ppleScript for Mac OS 8.5

Part One - Steve Harris

Back in the dawn of time (issues 2 to 6 of 1984-Online, anyway) I wrote a series of features on AppleScript and how to get to grips with it. Seeing that Apple has added a heap of new features to AppleScript now seems a good time to take a look at what new things it can do. I am not going to go over the basics of AppleScript, as this was done in issues 2 to 6 of 1984-Online, and are readily downloadable from 1984-Online's web site.

New Features

AppleScript is typical of brilliant Apple technology. With (compared to other programming languages) the tiniest amount of practice even non-techie types can create very clever scripts in a very English-like way.. It's brilliant, but suffered when Apple lost the plot. Steve Jobs returned, saw it was good, and demanded it be made better. What a guy!

So, after languishing in the hall of neglected technologies, AppleScript has been brought back to life with a whole raft of fab new features. Here's a summary of them:

Clipboard: support for the Clipboard is now included in the 'Standard Additions' file, making at available to all applications.
New Scriptable Control Panels: Control Panels such as Appearance, Apple Menu Options, File Exchange (previously PC Exchange) and the Location Manager are now scriptable, as are the File Sharing, Users and Groups and the Web Sharing control panels.
New Scriptable Extensions: Application Switcher (the thing you see when you 'tear-off' the Application menu) and ColorSync. In fact, using AppleScript is the only way you can customise the Application Switcher beyond toggling between an icon-only view and an icon and text view, and it can be made to do some interesting things...
New Scriptable Applications: Sherlock is scriptable. The Apple Help Viewer is also scriptable along with the Desktop Printer Manager and Network Setup.
New Toys: You can now write scripts to choose items from a list, pause scripts for a specified period of time, speak text, mount remote volumes (which is nowhere near as interesting as it sounds), summarise text (like in Sherlock), and have timed dialogue boxes, which could be incredibly handy for scripts that might run when there's nobody around.

PowerPC Native: And about time, too!
Folder Actions: Attach scripts to folders.
Documentation: At last there's some genuinely useful online documentation on AppleScript available in the Help Centre. It's packed with useful tips, very useful examples and even a few pre-written scripts. It also makes me redundant. Oh dear.

Putting This Into Practise

The only real way to learn about something is to do it. So we're going to write a script that takes advantage of some of the more useful features -- choosing from lists and timed dialogue boxes, along with speech, which may not be that useful but is a lot of fun.

The Script

We're going to write a script that displays a list of text-to-speech voices, from which the user can choose a single voice. Then we'll display a dialogue box where the user can type in some text for the Mac to speak, but they must be quick about it because they've only got five seconds or the dialogue disappears and the Mac makes a smart remark.

Choosing From Lists

This is probably the most useful addition of all. Included in the Standard Additions scripting addition this features is available in every scriptable application. You can display a list of items and the user can be allowed to select multiple(with the aid of the shift key) or single items which are then passed back into the script as a list variable.

Open the Standard Additions dictionary now. To do this:

Open the Script Editor application, which is usually located in the AppleScript folder in the Apple Extras folder.
Choose Open Dictionary from the File menu.
Click the "Go to Scripting Additions folder" button at the bottom of the Open dialogue.
Scroll down the list and choose the Standard Additions file.
In the dictionary window, if you then click on Choose From List, you'll see the choose from list command in full in the right pane of the window:

choose from list: Allows user to select an item from a list of strings

choose from list list -- a list of strings to display (an empty list if no selection)

[with prompt string] -- the prompt to appear at the top of the list selection dialog

[default items list] -- list of strings to initially select

[OK button name string] -- the name of the OK button

[cancel button name string] -- the name of the Cancel button

[multiple selections allowed Boolean] -- Allow multiple items to be selected?

[empty selection allowed Boolean] -- Can the user make no selection and then choose OK?

Result: list -- the list of strings chosen

All items enclosed in square brackets are optional.

We're now going to write the part of the script that gets a list of all available text–to–speech voices. To do this we're going to create a list variable containing the names of every file in the Voices folder, which is in your extensions folder. Type this into a new script document :

```
tell application "Finder"

set myList to the name of every item in folder "Voices" of extensions folder
end tell
```

These three lines need little explanation, to see what happens when this script runs do this:

Choose Show Result from the Controls menu.
Click the Run button.

You should see the result window looking something like this:

This is the format of a list variable containing all available voices. Lists start and end with curly braces with each item separated by commas. We've saved this result in a variable called myList.

We then need to show the choose from list dialogue box so the user can choose a voice from the list.

After 'end tell', type this:

```
choose from list myList ¬

with prompt "Choose a voice:" default items "Fred" without empty selection allowed
```

This doesn't use the full range of options available for the 'choose from list' command, but it will display the variable we created called 'myList' with a prompt to choose a voice. The default item is set to the voice 'Fred' and an item must be selected before the user can proceed. By default, multiple selection is not allowed, which is what we need as we can only speak with one

voice at a time.

Run the script again to see what happens. You should see a dialogue box.

Now we need to react to the user's choice from the dialogue box. They can either choose a voice and click OK, or they can click Cancel. We need to analyse what was returned in 'the result', but it's best to save this into our own variable as 'the result' changes with each action we perform.

To save the result into a variable called myResult, type this after the 'choose from list' command:

```
set myResult to the result
```

Now that's saved off, it's time to make some decisions. If the user clicked the Cancel button the 'the result' will contain a value of false. If they clicked OK then the result will contain a list of choices made. In other words, if they click Cancel we don't need to do anything else.

Now type this (but don't run it yet!):

```
if myResult is not false then

set myVoice to myResult   as string
```

So if the value contained in myResult is not false then we save myResult into a new string variable called myVoice.

Timed Dialogue Boxes
Next we want to display a dialogue box where the user can either type some text to speak and click OK or click Cancel. The intricacies of dialogue boxes were explained in my feature 'AppleScript Part 2', which appeared in Issue 3 of 1984-Online. The twist with our dialogue box this time is that it can also time out after 5 seconds.

Type this (the ¬ characters are optional and indicate the continuation of a line. They can be typed in the script editor by pressing option–return.):

```
display dialog ¬


"Type the text to speak (but be quick!):" default answer ¬


"" buttons "Cancel", " Speak! " default button 2 with icon note giving up after 5

set myResult to the result
```

Don't run it yet! The dialogue box will look like this:

In detail: The first command will display a dialogue box with a prompt reading "Type the text to speak (but be quick!):". Using 'default answer ""' puts a blank text-entry box on the dialogue where the user will type their text. There are two buttons called Cancel and Speak! and the default button (the button which is 'clicked' when you press Enter) is the second button. We add a pretty icon called note and 'give up' after 5 seconds.

Then we set the variable myResult to the result.

Speech
Then we must react to the user's choices from this dialogue. They can either type something and click the 'Speak!' button, not type anything and click the 'Speak' button or click Cancel.

So, firstly we only want to do something if the user clicked Speak.

Type this:

```
if the button returned of myResult is "Speak!" then
```

Self explanatory?

After that, type this:

```
if the text returned of myResult is "" then

say "You didn't type anything!" using myVoice
```

So if the 'text returned' is "" (meaning nothing was typed) then we make the computer say "You didn't type anything!" using the voice chosen from the 'choose from list' dialogue.

After that, type this:

```
else

set myWords to the text returned of myResult

say myWords using myVoice
end if
```

So if the user did type something to speak then we save it into a variable called myWords and say it using myVoice -- the voice chosen from the 'choose from list' dialogue.

Now the only situation left to deal with is if the dialogue box timed out after five seconds.

Type this:

```
else if myResult is gave up then
```

```
say "You'll have to be quicker than that!" using myVoice

end if
end if
```

Translated this means ‘else’ (this is the else to the ‘if’ questioning whether the button clicked was ‘Speak!’) if ‘gave up’ in myResult is true, then say “You’ll have to be quicker than that!” using the voice previously chosen. Then we end the ‘else if’ and end the first if which checked whether the Speak button was chosen.

Testing
Now you can run the script. Try it in the following combinations to make sure it all works:

Click Cancel when asked to choice a voice.

The script should finish.
Choose a voice, click OK, then click Cancel when asked to type some text to speak.

The script should finish.
Choose a voice, click OK, then quickly type some text and click Speak.

The text should be spoken in the voice you chose.
Choose a voice, click OK, then click Speak without typing anything.

You should be told that you didn’t type anything.
Choose a voice, click OK and wait to be told off.

You should be told that you have to be quicker than you were.

Here’s a summary of technical terms used in this feature:

TECHNO–BABBLE
Dialog(ue) Box
A window that appears on screen when the Mac needs to tell you some important information, or needs to get some information from you. Dialogue boxes are achieved by using the ‘display dialog’ command.

Variables
A named container of information. To create a variable, just specify its contents. Example: set myVariable to “Joe Bloggs”, creates a variable called myVariable with the text “Joe Bloggs” as its contents.

Dictionaries
A dictionary defines words (commands) that can be used in scripts. Dictionaries are found in a scriptable application’s file and in Scripting Additions. Scripting Additions can be found, oddly enough, in the Scripting Additions folder in your System Folder. To open a dictionary choose Open Dictionary from the File menu in the Script Editor.

The Result
An AppleScript-defined variable (i.e. you can’t define this as a variable yourself) that contains the result, if any, of the last command which would have returned a result to the script, like the button

clicked or text entered. To see the result in the Script Editor, choose Show Result from the Controls menu.

Conditions
Conditions are a way of making decisions in scripts, and controlling which actions are necessary. Conditions use the if... else... end if structure.

Steve Harris
<steve_harris@1984-online.com>