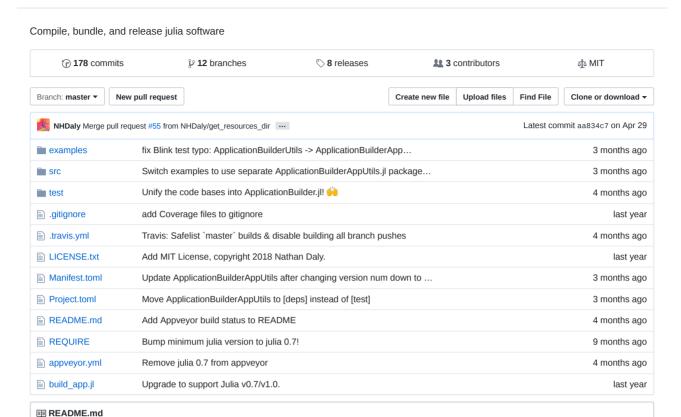
#### NHDaly / ApplicationBuilder.jl



# Julia Application Builder

```
build passing obuild 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**

 $To \ compile \ and \ bundle \ your \ julia \ program \ into \ a \ distributable \ app, \ use \ Application Builder.build\_app\_bundle:$ 

```
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()
```

1 of 3 8/5/19, 5:46 PM

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 <code>build\_app.jl</code>, 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
```

2 of 3 8/5/19, 5:46 PM

```
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
- lucatry
- simondanish
- vtjnash

3 of 3 8/5/19, 5:46 PM