A STEP-BY-STEP FOR MAKING A DROP-ON INTERFACE TO A SHELL SCRIPT

MICHAEL SHARPE

1. Introduction

Suppose you have an executable script (shell, python, perl ...) which expects one or more files or folders as inputs on the command line. You may of course type the name of the script in a Terminal window, a space, and then drag and drop the files and folders from the Finder. That's not especially Mac-like, and it's not always convenient. This article provides a step-by-step method for constructing a drop-on face for the script. There are other methods based on Platypus, but I'll use the Mac's own Automator.

For a very simple example, here's a shell script named `nabl` to enable a single font map located in a texmf tree in the system area `/usr/local/texlive`:

```
#!/bin/bash
f=${1##*/}
sudo -H mktexlsr # update the lsr database, just in case
sudo -H updmap-sys --enable Map ${f}
```

The `${1}` is the first argument given on the command line after `nabl`, and `${f}` is the name of the file without any directory information, as this is what `updmap` expects. (This is not a very useful example, just an illustration of what we're doing. A more useful script would check for a number of error possibilities and alert the user before carrying out the execution.) We are going to provide a Mac application onto which a file can be dropped, the end result of which will be to pass the file name to the script in a Terminal window so that errors can be viewed.

As a second example, which I find quite useful, here is a script `png512` to make a `png` file exactly 512 by 512 with a transparent background, which is just what is needed for an icon. This script is called with the path to the file as the first argument and the resolution (in `dpi`) to render it as the second. Eg,

```
png512 ~/Documents/x.ps 580
```

```
if [ $# -lt 1 ]; then
echo "Filename and resolution required"
exit 1
fi
if [ $# -gt 1 ]; then
res=$2
else
res="580"
fi
f=${1##*/} # just the filename
fb=${f%.*} # without extension
fd="${1%/*}" # the dirname
cmd='/usr/local/bin/gs -q -dNOPAUSE -dBATCH -sDEVICE=pngalpha -dEPSCrop -r'
cmd=${cmd}"${res} -sOutputFile=${fb}.png -g512x512 ${f}"
```

Date: January 27, 2014.

```
if [ "${fd}" != "${1}" ]
then
  cd "${fd}"
fi
${cmd}
```

2. THE STEPS

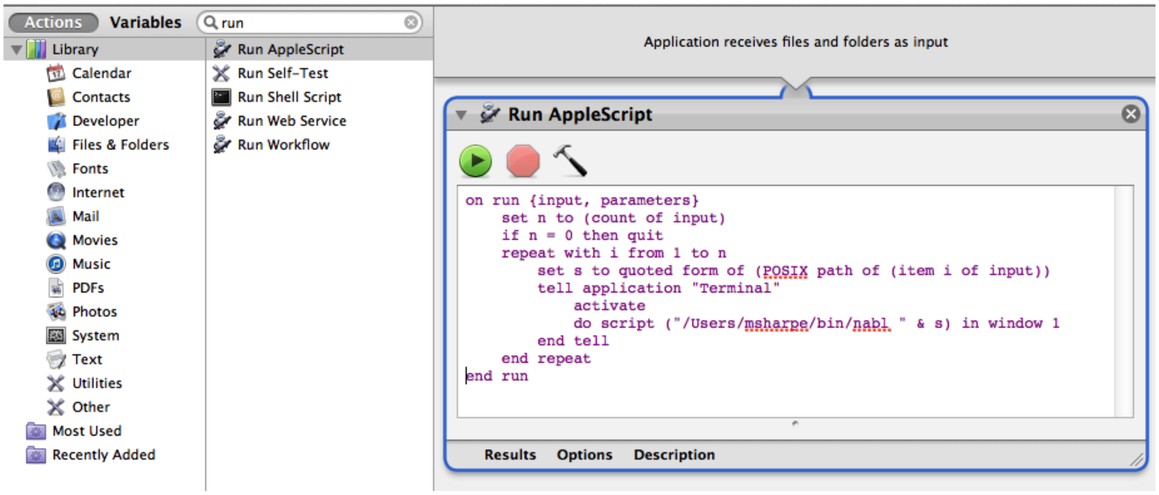
1. Make sure your script works exactly as expected when run from the command line. Be sure to make it executable using the command line. Eg,

```
cd ~/bin # or wherever nabl is located
chmod 755 nabl
```

2. Open the application Automator, and, from the opening splash screen, choose to create an Application. Near the top left of the Automator window, you'll see a small search-box just to the right of the labels **Actions** and **Variables**. Type "Run Applescript" (without the quotes) into that search box, then drag the resulting component onto the workflow list to the right of the window. The AppleScript you enter there should be as follows:

```
on run {input, parameters}
    set n to (count of input)
    if n = 0 then quit
    repeat with i from 1 to n
        set s to quoted form of (POSIX path of (item i of input))
    tell application "Terminal"
            activate
            do script ("/Users/msharpe/bin/nabl " & s) in window 1
    end tell
    end repeat
end run
```

Here's a screenshot of part of the finished Automator window.



```
on run {input, parameters}
    set n to (count of input)
    if n = 0 then quit
    repeat with i from 1 to n
        set s to quoted form of (POSIX path of (item i of input))
        tell application "Terminal"
            activate
            do script ("/Users/msharpe/bin/nabl " & s) in window 1
        end tell
    end repeat
end run
```

Save your application to any convenient location.

3. A custom icon may be considered a frill, but becomes almost very useful if you have a number of such application as constructed in step 2. You may use any icon you wish, but I like to make a simple square box with a few letters to remind me of the purpose, and such icons are easy to construct in TEX. The following model may be varied according to your needs, but don't idly change the pagesize or the picture size—they are computed to give a good fit with little roundoff.

```
\documentclass[10pt]{article}
\parindent=0pt
\usepackage[papersize={63.797pt,63.797pt},margin=0pt]{geometry}
\usepackage{libertine-type1}
\usepackage[T1]{fontenc}
\usepackage[dvips]{pstricks}
\usepackage[libertine]{newtxmath}% in case math symbols
\newcommand{\mytxt}[1]{{\usefont{T1}{LinuxLibertineT-LF}{b}{it}%
\fontsize{22pt}{27pt}\selectfont #1}}
% Adjust the font and the fontsize entries to your taste
\pagestyle{empty}
\begin{document}
\noindent
\psset{unit=.1246pt}
\begin{pspicture}(0,0)(512,512)
\psframe[linewidth=2pt,linecolor=red](8,8)(504,504)
\rput[c](256,256){\mytxt{NABL}}%may need to adjust center horizontally
\end{pspicture}
\end{document}
% render at 580dpi to get image 512px
```

If you wish two lines of text on your icon, you could replace the `\rput` line with, eg,

```
\rput[c](256,256){\parbox{60pt}{\centering\mytxt{PNG\\[5pt] 512}}}
```

IMPORTANT: Because of the uncertainties of text spacing, you may need to adjust the horizontal position of the center, replacing `(256,256)` with, eg, `(250,256)`, based on appearance.

The next step is to make a 512x512 `.png` from the .ps file output by LATEX. Open a Terminal window, `cd` to the folder containing your `.tex` file (we'll assume that its name is `nabl-icon.tex`), and type

```
png512 nabl-icon.pdf 580
```

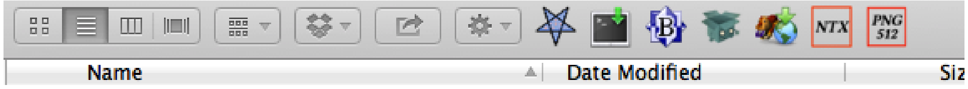
Finally, we convert the `.png` to the icon format `.icns`. The simplest way I've found is to use the free version of the commercial program **Img2icns.app** from

```
http://www.img2icnsapp.com
```

which both converts images (use a format like `.png` which supports transparency) to `.icns` format and, optionally, applies the `.icns` to any application dragged onto it following making the `.icns`. That is

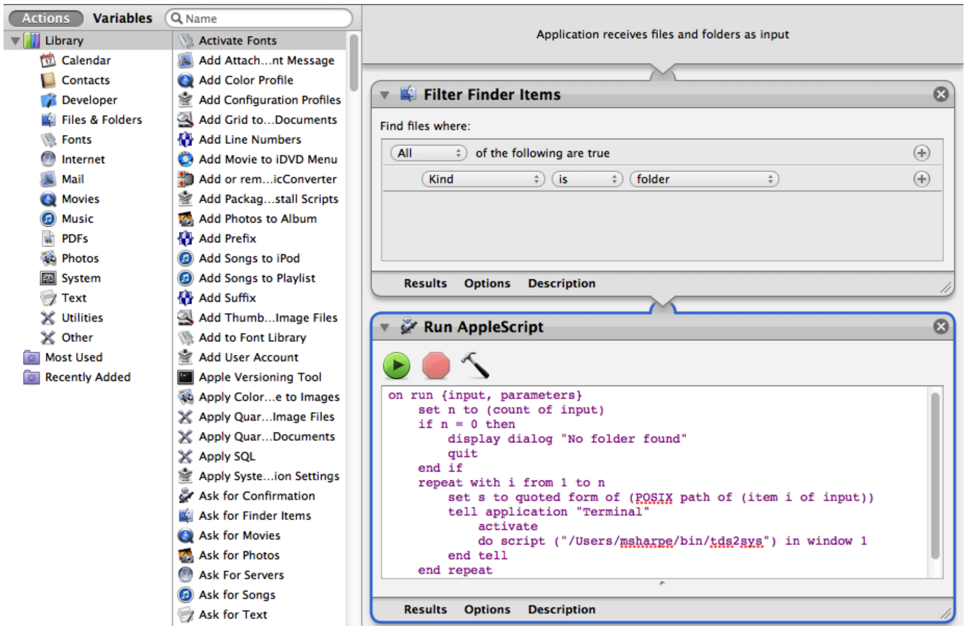
- Start **Img2icns**;
- drag your `.png` onto the box in its window;
- to make an `.icns` file, click the **icns** output button. To apply the icons to an application, drag the application onto the same box.

4. Having applied a suitable icon to your application, you may drag map files onto its icon to enable them. I find it useful with frequently used script applications to open the folder in which they reside and drag the application to the top bar of the window—you may need to hold it there a couple of seconds or wiggle the mouse a bit to have Finder understand that you want to mount the icon there so it's most convenient for dragging onto. The result is like this:



3. Some other example scripts

Install a font package in texmf-local.



Find files where:

All of the following are true

Kind is folder

```
on run {input, parameters}
    set n to (count of input)
    if n = 0 then
        display dialog "No folder found"
        quit
    end if
    repeat with i from 1 to n
        set s to quoted form of (POSIX path of (item i of input))
        tell application "Terminal"
            activate
            do script ("/Users/msharpe/bin/tds2sys") in window 1
        end tell
    end repeat
```

The Automator script is a little more complicated, filtering out in the first step ordinary files, passing along to the AppleScript only folders. Saved as an application named tds2sys.app, the main work occurs in the bash script `tds2sys`, which copies the structure of the folder passed to it to texmf-local, updating only files that are newer than existing files. It then runs

```
sudo -H mktexlsr
```

and runs `sudo -H updmap-sys --enable Map` on every map file it finds in the tds folder's

`fonts/map/dvips`

Install a package in `~/Library/texmf` with map files symlinked to texmf-local. The Automator script is almost identical, but calls `tds2home` instead of `tds2sys`, where the major difference is that files are copied to `~/Library/texmf` and the map files are symlinked to texmflocal and processed as in the preceding case. The advantage is that there is no complication of two separate `updmap.cfg` files, but it is still very simple to modify files in the package. (There is of course a slight timing hit because there is no lsR database for file searching in the home texmf tree.) The icons I made for the corresponding Automator applications are: