Technical Q&A QA1881 v2 - Embedding Content with Swift in Objective-C

###Background: There appears to be some very vague handling of how to get this to work properly from Apple, this post is meant to explain the situation and problems in detail with the goal that it can help people work around these errors until Apple implements a fix or better explains the potential problems. The sample app was provided to me by Samuel Giddins, and I worked with them to implement a patch for CocoaPods.

###Issue: Scenario: You have an application written in Objective-C. This application contains no Swift code. This application has a couple of dependencies. You are targetting iOS 8+ and have made these dependencies compile as frameworks or dylibs. One or more of these dependencies is written in Swift. When you try to run the application on a device you get an error message such as:

```
dyld: Library not loaded: @rpath/libswiftCore.dylib
  Referenced from: /private/var/mobile/Containers/Bundle/Application/..
  Reason: image not found
```

This is an error thrown by the dynamic linker (dyld) that notifies us that the requested library wasn't found in the runtime search paths. Quick way to check this would be to jump to where application is built and then search for any Swift libraries.

```
$ ls -lsa
total 0
0 drwxr-xr-x+  9 sam  staff  306 Nov 30 09:53 .
0 drwxr-xr-x@  5 sam  staff  170 Nov 30 09:53 ..
0 drwxr-xr-x+ 16 sam  staff  544 Nov 30 09:53 Pods-SwiftTorture
0 drwxr-xr-x+  7 sam  staff  238 Nov 30 09:53 Pods_SwiftTorture.framewo
0 drwxr-xr-x+  3 sam  staff  102 Nov 30 09:53 Pods_SwiftTorture.framewo
0 drwxr-xr-x+  8 sam  staff  272 Nov 30 09:53 SwiftTorture.app
0 drwxr-xr-x+  3 sam  staff  102 Nov 30 09:53 SwiftTorture.app.dSYM
0 drwxr-xr-x+  6 sam  staff  204 Nov 30 09:53 SwiftTortureTests.xctest
0 drwxr-xr-x+  3 sam  staff  102 Nov 30 09:53 SwiftTortureTests.xctest.
$ find . -name "libswift*" | wc -l
       0
```

From this we can tell that the Swift runtime libraries are not being copied into the application bundle so when the application is launched, it fails because it does not have the necessary libraries included.

###Analysis: I would consider this to be a very common case: over-hauling an existing application is a monumental task, but updating an existing common library to Swift would be a good way to integrate new technology and update older code. So, how is it that Xcode seems to fail in this seemingly straight-forward use-case.

Based on some analysis of the build system process, this seems like it was solved at one point. There is a build setting named `EMBEDDED_CONTENT_CONTAINS_SWIFT` which stores a boolean value. There is an Apple Q&A document that describes how to use this, however there are some assumptions made as to how this flag is to be used.

When enabling the `EMBEDDED_CONTENT_CONTAINS_SWIFT` flag on a target, a new step is added to the build process. This step runs the target's build product through a tool called `swift-stdlib-tool`, which parses the binary header to get the list of linked dependencies (frameworks/libraries). It will then check the paths of these linked dependencies to see if any of them contain references to the Swift runtime libraries.

####Background on linked and install paths When a dynamic library is linked, you supply the path to the library and the library's install path gets added to the binary you are linking it to. This install path tells the linker where to look for this library when the binary is loaded and launched by the dynamic linker. To see these paths for yourself, you can dump them by running a binary through otool:

```
$ otool -L /Users/sam/Desktop/SwiftTorture/build/Debug-iphoneos/SwiftTo
/Users/sam/Desktop/SwiftTorture/build/Debug-iphoneos/SwiftTorture.app/S
    @rpath/AFNetworking.framework/AFNetworking (compatibility version 1
    @rpath/Alamofire.framework/Alamofire (compatibility version 1.0.0,
    /System/Library/Frameworks/CFNetwork.framework/CFNetwork (compatibi
    /System/Library/Frameworks/CoreData.framework/CoreData (compatibili
    /System/Library/Frameworks/CoreGraphics.framework/CoreGraphics (com
    @rpath/ISO8601DateFormatterValueTransformer.framework/ISO8601DateFo
    /System/Library/Frameworks/MobileCoreServices.framework/MobileCoreS
    @rpath/RKValueTransformers.framework/RKValueTransformers (compatibi
    @rpath/RestKit.framework/RestKit (compatibility version 1.0.0, curr
    @rpath/SOCKit.framework/SOCKit (compatibility version 1.0.0, curren
    /System/Library/Frameworks/Security.framework/Security (compatibili
    /System/Library/Frameworks/SystemConfiguration.framework/SystemConf
    @rpath/TransitionKit.framework/TransitionKit (compatibility version
    @rpath/Pods_SwiftTorture.framework/Pods_SwiftTorture (compatibility
    /System/Library/Frameworks/Foundation.framework/Foundation (compati
    /usr/lib/libobjc.A.dylib (compatibility version 1.0.0, current vers
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current ve
    /System/Library/Frameworks/UIKit.framework/UIKit (compatibility ver
```

This lists all the paths to the linked libraries that must be loaded when the binary is loaded by the dynamic linker. The paths that begin with `@rpath/` are resolved by looking up the `LC_RPATH` load command in the binary header. In this case it contains the path `@executable_path/Frameworks`, which will resolve to the path to a directory named "Frameworks" that is alongside the binary executable. To show this here are all the linked libraries that use `@rpath/`:

```
@rpath/AFNetworking.framework/AFNetworking (compatibility version 1.0.0
@rpath/Alamofire.framework/Alamofire (compatibility version 1.0.0, curr
@rpath/ISO8601DateFormatterValueTransformer.framework/ISO8601DateFormat
@rpath/RKValueTransformers.framework/RKValueTransformers (compatibility
@rpath/RestKit.framework/RestKit (compatibility version 1.0.0, current
@rpath/SOCKit.framework/SOCKit (compatibility version 1.0.0, current ve
@rpath/TransitionKit.framework/TransitionKit (compatibility version 1.0
@rpath/Pods_SwiftTorture.framework/Pods_SwiftTorture (compatibility ver
```

To double check that these resolve correctly, here are the contents of the resolved `@executable_path/Frameworks` directory:

```
$ ls -ls /Users/sam/Desktop/SwiftTorture/build/Debug-iphoneos/SwiftTort
total 0
0 drwxr-xr-x+ 7 sam  staff  238 Nov 30 09:52 AFNetworking.framework
0 drwxr-xr-x+ 8 sam  staff  272 Nov 30 10:07 Alamofire.framework
0 drwxr-xr-x+ 7 sam  staff  238 Nov 30 09:52 ISO8601DateFormatterValueT
0 drwxr-xr-x+ 7 sam  staff  238 Nov 30 10:07 Pods_SwiftTorture.framewor
0 drwxr-xr-x+ 7 sam  staff  238 Nov 30 09:52 RKValueTransformers.framew
0 drwxr-xr-x+ 7 sam  staff  238 Nov 30 09:53 RestKit.framework
0 drwxr-xr-x+ 7 sam  staff  238 Nov 30 09:52 SOCKit.framework
0 drwxr-xr-x+ 7 sam  staff  238 Nov 30 09:52 TransitionKit.framework
```

From these result we can see that the framework paths correctly resolve to where they are found inside of the application bundle.

####Problematic behavior with checking @rpaths From the example given, you can see how linked dependencies are found when an application is launched. This process is repeated across each dependency that is linked. So, using `Alamofire.framework` as an example, I will repeat these steps again to demonstrate the problematic behavior with using `swift-stdlib-tool`.

```
$ otool -L /Users/sam/Desktop/SwiftTorture/build/Debug-iphoneos/SwiftTo
/Users/sam/Desktop/SwiftTorture/build/Debug-iphoneos/SwiftTorture.app/F
    @rpath/Alamofire.framework/Alamofire (compatibility version 1.0.0, 
    /System/Library/Frameworks/Foundation.framework/Foundation (compati
    /usr/lib/libobjc.A.dylib (compatibility version 1.0.0, current vers
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current ve
    /System/Library/Frameworks/CoreFoundation.framework/CoreFoundation 
    @rpath/libswiftCore.dylib (compatibility version 0.0.0, current ver
    @rpath/libswiftCoreGraphics.dylib (compatibility version 0.0.0, cur
    @rpath/libswiftCoreImage.dylib (compatibility version 0.0.0, curren
    @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/libswiftSecurity.dylib (compatibility version 0.0.0, current
    @rpath/libswiftUIKit.dylib (compatibility version 0.0.0, current ve
```

Here we see this framework contains references to loading the Swift runtime libraries through `@rpath`. To work out where the locations of these libraries are we must consult the `LC_RPATH` commands in the framework:

```
$ otool -l /Users/sam/Desktop/SwiftTorture/build/Debug-iphoneos/SwiftTo
    cmd LC_RPATH
    cmdsize 40
    path @executable_path/Frameworks (offset 12)
--
    cmd LC_RPATH
    cmdsize 36
    path @loader_path/Frameworks (offset 12)
```

The framework gives us two paths to use for subsituting and searching for these dependencies. First being `@executable_path/Frameworks` which gets resolved to being the path to the app path. In this case it would resolve to be searching `/Users/sam/Desktop/SwiftTorture/build/Debug-iphoneos/SwiftTorture.app/F` The second search path is `@loader_path/Frameworks`, which would resolve to be `/Users/sam/Desktop/SwiftTorture/build/Debug-iphoneos/SwiftTorture.app/Framewo` For this library to load correctly, the Swift libraries must exist in at least one of these locations.

####Using swift-stdlib-tool and EMBEDDED_CONTENT_CONTAINS_SWIFT As mentioned, the purpose of `swift-stdlib-tool` is to check the linked libraries for the Swift runtime libraries and copy the respective library into the correct location. The tool does this by checking each library path for a string starting with "@rpath/libswift". If these paths are found, it will copy and then sign the matching libraries into the target bundle.

To mark specific targets for this analysis and to have them include the Swift runtime libraries you must set the flag `EMBEDDED_CONTENT_CONTAINS_SWIFT` to `YES` in the target's build settings. This flag is displayed as "Embedded Content Contains Swift Code".

This is where it becomes problematic: If your app does not contain Swift code and use multiple Swift frameworks, your app will ballon in size. This is because enabling the flag, `EMBEDDED_CONTENT_CONTAINS_SWIFT`, on multiple frameworks will result in the Swift runtime libraries being copied into each of the framework bundles you have enabled that flag on. The Q&A document goes on to say that this is a problem and the solution is to leave that flag turned off on your frameworks and to enabled it on your app target instead.

This statement is only true and will only work if your app target also links against the same Swift runtime libraries that your framework depends on.

###Proposed Solutions:

####XCConfig Fix Last week I posted a link to a gist that included a script for generating an xcconfig file that is intended to fix an issue with using Swift dynamic libraries in with non-Swift apps. Below is the python script I wrote to generate the xcconfig file, and the resulting xcconfig file:

Script:

```
import sys
import os
import string
import subprocess
from subprocess import CalledProcessError

def make_call(call_args):
    error = 0;
    output = '';
    try:
        output = subprocess.check_output(call_args);
        error = 0;
    except CalledProcessError as e:
        output = e.output;
        error = e.returncode;
    return (output, error);
def make_linker_string(libs):
    linker_string = '';
    for lib in libs:
        linker_string += '-Wl,${SWIFT_STDLIB_PATH}/'+lib+' ';
    return linker_string;
def main(argv):
    swift_compiler_lookup = make_call(('xcrun','-f','swift'));
    swift_usr_path = os.path.dirname(os.path.dirname(swift_compiler_loo
    swift_runtime_path = os.path.join(swift_usr_path, 'lib/swift/');
    swift_platforms = ['iphoneos', 'iphonesimulator', 'macosx'];
    swift_libraries = {
        'iphoneos': [],
        'iphonesimulator': [],
        'macosx': []
    };
    for platform in swift_platforms:
        swift_platform_runtime_path = os.path.join(swift_runtime_path,
        find_dylib_results = make_call(('find',swift_platform_runtime_p
        for lib_line in find_dylib_results[0].split('\n'):
            if lib_line != '':
                lib_name = lib_line.split(swift_runtime_path)[1].split(
                if lib_name.find("XCTest") == -1 and lib_name.find("Uni
                    swift_libraries[platform].append(lib_name);

    swift_all_libs = [set(swift_libraries['iphoneos']), set(swift_libra
    swift_universal_libs = list(set.intersection(*swift_all_libs));
    swift_macosx_libs = list(set(swift_libraries['macosx']) - set(swift
    swift_iphoneos_libs = list(set(swift_libraries['iphoneos']) - set(s
    swift_iphonesimulator_libs = list(set(swift_libraries['iphonesimula

    print 'SWIFT_STDLIB_PATH = "$DT_TOOLCHAIN_DIR/usr/lib/swift/$PLATFO
    print 'SWIFT_UNIVERSAL_LIBS = '+make_linker_string(swift_universal_
    # iphoneos
    print 'SWIFT_IPHONEOS_LIBS = '+make_linker_string(swift_iphoneos_li
    # iphonesimulator
    print 'SWIFT_IPHONESIMULATOR_LIBS = '+make_linker_string(swift_ipho
    # macosx
    print 'SWIFT_MACOSX_LIBS = '+make_linker_string(swift_macosx_libs)+
    # OTHER_LDFLAGS
    print 'OTHER_LDFLAGS[sdk=iphoneos*] = ${SWIFT_UNIVERSAL_LIBS} ${SWI
    print 'OTHER_LDFLAGS[sdk=iphonesimulator*] = ${SWIFT_UNIVERSAL_LIBS
    print 'OTHER_LDFLAGS[sdk=macosx*] = ${SWIFT_UNIVERSAL_LIBS} ${SWIFT

if __name__ == "__main__":
    main(sys.argv[1:]);
```

XCConfig File:

```
SWIFT_STDLIB_PATH = "$DT_TOOLCHAIN_DIR/usr/lib/swift/$PLATFORM_NAME"
SWIFT_UNIVERSAL_LIBS = -Wl,${SWIFT_STDLIB_PATH}/libswiftCoreGraphics.dy

SWIFT_IPHONEOS_LIBS = -Wl,${SWIFT_STDLIB_PATH}/libswiftCoreImage.dylib

SWIFT_IPHONESIMULATOR_LIBS = -Wl,${SWIFT_STDLIB_PATH}/libswiftCoreImage

SWIFT_MACOSX_LIBS = -Wl,${SWIFT_STDLIB_PATH}/libswiftAppKit.dylib -Wl,$

OTHER_LDFLAGS[sdk=iphoneos*] = ${SWIFT_UNIVERSAL_LIBS} ${SWIFT_IPHONEOS
OTHER_LDFLAGS[sdk=iphonesimulator*] = ${SWIFT_UNIVERSAL_LIBS} ${SWIFT_I
OTHER_LDFLAGS[sdk=macosx*] = ${SWIFT_UNIVERSAL_LIBS} ${SWIFT_MACOSX_LIB
```

This approach allows the developer to listen to Apple's Q&A document of disabling the `EMBEDDED_CONTENT_CONTAINS_SWIFT` on all targets except for the app target. The XCConfig file adds additional linker flags to include the Swift runtime libraries so that they are detected and only copied once into the app's bundle.

However, this fix has a lot of downsides to it:

- Hard-coding the names of the Swift runtime libraries
- Does not properly support XCTest or the SwiftStdlibUnittest libraries
- Copies all Swift runtime libraries into the app bundle instead of only the required ones

####CocoaPods Fix Samuel Giddins created a fix which addresses the problem in a more direct and less fragile way. By iterating over all of the dependencies of an app and running `otool` on them for linked libraries, it is possible to create a list of just the required frameworks to be linked. This approach ignores the setting of `EMBEDDED_CONTENT_CONTAINS_SWIFT` flag and finds and copies in only the required Swift runtime libraries as needed to the app's bundle in the `Frameworks/` directory. If you have frameworks with the

`EMBEDDED_CONTENT_CONTAINS_SWIFT` flag set to `YES`, then those will also have copies of the Swift runtime libraries embedded in them. This is a more holistic approach to solving this problem.

####Apple Fix I think the most ideal solution would be for Apple to remove this problem entirely by analyzing the linked paths on from `swift-stdlib-tool` and then step through the dependencies to avoid copying the Swift libraries into the app bundle multiple times. This is the solution that needs to happen automatically so that there is no need for developers to deal with management of language runtime dependencies ever.

I think that the existing behavior of `swift-stdlib-tool` needs to be changed from only looking at the binary it is pointed at, to also descend to do checking and validation of the dependencies of any additionally linked library so that the Swift runtime doesn't get duplicated unnecessarily into the app bundle.