On the way to implementing scripting support in your applications, you’re bound to confront a variety of issues. In this column, I’ll give you some pointers for devising and testing property names and discuss the techniques for handling preferences through scripting.

PROPERTIES
In an application’s scripting vocabulary, a property is an attribute of an object. Properties can replace variables in if and repeat statements, as well as in expressions, and a script writer normally uses the AppleScript verbs set and get with them. Here I’ll give some guidelines for coming up with human-language names for properties and testing the viability of those names within the overall natural style of the AppleScript language.

It’s important that properties have names that users can easily become familiar with. Ideally, users should be able to refer to properties in a script the way they think or speak about them.

Don’t start property names with verbs. Starting property names with verbs leads to confusion when the property appears in the middle of a sentence. For example, naming a property disable call waiting leads to commands that don’t read smoothly:

set disable call waiting to true
if disable call waiting then ...

This is somewhat clearer:

set call waiting enabled to false
if not call waiting enabled ...

In fact, in the above case, it would be even better to name the property call waiting and use an enumeration as its value type (for a discussion of enumerations, see my article “Designing a Scripting Implementation” in develop Issue 21). The choices enabled and disabled allow grammatically correct sentences, as in the following:

set call waiting to enabled
if call waiting is disabled ...

A little creative thinking goes a long way in making it easy for users to work with the language.

The “the” test. AppleScript allows you to add or remove the word the almost anywhere in a script without changing the meaning of the script. Many script writers precede object and property names with the word the to make their scripts easier to read. Writing your test scripts in this way helps you determine the degree to which your property names facilitate forming natural sentences.

set the service to “America Online”
if the priority is high then ...

Don’t confuse attributes and actions. Sometimes setting a property can cause an immediate change on the screen. In deciding whether to use a property in this situation, a helpful rule is: When an action is initiated, use a verb; when an attribute changes (even if it produces immediate visible results), use a property. Another way of looking at this is if a visible change is immediate, it’s OK to use a property, but if an action has a duration, use a verb.

As an example, the following command causes an immediate change on the screen:

set the font of the third paragraph to “Courier”

Even though setting the font property creates a visible change, the font is still an attribute of the text, not an action. On the other hand, naming a property or enumerator playing, as shown in the next two commands, is a poor choice, because playing actually initiates an action:

set playing to true
set [the] status to playing

**CAL SIMONE** (AppleLink MAIN.EVENT, Internet mainevent@his.com) wants your dictionary for the Webster database, which will be used to help resolve human-name conflicts between different applications and scripting additions. He’ll be analyzing the terms in your vocabulary against others in search of similarities and differences. Send your ‘aete’ resources to Cal via AppleLink or the Internet.
The playing enumerator value in the second command is fine for obtaining state information, but a status property should be read-only. Instead of creating a property to control an action, use a verb. Verbs such as play or start playing are better suited for actions, as shown here:

play the movie "Wowie Zowie"
start playing the movie "Wowie Zowie"

Note that the commands are play and start playing, not play movie or start playing movie. In an application based on the object model, movie would be an object class.

The properties property. A properties property enables script writers to obtain all the properties for a given object in the form of a record by using a get properties construct. (I first suggested using records in this column in develop Issue 22.) The properties property can also be set with the set command. The sample properties property shown in Listing 1 can be included as a property of any object for which you allow the setting of more than one property at a time.

### Listing 1. A sample properties property

```plaintext
{ /* array Properties: 5 elements */ /* [5] */
  "properties",
  'Prop',
  'reco',
  "Property that allows setting of a list of properties.",
  reserved, singleItem, notEnumerated,
  readWrite, reserved,
  ... 
},
```

Don’t require the user to supply all the properties when setting the properties property — allow the setting of just one or a few properties.

get the properties of the fourth paragraph -- returns font, size, style, and so on
set the properties of the fourth paragraph to ¬
  {font:"Helvetica", size:14}

PREFERENCES

Developers use a variety of techniques to allow users to set preferences through scripts. I’ll describe three common and easily implemented approaches for dealing with preference properties in your application class. (These same approaches can be used to implement document settings or group properties for individual objects within your application.)

**Separate properties for each preference.** Implementing preferences as individual properties works well when you have only a few preferences. For example:

set the connect sound to "Shriek"
set the receive folder to alias "HD:Drop Folder"

If you have many preferences, it’s inefficient for the user to have to set each property individually. To solve this, you can implement your preferences as individual properties (usually in your vocabulary’s application class definition) and also include a preferences property, described next.

**A property that includes all the preferences.** You can make a single preferences or settings property, which is a record that’s defined elsewhere in your vocabulary. To define the elements of the record, create a fake “class” in your vocabulary, preferably in your Type Definitions Suite, to serve as the definition of the element labels in a record definition. In the comment field for your “class,” be sure to document clearly that this is a record definition, not an object class. Listing 2 illustrates this technique; for more information, see the section “Define Record Labels in a Record Definition” in “Designing a Scripting Implementation” in develop Issue 21.

Lists and records are the two principal constructs in AppleScript that don’t lend themselves to human sentence structure. They are, however, an integral part of the language and can occasionally help to make the script writer’s life easier. When you use a record to create a preferences property, it’s OK to stray a little from strict natural-language style. Of course, when referring to elements of a list or record, you should use natural-language style.

As with the properties property described earlier, don’t require the user to set all the individual preferences at once. Allow the setting of just one or a few preferences at a time:

set the preferences to ¬
  {connect sound:"Shriek", ¬
    receive folder:alias "HD:Drop folder"}

A user can address individual preferences as if they were defined as separate application properties. To allow for varying user experience with AppleScript, your application should always accept property
specifications for individual preferences using the technique described above, regardless of whether the user includes the qualifying phrase of the preferences.

For example, both of the following statements should be allowed:

- set the receive folder of the preferences to ¬ alias "HD:Drop Folder"
- set the receive folder to alias "HD:Drop Folder"

### Multiple “group” properties for grouping preferences.

If you have many preferences or want to group the preferences according to similar functionality, such as those often found in multipaned dialog boxes, you can create separate properties for groups of preferences or settings (using the record definition technique just described). The properties can reflect the groupings you’ve set up in your graphical interface:

- set the compiler preferences to ¬
  {warnings included:true, ¬
   default integer size:short integer}
- set the drawing settings to ¬
  {pen size:{1,2}, shape:circle}

A user addresses an individual preference by including in the object property specification the record that the preference is an element of, as follows:

- the pen size of the drawing settings
- set the shape of the drawing settings to rectangle
- set the default integer size of the compiler preferences to short integer

### PARTING WORDS

Following these guidelines in implementing scriptability in your applications makes it easier for users to write scripts. Although they may seem like small points, it’s the details that mean the difference between frustration and smooth sailing for the script writer. Remember to think about the way a user would write or speak about accomplishing what they want to do. Until next time, I remain your obedient servant on the AppleScript front. I’ll see you on applescript-implementors@abs.apple.com, the mailing list for scriptability.

---

**Thanks** to Eric Gundrum, Jon Pugh, and Derrick Schneider for reviewing this column.

---