AppBundler.jl

AppBundler.jl

Bundle your Julia GUI Application

Dr. Janis Erdmanis

GitHub: JanisErdmanis, janiserdmanis.org

Aspirational Deployment

```
import Pkg; Pkg.instantiate()
include("main.jl")
```

Why Bundling?

Advantages over web application

- Performance (CPU optimal code, GPU, Cluster, local resources)
- Hardware Interfacing (drivers, systems, measurement apparatus)
- Offline functionality
- Security and Privacy (Secrets and sensitive data do not leave the device)
- Ergonomics (Opportunity using Julia, QML over web technologies)

Challenges

In deploying local apps

- Development Complexity

  Does the app work the same on Linux, MacOS and Windows? Maintaining separate package formats and build infrastructure for each OS

- Distribution and Deployment

  Obtaining and maintaining signing certificates, navigating different marketplace submissions and review processes

- User Adoption Barriers

  Installation is required and can leave traces in the system when uninstalled. Does the software contain malware?

- Maintenance

  Addressing OS-specific bugs and issues; Version fragmentation among users

Solutions

To deployment complexity

- Julia Artifacts system with BinaryBuilder
- Distribute over well-established formats that offer application-level sandboxing and remove local storage after removal
- Use **AppBundler** to vendor package dependencies and artifacts and provide post-processing scripts and steps
- Use GitHub Actions or other CI integration pipelines to finalise created bundles. Alternatively, send the bundles over SSH to self-hosted systems for finalisation.

Compilation

PkgImages or SysImages?

- GUI frameworks have a long TTFX
- Since Julia 1.9 pkgimages are cached, and with Julia 1.11 are relocatable
- SysImages offers instant startup time; PkgImages can take a second to load
- PkgImages are more tested, and the compilation process is more transparent and faster
- Neither method currently offers cross-compilation support; it must be combined with finalising the bundles on the host systems

Let’s make bundles

MacOS Bundle Finalization

What’s in make-dmg script

- Precompilation:

  MyApp/Contents/MacOS/precompile

- Launcher formation:

  gcc -arch arm64 -o "Contents/MacOS/MyApp" "Contents/Resources/launcher.c"

- Codesigning:

  codesign --entitlements "MyApp.app/Contents/Resources/Entitlements.plist"
  --force --sign "JanisErdmanis" --deep "MyApp.app"

- Formation of DMG with a neat installer:

  dmgbuild -s "MyApp.app/Contents/Resources/dmg_settings.py"
  -D app="MyApp.app" "MyApp Installer" "MyApp.dmg"

Snap bundles

Finalization & Installation

- Snap bundles bundles can be installed with:

```
snap install -classic -dangerous myapp.snap
```

- Configure hook runs precompilation after installation
- Alternativelly precompilation can be done:

```
unsquashfs myapp.snap
squashfs-root/bin/precompile
mksquashfs squashfs-root myapp-comp.snap -noappend -comp xz
```

Windows MSIX

make-msix

- Precompiling with MyApp/precompile.ps1
- Changing subsystem with editbin for lld.exe and julia.exe

```
editbin /SUBSYSTEM:WINDOWS "MyApp\julia\bin\julia.exe"
```

- Forming an archive

```
makeappx pack /d "MyApp" /p "MyApp.msix"
```

- Signing the result

```
signtool sign /fd SHA256 /a /f "SigningKey.pfx" "MyApp.msix"
```

Tips

- Use PrecompileTools to precompile the startup of the application
- RelocatableFolders can be useful for QML files
- Use Add-AppPackage -register .\MyApp\AppxManifest.xml for debugging Windows bundles
- Use snap try MyApp and snap run --shell MyApp to debug snap bundles

Customization

The Recipe System

- Every recipe is made of list of rules executed sequentually
- A rule specifies files that need to be moved from origin to destantion
- If a destination already contains a file written by previous rule it is skipped
- If a recipe path exists in app folder it overrides the default from AppBundler/recipes

Recipe System Demo

Sandboxing

User Data

- User data is set to USER_DATA environment variable
  - MacOS: ~/.config/{{APP_NAME}} ~/Library/Containers/{{APP_ID}}/Data
  - Linux: ~/snap/{{APP_NAME}}/common
  - Windows: ~\AppData\Local\Packages\{{APP_ID}}\LocalState
- Additionally a $USER_DATA/cache is set as DEPOT_PATH first entry

Future Work

Sandboxing

- Application marketplaces expect applications to use the least number of system resources for favourable reviews.
- Currently, none of the recipes works:
  - MacOS: application loads but is unresponsive to input;
  - Linux: OpenGL does not work with QML and Gtk but works with GLFW;
  - Windows: Julia does not load; some progress had been made recently issue #52007;

Future Work

Some other things

- Writing a GitHub action that automatically bundles applications when a new app version is tagged;
- Making a flatpack recipe;
- Adding a PackageCompiler support for postprocessing the application bundles.

The End