Tutorial

Converting your scripts to Mac OS X applications is easy with py2app.

Create a setup.py file

The first step is to create a `setup.py` file for your script. `setup.py` is the "project file" that tells setuptools everything it needs to know to build your application. We'll use the py2applet script to do that:

```
$ py2applet --make-setup MyApplication.py
Wrote setup.py
```

If your application has an icon (in `.icns` format) or data files that it requires, you should also specify them as arguments to py2applet.

Clean up your build directories ¶

Before starting development or switching development modes it's usually a good idea to ensure that your `build` and `dist` directories are cleaned out:

```
$ rm -rf build dist
```

Development with alias mode

Alias mode (the `-A` or `--alias` option) instructs py2app to build an application bundle that uses your source and data files in-place. It does not create standalone applications, and the applications built in alias mode are not portable to other machines. This mode is similar to the setuptools `develop` command, or Xcode's zero-link feature.

To build the application in alias mode, execute `setup.py` with the `py2app` command and specify the `-A` option (or `--alias`):

```
$ python setup.py py2app -A
```

After this, py2app will spit out a bunch of messages to your terminal and you'll end up with new `build` and `dist` folders. The `build` folder contains build sludge that you'll never need to touch, and the `dist` folder contains your application bundle. The application bundle will be named after your script; if your script was named `MyApplication.py`, then your application bundle will be named `MyApplication.app`. Note that Finder displays application bundles without the `.app` extension.

You only need to run this command again when you add data files or change options. Changes to your source code won't require rebuilding!

Running your application

During development, it's often useful to have your application attached to the Terminal. This allows you to better debug it, e.g. by inserting `import pdb; pdb.set_trace()` into your code to inspect it interactively at runtime.

To run your application directly from the Terminal:

```
$ ./dist/MyApplication.app/Contents/MacOS/MyApplication
```

To start your application normally with LaunchServices, you can use the `open` tool:

```
$ open -a dist/MyApplication.app
```

If you want to specify "open document" events, to simulate dropping files on your application, just specify them as additional arguments to `open`.

You may of course also double-click your application from Finder.

When run normally, your application's stdout and stderr output will go to the Console logs. To see them, open the Console application:

```
$ open -a Console
```

Building for deployment

After you've got your application working smoothly in alias mode, it's time to start building a redistributable version. Since we're switching from alias mode to normal mode, you should remove your `build` and `dist` folders as above.

Building a redistributable application consists of simply running the `py2app` command:

```
$ python setup.py py2app
```

This will assemble your application as `dist/MyApplication.app`. Since this application is self-contained, you will have to run the `py2app` command again any time you change any source code, data files, options, etc.

The easiest way to wrap your application up for distribution at this point is simply to right-click the application from Finder and choose "Create Archive".