Linux Application Suite (LAS)

Application Developer short paper

by Felix Caffier (broozar@web.de) draft 2015-03-30

Content

1. Application bundle structure

Application bundles are folders containing all files belonging to one user-space application. They (currently) can have the extension **.apx** (GUI app) or **.cap** (CLI app). If you know Mac OSX .app bundles, the idea is similar.

Linux application bundles are structured the following way, while a yellow background indicates optional items:

```
+ appname.apx/.cap (top folder)
-- launch[-[arch]].sh
-- icon.png
-- info.xml
-- appname.cert
-- file.png
-- LICENSE[_LANG]
-- README[_LANG]
-- EULA[_LANG]
-- CREDITS[_LANG]
-+ lib[-[arch]] (sub folder)
```

Required Elements:
appname.apx/.cap - The app can have any name, preferably alphanumeric lowercase with no special characters. The "extension" indicates a GUI application (apx) or a console application (cap).
launch[-[arch]].sh - The shell script that will be executed when you tell the system to launch the app. All apps are launched through a shell script. If no arch is found ("") or you are trying to run a platform-agnostic script, the fallback "launch.sh" will be executed instead.
info.xml - holds the app name in multiple languages, the app description, and additional information about the program, like version number, build, author, etc.
icon.png - application icon

Optional Elements:
file.png - if you want the files associated to the program have a special icon
LICENSE, README, EULA, CREDITS [_LANG] - common text files
appname.cert - the application certificate
lib[-[arch]] - libraries required by the application that do not ship by default with the linux distribution. If the arch-specific folder "lib-*archname*" is not found, appopen will try to use "lib". This folder will be automatically added to LD_LIBRARY_PATH

2. Specifications for individual bundle items

appname.apx/.cap - the root folder of the application
apx indicates a GUI app, cap indicates a terminal/CLI app. They are structured equally, the only difference: the cap bundle launcher opens a terminal first, then executes launch-[arch].sh. This gets rid of the common CLI app steps “open terminal - cd to directory - type script interpreter name and file name to execute”.

launch[-[arch]].sh - the entry point for the application
Main Script that will be executed. It can contain whatever bash code you need. In its most minimal form, simply declare the name of the GUI executable or the interpreter and the name of the main script. The CWD for the script will be set to appname.apx/cap automatically, so no need for “cd” code. The simplest lauch script could look like this:

```
./binary $@
```

where “binary” is the local executable and “$@“ transmits all arguments to the binary.
Since this is just a shell script, it is easy to create dummy applications like a Desktop Environment settings manager or one-click launchers to system-built-in tools like “alsamixer” (see provided test applications).
[arch] is determined by ‘uname -m’ upon launch, so it is possible to pack multiple launch scripts into one apx/cap bundle (and the respective target binaries), which get automatically selected depending on the host platform. If arch is empty or you are trying to run a platform-agnostic script, the fallback “launch.sh” will be executed instead.

icon.png - application icon
A square power-of-two (preferably 128x128px with transparency) PNG image for the application. This will be read by the Desktop Enviroment’s application launcher.

file.png - file icon
A square power-of-two (preferably 128x128px with transparency) PNG image for the application-related files. This will be read by the Desktop Enviroment’s file manager.

LICENSE[_LANG], README[_LANG], EULA[_LANG], CREDITS[_LANG]
Common text files that ship with most programs already. appinfo and appinstall provide a GUI frontend for these files. [_LANG] can be specified if you want different files for different languages. If no file matches the system language, [_LANG] will be ignored. Language codes correspond to ISO 639-1.

appname.cert
Certification file for the application. I am no expert on this, so someone will have to fill the gaps here.

-- TODO --

lib-[arch] - non-system library folder
Every app bundle can ship the libraries it needs, or opt to ship with other versions of the libraries present on the system to avoid incompatibilities. The lib[arch] folder is automatically included into the app launch through LD_LIBRARY_PATH.
[arch] is determined by ‘uname -m’ upon launch, so it is possible to pack multiple libraries into one apx/cap bundle.

3. info.xml

UTF-8 XML file for file info and multilanguage support. It is interpreted by appopen, appinfo and the Desktop Environment program launcher. Language codes are ISO 639-1. Example file:

```
<?xml version="1.0"?>
<app>
	<author>L. Inux</author>
	<website>http://www.linux.org</website>
	<email>support@linux.org</email>
	<version>3.5</version>
	<build>1234</build>
	<date>2014-12-31</date>
	<category>Web;Games;Fun;</category>
	<name>
		<en>My Application</en>
		<de>Mein Programm</de>
		<en>Mi Aplicación</en>
	</name>
	<description>
		<en>A great program.</en>
		<de>Ein tolles Programm.</de>
		<en>Una aplicación muy bien.</en>
	</description>
	<lds>2016</lds>
</app>
```

<?xml: because we like standards
root <app></app>: node name can be anything, will be ignored - but must be present
<author>: main author or corporation
<website>: main website (optional)
<email>: support email (optional)
<version>: program version (optional)
<build>: program build number (optional)
<date>: program build/release date (optional)
<category>: for launcher menus/desktop environments (optional)
<name> in <language>: program name, <en> is required, all other languages optional
<description> in <language>: app description, <en> is required, all other languages optional
<lds>: version number of LDS (optional, see Linux Desktop Standard concept paper)