Mixing Rust and iOS

Sebastian Imlay

About me

- Rust Open Source Developer
- github.com/simlay/
- hachyderm.io/@simlay
- simlay.net
- co-maintainer of coreaudio-rs and coreaudio-sys

This talk & code at: https://github.com/simlay/presentations/

What's included (and what's not included)

- Why do this?
- Rust "Hello World" for iOS
- Bundling and Running "Hello World" iOS
- Exit status of "Hello World" for CI
- Helpful Rust features and tools for rust iOS
- Expose Rust library to React Native with an Expo Native Module

Not included:

- GUI programming in Rust

Why go to the effort?

- Memory Safe
- Stable
- Cross compiling rust is easier than C and C++
- Write once rather than Kotin and Swift

Hello World

- Install Rust, install xcode, install iOS simulator and runtime
- `$ rustup target install aarch64-apple-ios-sim`
- `$ cargo new --bin hello-world && cd hello-world`
- `$ cargo build --target aarch64-apple-ios-sim`
- Executable at `./target/aarch64-apple-ios-sim/debug/hello-world`

What is an iOS “app"?

```
RustWrapper.app/
├── Info.plist
└── hello-world
```

Upgrade `src/main.rs`

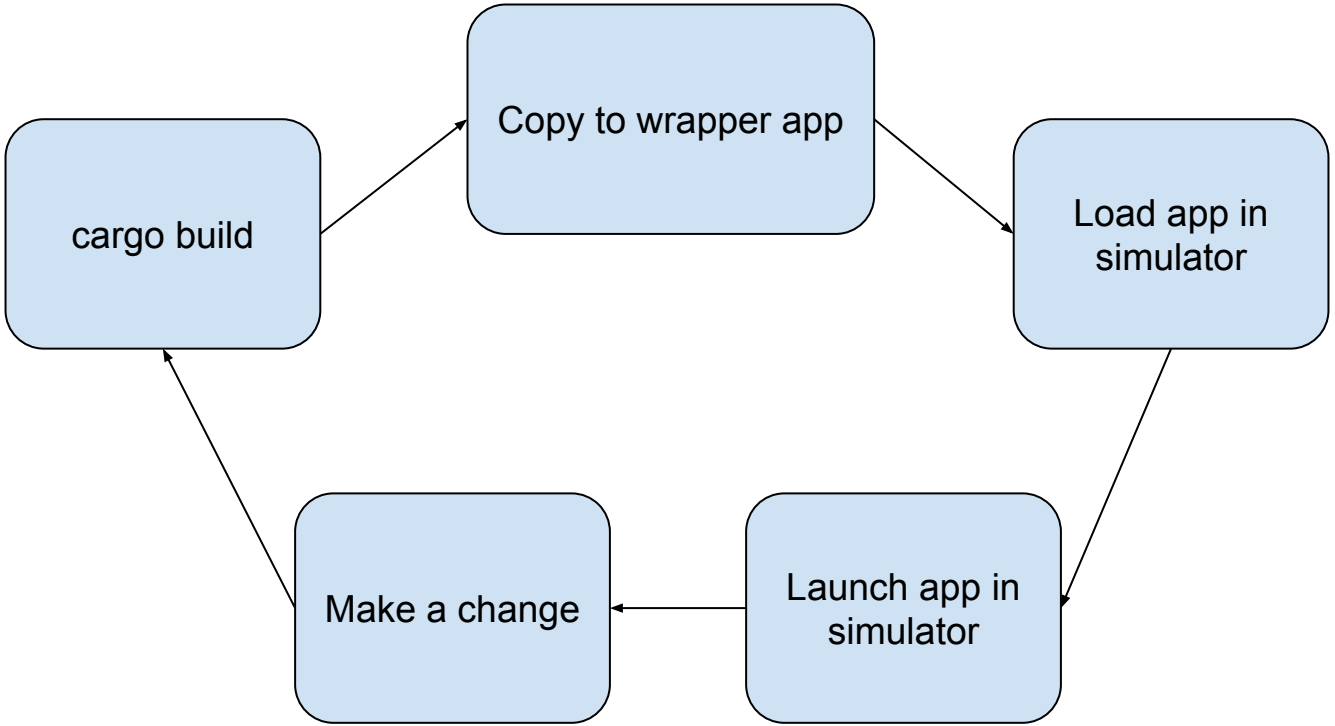
```
use std::{
    thread::sleep,
    time::{Duration, SystemTime, UNIX_EPOCH},
};

fn main() {
    for i in 1..100 {
        // How long has it been since unix epoc?
        let since_unix_epoc = SystemTime::now().duration_since(UNIX_EPOCH).unwrap();
        println!(
          "Hello, world for the {i}'th time, {:?} seconds from unix epoc",
          since_unix_epoc,
        );
        // Let's time out
        sleep(Duration::from_millis(500));
    }
}
```

Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0"><dict>
        <key>CFBundleName</key><string>RustWrapper</string>
        <key>CFBundleExecutable</key><string>hello-world</string>
        <key>CFBundleIdentifier</key><string>RustWrapper</string>
        <key>CFBundleVersion</key>
        <string>arm64
    </string>
    <key>CFBundleShortVersionString</key>
    <string>arm64</string>
    <key>UIRequiredDeviceCapabilities</key>
    <array><string>arm64 </string></array>
    <key>UILaunchStoryboardName</key>
    <string></string>
</dict></plist>
```

Development loop



Manually doing this loop

```
$ cargo build --target aarch64-apple-ios-sim
$ cp ./target/aarch64-apple-ios-sim/debug/hello-world ./Wrapper.app/
$ xcrun simctl install booted ./Wrapper.app/
$ xcrun simctl launch --console booted RustWrapper
```

Put it in a `makefile`

```
build:
	cargo build --target aarch64-apple-ios-sim

bundle: build
	cp ./target/aarch64-apple-ios-sim/debug/hello-world ./RustWrapper.app/

install: bundle
	xcrun simctl install booted ./RustWrapper.app/

run: install
	xcrun simctl launch --console --terminate-running-process booted RustWrapper

watch:
	cargo watch -s 'make run' -w ./src
```

How do we get an exit status?

- Don’t.
- Start the iOS app in debug
- attach to the iOS app in with lldb.
- Run “continue” lldb command
- Parse stdout to find something like: `Process 34163 exited with status = 101`

Dinghy for the rescue

- github.com/sonos/dinghy
- Bundles the executable into a wrapper iOS app
- Loads into a simulator
- Runs and attaches to the iOS app to get the exit status
- Does not stream stdout to the command line
- Can sign and load onto a device
- Works in CI
- Supports watchOS, tvOS, and soon to visionOS

Use dinghy in a target runner

```
$ cat .cargo/config.toml
[target.aarch64-apple-ios-sim]
runner = "cargo dinghy -p auto-ios-aarch64-sim runner --"

[target.x86_64-apple-ios]
runner = "cargo dinghy -p auto-ios-x86_64 runner --"

$ cargo test --target aarch64-apple-ios-sim
   Compiling hello-world v0.1.0
    Finished test [unoptimized + debuginfo] target(s) in 0.17s
     Running unittests src/main.rs
(target/aarch64-apple-ios-sim/debug/deps/hello_world-ced75443a4c5495b)
  Installing hello_world-ced75443a4c5495b to 820BA6FB-5A8C-4CAC-9952-C8F6841971EA
     Running hello_world-ced75443a4c5495b on 820BA6FB-5A8C-4CAC-9952-C8F6841971EA

running 1 test
test add ... ok
```

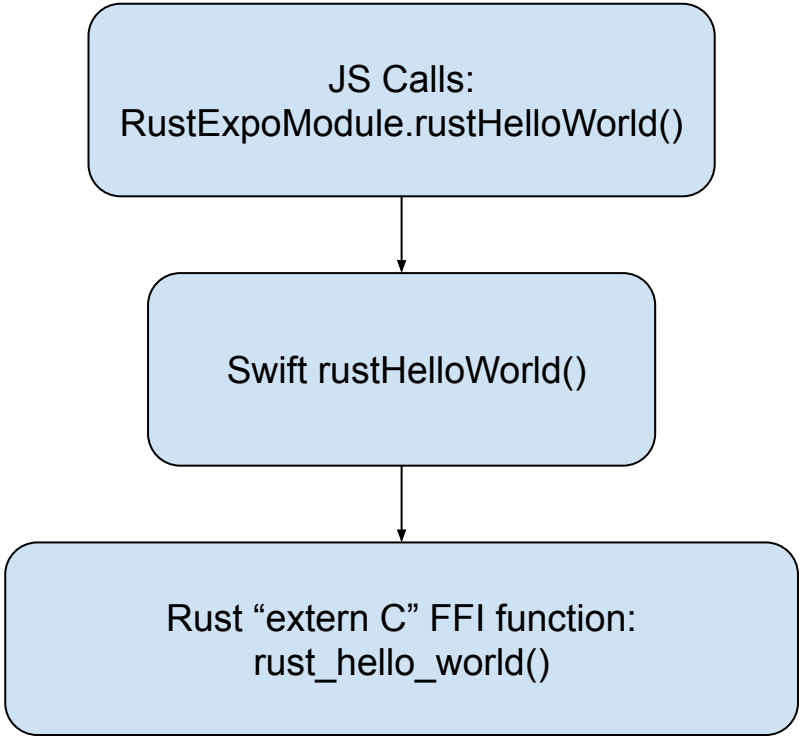
Tools and Tips

- Use the cargo target runner in `.cargo/config.toml`
- Most rust crates have a `vendored` feature flag to statically link big dependencies
  - `reqwest` (a rust request library)
  - `libsqlite-sys`
- Use conditional compilation for platform specific work
  - `#[cfg(target_os = "ios")] mod foo;`
  - Example: `cpal` (cross platform audio library)

Expo Native Module with Rust

- Compile rust as staticlib for iOS rust targets
- Rust functions are `#[no_mangle] extern "C" fn`
- Add the static lib as a vendored library to the Podspec
- Use C Strings (null terminated)
- https://github.com/simlay/rust-expo-module

Call Tree


JS Calls:
RustExpoModule.rustHelloWorld()
Swift rustHelloWorld()
Rust “extern C” FFI function:
rust_hello_world()

Adding a new call

- Add new `#[no_mangle] extern "C" fn rust_foo()`
- Add new Swift wrapper `rustFoo()`
- Add new typescript function `rustFoo()`

Questions