Linux Application Suite (LAS)

a design proposal

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

Content

1. What and Why

"We made a great 3d game editor. It runs on Windows (exe + installer), Mac (.app + dmg), and Linux (binary+resources + ... oh wait. We have a problem - we do not know which distribution our target audience runs, we do not know which libraries this distro has installed by default, and we are not familiar with every nuance between the different distros, packet formats, file locations and permission sets. This makes it quite cumbersome to distribute our (otherwise ready & working) application to linux users.
Wouldn't it be great if there were an application bundle format that all distros would accept? And while we are at it, solve other problems like multiplatform (universal) applications, file system clutter, easy (un)install, multilanguage support etc. as well?"

Application Bundles do not reinvent the wheel, nor do they aim to replace the existing package managers (which work well for system libraries, kernel updates, system core binaries etc), but rather supplement them. The bundle system I propose is exclusively for common user space applications, email clients, browsers, video editors, games etc. that neither need root access nor need to be installed into a specific location in the file system to function. Prime examples for this would be applications like Open Office, Firefox, Inkscape, your favourite audio editor, Gimp and Blender - in short, applications that have been endlessly and needlessly repackaged for every linux distro the world over without any particular benefit.

Goals & benefits

The new system sets out to have the following benefits:

- clear separation of the “system/root/admin” tools and applications from the “user applications”
- simplified app development and distribution, elimination of distro-specific re-packaging: offering e.g. “firefox” on linux should not be the task of the distro maintainers
- file system de-cluttering (all app-related files are bundled in 2 places: the application’s own directory and a hidden settings/savings/… folder in the user home directory)
- easy & user-friendly (de-)installation of user applications
- standardized (relative) locations for icons, certificates, app-specific libraries
- (optional) shipping of critical application libraries inside the application itself instead of relying on system libraries
- easy multilanguage application names and description texts
- universal application bundles with multiple architecture support through internal [arch] naming scheme
- automatically launch console-based applications in a console window

2. App 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

3. 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.

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 used after ISO 639-1.
Example file:

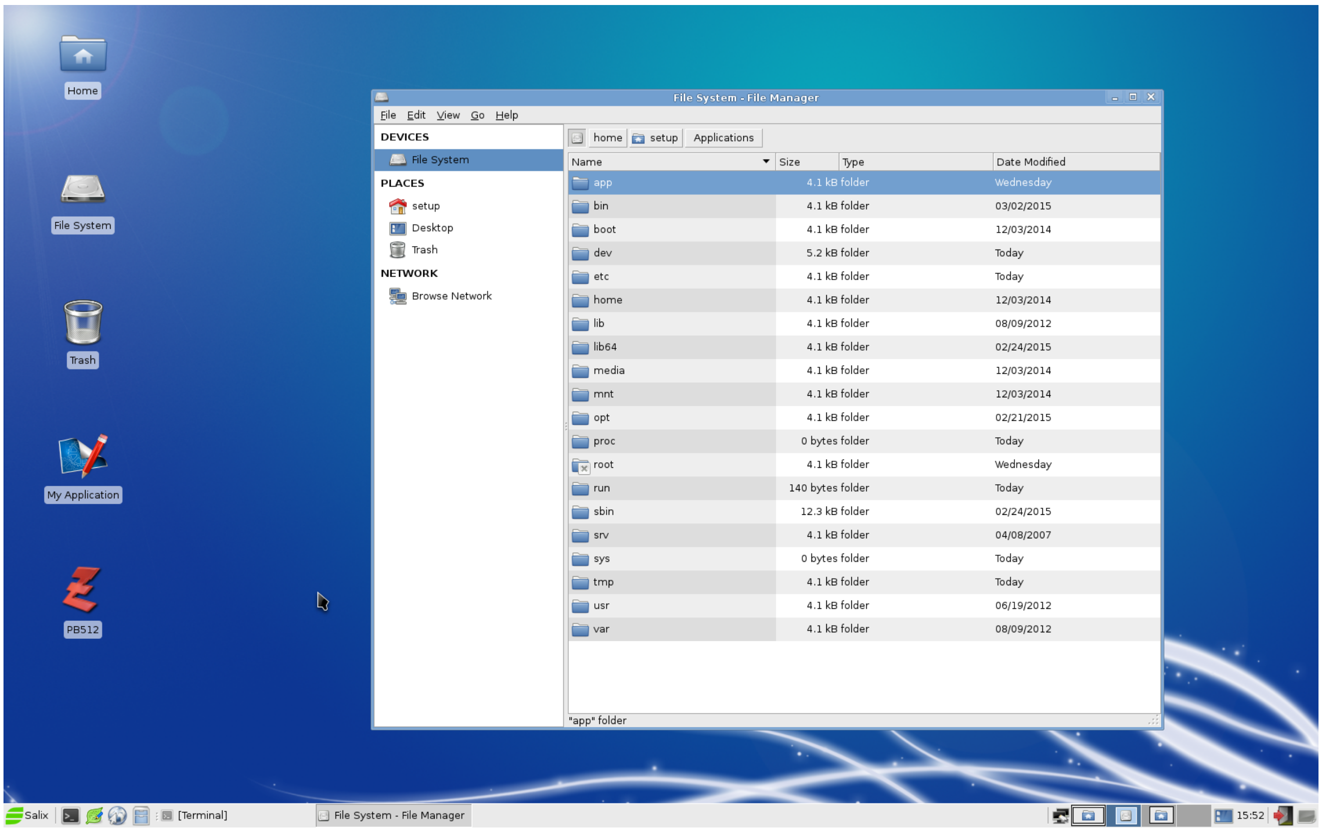
```
<?xml version="1.0"?>
<app>
	<author>L. Inux</author>
	<website>http://www.linux.org</website>
	<email>support@linux.org</email>
	<version>3.5</version>
	<build>1234</build>
	<date>2014-12-31</date>
	<category>Web;Games;Fun;</category>
	<name>
		<en>My Application</en>
		<de>Mein Programm</de>
		<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)

4. proposed file system structure / DE additions

/app/ - storage of all cap/apx folders

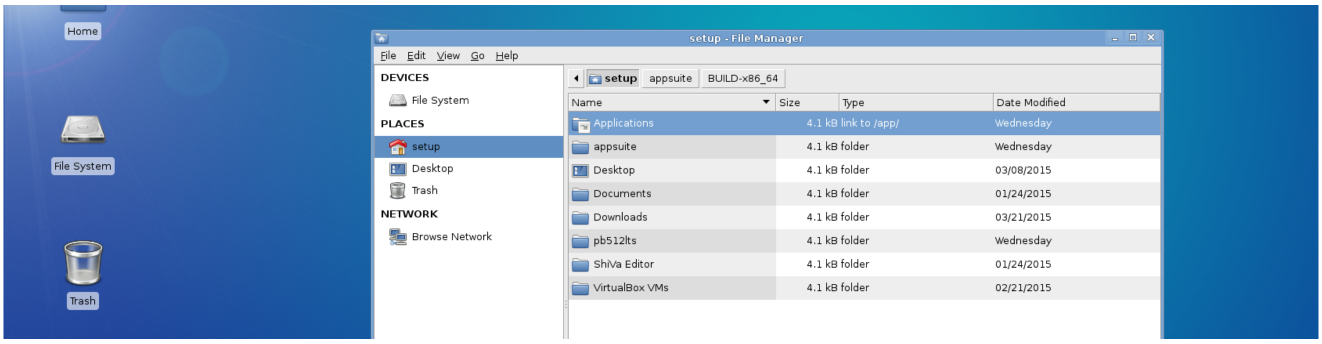
All (cap/apx user) applications should be stored in a central location. I propose a new directory called /app in order to not mess up any established directory structures. Applications can be installed into this directory through appinstall (see below).



Name	Size	Type	Date Modified
app	4.1 kB	folder	Wednesday
bin	4.1 kB	folder	03/02/2015
boot	4.1 kB	folder	12/03/2014
dev	5.2 kB	folder	Today
etc	4.1 kB	folder	Today
home	4.1 kB	folder	12/03/2014
lib	4.1 kB	folder	08/09/2012
lib64	4.1 kB	folder	02/24/2015
media	4.1 kB	folder	12/03/2014
mnt	4.1 kB	folder	12/03/2014
opt	4.1 kB	folder	02/21/2015
proc	0 bytes	folder	Today
root	4.1 kB	folder	Wednesday
run	140 bytes	folder	Today
sbin	12.3 kB	folder	02/24/2015
srv	4.1 kB	folder	04/08/2007
sys	0 bytes	folder	Today
tmp	4.1 kB	folder	Today
usr	4.1 kB	folder	06/19/2012
var	4.1 kB	folder	08/09/2012

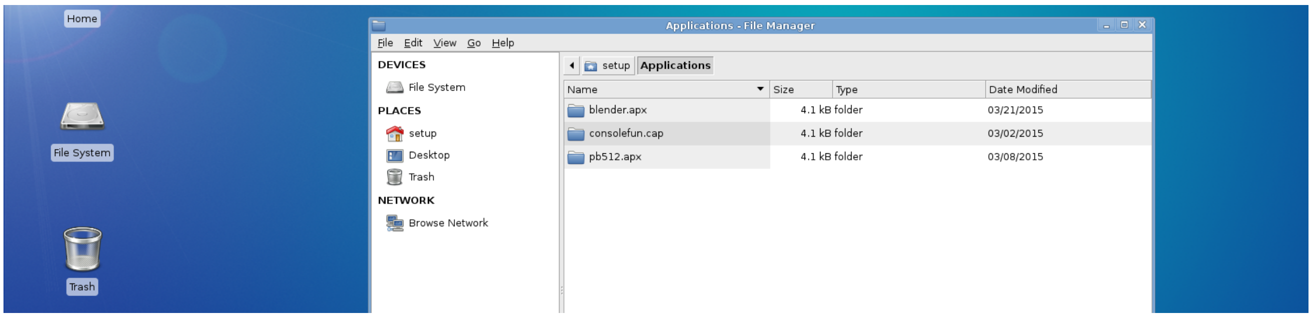
You can start a cap/apx bundle from anywhere though, you just have to supply an absolute path to "appopen". Similarly, "appinfo" can analyze any bundle as long as you provide an absolute path. In contrast, the current "appinstall" tool already assumes /app/ is there.

I would suggest that this directory is linked to in the user's /home in his language:



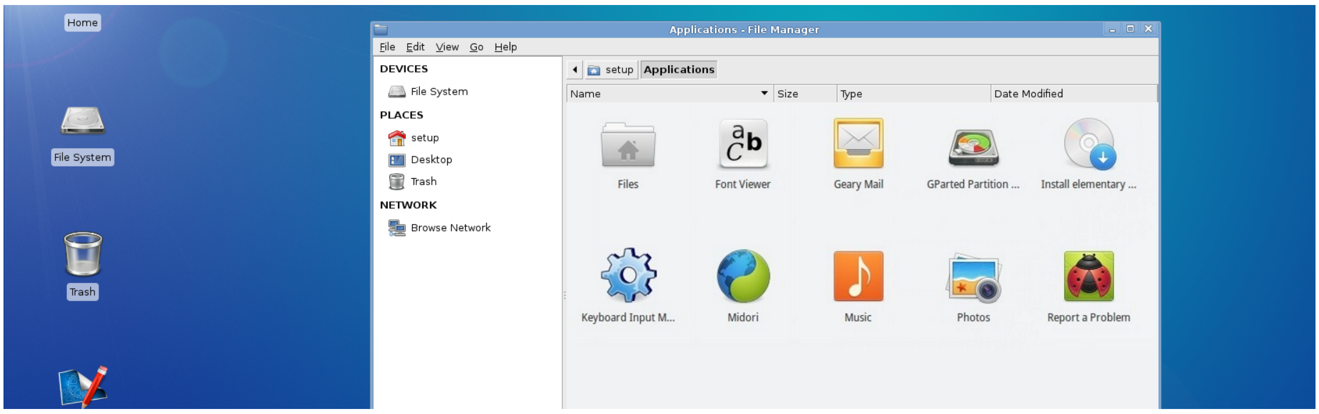
Name	Size	Type	Date Modified
Applications	4.1 kB	link to /app/	Wednesday
appsuite	4.1 kB	folder	Wednesday
Desktop	4.1 kB	folder	03/08/2015
Documents	4.1 kB	folder	01/24/2015
Downloads	4.1 kB	folder	03/21/2015
pb512lts	4.1 kB	folder	Wednesday
ShiVa Editor	4.1 kB	folder	01/24/2015
VirtualBox VMs	4.1 kB	folder	02/21/2015

Right now, the contents of /app/ looks like this:



Name	Size	Type	Date Modified
blender.apx	4.1 kB	folder	03/21/2015
consolefun.cap	4.1 kB	folder	03/02/2015
pb512.apx	4.1 kB	folder	03/08/2015

… but with a little tweaking of the standard file managers, it should look like this (mockup):

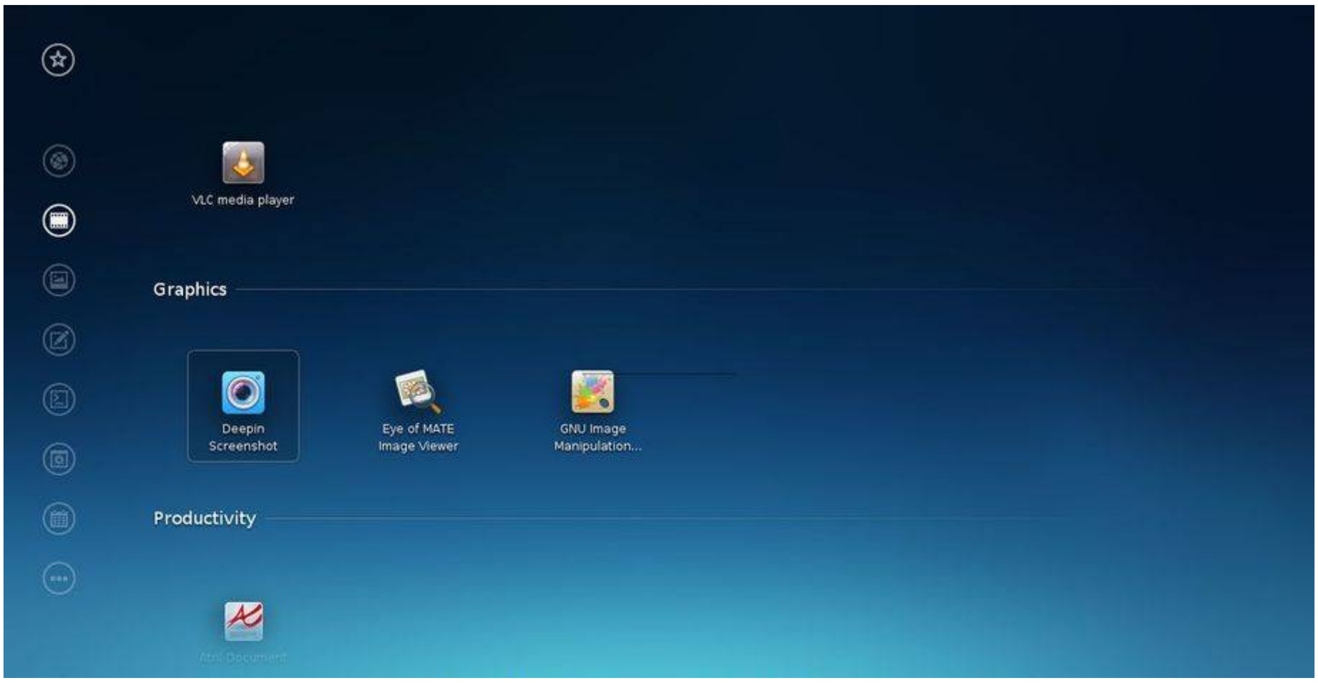


The file manager only has to read the info.xml and icon.png to determine the bundle name and app description (both in the correct language) and replace the default folder icon with the bundle icon.

Double-clicking on those icons should launch “appopen /app/bundelname.apx” A context menu should pop up on right-click and contain the following entries:

- Show Contents (opens the bundle folder in the file manager)
- Get Info (launches “appinfo /app/bundlename.apx”)
- Create Desktop Shortcut (launches “appdesktopshortcut /app/bundlename.apx”)
- Remove (launches some variant of gksudo with an “rm” command)

Since all user applications are in one folder now, it should be very easy to have a launchpad-like application launcher. Deepin Linux has something very pretty like it already, complete with categories and right-click menu:



user "appmin" and group "apps"
I furthermore propose that /app/ should be owned by a new user called "appmin" (a nonsense word made up from app + admin) and belong to a group called "apps". Ordinary users should be members of the group "apps" if you intend for them to be able to execute application bundles. In short:

```
$> ls -l /app
drwxr-x--x   {n}   appmin   apps   4096   {date}   {bundlename}
```

It is the job of "appinstall" to put the bundle in the right place and assign these permissions (currently 751, maybe 750 later?) to all the files in the bundle recursively. The current build of "appinstall" assumes these user/group names already.

5. LAS tools

… first, of course, the application packs themselves:
/app/appname.apx(or .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".

… and now the tools:
/usr/bin/appopen
Opens .cap and .apx bundles
Usage: appopen /path/to/bundle.cap(or .apx) other parameters here

/usr/bin/appinfo
Reads info.xml, README, LICENSE etc. files from the app bundle; depends on xdg-open
Usage: appopen /path/to/bundle.cap(or .apx)

/usr/bin/appdesktopshortcut
Creates an appname.desktop file of a given .cap/.apx on /home/username/Desktop/
Usage: appdesktopshortcut /path/to/bundle.cap(or .apx)

/usr/bin/messager
General purpose messaging window, can be used to display GUI messages from shell scripts.
Usage: messager "title" "the message you want to display\ncan contain line breaks" ["ync"]
- the last parameter can be: "ync" (yes/no/cancel), "yn" (yes/no), "" (ok)
-> yes returns 1, no returns 2, ok/cancel return 0

/usr/bin/st
Simple terminal by http://st.suckless.org/, included as lightweight and DE-independent solution for CLI-based applications.

/usr/bin/libmakegeneric
Creates generic name links for .so files.
Usage: libmakegeneric /path/to/libfolder
- logs verbose output, like: /lib64/liblzma.so.5.0.5
/lib64/liblzma.so.5.0 (-> /lib64/liblzma.so.5.0.5)
/lib64/liblzma.so.5 (-> /lib64/liblzma.so.5.0.5)
/lib64/liblzma.so (-> /lib64/liblzma.so.5.0.5)

/usr/bin/appcrashreporter
currently a dummy app, gets invoked when an app started by appopen exits with exitcode > 0

/usr/bin/appinstall
Installs app bundles into /app from .tar.gz archives. Must be launched as ROOT or APPMIN. Depends on xdg-open. Installer packs must be named *appbundle.ext*.tar.gz and contain a root folder named *appbundle.ext,* please see “testinstallpacks” for some samples.

/usr/bin/applicense
Verifies the application with its .cert certificate. Since I have no idea how such a cert would look like, nor how you could verify it, this application does not exist yet. Drop me a note if you have any suggestions!

-- TODO --

6. sample applications

The sample applications on my website each demonstrate a specific app type.

alsamixer.cap
Essentially a shortcut/launcher for the (hopefully) preinstalled application “alsamixer”. The bundle does not contain an executable binary of its own.

consolefun.cap
A console application with its own binary. Since it is 64bit only, there is only one launcher named launch-x86_64.sh
Try to invoke the crash reporter by exiting the application with CTRL+C.

testapp.apx
A graphical program that logs console application with its own binary. The application is x86_64 and i686, hence the two launch scripts. The program logs every parameter you give to it, for instance
“appopen /path/to/testapp.apx param1 param2” will echo “param1” and “param2”.

testapp.apx.tar.gz
The corresponding install pack for testapp.apx, run “appinstall /path/to/testapp.apx.tar.gz”.

7. Connection to LDS and the future of LAS

LAS can only work properly if there is a base of guaranteed libs, tools and names that a bundle application can rely on, since bundles do not have the benefit of dependency checking as system apps do via the distro-specific packet managers. In order to avoid making developers ship lots of libraries in their bundles, I thought of establishing the LDS as a new standard. Please read the LDS paper to learn more.

4 main challenges lie in the future of LAS:

1. .cert bundle certification (suggestions please!)
2. bug reporter (currently only a dummy application)
3. transitioning the codebase from PureBasic to C/C++
4. general adoption of LAS and LDS

1-3 are challenges a lone programmer can solve. 4 is a political decision involving many as well as a question of whether or not distro makers, DE developers and 3rd party application developers are interested in a more user-friendly, cross-distro compatible solution.