Custom Frameworks and Swift

###Setup

To build a framework that is compatible with swift it must generate a module as part of the build process. All new framework targets in Xcode 6 have this enabled by default. For older projects, you will have to enable this inside of the “Build Settings” for the framework target.

Under the heading “Apple LLVM 6.0 - Language - Modules” enable the setting “Enable Modules (C and Objective-C)”. This will create the module file needed for swift to auto-generate the swift headers from C and Objective-C headers.

Now, drop the Custom Framework into the respective “Frameworks” folder for the target SDK (iOS or MacOSX). For many frameworks this is going to be all that is needed for the Playground or REPL to recognize and load the framework through the “import” command.

However sometimes there will be “Undefined Symbol” errors, here is how to fix that.

###REPL

Loading custom frameworks in the REPL is very easy, after calling

```
import MyFramework
```

Just supply the following call:

```
var handle = dlopen("/Applications/Xcode6-Beta.app/Contents/Developer/P
```

This will load the framework into the REPL’s memory space and allow the symbol lookups to be performed.

###Playground

Loading custom frameworks into the Playground is more tricky as it doesn’t work the same way as the REPL does. Unlike the REPL, adding the dlopen() call to your playground code doesn’t get the framework loaded into the Playground process. This is because there is a different binary that is run for the Playground execution, it is located at:

```
/Applications/Xcode6-Beta.app/Contents/Developer/Platforms/MacOSX.platf
```

Additionally, this process seems to be continually killed off and restarted, based on the execution time specified in the Playground environment. In order to make the process always be able to load the desired custom frameworks, I hacked the binary file to add in my own DYLIB load command.

Here is what the “PlaygroundStub_OSX” binary, that ships with the Xcode 6 beta, links loads on launch:

```
sam@Pegasus/Users/sam $ otool -L /Applications/Xcode6-Beta.app/Contents
/Applications/Xcode6-Beta.app/Contents/Developer/Platforms/MacOSX.platf
    @rpath/libswift_stdlib_core.dylib (compatibility version 0.0.0, cur
    @rpath/XCPlayground.framework/Versions/A/XCPlayground (compatibilit
    @rpath/PlaygroundLogger.framework/Versions/A/PlaygroundLogger (comp
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current ve
    /System/Library/Frameworks/AppKit.framework/Versions/C/AppKit (comp
    /System/Library/Frameworks/CoreFoundation.framework/Versions/A/Core
    /usr/lib/libobjc.A.dylib (compatibility version 1.0.0, current vers
    @rpath/DVTPlaygroundCommunication.framework/Versions/A/DVTPlaygroun
    @rpath/libswiftAppKit.dylib (compatibility version 0.0.0, current v
    @rpath/libswiftCoreGraphics.dylib (compatibility version 0.0.0, cur
    @rpath/libswiftDarwin.dylib (compatibility version 0.0.0, current v
    @rpath/libswiftDispatch.dylib (compatibility version 0.0.0, current
    @rpath/libswiftFoundation.dylib (compatibility version 0.0.0, curre
    @rpath/libswiftObjectiveC.dylib (compatibility version 0.0.0, curre
```

Here is what it loads after patched to load my custom framework loader:

```
sam@Pegasus/Users/sam $ otool -L /Applications/Xcode6-Beta.app/Contents
/Applications/Xcode6-Beta.app/Contents/Developer/Platforms/MacOSX.platf
    @rpath/libswift_stdlib_core.dylib (compatibility version 0.0.0, cur
    @rpath/XCPlayground.framework/Versions/A/XCPlayground (compatibilit
    @rpath/PlaygroundLogger.framework/Versions/A/PlaygroundLogger (comp
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current ve
    /System/Library/Frameworks/AppKit.framework/Versions/C/AppKit (comp
    /System/Library/Frameworks/CoreFoundation.framework/Versions/A/Core
    /usr/lib/libobjc.A.dylib (compatibility version 1.0.0, current vers
    @rpath/DVTPlaygroundCommunication.framework/Versions/A/DVTPlaygroun
    @rpath/libswiftAppKit.dylib (compatibility version 0.0.0, current v
    @rpath/libswiftCoreGraphics.dylib (compatibility version 0.0.0, cur
    @rpath/libswiftDarwin.dylib (compatibility version 0.0.0, current v
    @rpath/libswiftDispatch.dylib (compatibility version 0.0.0, current
    @rpath/libswiftFoundation.dylib (compatibility version 0.0.0, curre
    @rpath/libswiftObjectiveC.dylib (compatibility version 0.0.0, curre
    @rpath/libPlaygroundLoad.dylib (compatibility version 0.0.0, curren
```

The last loaded library is one I created ensure that custom frameworks would get loaded into process to have working symbol resolution each time the Playground is reloaded.

Because we are editing signed binaries, I did have to resign it using my Developer ID cert (acquired through the Mac Developer program).

###This next step is DIY. I am not supporting it, this may break.

I have uploaded the patched version of PlaygroundStub_OSX:

- Beta 1 PlaygroundStub_OSX
- Beta 2 PlaygroundStub_OSX

The code for libPlaygroundLoad.dylib.

In the PlaygroundStub_OSX.zip, there are two files:

- PlaygroundStub_OSX : Patched file to load custom framework loading dylib
- PlaygroundStub_OSX-o : Original copy of the file from Xcode 6 Beta

Place the patched version into this directory:

```
/Applications/Xcode6-Beta.app/Contents/Developer/Platforms/MacOSX.platf
```

In the PlaygroundLoader.zip, build the dynamic library target and place it into this directory:

```
/Applications/Xcode6-Beta.app/Contents/Developer/Toolchains/XcodeDefaul
```

Finally, go to your Application Support folder:

```
~/Library/Application Support/
```

Create a directory named “PlaygroundLoad” and inside of that create a plist called “Plugin.plist”. To have the framework loader resolve symbols you must add some entries to this, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://w
<plist version="1.0">
    <dict>
        <key>Test</key>
        <string>Test.framework/Versions/A/Test</string>
        <key>SDMMobileDevice</key>
        <string>SDMMobileDevice.framework/Versions/A/SDMMobileDevice</s
    </dict>
</plist>
```

You can name each key entry whatever you like, but the string value for that key must be the full path to the framework binary for each custom framework you want to be fully loaded into the Playground process.
