To wind up my first year of writing about scripting in develop, this time I’ll solidify the sequence of steps involved in making an application scriptable. A few of these steps have been mentioned before, while some material is new; here all the steps are organized so that you can work out a strategy for implementing scriptability. You may be surprised at what you’ll find.

THE WRONG WAY
In the past, a programmer who was responsible for implementing Apple events support in a scriptable application usually set about this task in one of two ways:

• writing the code for the event handlers and object accessor functions first, then, just before shipping, deciding what to call things and throwing together a dictionary at the last minute

• jumping into the design of an object model hierarchy (in an attempt to implement the Core suite), then writing the event handlers and object accessor functions, and, again, putting together the dictionary last

These methods were fine back in the days when Apple events were used principally for direct communication between two applications — one program was usually the client of the other. But in today’s world of scripting, it is users who are the clients. So in order to accomplish the goal of creating a human-friendly scripting vocabulary, developers need different methods for development.

THE NEW, BETTER WAY
Since your scripting interface is also a user interface to your application, it should be as full and rich as the graphical interface, and should be as intuitive as you can make it. In creating human-oriented scriptability, your goal is to make it as natural and as easy as possible for users to write sentences to communicate with and control your application. You want users to be able to write sentences that are as close as possible to the way they might think about what they want to do. Prepare to open up the full functionality of your application through scripting — you’ll want to make it complete.

The following plan will help you develop a clean vocabulary that allows users to easily work with your application.

PRACTICE YOUR WRITING
The first set of steps will help you home in on the terms you’ll use in your vocabulary.

Write down sentences. The very first thing to do is to write down as many sentences as possible describing actions that can be accomplished with your application. At this stage, don’t try to make real scripting commands; just write down basic ideas. For example:

• play movies
• grab the customer’s profile
• print pages 2 through 5
• translate this book from English to French
• send this message to Bob at the Redmond office
• find all the records containing “University”
• delete all paragraphs containing the word “Windows”

Have users write sentences. Users think differently about the way they accomplish things with applications than programmers do. Invite users of your application to write down some general sentences. Encourage them to think about how they want to accomplish what they do. Ask them to write the sentences as if they were directing the computer by speaking to it. (You can do this simultaneously with the above step.)

Include users who are experienced with earlier versions of your application. These users don’t need AppleScript experience. Consider inviting your documentation writers and your support people to participate. You’ll see quickly how users think about your application from a task-oriented perspective.

Don’t attempt to write code yet or design your object hierarchy around what users write. Just use this to help
you think in broad terms about how something might be accomplished.

**Write some commands.** Write more sentences, this time attempting to make script commands. Try to fit them into the context of a possible scripting vocabulary. This is an iterative process, through which you can distill your broad ideas into useful terms.

When writing commands, keep one eye open for consistency — think a bit about existing AppleScript commands and objects. At this juncture, it may help to have some people with AppleScript experience write sample sentences to describe how they want to control your application.

The sentences should begin to take on the flavor of AppleScript statements, with verbs followed by objects. For instance:

```plaintext
tell "emailer" to send the file "Weekly Report"
   to "Bob" at "Redmond"
tell "Mail Order Store" to order item "CW056"
   with nextday delivery
tell the front window to select the first
   paragraph containing "Macintosh"
```

**WRITING ANALYSIS**

In the next set of steps, you’ll develop your object model hierarchy from your early command writing.

**Analyze your initial commands.** The consumers of your product may surprise you. Some of the sentences they write will be too large in scope, but others will be highly focused to specific tasks. You’re likely to find that they’ll focus on the action first, then the objects. From those sentences, begin to determine the common verbs and objects. For example:

- verbs: play, get, set, translate, send, print, select, delete
- objects: movie, customer, paragraph, document, record, message
- properties: profile, leading
- enumerators: English, French, PowerTalk

**Make a crude object model hierarchy.** Based on the analysis of your commands, make a first cut at your object model hierarchy. Although many object classes in your vocabulary are types of objects that can be physically manipulated by your application, objects in scripting do not have to correspond to the objects on your screen. Nor should they match the objects in your internal code created by the programmers. Rather, script objects should be the most natural representation of what the user is trying to manipulate. Often these three — scripting, onscreen, and internal — will be nearly the same, but they don’t have to be.

Remember that consistency in a scriptable application is often accomplished through the liberal use of setting and getting properties instead of through large numbers of verbs. For more information, read the section “Designing Your Object Model Hierarchy” in my article, “Designing a Scripting Implementation,” in *develop Issue 21.*

**WORK ON YOUR DICTIONARY**

The key to a clean, intuitive scriptable application is its dictionary. It’s now time to develop this all-important “window” to your application’s soul.

**Look at other application terminologies for consistency.** Creating the AppleScript interface is a lot like creating the graphical interface. When designing dialog boxes, for example, most developers look at many other applications for examples of what works and what doesn’t. Similarly, you should view and use the AppleScript terminology of other applications to see how well they work. Remember that AppleScript hasn’t been around long enough for strong guidelines to be developed. Often you can do better than another application (in some cases, you can learn what not to do), but you also want your application to share as many elements as make sense with other applications your users might be familiar with. (When in doubt, refer to and practice with the Scriptable Text Editor; it’s clean and simple.)

**Make your first rough 'aete' and write commands.** When you’re ready, take a stab at making an ’aete’. Don’t expect too much at this stage; just get comfortable with the structure of this resource. Write some commands with your crude ’aete’. You can even open up your ’aete’ in the Script Editor and check the syntax of your commands against your dictionary. Even though you won’t be able to execute the commands, you’ll be able to practice writing sentences using the terms in your early dictionary.

**Adjust the 'aete'.** Looking at the commands written with your early terms, you’ll begin to see where the sentences look more or less natural, and where they’re awkward. Based on this, you can start improving on the terms in your ’aete’.

**Make more commands; have users write commands.** At this point, you’re ready to write some serious commands. By now you should be able to write real sentences that follow the AppleScript command structure: verb [object] [keyword value] … These
sentences should be similar in structure to standard commands that you can write for other scriptable applications. They should "feel" like AppleScript:

- play the movie "1984 Commercial"
- get the profile of customer "Caroline Rose"
- print pages 2 through 5
- translate the document "Tech Manual" from English to French
- set the leading of paragraphs 1 through 3 to 10
- send the document "Order 578" via PowerTalk

Note that the use of the word "the" is allowed in many places in AppleScript. Many of your users will include it in their commands. You should name your objects and properties so that they won’t sound awkward when preceded by the word “the.” And try to avoid property names that start with a verb.

Give your sample 'aete' to users and ask them to begin writing scripts to see how good your terminology feels and how it integrates and interacts with other applications. This interaction is crucial to understanding the value of AppleScript. All this can be done before any code is connected to the commands in the 'aete'. (Be sure to tell them that they can’t run their scripts.)

**NOW TO YOUR CODE**

A well-conceived dictionary will serve as a specification for programmers. Only after you've gotten your vocabulary in fairly good shape and done some preliminary testing with users should you (or your programmers) begin to write the code behind the vocabulary.

**Write object accessor functions.** It's probably a good idea to begin writing some of your object accessor functions first, so that you'll have something to test your Apple event handlers against. Accessor functions must cover all possible combinations of object classes and containers. However, accessor functions can be combined to handle more than one object class in a container if the objects are similar or lend themselves to code that can be shared.

For example, the Scriptable Text Editor has an accessor function for document objects, such as windows, within the application (the null container). It has another accessor function for all text objects within documents, such as characters, words, and paragraphs, and a third accessor for text objects within other text objects, such as characters within words, or words within paragraphs. Characters, words, and paragraphs were combined because the code to handle each of them was easily shared.

Also consider the language, framework, and structure of your existing code. Some frameworks, such as MacApp, use internal object member functions that are very similar to the accessor functions you'll write, lending themselves to individual accessors for each object class. You'll certainly want your accessor functions to make use of the existing internal functions.

**Write Apple event handlers.** Now you're ready to write the code to handle the Apple events. Since you've made the effort to lay the groundwork, this should be relatively easy. If your dictionary contains a lot of properties, consider implementing set and get early in the game.

**Test your code.** AppleScript is very useful for testing your Apple event code. You can easily write AppleScript commands that accurately send Apple events to your application. This is considerably easier than writing test code to fake sending Apple events to yourself. Scripter from Main Event makes an ideal tool for this task because you can observe what's going on in a script as it happens.

Once the code is connected, let a wider audience try your scripting. See how well the previously written scripts perform.

**Clean up your dictionary.** After you've gotten your code working, go back and carefully look over your 'aete' one more time. Make sure that you've organized the terms well and that your comments are understandable and innovative. Use the guidelines in my last column, “Thinking About Dictionaries,” in Issue 23.

**A NEW PLACE TO GET HELP**

There's now a resource on the Internet for posing questions relating to scriptability issues. It's a new mailing list: applescript-implementors@abs.apple.com. To subscribe, just send the following message to listproc@abs.apple.com:

```
SUBSCRIBE applescript-implementors Your Name
```

As always, happy implementing!