AUTOMATOR FOR LION++

MICHAEL SHARPE

1. Briefly

Apple's latest incarnation of Automator in Snow Leopard, Lion and Mountain Lion provides new reasons to consider learning to use it, mostly because Snow Leopard reworked the Services menu to be much less cumbersome than before. The Keyboard item in System Preferences now allows the switching on/off of many individual items in the Services menu, and Automator may be used to generate entries in the Services menu.

There is a substantial overhead incurred in using Automator. If a shell script or python script can do the job, it will work much faster than the equivalent Automator script. You should reserve Automator for those cases where you need to either (a) take input in a form not understood by traditional scripts, eg data dropped on script, or selected items become input; (b) use a GUI (but scriptable) Mac program as part of the Automator stream.

2. Defining a new Automator workflow

When you start Automator it begins by asking you to choose a template. The most important among these are:

Workflow: The input is not defined automatically—it is up to you to provide the input it expects. There are several ways in which a saved Workflow may be called and provided with input;

- The workflow may be run directly from within the Automator application.
- A saved Workflow may be inserted as part of another Workflow, which provides its input and processes its output.
- A Workflow may be run and provided with one or more inputs from the command line using

  ```
  /usr/bin/\textsf{Automator} -i <input> <path_to_workflow>
  ```

Application: The input is whatever you drop on the application, or call from, say, Applescript with a line of the form

```
tell application "Finder" to open "<inputfile>" using "<path_to_app>"
```

An Application may also run as part of another workflow, which provides its input. If the application requires no input, it may be run just like any other application, by double clicking its icon.

Service: The input is provided by what is selected at the time the Service is called from a menu, provided the service is defined to allow that kind of input. (Services may specify No input.) This could be one or more items in the Finder, a piece of text in a document, or other possibilities that you

Date: June 20, 2013.

may select from in setting up the Service workflow. User-defined services are stored by default in `~/Library/Services`. (Other services may be found in `/System/Library/Services` and `/Library/Services`.) Services appear in the list accessed from System Preferences/Keyboard/Keyboard Shortcuts/Services, and may be given a hotkey by selecting the line (left-click), then left-click about a half inch from the end of the selection to highlight the hotkey area and press the hotkey combination.

Folder Action: These are workflows attached to a folder in the Finder. Items added to the folder cause the workflow to run and become the input to the workflow.

Workflows and the like may feel a bit simpler in structure and a bit more confining than other scripting languages, but they often allow you to accomplish a lot with little effort, and, since an arbitrary shell script, python/perl/ruby script or AppleScript may be an element of a workflow, the real constrictions occur at initial input, final output, and error handling. The most important thing to remember is that the information flows from one black (or, at least, gray) box to the next, each doing its own specialized processing before passing the results to the next. Variables may also be defined at any stage to allow information to be stored and retrieved at a later stage. It is puzzling at first to follow what must be going on under the surface, but let's examine some simple cases.

3. Some Basic Examples

The cases are different according to the type of input.

3.1. No input.

This would normally not be appropriate for a Folder Action. Most commonly, it could be an Application which acts as a GUI wrapper for a script of some kind having no arguments. A simple example would be a shell script that copies a list of files (filenames specified in the script) to a folder, assembles them into a `.dmg` and uploads the `.dmg` to a server.

3.2. Input is a single file.

The workflow must be set up to handle input files. This is automatic for Applications and Folder Actions, and can be specified as input to a Service.

3.3. Input is multiple files or folders.

There are two ways to proceed. The seemingly simpler approach is to set up the workflow as if to handle a single file or folder, followed at the end by a Loop item. What appears to happen is that each of the files or folders is sent in parallel (ie, almost simultaneously) through a new copy of the workflow. This is not likely to cause problems if the number of files is small, but could lead to memory issues if there are thousands of input files. For example, if the workflow had just one item—'Open Finder Items'—then every file passed would be opened (almost) simultaneously. If the goal is instead to open them one by one to do something in each file, one solution would be Nyhthawk Productions' Dispense Items Incrementally, which fixes this problem by taking a batch of files as input and releasing them to an Automator Loop one at a time. (It should be the first action in the workflow, and Loop should be the last action in the loop.) Once all the items have been through the loop, the action stops the workflow.

In general, if you know a little shell scripting, it is preferable to write a script to handle the input items one at a time. If the first item of your workflow is Run Shell Script, and for Pass inputs you select as arguments, then Automator presents the list of files/folders in the argument stream, ready for you to process one by one in your script. The default bash script it presents shows how:

```
for f in "$@"
do
      echo "$f"
done
```

This says: for each file `f` in the argument stream, write the file/folder name to the output stream. (The quotes are necessary in case the path names contain spaces.) In other words, this simple script does nothing but pass through to the next stage everything it received as input. Likewise, the default Run AppleScript is:

```
on run {input, parameters}
          (* Your script goes here *)
          return input
end run
```

Its effect is exactly the same as the bash script above—it sends the input list along to the next stage as output. In AppleScript, `input` can be either a string or a list, and a list with one string element is handled as if it were a string. The `parameters` can in most cases be ignored — they convey information specified by the GUI environment from which the script was called. There is a rather vague explanation at

```
http://developer.apple.com/mac/library/documentation/_
AppleApplications/Conceptual/AutomatorConcepts/_
Articles/ImplementScriptAction.html
```

4. Copying Automator actions to another machine

The Automator actions are spread out over a number of locations, so some method is required to keep track of their location. For example, actions installed by the operating system are usually found in one of

```
/System/Library/Automator
/Library/Automator
/Library/Services
```

and are available to all users. Individual users normally save Automator actions in one of

```
~/Library/Automator
~/Library/Services
~/Library/Workflows
~/Library/PDF Services
```

If you want to be able to find them easily, and copy them between machines, you could organize them as follows.

- Automator Applications you write are normally stored in `~/Library/Workflows/Applications`. For quick access, drag that folder onto the Dock and make an alias to `~/Library/Workflows` in your Favorites folder.
- Make a new folder on your desktop named, for example, `Automator Folders`, and drag the folder onto your Places listing (left side of any Finder window), and into your Favorites folder so that it may be found easily using a Get File or Folder dialog. If you use Default Folder, add an entry for the folder in its defaults list as well.
- Make an alias in `Automator Folders` to each of the folders listed above that are in your home folder.

- To copy your personal Automator actions to another machine, create a zip file named, for example, `Automator_personal.zip` using the following commands in a Terminal window.

  ```
  cd ~/Library
  zip -r Automator_personal Automator Services Workflows "PDF Services"
  ```

  Copy the file to another machine and, in a Terminal window on the other machine, change directory to the folder containing the zip file and type

  ```
  unzip Automator_personal -d ~/Library
  ```

  The effect will be to overwrite any existing Automator action of the same name, add actions that do not exist, and leave in place those that are not installed on the machine you copied from.

It is another matter to copy the services selection together with the associated short-cut keys. This information appears to be stored in `/Library/Preferences/com.apple.systempreferences.plist`, but not in human-readable form, and there could be unpredictable results if it were copied directly to to another machine.

5. Notes on debugging services

The normal means of debugging any Automator workflow is to run it one step at a time, checking the results of each step and keeping the log panel open (from Automator's View menu) for information. There are some traps though in the case of services which define the input. Suppose, for example, your service expects to get files and folders selected in the Finder as its inputs. As far as I can see, when debugging the service, those inputs do not function, and it is necessary while debugging to replace the normal input mechanism temporarily with a Get Specified Finder Items action.