Building Julia Apps and So Can You!

Compiling and distributing desktop applications written in Julia

Nathan Daly
nhdaly@gmail.com
http://nhdaly.github.io (http://nhdaly.github.io)

Overview

- Definitions
- Demo
- Building a simple command line Application
- Building an Application with a GUI (dealing with binary dependencies)

What is an Application?

Julia 0.7 Pkg docs (https://docs.julialang.org/en/latest/stdlib/Pkg/#Glossary-1):

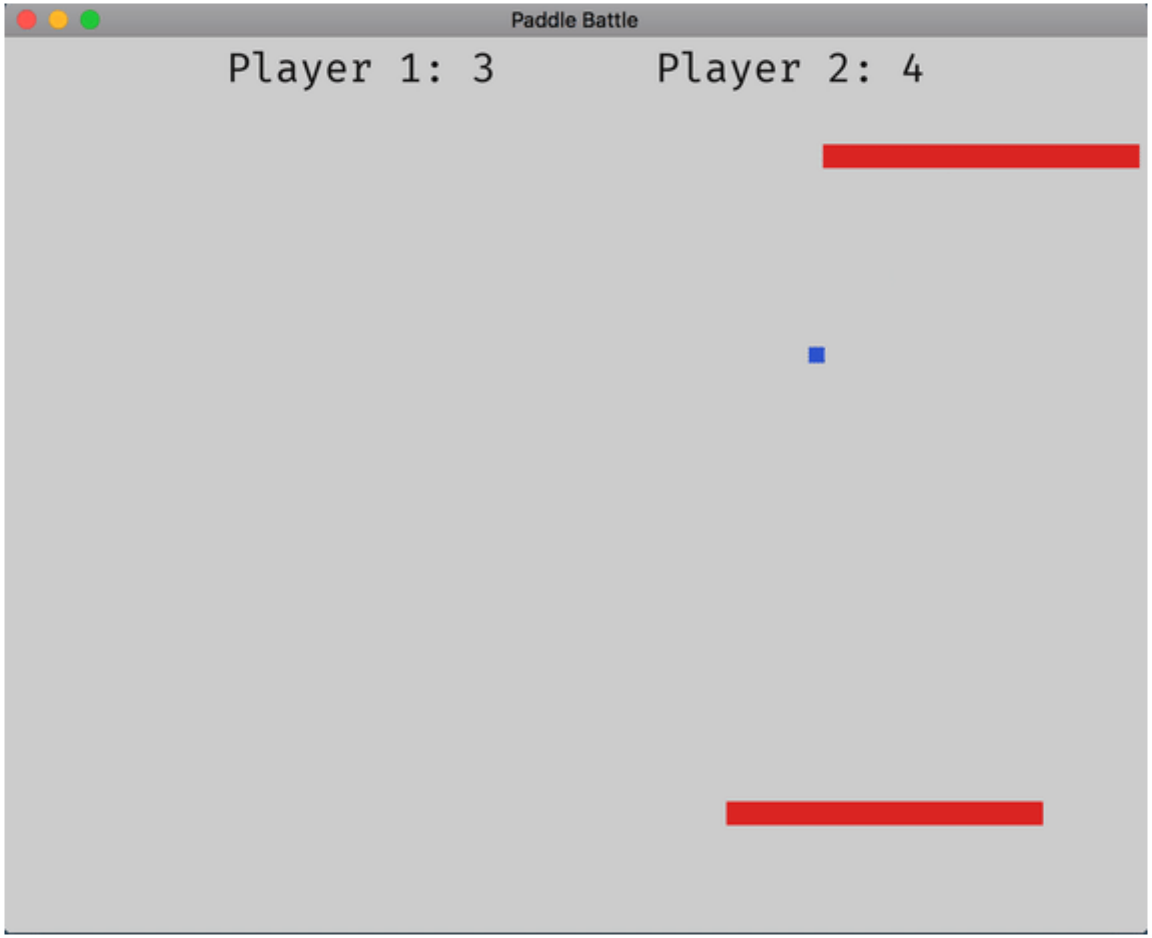
> *Application: a project which provides standalone functionality not intended to be reused by other Julia projects. For example a web application or a commmand-line utility, or simulation/analytics code accompanying a scientific paper.*

What am I talking about when I say "Application"?

- Distributable
- Self-contained (no Julia installation required)
- Compiled binary

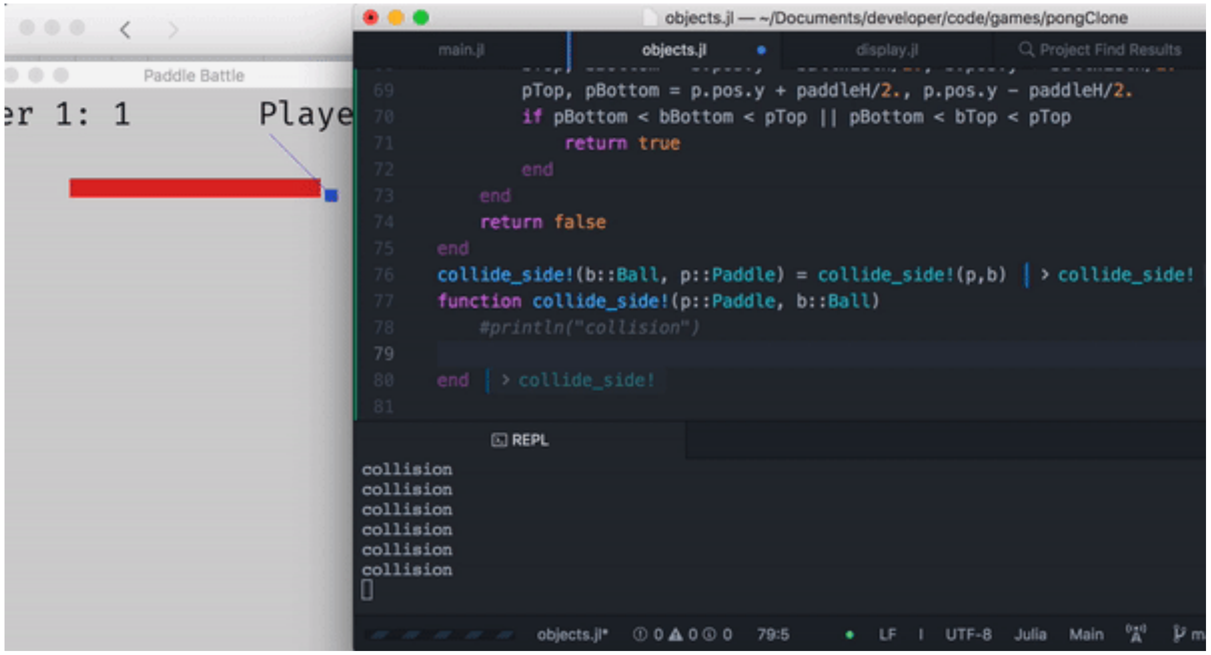
Maybe a better name would be "Application Bundle" or "App".

Paddle Battle



An example Application: Paddle Battle is a simple Pong-style game written entirely in Julia using a C graphics library called SDL.
https://github.com/NHDaly/PaddleBattleJL (https://github.com/NHDaly/PaddleBattleJL)

Building a game in Julia was fun! Look at me live-editing code in Juno:



```
        pTop, pBottom = p.pos.y + paddleH/2., p.pos.y - paddleH/2.
        if pBottom < bBottom < pTop || pBottom < bTop < pTop
            return true
        end
    end
    return false
end
collide_side!(b::Ball, p::Paddle) = collide_side!(p,b)  > collide_side!
function collide_side!(p::Paddle, b::Ball)
    #println("collision")

end  > collide_side!
```

```
collision
collision
collision
collision
collision
collision
```

ApplicationBuilder

In [2]:
```
using ApplicationBuilder; using BuildApp  # BuildApp is a weird "inner module"
```

```
WARNING: Your Julia system image is not compiled natively for this CPU architectur
e.
        Please run `PackageCompiler.force_native_image!()` for optimal Julia perfor
mance
```

In [3]:
```
# Setup
try rm("playground", recursive=true); end
mkpath("playground"); cd("playground");
```

Running a Simple Example

```
In [4]: run(`ls $(joinpath(Pkg.dir("ApplicationBuilder"), "examples"))`)
```

```
blink.jl
commandline_hello.jl
hello.jl
libui.jl
```

```
In [5]: BuildApp.build_app_bundle(joinpath(Pkg.dir("ApplicationBuilder"),"examples","hello.jl"))

          Using calculated bundle_identifier: 'com.daly.hello'
        ~~~~~~ Creating mac app in "/Users/daly/Documents/developer/talks/jupyter/playground/builddir/hello.app" ~~
        ~~~~~
        ~~~~~~ Compiling a binary from '/Users/daly/.julia/v0.6/ApplicationBuilder/examples/hello.jl'... ~~~~~~~
        Julia program file:
          "/Users/daly/.julia/v0.6/ApplicationBuilder/examples/hello.jl"
        C program file:
          "/Users/daly/.julia/v0.6/ApplicationBuilder/src/program.c"
        Build directory:
          "/Users/daly/Documents/developer/talks/jupyter/playground/builddir/hello.app/Contents/MacOS"
        All done
        ~~~~~~ Generating 'Info.plist' for 'com.daly.hello'... ~~~~~~~

Out[5]:  0

        ~~~~~~ Cleaning up temporary files... ~~~~~~~
        ~~~~~~ Signing the binary and all libraries ~~~~~~~
        ~~~~~~ Done building '/Users/daly/Documents/developer/talks/jupyter/playground/builddir/hello.app'! ~~~~~~~
```

```
In [6]: run(`open .`)
```

Anatomy of a Julia App

In [7]:
```
using PrintFileTree
```

In [8]:
```
printfiletree("builddir/hello.app")
```

```
builddir/hello.app
 └── Contents
     ├── Info.plist
     ├── MacOS
     │   ├── hello
     │   ├── hello.dylib
     │   ├── libLLVM.dylib
     │   ├── libamd.dylib
     │   ├── libarpack.2.dylib
     │   ├── libarpack.dylib
     │   ├── libcamd.dylib
     │   ├── libccalltest.dylib
     │   ├── libccolamd.dylib
     │   ├── libcholmod.dylib
     │   ├── libcolamd.dylib
     │   ├── libcurl.4.dylib
     │   ├── libcurl.dylib
     │   ├── libdSFMT.dylib
     │   ├── libfftw3.3.dylib
     │   ├── libfftw3.dylib
     │   ├── libfftw3_threads.3.dyli
b
     │   ├── libfftw3_threads.dylib
     │   ├── libfftw3f.3.dylib
     │   ├── libfftw3f.dylib
     │   ├── libfftw3f_threads.3.dyl
ib
     │   ├── libfftw3f_threads.dylib
     │   ├── libgcc_s.1.dylib
     │   ├── libgfortran.4.dylib
     │   ├── libgit2.0.25.1.dylib
     │   ├── libgit2.25.dylib
     │   ├── libgit2.dylib
     │   ├── libgmp.10.dylib
     │   ├── libgmp.dylib
     │   ├── libjulia.0.6.4.dylib
     │   ├── libjulia.0.6.dylib
     │   ├── libjulia.dylib
     │   ├── libmbedcrypto.0.dylib
     │   ├── libmbedcrypto.2.3.0.dyl
ib
     │   ├── libmbedcrypto.dylib
     │   ├── libmbedtls.10.dylib
     │   ├── libmbedtls.2.3.0.dylib
     │   ├── libmbedtls.dylib
     │   ├── libmbedx509.0.dylib
     │   ├── libmbedx509.2.3.0.dylib
     │   ├── libmbedx509.dylib
     │   ├── libmpfr.4.dylib
     │   ├── libmpfr.dylib
     │   ├── libopenblas64_.dylib
     │   ├── libopenlibm.2.3.dylib
     │   ├── libopenlibm.2.dylib
```

```
    │    ├─── libopenlibm.dylib
    │    ├─── libopenspecfun.1.3.dyli
b
    │    ├─── libopenspecfun.1.dylib
    │    ├─── libopenspecfun.dylib
    │    ├─── libpcre2-8.0.dylib
    │    ├─── libpcre2-8.dylib
    │    ├─── libpcre2-posix.1.dylib
    │    ├─── libpcre2-posix.dylib
    │    ├─── libquadmath.0.dylib
    │    ├─── libspqr.dylib
    │    ├─── libssh2.1.0.1.dylib
    │    ├─── libssh2.1.dylib
    │    ├─── libssh2.dylib
    │    ├─── libsuitesparse_wrapper.
dylib
    │    ├─── libsuitesparseconfig.dy
lib
    │    └─── libumfpack.dylib
    └─── Resources
         └─── hello.icns
```

`hello.app` Table of Contents (Mac)

- `Contents/Info.plist`: Mac-specific metadata for the application.
- `Contents/MacOS/hello`: Our compiled executable. It runs our static julia code which is compiled into `hello.dylib`
- `Contents/MacOS/hello.dylib` (`.dll` on windows): Our compiled julia code, saved as a shared library. All the julia code we write ends up here, and gets loaded by the `hello` executable.
- `Contents/MacOS/[One million julia libraries]`: This is the "julia runtime", which your code needs in order to run an a computer that doesn't have julia installed.

Ship it!!

And that's it! `hello.app` is a real, complete application that can be distributed to real users, either by downloading from a website, or getting it from an App Store.

Of course, it doesn't *do* anything interesting, so let's fix that!

Making our own App

```
In [9]:  mkpath("OurApp"); cd("OurApp")
         run(`pwd`)

         /Users/daly/Documents/developer/talks/jupyter/playground/O
         urApp
```

```
In [10]: mkdir("src")  # For our App's source code
```

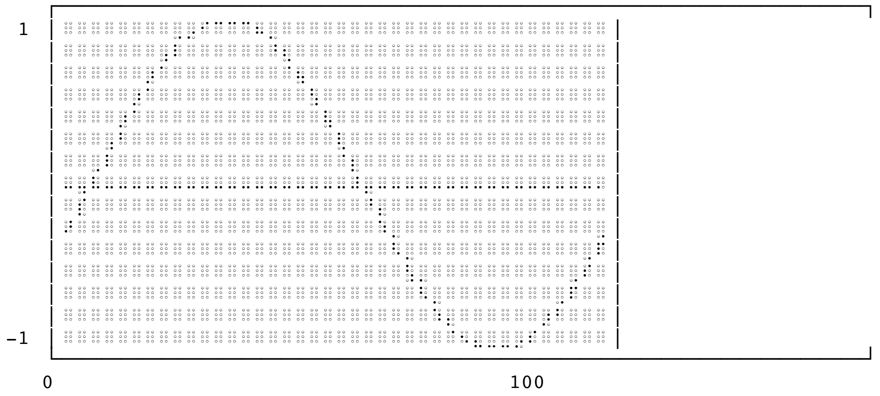
```
In [11]: write("src/app.jl",
          """
             println("**** Hello From Julia! ****")
          """
         )

Out[11]: 42
```

```
In [12]: include("src/app.jl")

         **** Hello From Julia! ****
```

```
In [13]: write("src/app.jl",
          """
             using UnicodePlots

             println("**** Hello From Julia! ****")
             r = rand(0:2π)
             println(lineplot(1:100, sin.(linspace(r, r+2π, 100))))

          """
         )

Out[13]: 147
```

```
In [14]: BuildApp.build_app_bundle("src/app.jl")
```

```
  Using calculated bundle_identifier: 'com.daly.app'
~~~~~~ Creating mac app in "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/app.app" ~~~~~~~
~~~~~~ Compiling a binary from 'src/app.jl'... ~~~~~~~
Julia program file:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/src/app.jl"
C program file:
  "/Users/daly/.julia/v0.6/ApplicationBuilder/src/program.c"
Build directory:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/app.app/Contents/MacOS"
**** Hello From Julia! ****
```



```
failed process: Process(`cc '-DJULIAC_PROGRAM_LIBNAME="app.dylib"' -o /Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddi
r/app.app/Contents/MacOS/app /Users/daly/.julia/v0.6/ApplicationBuilder/src/program.c /Users/daly/Documents/developer/talks/jupyter/playgroun
d/OurApp/builddir/app.app/Contents/MacOS/app.dylib -m64 -std=gnu99 -I/Applications/Julia-0.6.app/Contents/Resources/julia/include/julia -DJUL
IA_ENABLE_THREADING=1 -fPIC -O3 -mmacosx-version-min=10.10 -L/Applications/Julia-0.6.app/Contents/Resources/julia/lib -Wl,-rpath,/Application
s/Julia-0.6.app/Contents/Resources/julia/lib -Wl,-rpath,/Applications/Julia-0.6.app/Contents/Resources/julia/lib/julia -ljulia -Wl,-rpath,@ex
ecutable_path`, ProcessExited(1)) [1]

Stacktrace:
 [1] pipeline_error(::Base.Process) at ./process.jl:682
 [2] run(::Cmd) at ./process.jl:651
 [3] build_executable(::String, ::String, ::String, ::Bool, ::String, ::String, ::String, ::String) at /Users/daly/.julia/v0.6/PackageCompile
r/src/static_julia.jl:269
 [4] #static_julia#1(::String, ::Bool, ::Bool, ::String, ::String, ::Bool, ::Bool, ::Bool, ::Bool, ::Bool, ::Bool, ::Bool, ::Void, ::Void, ::
Void, ::Void, ::Void, ::Void, ::Void, ::String, ::String, ::String, ::Void, ::Void, ::Void, ::Void, ::Void, ::String, ::PackageCompiler.#stat
ic_julia, ::String) at /Users/daly/.julia/v0.6/PackageCompiler/src/static_julia.jl:131
 [5] (::PackageCompiler.#kw##static_julia)(::Array{Any,1}, ::PackageCompiler.#static_julia, ::String) at ./<missing>:0
 [6] (::BuildApp.##2#7{Bool,String})() at /Users/daly/.julia/v0.6/ApplicationBuilder/src/BuildApp.jl:134
 [7] withenv(::BuildApp.##2#7{Bool,String}, ::Pair{String,String}, ::Vararg{Pair{String,String},N} where N) at ./env.jl:157
 [8] #build_app_bundle#1(::String, ::String, ::Array{String,1}, ::Array{String,1}, ::Bool, ::Void, ::String, ::Void, ::Void, ::Void, ::Bool,
::BuildApp.#build_app_bundle, ::String) at /Users/daly/.julia/v0.6/ApplicationBuilder/src/BuildApp.jl:130
 [9] build_app_bundle(::String) at /Users/daly/.julia/v0.6/ApplicationBuilder/src/BuildApp.jl:37

Undefined symbols for architecture x86_64:
  "_julia_main", referenced from:
      _main in program-695139.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

Alright, what's going on there?

Alright, what's going on there?

First things to notice:

- that's really slow!
- There's a lot of output...
- It automatically created an output directory (`builddir`).
- Our `println` was executed!
- It looks like it failed!? Undefined symbols...?

What happens when you build an ”app bundle?”

1. ApplicationBuilder creates an empty "Application Bundle" based on the supplied configuration.

"Application Bundle" is my term for a standard, OS-native application. One that can be installed on a machine in the *standard* ways, and looks like a normal application to the operating system.

These "bundles" wrap up all the things necessary to run the application:

- an executable,
- supporting runtime libraries, and
- supporting resources (graphics, fonts, sounds, etc.)

On Mac, this is called an "app", and it's actually just a directory with the `.app` extension. **TODO: Windows Linux?**

```
build_app_bundle(juliaprog_main;
    appname, builddir, resources, libraries,
    verbose, bundle_identifier, app_version,
    icns_file, certificate, entitlements_file,
    commandline_app)
```

Compile `juliaprog_main` into an executable, and bundle it together with all its `resources` and `libraries` into an App called `appname`.

`juliaprog_main`: Path to a ".jl" file that defines the function `julia_main()`

Example

```
julia-repl
julia> build_app_bundle("main.jl", appname="MyApp",
           resources=["img.jpg"],
           libraries=[MyPackage._libp])
```

2. ApplicationBuilder creates an executable inside that App Bundle

It runs the `julia` process with your code (`"main.jl"`), with flags to emit statically compiled output.

This step is done using the `PackageCompiler` package's `build_executable` function (https://github.com/JuliaLang/PackageCompiler.jl/blob/master/src/static_julia.jl).

We'll talk about this more soon.

We can see that the Application Bundle was created (even though the next step failed) by examing the file structure:

```
In [15]: printfiletree("builddir")
```

```
builddir
└── app.app
    └── Contents
        ├── MacOS
        │   ├── app.dylib
        │   └── app.o
        └── Resources
```

On Compiling julia code

A statically compiled julia program consists of two pieces:

1. Your julia code, compiled as a static shared library
2. A "driver" executable which loads the shared library and runs it

The entry point that the *driver executable* uses for your code is a function called `julia_main()`. This maps to the the way `C` programs are invoked.

What this means for you is that your julia program *must* contain a `julia_main` function, which will be the first thing called when your application is run.

In [16]:
```julia
# Let's look at the previous example, `hello.jl`:

run(`open https://github.com/NHDaly/ApplicationBuilder.jl/blob/master/examples/hello.jl`)
```

Here we can see that this program's code is all running from withing the `julia_main` function, which has this ugly declaration:

```
Base.@ccallable
function julia_main(ARGS::Vector{String})::Cint
```

- `Base.@ccallable` allows the function to be called by the *driver* executable.
- `ARGS` is a vector of all the arguments passed to the program (like `argv` in C/C++).
- `::Cint` is the status code returned from your program (the same as in C/C++).

```
In [17]: write("src/app.jl",
          """
             using UnicodePlots

             Base.@ccallable function julia_main(ARGS::Vector{String})::Cint
                 println("**** Hello From Julia! ****")
                 r = rand(0:2π)
                 println(lineplot(1:100, sin.(linspace(r, r+2π, 100))))

                 return 0
             end
          """
         )

Out[17]: 249
```

```
In [18]: rm("builddir", recursive=true)
         BuildApp.build_app_bundle("src/app.jl", commandline_app=true)  # This is a commandline-only app.
```

```
  Using calculated bundle_identifier: 'com.daly.app'
~~~~~~ Creating mac app in "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/app.app" ~~
~~~~~
~~~~~~ Compiling a binary from 'src/app.jl'... ~~~~~~~
Julia program file:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/src/app.jl"
C program file:
  "/Users/daly/.julia/v0.6/ApplicationBuilder/src/program.c"
Build directory:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/app.app/Contents/app"
All done
```

```
Out[18]: 0
```

```
~~~~~~ Generating 'Info.plist' for 'com.daly.app'... ~~~~~~~
~~~~~~ Cleaning up temporary files... ~~~~~~~
~~~~~~ Signing the binary and all libraries ~~~~~~~
~~~~~~ Done building '/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/app.app'! ~~~~~~~
```

```
In [19]: printfiletree("builddir/app.app")
```

```
builddir/app.app
 └──── Contents
     ├──── Info.plist
     ├──── MacOS
     │    └──── app
     ├──── PkgInfo
     ├──── Resources
     │    ├──── Scripts
     │    │    └──── main.scpt
     │    ├──── app.icns
     │    ├──── applet.icns
     │    └──── applet.rsrc
     └──── app
          ├──── app
          ├──── app.dylib
          ├──── libLLVM.dylib
          ├──── libamd.dylib
          ├──── libarpack.2.dylib
          ├──── libarpack.dylib
          ├──── libcamd.dylib
          ├──── libccalltest.dylib
          ├──── libccolamd.dylib
          ├──── libcholmod.dylib
          ├──── libcolamd.dylib
          ├──── libcurl.4.dylib
          ├──── libcurl.dylib
          ├──── libdSFMT.dylib
          ├──── libfftw3.3.dylib
          ├──── libfftw3.dylib
          ├──── libfftw3_threads.3.dyli
b
          ├──── libfftw3_threads.dylib
          ├──── libfftw3f.3.dylib
          ├──── libfftw3f.dylib
          ├──── libfftw3f_threads.3.dyl
ib
          ├──── libfftw3f_threads.dylib
          ├──── libgcc_s.1.dylib
          ├──── libgfortran.4.dylib
          ├──── libgit2.0.25.1.dylib
          ├──── libgit2.25.dylib
          ├──── libgit2.dylib
          ├──── libgmp.10.dylib
          ├──── libgmp.dylib
          ├──── libjulia.0.6.4.dylib
          ├──── libjulia.0.6.dylib
          ├──── libjulia.dylib
          ├──── libmbedcrypto.0.dylib
          ├──── libmbedcrypto.2.3.0.dyl
ib
          ├──── libmbedcrypto.dylib
          ├──── libmbedtls.10.dylib
          ├──── libmbedtls.2.3.0.dylib
          ├──── libmbedtls.dylib
          ├──── libmbedx509.0.dylib
          ├──── libmbedx509.2.3.0.dylib
          ├──── libmbedx509.dylib
          ├──── libmpfr.4.dylib
          ├──── libmpfr.dylib
          ├──── libopenblas64_.dylib
          ├──── libopenlibm.2.3.dylib
          ├──── libopenlibm.2.dylib
```

```
        ├── libopenlibm.2.dylib
        ├── libopenlibm.dylib
        ├── libopenspecfun.1.3.dyli
b
        ├── libopenspecfun.1.dylib
        ├── libopenspecfun.dylib
        ├── libpcre2-8.0.dylib
        ├── libpcre2-8.dylib
        ├── libpcre2-posix.1.dylib
        ├── libpcre2-posix.dylib
        ├── libquadmath.0.dylib
        ├── libspqr.dylib
        ├── libssh2.1.0.1.dylib
        ├── libssh2.1.dylib
        ├── libssh2.dylib
        ├── libsuitesparse_wrapper.
dylib
        ├── libsuitesparseconfig.dy
lib
        └── libumfpack.dylib
```

```
In [20]: run(`./builddir/app.app/Contents/MacOS/app`)
```

Code structure

One last thing here, now that we've moved all our code inside `julia_main`, it's stopped working the normal way! Now it *only* works when it's compiled:

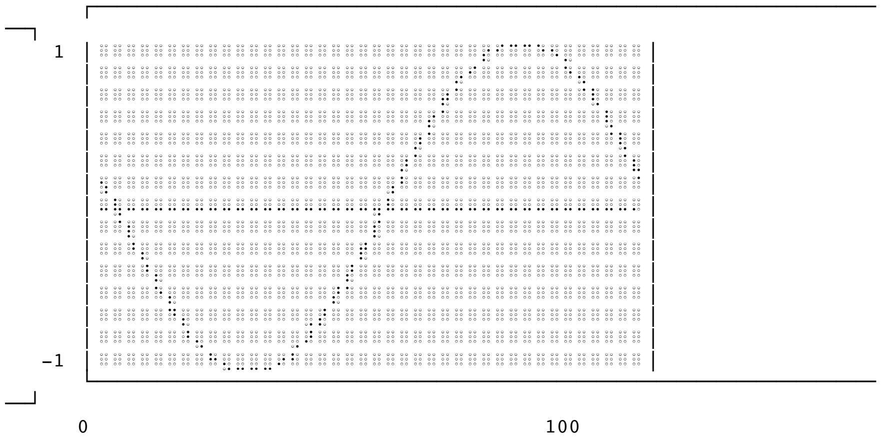
```
In [21]: include("src/app.jl")
```

The recommended way to fix this is to have multiple julia "driver" files, one to run your code through the normal `julia` interpreter, and one for compiling.

To support that, we can restructure our code into a new function:

In [22]:
```julia
write("src/app.jl",
 """
    using UnicodePlots

    println("**** Hello from the outside! ****")

    function app_main()
        println("**** Hello From Julia! ****")
        r = rand(0:2π)
        println(lineplot(1:100, sin.(linspace(r, r+2π, 100))))
    end
 """
)
write("src/julia_main.jl",
 """
    include("app.jl")
    Base.@ccallable function julia_main(ARGS::Vector{String})::Cint
        app_main()
        return 0
    end
"""
)
write("src/app_jl.jl",
 """
    include("app.jl")
    app_main()
 """
)

run(`julia src/app_jl.jl`)
```

```
**** Hello from the outside! ****
**** Hello From Julia! ****
```



In [23]:
```julia
printfiletree("src")
```

```
src
├── app.jl
├── app_jl.jl
└── julia_main.jl
```

When compiling, `app_main()` won't execute.

(But note that the global print statement is still executed!)

```
In [24]: BuildApp.build_app_bundle(
             "src/julia_main.jl",
             appname="HelloWorld",  # Let's set an app name so it doesn't keep changing when we change the file we're compiling.
             commandline_app=true,
         )
```

```
  Using calculated bundle_identifier: 'com.daly.helloworld'
~~~~~~ Creating mac app in "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/HelloWorld.app" ~~
~~~~~
~~~~~~ Compiling a binary from 'src/julia_main.jl'... ~~~~~~~
Julia program file:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/src/julia_main.jl"
C program file:
  "/Users/daly/.julia/v0.6/ApplicationBuilder/src/program.c"
Build directory:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/HelloWorld.app/Contents/app"
**** Hello from the outside! ****
All done
```

```
Out[24]: 0
```

```
~~~~~~ Generating 'Info.plist' for 'com.daly.helloworld'... ~~~~~~~
~~~~~~ Cleaning up temporary files... ~~~~~~~
~~~~~~ Signing the binary and all libraries ~~~~~~~
~~~~~~ Done building '/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/HelloWorld.app'! ~~~~~~~
```

Adding a GUI to our App

Julia desktop GUI Packages

And my success using them with ApplicationBuilder.

- ✅ [Blink.jl](https://github.com/JunoLab/Blink.jl) -
❌ [Gtk.jl](https://github.com/JuliaGraphics/Gtk.jl) -
✅ [Libui.jl]
(https://github.com/jonathanBieler/SimpleDirectMediaLayer.jl)
- ✅ [Cario.jl]
(https://github.com/JuliaGraphics/Cairo.jl)

- ✅ [SimpleDirectMediaLayer.jl](https://github.com/jonathanBieler/SimpleDirectMediaLayer.jl) -
❌ [Tk.jl](https://github.com/JuliaGraphics/Tk.jl) - ❌ [QML.jl]
(https://github.com/JuliaGraphics/Tk.jl) - **?** [Win32GUIDemo.jl]
(https://github.com/ihnorton/Win32GUIDemo.jl)

I'm still working on support for the ones that currently aren't working.

In [25]:
```
using Blink

win = Window(); sleep(2)
```
```
INFO: Loading HttpServer methods...
```

In [26]:
```
body!(win, """
    <input id="mySlider" type="range" min="1" max="100" value="50">
    <script>
        mySlider = document.getElementById("mySlider")
    </script>
"""); sleep(2)
tools(win)

Blink.@js_ win console.log("HELLO!")
Blink.@js_ win mySlider.oninput =
    (e) -> (Blink.msg("sliderChange", mySlider.value);
            console.log("sent msg to julia!"); e.returnValue=false)
Blink.handlers(win)["sliderChange"] = (val) -> (println("msg from js: $val"))
```

Out[26]:
```
(::#5) (generic function with 1 method)
```

In [27]:
```
using Plots
plotly()
```

```
INFO: Recompiling stale cache file /Users/daly/.julia/lib/v0.6/Plots.ji for module Plots.
```

Out[27]:
```
Plots.PlotlyBackend()
```

In [28]:
```
r = rand(0:2π)
p = plot(r:2π/100:r+2π, sin)
```

Out[28]:

In [29]:
```
# Let's capture that plot and show it in our html window!
buf = IOBuffer()
show(buf, MIME("text/html"), p)
plothtml = String(take!(buf))

body!(win, """<script>var Plotly = require('$(Plots._plotly_js_path)');</script>
              <div id="plotHolder"></div>"""); sleep(4)
content!(win, "#plotHolder", plothtml); sleep(4)
```

Putting it together!

In [32]:
```
using Blink, Plots
plotly()

win = Window(); sleep(2)
body!(win, """
    <input id="mySlider" type="range" min="1" max="100" value="50">
    <div id="plotHolder">
        plot goes here...
    </div>
    <script>
        mySlider = document.getElementById("mySlider")
        var Plotly = require('$(Plots._plotly_js_path)');
    </script>
"""); sleep(2)
tools(win)

Blink.@js win console.log("HELLO!")
Blink.@js win mySlider.oninput =
    (e) -> (Blink.msg("sliderChange", mySlider.value);
            console.log("sent msg to julia!"); e.returnValue=false)

function sliderChange(val)
    r = parse(val)
    p = Plots.plot(r:2π/100:r+2π, sin)
    buf = IOBuffer()
    show(buf, MIME("text/html"), p)
    plothtml = String(take!(buf))

    content!(win, "#plotHolder", plothtml, fade=false)
end

Blink.handlers(win)["sliderChange"] = sliderChange
```

Out[32]:
```
sliderChange (generic function with 1 method)
```

Building a static Application!

```
In [33]: write("src/app.jl",
          raw"""
             using Blink, Plots
             plotly()

             function app_main()
                 win = Window(); sleep(2)
                 body!(win, \"\"\"
                     <input id="mySlider" type="range" min="1" max="100" value="50">
                     <div id="plotHolder">
                         plot goes here...
                     </div>
                     <script>
                         mySlider = document.getElementById("mySlider")
                         var Plotly = require('$(Plots._plotly_js_path)');
                     </script>
                 \"\"\"); sleep(2)
                 tools(win)

                 Blink.@js_ win console.log("HELLO!")
                 Blink.@js_ win mySlider.oninput =
                     (e) -> (Blink.msg("sliderChange", mySlider.value);
                             console.log("sent msg to julia!"); e.returnValue=false)

                 function sliderChange(val)
                     r = parse(val)
                     p = Plots.plot(r:2π/100:r+2π, sin);  # Don't forget this ';' to prevent it opening a plot window!
                     buf = IOBuffer()
                     # invokelatest b/c show compiles more functions, and fails due to world age (https://discourse.julialang.org/t/running-in-world-age-x
         -while-current-world-is-y-errors/5871/5)
                     Base.invokelatest(show, buf, MIME("text/html"), p);
                     plothtml = String(take!(buf))

                     Blink.content!(win, "#plotHolder", plothtml, fade=false)
                 end

                 Blink.handlers(win)["sliderChange"] = sliderChange

                 # Keep the process alive until the window is closed!
                 while active(win)
                     sleep(1)
                 end

                 return 0
             end
          """
         )

Out[33]: 1475
```

```
In [34]: run(`julia src/app_jl.jl`)

         INFO: Loading HttpServer methods...
```

```
In [35]: build_app_bundle(
    appname="SinePlotter",  # New App name
)
```

```
  Using calculated bundle_identifier: 'com.daly.sineplotter'
~~~~~~ Creating mac app in "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/SinePlotter.app" ~~~~~~~
~~~~~~ Compiling a binary from 'src/julia_main.jl'... ~~~~~~~
Julia program file:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/src/julia_main.jl"
C program file:
  "/Users/daly/.julia/v0.6/ApplicationBuilder/src/program.c"
Build directory:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/SinePlotter.app/Contents/MacOS"
All done
```

```
Out[35]: 0
```

```
~~~~~~ Generating 'Info.plist' for 'com.daly.sineplotter'... ~~~~~~~
~~~~~~ Cleaning up temporary files... ~~~~~~~
~~~~~~ Signing the binary and all libraries ~~~~~~~
~~~~~~ Done building '/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/SinePlotter.app'! ~~~~~~~
```

```
In [37]: run(`open /Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/SinePlotter.app`)
```

In [38]:

```
# Here's how to make a _distributable_ Blink app:

run(`open https://github.com/NHDaly/ApplicationBuilder.jl/blob/master/examples/blink.jl`)
```

In [49]:

```
# Apply that to our program, and this is what we have:
write("src/julia_main.jl",
 """
    using ApplicationBuilder

    include("app.jl")

    Base.@ccallable function julia_main(ARGS::Vector{String})::Cint
        # THIS IS NEEDED FOR YOUR CODE TO RUN ON ANY COMPUTER
        ApplicationBuilder.App.change_dir_if_bundle()

        app_main()
        return 0
    end
"""
)

write("src/app.jl",
 raw"""
    using Blink, Plots

    # THIS IS NEEDED FOR YOUR CODE TO RUN ON ANY COMPUTER
    if get(ENV, "COMPILING_APPLE_BUNDLE", "false") == "true"
        println("Overriding Blink dependency paths.")
        eval(Blink.AtomShell, :(_electron = "Julia.app/Contents/MacOS/Julia"))
        eval(Blink.AtomShell, :(mainjs = "main.js"))
        eval(Blink, :(buzz = "main.html"))
        eval(Blink, :(resources = Dict("spinner.css" => "res/spinner.css",
                                 "blink.js" => "res/blink.js",
                                 "blink.css" => "res/blink.css",
                                 "reset.css" => "res/reset.css")))
        eval(Blink, :(const port = get(ENV, "BLINK_PORT", rand(2_000:10_000))))
        # Clear out Blink.__inits__, since it will attempt to evaluate hardcoded paths.
        # (We've defined all the variables manually, above: `resources` and `port`.)
        eval(Blink, :(empty!(__inits__)))

        eval(HttpParser, :(lib = basename(HttpParser.lib)))
        eval(MbedTLS, :(const libmbedcrypto = basename(MbedTLS.libmbedcrypto)))
        eval(MacroTools, :(const animals_file = "animals.txt"))

        println("Overriding Plotly dependency paths.")
        eval(Plots, :(_plotly_js_path = "plotly-latest.min.js"))
        println("Done changing dependencies.")
    end

    function app_main()
        # This must be inside app_main() b/c must be after `change_dir_if_bundle()`
        plotly()

        win = Window(); sleep(5)
        body!(win, \"\"\"
            <input id="mySlider" type="range" min="1" max="100" value="50">
            <div id="plotHolder">
                plot goes here...
            </div>
            <script>
                mySlider = document.getElementById("mySlider")
                var Plotly = require('../../../../../$(Plots._plotly_js_path)');
```

```
      </script>
    \"\"\"); sleep(2)
    tools(win)

    Blink.@js_ win console.log("HELLO!")
    Blink.@js_ win mySlider.oninput =
      (e) -> (Blink.msg("sliderChange", mySlider.value);
              console.log("sent msg to julia!"); e.returnValue=false)

    function sliderChange(val)
      r = parse(val)
      p = Plots.plot(r:2π/100:r+2π, sin);  # Don't forget this ';' to prevent it opening a plot window!
      buf = IOBuffer()
      # invokelatest b/c show compiles more functions, and fails due to world age (https://discourse.julialang.org/t/running-in-world-age-x
-while-current-world-is-y-errors/5871/5)
      Base.invokelatest(show, buf, MIME("text/html"), p);
      plothtml = String(take!(buf))

      Blink.content!(win, "#plotHolder", plothtml, fade=false)
    end

    Blink.handlers(win)["sliderChange"] = sliderChange

    # Keep the process alive until the window is closed!
    while active(win)
      sleep(1)
    end

    return 0
  end
"""
)
```

Out[49]: 2914

In [53]:

```
# Build a distributable SinPlotter.app!
using ApplicationBuilder; using BuildApp
using Blink, Plots
blinkPkg = Pkg.dir("Blink")
macroToolsPkg = Pkg.dir("MacroTools")

BuildApp.build_app_bundle(
  "src/julia_main.jl",
  appname="SinePlotterBundled",
  resources = [
    # Blink resources
    joinpath(blinkPkg, "deps","Julia.app"),
    Blink.AtomShell.mainjs,
    joinpath(blinkPkg, "src","content","main.html"),
    joinpath(blinkPkg, "res"),
    joinpath(macroToolsPkg, "animals.txt"),
    # Plots resources
    Plots._plotly_js_path,
  ],
  libraries = [
    HttpParser.lib,
    MbedTLS.libmbedcrypto,
  ],
)
```

```
 Using calculated bundle_identifier: 'com.daly.sineplotterbundled'
~~~~~~ Creating mac app in "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/SinePlotterBundled.app" ~~~~~~~
~~~~~~ Copying user-specified libraries & resources to bundle... ~~~~~~~
 Resources:
   - /Users/daly/.julia/v0.6/Blink/deps/Julia.app                    done
```

```
    - /Users/daly/.julia/v0.6/Blink/deps/Julia.app ............... done
    - /Users/daly/.julia/v0.6/Blink/src/AtomShell/main.js ............... done
    - /Users/daly/.julia/v0.6/Blink/src/content/main.html ............... done
    - /Users/daly/.julia/v0.6/Blink/res ............... done
    - /Users/daly/.julia/v0.6/MacroTools/animals.txt ............... done
    - /Users/daly/.julia/v0.6/Plots/src/backends/../../deps/plotly-latest.min.js ............... done
  Libraries:
    - /Users/daly/.julia/v0.6/HttpParser/deps/usr/lib/libhttp_parser.dylib ............... done
    - /Users/daly/.julia/v0.6/MbedTLS/src/../deps/usr/lib/libmbedcrypto.2.11.0.dylib ............... done
~~~~~~ Compiling a binary from 'src/julia_main.jl'... ~~~~~~~
Julia program file:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/src/julia_main.jl"
C program file:
  "/Users/daly/.julia/v0.6/ApplicationBuilder/src/program.c"
Build directory:
  "/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/SinePlotterBundled.app/Contents/MacOS"
Overriding Blink dependency paths.

WARNING: redefining constant _electron
WARNING: redefining constant mainjs
WARNING: redefining constant resources

Overriding Plotly dependency paths.
Done changing dependencies.

WARNING: redefining constant lib
WARNING: redefining constant libmbedcrypto
WARNING: redefining constant _plotly_js_path

All done
```

```
Out[53]: 0
```

```
~~~~~~ Generating 'Info.plist' for 'com.daly.sineplotterbundled'... ~~~~~~~
~~~~~~ Cleaning up temporary files... ~~~~~~~
~~~~~~ Signing the binary and all libraries ~~~~~~~
~~~~~~ Done building '/Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/SinePlotterBundled.app'! ~~~~~~~
```

```
In [43]: run(`open /Users/daly/Documents/developer/talks/jupyter/playground/OurApp/builddir/SinePlotterBundled.app/Contents/MacOS`)
```