# (Swift) Plays Well with Others

Matthew Burke, Capital One

*CocoaConfDC, 10 Sep 2016*

# Swift from C

```
@_silgen_name("retrieveAnInt") // override name mangling
func retrieveAnInt() -> Int {
     return 17 // the most random number
}

@_silgen_name("retrieveADouble")
func retrieveADouble() -> Double {
     return 8.124038404635961 // root 66
}
```

# Swift from C

```
swiftFromC: library
	$(CP) -f libswiftfromc.$(LIBSUFFIX) csrc
	$(MAKE) -C csrc

library: Sources/library.swift
	 swift build -Xswiftc -emit-library
```

The Swift build system and PM are shaping up nicely. If you dig around in the command line options, there are all kinds of interesting things you can do. (Mention importing your own modules into the REPL)

# Using a C Library

We'll use the GD (https://libgd.github.io/) library as an example.

I've been using this library for almost 20 years…

# Using a C Library

```
module CGd [system] {
        header "/usr/local/include/gd.h"
        header "/usr/local/include/gdfontt.h"
        header "/usr/local/include/gdfonts.h"
        header "/usr/local/include/gdfontl.h"
        header "/usr/local/include/gdfontg.h"
        header "/usr/local/include/gdfontmb.h"
        link "gd"
        export *
}
```

# Using a C Library

That’s it; we’re done.

# Using a C Library

Well, not really.

# Using a C Library

```
public class Gd {

    public let width: Int32
    public let height: Int32
    let im: gdImagePtr

    public func drawArc(cx: Int32, cy: Int32, w: Int32, h: Int32,
                        s: Int32, e: Int32, color: Int32) {
        gdImageArc(im, cx, cy, w, h, s, e, color)
    }

    public func drawString(text: String, x: Int32, y: Int32, color: Int32) {
        let cString = text.cString(using: NSASCIIStringEncoding)!
        let cPtr = UnsafeMutablePointer<UInt8>(cString)
        gdImageString(im, gdFontGetMediumBold(), x, y, cPtr, color)
    }
```

# Using Java

```
module swiftrixmodule [system] {
    header "/usr/local/include/swiftrix.h"
    link "swiftrix"
    export *
}
```

Why isn’t it

```
header “/System/Library/Frameworks/
JavaVM.framework/Versions/A/Headers/jni.h”
```

I’m working towards just wrapping JNI directly…types are a pain.

# Using Java

```
obj = (*env)->NewObject(env, cls, ctor, JNI_TRUE);
 if (obj) {
   mid = (*env)->GetMethodID(env, cls,
              "execute", "()Ljava/lang/Object;");
   if (mid) {
     jstring result = (*env)->CallObjectMethod(env,
                                               obj, mid);
     if ((*env)->ExceptionOccurred(env)) {
       (*env)->ExceptionDescribe(env);
     }
```

Here’s a snippet showing the JNI functions in action

# Using Java

```
import swiftrixmodule

initLibrary()

typealias Callback = @convention(c) () ->
                  UnsafeMutablePointer<Int8>?

func swiftyFallback() -> UnsafeMutablePointer<Int8>?
{
    return stringToCPointer(string: "fallback")
}

setfallback(swiftyFallback as Callback)
```

# Using Lua

```
class LuaState {
    let L: COpaquePointer

    init()  {
        L = luaL_newstate()
        luaL_openlibs(L)
    }

    func eval(script: String) -> EvalResult  {
        if status == LUA_OK {
            removeInput()
            status = lua_pcallk(self.L, 0, LUA_MULTRET, 0, 0, nil)
```

Note this is all Swift 2.1(?) and I need to update it

http://github.com/profburke