NHDaly / **ApplicationBuilder.jl**

Compile, bundle, and release julia software

178 commits | **12** branches | **8** releases | **3** contributors | MIT

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find File** | **Clone or download**

NHDaly Merge pull request #55 from NHDaly/get_resources_dir ... Latest commit aa834c7 on Apr 29

examples	fix Blink test typo: ApplicationBuilderUtils -> ApplicationBuilderApp…	3 months ago
src	Switch examples to use separate ApplicationBuilderAppUtils.jl package…	3 months ago
test	Unify the code bases into ApplicationBuilder.jl! 🙌	4 months ago
.gitignore	add Coverage files to gitignore	last year
.travis.yml	Travis: Safelist `master` builds & disable building all branch pushes	4 months ago
LICENSE.txt	Add MIT License, copyright 2018 Nathan Daly.	last year
Manifest.toml	Update ApplicationBuilderAppUtils after changing version num down to …	3 months ago
Project.toml	Move ApplicationBuilderAppUtils to [deps] instead of [test]	3 months ago
README.md	Add Appveyor build status to README	4 months ago
REQUIRE	Bump minimum julia version to julia 0.7!	9 months ago
appveyor.yml	Remove julia 0.7 from appveyor	4 months ago
build_app.jl	Upgrade to support Julia v0.7/v1.0.	last year

README.md

Julia Application Builder

build passing | build passing | coverage 62%

Turn your julia program into a standalone, distributable, statically-compiled "App"!

ApplicationBuilder compiles a julia program and bundles it up into a distributable application, on macOS, Windows and Linux! After building, your users can download your application and run it without having Julia installed.

ApplicationBuilder

To compile and bundle your julia program into a distributable app, use `ApplicationBuilder.build_app_bundle`:

```
julia> using ApplicationBuilder
help?> build_app_bundle()
  # 1 method for generic function "build_app_bundle":
  build_app_bundle(juliaprog_main; appname, builddir, resources, libraries, verbose, bundle_identifier, ap
```

Usage

To build a julia program into an application, you'll need to do two steps:

1. Wrap your code in `julia_main` function. If the main entry-point to your code is a function, `my_code()`, it would look like this:

```
# my_julia_main.jl
include("my_code.jl")
Base.@ccallable function julia_main(ARGS::Vector{String})::Cint
    return my_code()
```

```
end
```

The easiest thing to do is to maintain this as a separate script, but you can put this anywhere in your project.

2. Call `build_app_bundle` with the file that provides `julia_main`. The easiest way to do this is to maintain a build script, e.g.:

```
# build.jl
using ApplicationBuilder
build_app_bundle("src/my_julia_main.jl", appname="MyCode");
```

Compatibility

`ApplicationBuilder` supports macOS, Windows, and Linux. Currently, ApplicationBuilder doesn't do cross-compilation, so to compile for Windows, you need to run it from a Windows machine, etc.

Running an example:

After cloning the repository, you can build an App out of the example program, `examples/hello.jl`, like this:

```
julia> build_app_bundle("$(homedir())/.julia/v0.6/ApplicationBuilder/examples/hello.jl", appname="HelloWor
```

or like this:

```
$ julia build_app.jl -v examples/hello.jl "HelloWorld"
```

This will produce `builddir/HelloWorld.app`, which you can double click, and it will indeed greet you!

The simple example HelloWorld.app has no binary dependencies -- that is, it doesn't need any extra libraries besides Julia. Many Julia packages come bundled with their own binary dependencies, and if you want to use them in your app, you'll have to add those dependencies via the `libraries` (`-L`) option for libs and `resources` (`-R`) for bundle resources.

More examples

There are many more examples in the `examples` directory, each of which have a corresponding *build file* in the `test/build_examples` directory. You can build an example simply by running the build file:

```
julia> include("$(homedir())/.julia/v0.6/ApplicationBuilder/test/build_examples/commandline_hello.jl")
```

build_app.jl (The command-line tool)

There is also a command-line interface, through `build_app.jl`, if you prefer it. The main development is on the Julia API, though, so this sometimes lags behind. Feel free to send a PR if it's missing anything! :)

Run `julia build_app.jl -h` for help:

```
usage: build_app.jl [-v] [-R <resource>] [-L <file>] [--icns <file>]
                    [-h] juliaprog_main [appname] [builddir]

positional arguments:
  juliaprog_main        Julia program to compile -- must define
                        julia_main()
  appname               name to call the generated .app bundle
  builddir              directory used for building, either absolute
                        or relative to the Julia program directory
                        (default: "builddir")

optional arguments:
  -v, --verbose         increase verbosity
  -R, --resource <resource>
                        specify files or directories to be copied to
                        MyApp.app/Contents/Resources/. This should be
                        done for all resources that your app will need
```

```
                        to have available at runtime. Can be repeated.
    -L, --lib <file>    specify user library files to be copied to
                        MyApp.app/Contents/Libraries/. This should be
                        done for all libraries that your app will need
                        to reference at runtime. Can be repeated.
    --icns <file>       .icns file to be used as the app's icon
    -h, --help          show this help message and exit

    examples:
       # Build HelloApp.app from hello.jl
       build_app.jl hello.jl HelloApp
       # Build MyGame, and copy in imgs/, mus.wav and all files in libs/
       build_app.jl -R imgs -R mus.wav -L lib/* main.jl MyGame
```

License

This project is licensed under the terms of the MIT license.

Thanks

Thanks for the help from these contributors and everyone else!:

- ranjanan
- lucatrv
- simondanish
- vtjnash