

Applescript Tutorial 2

This is the second installment of the Applescript Tutorials for Scientists, one response I got from the [first article](#) was to suggest a few reference books or articles.

[Getting Started with AppleScript](#) is a comprehensive overview of Applescript that is part of the Apple Developers reference library. I'm not sure if you need to be signed up as a developer to read it but since you can be an [online member](#) for free it might be worth signing up.

There is also the [Applescript Users list](#) which gives you access to an extensive network of scripting experts.

The [MacScripter](#) website offers a great place to find help, tutorials, scripting additions, etc.

There are also a range of excellent books, a couple that I'd suggest looking at are AppleScript: The Definitive Guide (ISBN 0-596-00557-1) by Matt Neuburg, and AppleScript Handbook (ISBN 0-9744344-9-3) by Danny Goodman.

There is also the excellent built in help which of course we can access with AppleScript :-)

Open up Script Editor and type:-

```
tell application "Help Viewer" activate
    search looking for "AppleScript"
end tell
```

Now click "Run" This should open up the Help Viewer application and run a search for "AppleScript"

Applescript and UNIX

Beneath the beautiful Mac OS X interface lies an industrial-strength UNIX system, many of the key UNIX-based scientific applications are now available under Mac OS X. While many users feel very comfortable in the command-line environment AppleScript offers a means to access the UNIX foundations without the need to remember a series of UNIX commands. However as you might expect there are a few things that you you need to be aware of when linking AppleScript and UNIX.

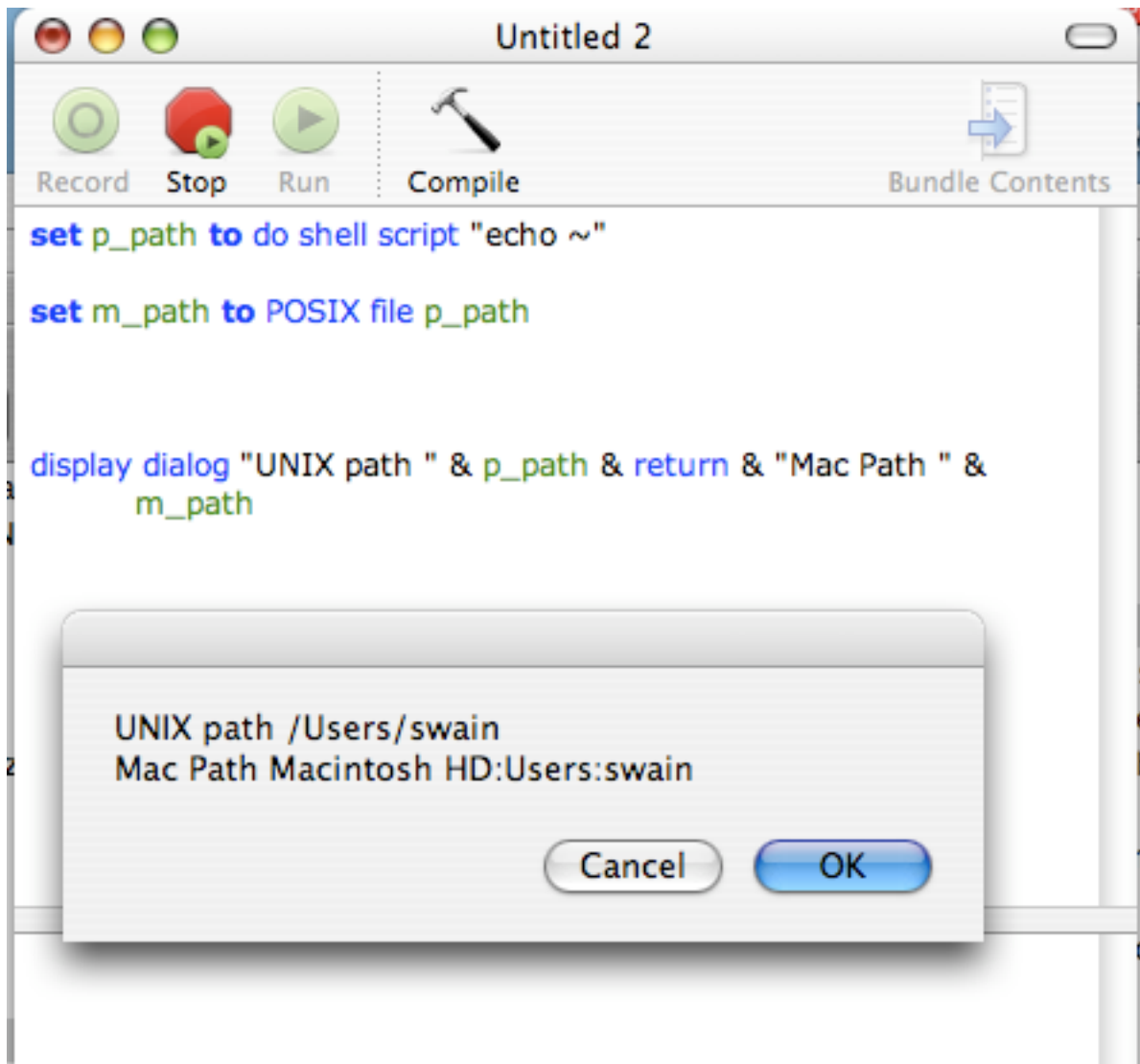
Leading you up the garden path

Applescript uses the colon ":" as a separator for directories however UNIX uses POSIX ([Portable Operating System for UNIX](#)) file paths in which the slash "/" is used as the directory separator. This could cause obvious problems, however one of the additions to AppleScript 1.8 was the ability to interconvert the two file reference systems.

Open up Script Editor and type/copy :-

```
set this_file to choose file
set this_file_text to (this_file as text)
display dialog this_file_text
set posix_this_file to POSIX path of this_file
display dialog posix_this_file
set this_file_back to (POSIX file posix_this_file) as
string
display dialog this_file_back
```

The first line allows the user to select a file, the second line converts the file reference to text and then it is displayed. The fourth line converts the Apple file reference to the POSIX path and then displays it. Of course you need to be able to do the reverse translation and the "POSIX file" command converts the POSIX path to a Mac OS X path as shown below.



Here is a very simple example of how this can be used, this takes a Macintosh Path to a folder, converts it to the POSIX path. The script then runs the Shell command "ls" to list the contents of the directory and displays the result.

```
set a to "Macintosh HD:usr:local:bin:" --Mac file path
set p to POSIX path of a --Convert to POSIX path
set the_text to (do shell script "ls " & p)
display dialog the_text
```

Whilst this works perfectly in most cases there are a couple of issues you should be aware of, a file or folder name may contain characters which need to be escaped to be passed to a shell command. Use quoted form to get the quoted form of a string. For example if a directory name contains a space.

```
set ap_sup to path to application support
```

```
set p_ap_sup to POSIX path of ap_sup
-- "/Library/Application Support/" -- contains a space
do shell script "ls " & p -- does not work
do shell script "ls " & quoted form of p -- works
```

Come out from under that shell

The default shell for Mac OS X is BASH and I don't want to get into a debate about the relative merits of the different shells that are available, the important thing to remember when running a "Do shell script" command is that do shell script always uses /bin/sh to interpret your command, **not** your default shell. So commands that work fine in the Terminal may not work when called using the "Do shell script" command. In addition when you use just a command name instead of a complete path, the shell uses a list of directories (known as your PATH) to try and find the complete path to the command. For security and portability reasons, do shell script ignores the configuration files that an interactive shell would read, so you don't get the customizations you would have in Terminal. You need to use the full path to the command. Compare the results of these two applescripts:

```
do shell script "echo $PATH"
```

Should give you something like "/usr/bin:/bin:/usr/sbin:/sbin"

On the other hand this applescript

```
set the_shell_script to "echo $PATH"
tell application "Terminal" activate
    do script the_shell_script
end tell
```

Will give you the complete PATH as defined in your .profile.

Having a little Fun with UNIX

This script gets a list of all the applications in the Applications folder, then finds those applications that contain the capital letter "B", then sorts them into reverse alphabetical order and displays the result. The "|" is the pipe command, the pipe command is one of

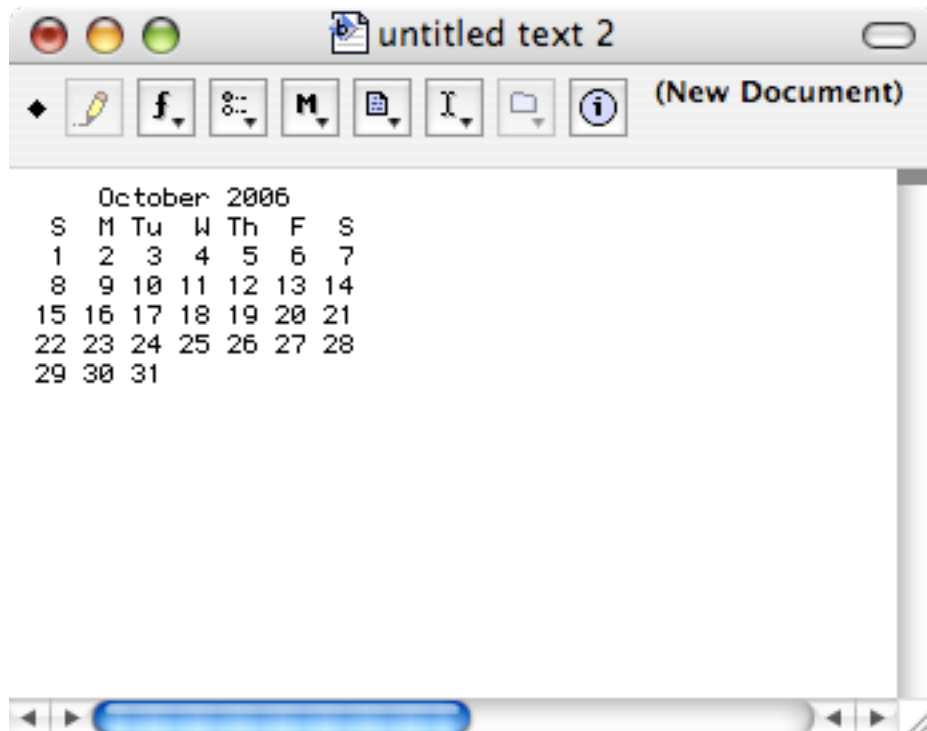
the most important commands in UNIX because it allows us to create powerful functions in a single statement. The pipe command is represented with the | character and it is used to connect the output from one command and send it as input to another command. On a Mac keyboard it is on the right-hand side next to the return key.

```
set a to "Macintosh HD:Applications:"  
set p to POSIX path of a  
set the_text to (do shell script "ls " & p & "| grep B | sort -r")  
display dialog the_text
```

Here is a fun feature:-

```
set the_script to "cal"  
set the_text to (do shell script the_script)  
set the clipboard to the_text
```

Run this script and then paste into a text editor, (I used BBEdit) and you should see this months calendar. Replace "cal" with "cal 10 1492" and you can see the month in 1492 when Christopher Columbus landed on a Caribbean Island.



Putting it under the Spotlight

Tiger introduced Spotlight, a wonderful searching mechanism that indexes the contents of all the files on your Mac without any effort, it also provides a means to search that information. However you can also search the metadata using UNIX, mdfind provides a command line search interface, we can interact with this using Applescript.

As you might expect as a chemist I have a lot of documents containing chemical information the following script uses mdfind to identify all files that contain the word 'ketone', it displays all the file names and then a count of how many files, too many.

```
set the_script to "mdfind 'ketone'"
set the_text to (do shell script the_script)
display dialog the_text
set num_records to count of paragraphs in the_text
display dialog num_records
```

I remember that I downloaded the file I want last week so I can limit the search.

```
set the_script to "mdfind -onlyin '/Downloads/' 'ketone'"
set the_text to (do shell script the_script)
display dialog the_text
set num_records to count of paragraphs in the_text
display dialog num_records
```

Now I only find two files. OK so I could do that with Spotlight, well download [ChemSpotlight](#) and install it. ChemSpotlight is a Spotlight metadata importer plugin for Mac OS X, which reads common chemical file formats (MDL .mol, .mdl, .sd, .sdf, Tripos .mol2, Protein Data Bank .pdb, Chemical Markup Language .cml, and XYZ) using the Open Babel chemistry library, it then adds molecular formulas (complete with subscripts in the Finder), molecular weight, and a variety of other chemical information.

Now if we modify the mdfind script as shown below, this script now searches your hard drive for all chemical structure files that contain a molecule with a molecular weight less than 250.

```
set the_script to "mdfind 'net_sourceforge_openbabel_Mass < 250'"
set the_text to (do shell script the_script)
display dialog the_text
```

ChemSpotlight adds a variety of metadata as shown in the table below, all of which can be searched using mdfind.

Metadata Field	Notes
net_sourceforge_openbabel_Chirality	True/False (1/0)
net_sourceforge_openbabel_Dimension	0D/2D/3D depending on the coordinates found
net_sourceforge_openbabel_DisplayFormula	Formula with subscripts for Finder gGet Info windows
net_sourceforge_openbabel_Formula	Chemical formula in standard gHill Orderh
net_sourceforge_openbabel_Mass	Standard molecular weight in a.m.u. (g/mol)
net_sourceforge_openbabel_ExactMass	Molecular mass of most common isotopes for mass spectra
net_sourceforge_openbabel_NumAtoms	Number of atoms in the molecule
net_sourceforge_openbabel_NumBonds	Number of bonds in the molecule
net_sourceforge_openbabel_NumMols	Number of molecules in the file
net_sourceforge_openbabel_NumResidues	Number of biomolecule residues
net_sourceforge_openbabel_SMILES	Daylight SMILES string for this molecule
net_sourceforge_openbabel_InChI	IUPAC/NIST canonical identifier
net_sourceforge_openbabel_Sequence	Biomolecule residue sequence

Give me some input

This is the mechanism by which [iBabel](#) searches chemical information, of course iBabel was built using [Applescript Studio](#) which allows the construction of rich user interfaces, and hopefully that will be the subject of a future tutorial. However we can use Applescript to provide a limited user interface, the following script asks the user to choose a file, it then uses the UNIX commandline tool OpenBabel to convert the file to the format chosen in the second dialog box and then saves it to the desktop. You will need to have installed [OpenBabel](#) or better/easier [ChemSpotlight](#). You could have yet another dialog box asking to choose where to save the file but I think you can see why Applescript Studio would be a better option.

```
--Get path to desktop to save result
set out_path to path to desktop
set out_path_posix to (POSIX path of out_path & "output_file")
```

```
--Get file to convert
```

```
set TheFile to (choose file with prompt "Choose a file for Conversion")
set the_posix_file to POSIX path of TheFile
display dialog "Choose output file format" buttons {"smi", "mol2", "sdf"} default
button 3
set out_format to the button returned of the result
```

```
--Generate shell script-- /usr/local/bin/babel '/Public/multimol.sdf' -osmi
'/Users/username/Desktop/outputfile.smi'
```

```
set babel_script to "/usr/local/bin/babel '" & the_posix_file & "' -o" & out_format &
"' & out_path_posix & "." & out_format & ""
do shell script babel_script
```