Contents

1 AppleScript mode Tutorial

This is a tutorial that demonstrates the "AppleScript mode". It requires Alpha8 or greater and won't work with Alpha7 because it relies on the Tclapplescript.shlb shared library which needs Tcl 8.0 or greater. For Alpha7 a minimal set of functionalities is implemented though: editing scripts, syntax coloring, running a compiled script.

AppleScript mode lets you use Alpha as the front end editor to create, manipulate and execute your AppleScripts and take advantage of its powerful editing capacities.

In order to use AppleScript mode in Alpha, you must first activate the AppleScript menu. This can be done in different manners :

- - selecting the menu item "Config –> Preferences –> Menus..." and choosing 'AppleScript Menu' (checking the corresponding checkbox)
- - editing a file in this mode: open a new file and select "Scrp" in the right most popup window on the status bar. Note that the mode will be triggered automatically if the file has one of the following extensions: ".script", ".scr", ".scpt" or ".ascr". When a file is in AppleScript mode, the AppleScript menu is automatically displayed in the menu bar.
- - opening with Alpha a source file which has been created by the application "AppleScript Editor" will also edit it in AppleScript mode automatically.

AppleScript mode is identified by the four-letters code "Scrp" in the mode pop-up at the right of the status bar.

The AppleScript mode allows you to achieve with maximum flexibility all kinds of operations concerning scripts written in the AppleScript language, such as editing, compiling, decompiling, executing and processing. Advanced features include compilation of context files, execution of scripts in contexts etc.

Note that the AppleScript Editor application does not have to be launched or even present on your computer for this mode to work. It relies on the Tclapplescript.shlb shared library which interacts directly with the MacOS Open Scripting Architecture. This shared library must be present and should be located (itself or an alias) in the ”Tool Command Language” subfolder of the ”Extensions” folder inside the System folder.

This Tutorial, when opened from the Help menu in Alpha, is a ”shell” window. None of the changes you make will affect the actual file. If you close the window and then click on the hyperlink again, you will start with the same example as before. Note that ”Undo” is not available in shell windows.

The AppleScript mode menu can also be invoked while you are working in any mode in Alpha. You could have AppleScripts related to various tasks and trigger them when necessary.

1.1 First example: creating a script file

To create a new script, choose ”New Script” from the AppleScript menu: this simply opens in Alpha, directly in AppleScript mode, a new editing window. You are asked to provide a name: name it for instance ”Weekday.scpt”.

Type in the following instructions:

```
tell application "Finder"
        weekday of (current date) as text
end tell
```

Then choose ”Execute” from the AppleScript menu. This compiles internally the contents of your window and executes the compiled script. The result is displayed in a second window called ”* AppleScript Results *”. All the data resulting from the execution of various scripts will be appended to this window which you can save if necessary. In our case, the result should be the name of the current day. For example :

```
"Thursday"
```

1.2 Second example: compiling a script

You can make a compiled version of your script using the ”Compile” item from the AppleScript menu. This will allow you later to execute the script by just choosing it in the ”Scripts” submenu. The ”Weekday.scpt” script will be compiled with the name ”Weekday”. When a script is compiled, if the window does not correspond yet to a file on disk (newly created script using the ”New Script” item), the compiled script will be written in the AppleScripts folder in Alpha’s folder. Otherwise the compiled script will be in the same folder as the text file. You can execute any of the scripts stored in this folder by just selecting them in the ”Scripts” submenu of the AppleScript menu.

If you want to re-edit a compiled script, choose the ”Open Script” item in the AppleScript menu or, alternatively, open the ”Scripts” submenu while holding the option key down. In this case all the items become editable instead of executable: for example, if the option key is pressed the item ”Weekday” changes to ”edit Weekday”.

You can also run any compiled script located on your computer using the ”Run. . . ” item in the AppleScript menu. A dialog window lets you browse in the files hierarchy to select the compiled script you want to run.

1.3 Third example: adding a comment

A descriptive comment can be stored in the compiled script to explain to the users how the script works or what it does. AppleScript mode is able to deal with these comments. You must check the ”Include Description” check box in the AppleScript mode preferences: to do this, open the ”Preferences” item of the ”Scrp Prefs” submenu of the ”Config” menu (you must already be in AppleScript mode).

When the ”Include Description” preference is set, AppleScript mode will display the comment (if there is one) when it edits a script. The comment will be displayed at the beginning of the script like this:

```
(* Description
     Text of the comment here
*)
```

You can also create a comment yourself using the same syntax. Try to add the following to the previous example (choose the ”Add Comment” item in the menu to insert a template automatically):

```
(* Description
   This script finds the name of the current day.
*)
```

Make sure that the ”Include Description” preference is on and compile the script. Next time you edit the script, your comment will be there. You can also open your script with the AppleScript Editor instead of Alpha and see that the comment is there in the description field at the top of the editing window.

1.4 Fourth example: evaluating a compiled script in a Tcl proc

You can run an already existing compiled script from a Tcl procedure or any piece of Tcl code with the Scrp::doRun procedure like this:

```
global HOME
set file [file join $HOME AppleScripts Weekday]
Scrp::doRun $file
```

You pass the full path name of your script as the first argument. This proc returns the result of the execution of the script.

1.5 Fifth example: evaluating a script on the fly

The procedures defined in the AppleScript Mode files can be used in other Tcl scripts and allow you to evaluate AppleScript scripts on the fly inside a Tcl script. This way some variables can be evaluated at run time.

Here is a simple example. It defines a Tcl proc which will add the currently opened file to the Favorites menu of the MacOS Finder.

```
proc addToFavorites {} {
    set currfile [win::Current]
    if {[file exists $currfile]} {
 set scripttext "tell application \"Finder\"
 add to favorites \{file \"$currfile\"\}
 end tell"
 Scrp::doExecute $scripttext
 message "Added '[file tail $currfile]' to favorites"
    }
}
```

Make sure you escape the quotes and braces with a backslash in the definition of the text string.

So there are basically two ways of using the AppleScript scripting component from Alpha:

- you can have predefined compiled scripts in the AppleScripts folder or anywhere on your computer and execute them with Scrp::doRun
- you can build a text script and have it executed on the fly, like in the previous example, with Scrp::doExecute.

2 Advanced matters

AppleScript Mode allows you to compile scripts in specific contexts and not only in the global context. A context, in AppleScript parlance, is like a namespace : it is an environment in which some variables, functions (aka handlers) etc. are defined and are accessible to all the scripts you want to run in this context. Thanks to the contexts different scripts can share variables and data.

2.1 Compiling a context script

To compile a script as a context, just select the flag "Make Context" in the Apple Script Flags submenu. The compiled script will be refered to as a context script.

For instance, let's create a new script called VarContext.scpt with the following instruction :

```
set theSig to "ALFA"
```

and compile it as a context. We obtain a compiled script called VarContext which will be added to the Scripts submenu.

2.2 Running a script in a context

To run a script in a context, you must:

1. set the flag ”Run in context” in the Apple Script Flags submenu.
2. select a context. This is done by opening the Scripts submenu while pressing the Control key down and selecting an item. This item will be marked with a bullet showing that it is the current context.

Here is an example. Let’s create a script called NameFromSig.scpt with the following instructions asking the Finder to return the name of the application whose signature is the value of the variable theSig:

```
tell application "Finder"
  name of application file id theSig
end tell
```

Note that theSig is a variable and that it is NOT defined in this script. You can compile this script normally. This will result in a compiled script called NameFromSig. If you run this script (with the Run item in the AppleScript menu), you will have an error message because the variable theSig is not defined. Now run the script in the context VarContext previously defined : set the flag ”Run in context” in the Apple Script Flags submenu, open the Scripts submenu with the Control key down to select VarContext then run NameFromSig (NameFromSig.scpt is still the frontmost window). This time no error and the result is written in the AppleScript Results window like this :

```
"Alpha8"
```

2.3 Inheriting from a parent context

A context script can inherit the properties, data, variables, handlers etc. defined in another (already compiled) context. This context is called a parent context: the new context will contain all the data it defines as well as all the data defined in its parent. To compile a context and let him inherit from a parent context you must:

1. set the flag ”Inherit from parent” in the Apple Script Flags submenu.
2. select a parent context. This is done by opening the Scripts submenu while pressing the Shift key down and selecting an item. This item will be marked with a diamond showing that it is the current parent.

Example: let’s make a context called Inherit.scpt with the following instruction:

```
tell application "Finder"
    set theFolder to {container of application file id theSig}
end tell
```

This instruction defines a variable theFolder which designates the folder containing the application whose creator type is theSig. Once again theSig is not defined in this file but we are going to compile it and let him inherit from the VarContext script which defines theSig : set the flag ”Inherit from parent” in the Apple Script Flags submenu, open the Scripts submenu with the Shift key down to select VarContext, then compile.

The resulting Inherit script can be tested like this. Suppose you have a file called "HD:Dissertation:Conclusion.tcl" that we want to copy to Alpha's folder. Let's make a script called CopyToFolder.scpt with the following instructions:

```
set thefile to "HD:Dissertation:Conclusion.tcl"
tell application "Finder" to duplicate file thefile to theFolder
```

Now compile this script normally, then run it in the VarContext context: the variable theFolder is defined in this context. Its definition uses the variable theSig which was defined initially in the parent context and transmitted to the VarContext context.

2.4 Augmenting a context

If a script has been compiled as a context and its instructions have been modified, there are two possibilities when you recompile the script: the new instructions can replace the previous ones or they can be added to what was previously defined. This is controlled by the flag "Augment Context" in the Apple Script Flags submenu.

Example: let's define a script Variables.scpt with the following instruction:

```
set theSig1 "ALFA"
```

and compile it as a context script. Then replace the instruction by this one (i-e delete the preceding instruction):

```
set theSig2 "MOSS"
```

and recompile the script with the flag "Augment Context" set. The resulting context script Variables will make both variables theSig1 and theSig2 available.

2.5 Getting the scripting terminology of an application

To drive an application with an AppleScript, one must know which terms this application understands. Each application defines its own terminology depending on the kind of service it provides. The terminology can be stored in different ways:

- in a static way. It is stored in the resource fork of the application (systems prior to OSX) in resources of type 'aete'. Static terminology is the most current situation.

- in a dynamic way. The application accepts an event called "Get AETE" and returns all the terminology to the caller. It concerns mainly applications who support plugins and extensions. All the plugins can have their own dictionary and the application is responsible for gathering the terminology for all of them. This can't be done in the static way since you don't know in advance which plugins are currently available or loaded.

Alpha8 has dynamic terminology and Alpha7 has static terminology. With dynamic terminology, the application must be running to be able to receive the "Get AETE" request and to reply: if it is not running, it will have to be launched, which in some cases can take time. With static terminology,the application does not have to be running, nor launched. AppleScript Mode takes care of launching the application when necessary. Usually, dynamic terminology includes the static terminology.

AppleScript Mode has a flag called "Launch To Get Terminology" which lets you decide if you accept that the application be launched to retreive its dictionary. Setting this flag or not is a blind guess because you do not know in advance if the application has dynamic terminology but, if it does and if the flag is not on, AppleScript Mode will let you know immediately and offer the choice of putting the flag on. If the application does not have dynamic terminology and the flag is on, AppleScript Mode will look for static terminology anyway.

3 AppleScript mode Key Bindings

Most of the actions can be triggered from the keyboard instead of opening AppleScript menu. All of them are obtained by typing 'ctrl-a' followed by a letter to specify which command you are interested in. Here is the meaning of the different letters you can use with 'ctrl-a':

'ctrl-a b'	to display the info about bindings
'ctrl-a c'	to compile the current window or selection
'ctrl-a d'	to get a scripting dictionary
'ctrl-a e'	to execute (compile and run) the current window
'ctrl-a f'	to free compiled scripts stored in memory
'ctrl-a l'	to insert the line continuation symbol (¬)
'ctrl-a m'	to dump memory info
'ctrl-a n'	to create a new script
'ctrl-a o'	to decompile and open a script
'ctrl-a r'	to select and run a script
'ctrl-a s'	to check the syntax of the current window

Please e-mail any problem or bug you encounter : bdesgraupes@easyconnect.fr
Go to the "Alpha utilities" page :
http://webperso.easyconnect.fr/bdesgraupes/alpha.html