ow To Write An Award–Winning Script
Part Three

Welcome to the third part of 1984-OnLine's thrilling series on AppleScript. For those who haven't read the first two parts, I'm afraid that I haven't the space to go over what was covered then. They can be found in Issues 1 and 2 of 1984 OnLine at http://www.1984-online.com.

No messing about at the back, let's get down to business.

Control Statements
Although they may sound daunting, they are nothing to be feared. If you've followed the first two parts of this AppleScript series then you will have already used them, for example 'if… then… else…' is a control statement, as is 'tell… end tell'.

A definition would be 'structured commands that let you control the flow of the script'. Their structure makes them easy to use, because they always follow the same pattern. Much like the MacOS, really: you can learn one method of doing something (like using menus) that is consistent across all applications.

Here are some 'control statements':

tell… end tell
if… then… else… end if
repeat… end repeat
try... on error... end try

tell… end tell
This tells an application to do something.

if… then… else… end if
This checks a condition, which we learned about in Part 2. If the condition is true then do

something otherwise (else) do something else (end if).

repeat… end repeat
This will loop through all commands between repeat and end repeat. A condition (like in the ‘if’ statement) is used to control the repeat, and repeats have an extra word after ‘repeat’ to control when the loop should stop.
Examples:

repeat until myVariable equals 5
repeat while myVariable is less than 5
repeat with myItem in myList

We’ll learn more about some of these later.

try... on error... end try
This is used to handle errors. You tell an application to ‘try’ to do something, ‘on error’ (when an error occurs) do something. For example, say your script is using the Finder to copy a file onto a floppy disk and the disk is damaged. Without using a ‘try’ in your script, the Finder would display an error and the script would fail. By using ‘try’ you can control what your script does when it hits an error, such as display a nice, friendly message to advise corrective action:

e shall be looking at how to code control statements after we’ve looked at subroutines.

Subroutines
Look, like it or lump it AppleScript is a computer programming language. And it’s a very clever very sophisticated language, too. I don’t believe I’ve ever seen an easier programming language to write that allows you to do so much. Subroutines might sound scary, but they’re easy, honest.

A subroutine lets you chop up your script into smaller, reusable bits. A subroutine starts with ‘on’ and ends with ‘end’. This is best illustrated with an example:

```
on addVat(theAmount)

set theAmount to theAmount * 1.175

return theAmount
```

```
end addVat
```

Line by line:

```
on addVat(theAmount)
```

The first line of a subfunction starts with ‘on’ then the subfunction’s name: addVat. Following that there must be brackets, and optionally these may contain one or more parameters, separated by commas. Don’t panic, parameters are just variables. Parameters are ‘passed’ to subroutines because subroutines cannot see any variables that have not been defined within them, apart from properties and global variables (we haven’t ‘done’ them yet) unless they are passed to it by another subroutine. This parameter is a variable called ‘theAmount’.

```
set theAmount to theAmount * 1.175
```

This does a mathematical calculation (adds VAT -- Value Added Tax -- at 17.5%) to theAmount.

```
return theAmount
```

This ‘returns’ the variable ‘theAmount’ to the subroutine that called the addVat subroutine. Whatever is returned using ‘return’ becomes ‘the result’. In effect, your subroutine returns a result just like the result returned when you use the ‘display dialog’ command.

```
end addVat
```

Ends the subroutine structure.

This brings us nicely onto…

Droplets
No, not the splendid Mac sound, or anything of the ‘spilt drink’ variety. Droplets are scripts (or Applets, as technonerds call them) that you can drop files or folders on in the Finder. If you look in your Automated Tasks folder in the AppleScript folder, you’ll see loads of them, and they have icons like this:

ou turn an ordinary script (or Applet) into a Droplet by using a subroutine called ‘on open’. Only if this subroutine is in the script will the Script Editor automatically save a script as a Droplet. The ‘on open’ statement works much the same as other subroutines.

Example (pinched from Apple’s ‘Add Alias to Apple Menu script’):

```
on open theList

MakeAppleAliases(theList)
end open
```

Note how the parameter ‘theList’ is not enclosed in brackets after ‘on open’. This is because parameters in brackets are optional, whereas ‘on open’ is only going to be used when an item is dropped onto the script’s icon in the finder, and it’s a list of the items (files or folders) that were dropped which becomes the parameter.

However, it’s always important when writing scripts to think out every scenario. If a Droplet is normally opened by dragging files onto it, what would happen when the user double clicks the file?

Well, if all the commands are contained within the ‘on open’ subroutine, then nothing will happen. Not unless you also write an ‘on run’ subroutine outside the ‘on open’ subroutine (Script Editor won’t let you write it anywhere else).

The ‘on run’ subroutine works just like ‘on open’ except it’s only used when a script is double-clicked, and it has no parameters.

The Practical Bit

This month our practical bit isn’t terribly practical. We’ve going to look at Apple’s ‘Add Alias To Apple Menu’ script / droplet, which conveniently uses just about everything we’ve discussed in this article so far. It’s just as important to be able to read a script as it is to write it, and I don’t suppose you’re going to find a better written script than an Apple one.

The script begins:

otice how all the control statements (‘on’, ‘tell’, ‘if’) in this subfunction (‘on run’) have matching ‘end’ statements (‘end run’, ‘end tell’, ‘end if’), sand also notice how the Script Editor indents each of these statements by about half an inch. This helps you to ensure that your control statements have matching ‘end’ statements.

The purpose of the statement ‘on run’ in this script is to set ‘theList’ -- a list variable containing a list of files to be processed -- to selected icons in the Finder. This can only be used when an icon is selected in the Finder and the script is run from the Apple menu. However it also checks that if only one icon is selected, and its name is “Add Alias To Apple Menu” (the name of the script

itself), then it sets the list to -- empty. Having the “Add Alias To Apple Menu” as the only icon selected in the Finder would be the case, should the droplet have just been double–clicked.

Following that check, the script then calls subroutine ‘MakeAppleAliases’ using the list variable called ‘theList’ as its parameter.

```
on open theList

MakeAppleAliases(theList)
end open
```

The above is only run when items are dropped onto the script’s icon in the Finder, and therefore the ‘on run’ validation is not required. So it immediately calls the ‘MakeAppleAliases’ subroutine using ‘theList’, as above.

Here is the MakeAppleAliases subroutine. Notice ‘theList’ is passed in brackets:

```
on MakeAppleAliases(theList)

set NoItemsSelectedFlag to true -- initialise the flag
```

The above created a variable called ‘NoItemsSelectedFlag’ and sets it to true.

```
tell application "Finder"


repeat with x in theList



set NoItemsSelectedFlag to false
```

In the three lines above, the script uses the ‘tell’ control statement to ‘tell application “Finder” to ‘repeat with x in theList’ -- another control statement.

This repeat loop will automatically loop through every item in theList. The variable ‘x’ will refer to the current item of the list in the script.

Notice that once the script is in the list it sets the variable ‘NoItemsSelectedFlag’ to ‘false’.

```
try
```

make new alias file at apple menu items folder to x

on error

error "There was an error making the alias."

end try

Now we have a ‘try’ control statement, which is asking the Finder to ‘try [to] make a new alias file at apple menu items folder to x’. If it hits an error then it’ll beep and show a message.

end repeat

end tell

End the ‘repeat’, end the ‘tell’.

if NoItemsSelectedFlag then

display dialog "You have not selected an item to create an alias." & return & return & ¬

"Select an item and then run this application " & ¬

"from the Apple menu." & return & return & ¬

"An alias can also be created by dragging an item onto this " & ¬

"application." buttons "OK" default button 1

If the script never entered the ‘repeat’ loop, then the variable ‘NoItemsSelectedFlag’ would remain true. If this is so, then the script displays a dialogue box with the above message.

```
else


if result is not  then



display dialog "The Alias(es) have been added to the Apple menu." buttons "OK" default
button 1


else



display dialog "Aliases to PowerTalk items are not supported." buttons "OK" default button 1


end if

end if
end MakeAppleAliases
```

Otherwise (if items were selected), and ‘the result’ is not blank then the script will display a nice message confirming that everything ran successfully. If ‘the result’ is blank, a message is displayed about PowerTalk items not being supported.

That’s it for this month. We’ve learned about control statements, control statements and some control statements. We’ve also learned about subroutines, how to make droplets (by using ‘on open’), and some control statements too.

Here’s a summary of the technical terms used in this month’s article:

s ever, next month we shall put all this into practise, and learn some more AppleScript features along the way.

Comments to:
helpdesk@1984-online.com