the AppleScript Finder Guide, the Finder’s dictionary resource (viewable from the Script Editor), or the Finder Event Suite document on this issue’s CD.

GETTING AND SETTING THE FINDER SELECTION

Determining which files have been selected by the Finder is something developers have been trying to do for a long time. Many ingenious and completely unsanctioned hacks and patches have been devised just to get this simple piece of information. Often, these patches fail to work beyond the Finder that they were designed for, and those that happen to work with multiple Finders may not be compatible with future versions. With the Scriptable Finder, there’s no need to patch, hack, or guess which items are selected in the Finder; one simple event will return the answer.

You can obtain the Finder’s selection by sending a Get Data event (event class kAECoreSuite, event ID kAEGetDataEvent) to the Finder, and specifying an object specifier for the property pSelection of the null container in the direct object. By

SCRIPTING THE FINDER FROM YOUR APPLICATION 69
### OVERVIEW OF FINDER EVENTS

The Scriptable Finder recognizes most of the events in the Required and Core suites, and defines a few events of its own in the Finder suite.

The Finder recognizes events that:

- count, get, and set data
- test whether an object exists
- open, reveal, select, and close objects
- tell applications to print documents
- move and copy files and folders
- make folders, aliases, and suitcases
- eject and unmount disks
- put away objects in the Trash or on the desktop
- empty the Trash
- query Gestalt

- quit and restart
- shut down and sleep

Of these events, Get Data and Set Data are the most versatile, as they can be used to determine and change a wide variety of properties of the Finder. Some of these properties are:

- the name, label, creation date, modification date, position, custom icon, and comment of an item, as well as its logical and physical size
- the creator, type, and version of a file
- the original item of an alias file
- the partition size of application files
- file sharing and view settings of folders
- the capacity and free space of a disk
- the memory used by a running process

Notice that before sending the event, we put in the optional “requested type” parameter. Coercing the data descriptor to a list isn’t necessary when sending an event to the Finder, but it is required by the OSL specification, so it’s a good habit to get into.

You can also change the Finder’s selection with a Set Data event (event class kAECoreSuite, event ID kAESetDataEvent): The direct object should again be an object specifier for the property pSelection, and the parameter keyAEData should contain the items to be selected. The key data parameter may contain an object specifier, an FSSpec, an alias record, an empty list (to clear the selection), or a list that contains multiple objects.

As you’ll see in the rest of this article, the Get Data and Set Data events are very powerful and can be used for a wide variety of purposes.

### GETTING THE FRONTMOST FINDER WINDOW

Although it’s not a terribly difficult thing for an ingenious bit of code to obtain a pointer to the Finder’s frontmost window, a well-behaved application never peeks at another process’s window list. The Scriptable Finder will tell you which windows are open if you ask nicely; once again, the event to use is the Get Data event. The direct object should be an object specifier to the window whose index is 1, as the frontmost window is always the first window in the window list.
Note that the event to get the frontmost window always returns an object specifier. It isn’t possible to get the Finder to return an FSSpec, alias record, or pathname to a window, because FSSpecs, alias records, and pathnames cannot represent a window — they always point to file system objects. For the event to return an alias to the file system item whose contents are displayed in the frontmost window, its direct object must specify “item of window 1,” that is, the item that owns window 1. In most applications, the window’s owner would be accessed via the specifier “document of window 1,” but because the Finder doesn’t have documents, its windows are owned by “items” instead. Listing 3 shows how to get the owner of the frontmost window.

The frontmost Finder window will usually be a folder window, but it could also be an information window, a sharing setup window, or even the About This Macintosh window or Finder Shortcuts window. To limit the window returned to only folder
windows, change the desired class from cWindow to cContainerWindow. Similarly, the open information windows can be identified by the class cInfoWindow.

The sample application Finder Snapshot on this issue's CD illustrates a very useful reason for requesting the list of open Finder windows. When launched, it records the set of open Finder windows in a document; opening the document results in the same

```
Listing 3. Getting the owner of the frontmost Finder window

// tell application "Finder"
// get item of window 1
// end tell
//
// Get the address of the Finder and make a Get Data event.
TAEEvent ae;

TDescriptor target = GetAddressOfFinder();
ae.MakeAppleEvent(kAECoreSuite, kAEGetData, target);
target.Dispose();

// Make an object specifier for "item of window 1" and put it into the // direct object of the event. Note that the Apple Event Registry class // for "item" is cObject.
TDescriptor directObjectSpecifier;
TDescriptor frontWindowSpecifier;
TDescriptor keyData;
TDescriptor nullDescriptor;

keyData.MakeLong(1);
frontWindowSpecifier.MakeObjectSpecifier(cWindow, nullDescriptor,  // formAbsolutePosition, keyData, true);
keyData.MakeDescType(cObject);
directObjectSpecifier.MakeObjectSpecifier(cProperty,  // formPropertyID, keyData, true);
frontWindowSpecifier, formPropertyID, keyData, true);
ae.PutDescriptor(keyDirectObject, directObjectSpecifier);
directObjectSpecifier.Dispose();

// Specify that we would like the result returned as an alias record // rather than an object specifier.
TDescriptor dataDescriptor;
dataDescriptor.MakeDescType(typeAlias);
dataDescriptor.CoerceInPlace(typeAEList);
ae.PutDescriptor(keyAERequestedType, dataDescriptor);
dataDescriptor.Dispose();

// Send the event, extract the reply, and dispose of the event and // reply. frontWindowOwner will contain an object specifier to the // frontmost window.
TAEEvent reply;

ae.Send(&reply, kAEWaitReply);
TDescriptor frontWindowOwner = reply.GetDescriptor(keyAEResult);
reply.Dispose();
ae.Dispose();
```
set of windows being opened again and positioned in the same locations that they were in at the time that the document was created. This application provides a simple way to make multiple “working sets” of Finder windows, easily accessible through items in the Apple Menu Items folder, or perhaps via documents sitting on the desktop.

**GETTING AND SETTING CUSTOM ICONS**

The icon bitmap of a file is available through ordinary file system calls, but there are a couple of different cases to contend with: the icon might be stored in the desktop database, or it could be a custom icon stored in the resource fork of the file. Some files are “special,” and only the Finder really knows what their icon bitmap should be. The simplest way to get the exact icon bitmap for a file is to ask the Finder what it is. Once again, Get Data and Set Data are the events to use.

The result of a Get Data event that specifies the icon property of some object is an AERecord that contains the entire icon family for the item’s icon. The record contains parameters whose key is the same as the individual resources of an icon family (for example, 'ICN#' and 'icl8'); the data stored in these parameters is identical to the data found in a resource of the same type. A Set Data event takes a record in the same format, or an empty list if the intention is to remove the custom icon.

Listing 4 shows how to remove the custom icon from every item in the selection. Note that the specifier “icon of selection” is equivalent to the more complex specifier “icon of every item of selection.”

The sample application Finder Tricks on the CD has a feature that changes the icons of all the items in the frontmost Finder window — each item is given some other item’s icon. Other than serving as a useful example of how to send events to the Finder, this sample doesn’t have much utility, although it does do an admirable job at messing up the appearance of Finder windows.

An application can change an item’s icon by writing the custom icon directly into the appropriate resources in the file and then setting the “custom icon bit” using the file system, instead of sending an event to the Finder — but the change won’t take effect right away. The reason for the delay is that the Finder isn’t notified when the contents of the disk change, so it must periodically poll the file system to find out whether it needs to redraw any items in its open windows. Polling happens only every now and again, so that the Finder doesn’t eat up every spare CPU cycle on the machine when it’s just sitting idle in the background.

**UPDATING FINDER CONTAINERS**

As just mentioned, the Finder sometimes takes a while to notice when the contents of the disk change. If an application writes new information into a folder, it may inform the Finder via an Apple event that the item changed (Listing 5). This event is most useful after an application has created a new file or has changed some visible attribute of an existing file — its type or creator, for example. If an update event isn’t sent, the Finder will eventually notice the change and redraw the item; however, there’s a several-second delay that’s somewhat disconcerting, particularly if the user has just saved a new document to the desktop with the Standard File dialog and expects to see it show up right away.

**SETTING UP SHARING**

Setting the sharing properties of a folder is a task that many users find confusing. Although scripting this task isn’t necessarily any easier, the availability of file sharing
scriptability makes possible applications that could walk through the process or could provide a more intuitive user interface than the Sharing dialog (commonly referred to in technical circles as “the evil grid of checkboxes”). Listing 6 shows how to enable file sharing on every folder in the current selection.

Unfortunately, not every file sharing feature is scriptable. It’s possible to set the sharing properties of a folder (everything that can be set from the Finder’s Sharing menu item), create a new user or a new group, and rename a user or a group; however, currently it’s not possible to set a user’s password, allow a user to connect to file sharing or program linking, add a user to a group, or remove a user from a group. This capability will be available in some future version of Macintosh system software.

Listing 4. Removing custom icons from the selection

```objective-c
// tell application "Finder"
// set icon of selection to empty
// end tell
TAEEvent ae;

TDescriptor target = GetAddressOfFinder();
ae.MakeAppleEvent(kAECoreSuite, kAESetData, target);
target.Dispose();

// Make an object specifier for "icon of selection" and put it into the
// direct object of the event.
TDescriptor directObjectSpecifier;
TDescriptor selectionSpecifier;
TDescriptor keyData;
TDescriptor nullDescriptor;

keyData.MakeDescType(pSelection);
selectionSpecifier.MakeObjectSpecifier(cProperty, nullDescriptor,
    formPropertyID, keyData, true);
keyData.MakeDescType(pIconBitmap);
directObjectSpecifier.MakeObjectSpecifier(cProperty, selectionSpecifier,
    formPropertyID, keyData, true);
ae.PutDescriptor(keyDirectObject, directObjectSpecifier);
directObjectSpecifier.Dispose();

// Obviously, a Set Data event needs data. In the case of this sample,
// the data we want is “empty,” which is represented by an empty list.
TDescriptor emptyList;

emptyList.MakeEmptyList();
ae.PutDescriptor(keyAEData, emptyList);
emptyList.Dispose();

// Send the event and dispose of it once it has been sent.
TAEEvent reply;

ae.Send(&reply, kAENoReply);
ae.Dispose();
```

Listing 5. Updating a Finder container

```bash
// tell application "Finder"
//   update alias "HD:Documents:"
// end tell
void UpdateFinderContainer(FSSpec& changedContainer)
{
    TAEEvent ae;

    TDescriptor target = GetAddressOfFinder();
age.MakeAppleEvent(kAEFinderSuite, kAEUpdate, target);
target.Dispose();

    // Make an object specifier for the FSSpec and put it into the direct
    // object of the event.
    TDescriptor directObjectSpecifier;

directObjectSpecifier.MakeFSS(changedContainer);
age.PutDescriptor(keyDirectObject, directObjectSpecifier);
directObjectSpecifier.Dispose();

    // Send the event and dispose of it once it has been sent.
    TAEEvent reply;

    ae.Send(&reply, kAENoReply);
ae.Dispose();
}
```

Listing 6. Sharing every folder in the selection

```bash
// tell application "Finder"
//   set shared of every folder of selection to true
// end tell
TAEEvent ae;

TDescriptor target = GetAddressOfFinder();
age.MakeAppleEvent(kAECoreSuite, kAESetData, target);
target.Dispose();

// Make a specifier for "selection."
TDescriptor selectionSpecifier;
TDescriptor keyData;
TDescriptor nullDescriptor;

keyData.MakeDescType(pSelection);
selectionSpecifier.MakeObjectSpecifier(cProperty, nullDescriptor, formPropertyID, keyData, true);

// Make a specifier for "every folder of..."
TDescriptor everySpecifier;

(continued on next page)```
MOVING FILES — AND AN UNDOCUMENTED PARAMETER

The Finder also has events that move and copy files from one container to another. Strictly speaking, there’s little reason for an application to use these events, since file copying can be done quite acceptably using the file system directly. However, it may take less code to tell the Finder to create a copy than to make the appropriate file system calls and put up a copy progress dialog. The events to use are kAEClone and kAEMove, both of which have the event class of kAECoreSuite.

A new parameter was added to the Move and Copy events of the Scriptable Finder after the *AppleScript Finder Guide* went to press, but before the Finder Scripting Extension shipped with System 7.5. The new parameter allows a Move event to specify the position of every item being moved inside the destination container. This parameter was not originally a part of the Finder Event Suite because a script that needed to position items being moved to another container could always go back and set the position property of the destination items after the move was completed.

The new Find File desk accessory included with System 7.5, however, needed to be able to move and position items all in one atomic operation; otherwise, the user would see the items move from an intermediate position to a final position, which would look jerky. The new parameter was added to fill this need; its use is shown in Listing 7.

The code in Listing 7 specifies the position of the item in local coordinates of the destination window. To specify the position in global screen coordinates, use the parameter keyGlobalPositionList instead of keyLocalPositionList.

---

**Listing 6. Sharing every folder in the selection (continued)**

```c
keyData.MakeOrdinal(kAEAll);
everySpecifier.MakeObjectSpecifier(cFolder, selectionSpecifier,
   formAbsolutePosition, keyData, true);

// Make a specifier for "shared of..."
TDescriptor directObjectSpecifier;
keyData.MakeDescType(pSharing);
directObjectSpecifier.MakeObjectSpecifier(cProperty, everySpecifier,
   formPropertyID, keyData, true);
ae.PutDescriptor(keyDirectObject, directObjectSpecifier);
directObjectSpecifier.Dispose();

// Set the property to true.
TDescriptor sharedSetting;
sharedSetting.MakeBoolean(true);
ae.PutDescriptor(keyAEData, sharedSetting);
sharedSetting.Dispose();

// Send the event and dispose of it once it has been sent.
TAEEvent reply;
ae.Send(&reply, kAENoReply);
ae.Dispose();
```
TEACHING THE FINDER NEW TRICKS

From the previous sections it should be clear that the events the Finder recognizes are all very similar, and the code to generate them looks pretty much the same. The event class and message ID may vary, and the contents of the direct object might specify different objects, but there’s nothing substantially different between the code.

---

**Listing 7. Moving a file with the optional position parameter**

```plaintext
// tell application "Finder"
//   move item "x" to preferences folder positioned at ¬
//       location {10, 10}
// end tell

TAEEvent ae;

TDescriptor target = GetAddressOfFinder();
ae.MakeAppleEvent(kAECoreSuite, kAEMove, &target);
target.Dispose();

// Make a specifier for item "x" and place it in the direct object.
TDescriptor directObjectSpecifier;
TDescriptor keyData;
TDescriptor nullDescriptor;

keyData.MakeString("\px");
directObjectSpecifier.MakeObjectSpecifier(cObject, nullDescriptor,
    formName, keyData, true);
ae.PutDescriptor(keyDirectObject, directObjectSpecifier);
directObjectSpecifier.Dispose();

// Make a specifier for the preferences folder and place it in the
// destination parameter.
TDescriptor preferencesSpecifier;

keyData.MakeDescType(pPreferencesFolder);
preferencesSpecifier.MakeObjectSpecifier(cProperty, nullDescriptor,
    formPropertyID, keyData, true);
ae.PutDescriptor(keyAEInsertHere, preferencesSpecifier);

// Put the point {10, 10} into the local position list.
Point destinationPosition;

destinationPosition.h = 10;
destinationPosition.v = 10;
keyData.MakePoint(destinationPosition);
keyData.CoerceInPlace(typeAEList);
ae.PutDescriptor(keyLocalPositionList, keyData);
keyData.Dispose();

// Send the event and dispose of it once it has been sent.
TAEEvent reply;

ae.Send(&reply, kAENoReply);
ae.Dispose();
```
that sends an event to open the System Folder and the code that sends an event to get the view setting of the frontmost window.

Be careful, though, when using the constants defined in AERegistry.h; they’re intended for use with the old System 7.0 Finder Event Suite. Using the old events (events whose class is kAEFinderEvents, or ‘FNDR’) has the advantage that they’re recognized by earlier System 7 Finders, but in general they should be avoided. The old events are buggy, they don’t work with the OSL, and they won’t ever be upgraded or changed to support new Finder features. Events in the new event suite (events whose class is kAEFinderSuite, or ‘fndr’) work better, return meaningful results, and are compatible with the OSL.

Programmer’s documentation of Finder events can be found in the *Apple Event Registry: Standard Suites* and the *Finder Event Suite* document on this issue’s CD. The old Finder events are described in the Finder Events chapter of the *Apple Event Registry*. The Scriptable Finder supports the Required and Core suites, as described in the *Apple Event Registry*, and also provides new events that are described in the Finder Event Suite. These documents list the events defined in each suite, the parameters that they take, the classes of objects defined in the suite, and the properties of those objects.

So, the next time you’re tempted to disassemble the Finder, poke around in private Finder data structures, or hack your way to Finder properties, remember the new event suite for the Scriptable Finder. Really cool integration with the Finder doesn’t have to be painful any more.

**RELATED READING**

- *Apple Event Registry: Standard Suites*, on this issue’s CD and available in print from APDA.

Thanks to our technical reviewers Sue Dumont, Max McFarland, Donald Olson, and Greg Robbins.