05
Aug 2015

AN INTRODUCTION TO CREATING AND DISTRIBUTING EMBEDDED FRAMEWORKS IN IOS

by Brett Stover · 32 Comments

Updates!

1. As JasonR1 points out in the comments, the bug/feature referred to in the "Working with Older Xcode Versions" section has made a comeback in Xcode 7.1, so please review the information below as it's once again relevant.

2. As featherless points out in the comments, distributing frameworks written in Swift via a pre-built .framework file (as opposed to via source code from which users can build the framework themselves) can be problematic. In order for your app to build properly, it and the frameworks it uses must have been built with the same version of the Swift compiler. Since Swift is changing so quickly, maintaining compatibility of the framework with whichever version of Xcode a consumer of your framework is using quickly becomes a hassle. For this reason, you may want to use Objective-C or distribute the source code instead.

With iOS 8, Apple introduced a number of app extensions such as the Share and Today extensions, as well as WatchKit extensions for interacting with WatchKit apps. To facilitate building these extensions, Apple introduced a new (to iOS) type of framework called an embedded framework. The typical use cases Apple has shown for using these frameworks revolve around projects where the embedded framework's code is included in the same project as the code that uses the framework (e.g., an app with a Share extension packaged up as an embedded framework residing in the same project). What hasn't been quite as clear is how one can distribute these frameworks when the framework's parent project and the framework consumer's parent project are not the same project.

At Hootsuite, as we create new apps beyond our flagship iOS app (such as Suggestions by Hootsuite), we've become interested in sharing code between our apps as well as between our individual apps' main target and their various extensions. For example, many projects (including the main Hootsuite iOS app) define their own branded fonts and colors in a category on UIFont and UIColor. Replicating this across several apps, each of which has several of its own extensions, quickly leads to duplication of this code many times over. The same can be said for common networking code, custom UI elements, etc. These new embedded frameworks appear to be a great way to share such common code across projects. In this blog post, we're going to walk through creating and distributing a simple embedded framework consisting of a single class extension on UIColor. The embedded framework we create won't be that exciting; it's the ability to distribute and consume the framework in multiple apps that is. We should note that we are taking the DIY approach here and that there are tools out there such as Carthage and CocoaPods that can handle most of the heavy lifting of this process for you. Even if you do decide to use one of these tools, it can be helpful to go through the process

yourself to better understand what's going on. We should also note that we'll be using Xcode 6.3.2 (with the command line tools installed) and there may be slight variations if you are using a different version of Xcode. To get started, let's review some important framework-related terminology.

Terminology

The iOS 8+ "embedded frameworks" are both embedded and dynamic. To distribute them properly, we must make them into fat frameworks containing all the necessary slices. What does this even mean!? Let's find out.

Static vs. Dynamic Frameworks

Static frameworks are linked at compile time. **Dynamic frameworks** are linked at runtime, and can be modified without relinking. If you've ever used a non-Apple framework prior to iOS 8, you were using a static framework which was compiled into the source code of your app. See Apple's "Dynamic Library Programming Topics" document for further discussion.

Embedded vs. System Frameworks

Embedded frameworks are placed within an app's sandbox and are only available to that app. **System frameworks** are stored at the system-level and are available to all apps. Apple reserves the ability to create system frameworks for itself; there is currently no way for third-party developers to create system frameworks on iOS.

Thin vs. Fat Frameworks; Slices

Thin frameworks contain compiled code for one architecture. **Fat frameworks** contain compiled code for multiple architectures. Compiled code for an architecture within a framework is typically referred to as a "**slice**." For example, if a framework had compiled code for just the two architectures used by the Simulator (i386 and x86_64), we would say it contained two slices. If we distributed this framework, it would only work when its consumer was built for the Simulator and would fail when the consumer was built for device. In order to ensure our frameworks can be consumed properly, we must also include the device architectures (currently arm64, armv7 and armv7s) for a total of five slices.

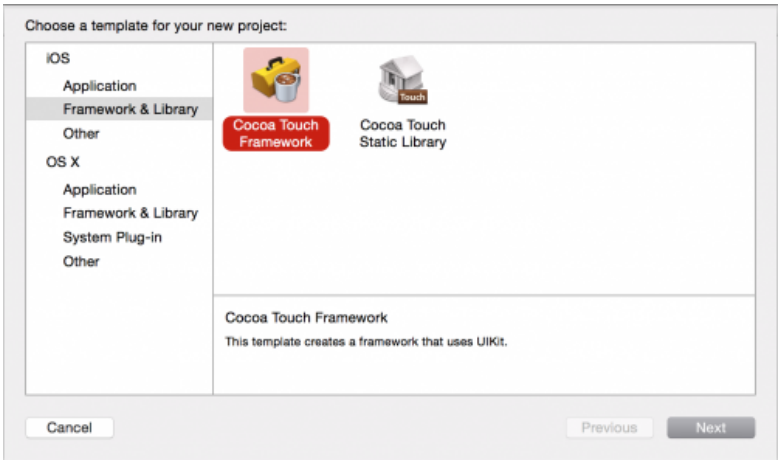
Well, that wasn't so bad. Next, let's create our embedded framework.

Creating the Framework

This is the easy part. Here we're going to create probably the world's simplest embedded framework.

- In Xcode, go to File > New > Project and select iOS > Framework & Library > Cocoa Touch Framework:

Categories

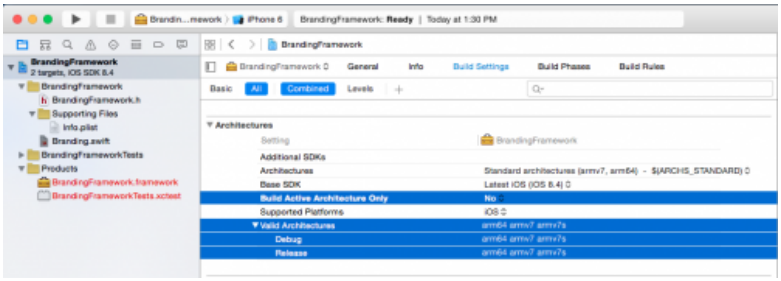


- Name your project “BrandingFramework”, enter your organization identifier, select Swift as your language and save it to disk.
- Add a Swift file to the project (File > New > File > Swift File) and name it “Branding.swift.” The following example will be in Swift, but you could do something similar in Objective-C.
- Add to this file your extension on UIColor:

```swift
extension UIColor {

    public class func brandedColor() -> UIColor {
        return UIColor.magentaColor()
    }

}
```

frameworks_blogpost_extension_on_uicolor.swift hosted with by GitHub view raw

- In the build settings for the BrandingFramework target, make sure that “Build Active Architectures Only” is set to “No” for both debug and release. As well, “Valid Architectures” should include “armv7s”, “armv7” and “arm64.” Depending on your version of Xcode, the “Standard Architectures” option will include all of these or just a subset of them. If you’re reading this in the future (relative to the summer of 2015), then there may be other architectures listed to support newer devices:

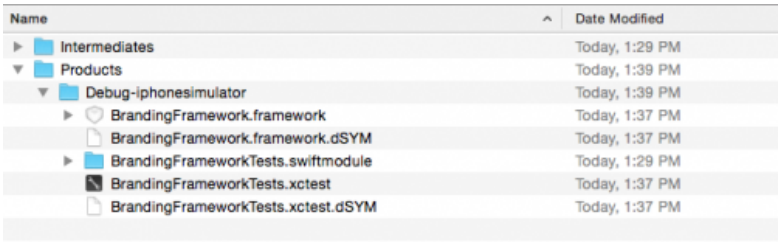


- Build the project against the Simulator. At this point, your framework will be created using only the slices for the Simulator (alternatively, if you built it for device it would only have the slices for the device). The default scheme for building is set to create a debug version, so this framework will be the debug version.
- Locate the .framework file that was created. It will be in your project’s Derived Data folder in one of the subfolders of Build/Products, but the exact folder will depend on your build settings. For example, when I built the debug version to the Simulator my .framework file was located at “/Users/Brett/Library/Developer/Xcode/DerivedData/MyFramework-hezzcnchdodkgegxmncdzjjhkkde/Build/Products/Debug-iphonesimulator/BrandingFramework.framework.” If built for device, the folder will have “iphoneos” in its name instead of “iphonesimulator”, and if built for release it’ll

Data
Default
Design
DevOps
High School
iOS
JavaScript
Life at Hootsuite
Microservices
Mobile
Open Source
Process
QA
React
Scala
Security
Testing
Words
ZooKeeper

Search

have “Release” instead of “Debug.” You can locate your Derived Data folder easily in Xcode by selecting Window > Projects in the menu, selecting your project,and clicking the arrow icon next to the Derived Data path displayed in the Projects window.



- Switch to the folder containing your framework in the Terminal. We can verify the architectures of the framework using the following command:

```
1  lipo -info BrandingFramework.framework/BrandingFramework
```

frameworks_blogpost_lipo_command.sh hosted with by **GitHub** view raw

- This should output something like this:

```
1  Architectures in the fat file: BrandingFramework.framework/BrandingFramework a
```

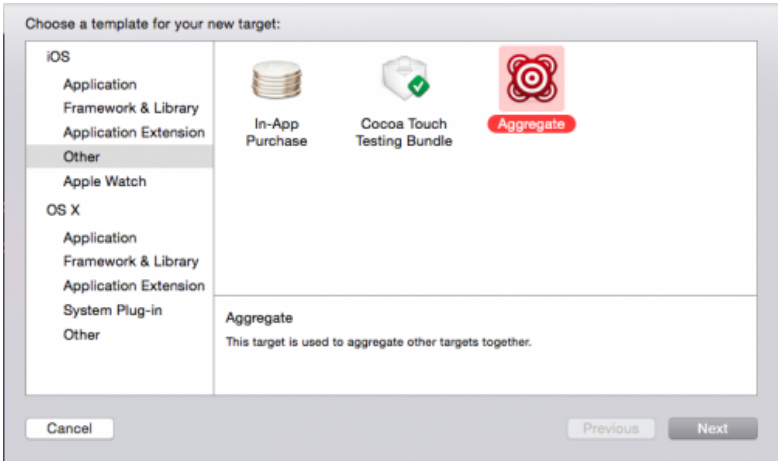
frameworks_blogpost_lipo_command_output.sh hosted with by **GitHub** view raw

This confirms that the framework has the Simulator’s (and only the Simulator’s) required architectures. If you repeat steps 5-7 and build to device, you will see a different set of architectures, which will be the device architectures. If you were to add either .framework file to another project right now, that project would only be able to build to either the Simulator or the device depending on which .framework file you used; it would not be able to build to both. In the next section, we will learn how to stitch together a .framework file containing both the Simulator and device architectures.

Creating the Fat Framework

As we’ve seen so far, in order to be able to consume our framework, we need it to contain the correct architectures that match with the build settings of the consumer. When the framework and the consumer of the framework are part of the same project, this isn’t a problem; Xcode will build the framework and the consumer using the same settings. A problem only arises when the framework and its consumer are in different projects. To solve this problem, we can use the same command line tool, lipo, that we used in the previous section. The basic idea is to use lipo to merge two frameworks together into one framework containing all the needed architectures. Of course, we don’t want to have to manually build for Simulator and device separately and then run lipo every time we want to build the framework, so we’ll use a build script to handle this for us. To automate the merging of the device and simulator slices into one framework, we’ll use a simple build script I wrote. It’s rather basic and geared toward demonstration purposes, but feel free to expand upon it and modify it as necessary.

- Create a new aggregate target in Xcode, by going to File > New > Target and selecting “Aggregate” from iOS > Other:



- Name the target “Build framework.”
- In the Build Phases section of the newly added aggregate target add a new run script phase.
- Copy and paste the following script into the editor area of your new run script. See the comments in the run script to understand what it’s doing.

```
# Merge Script

# 1
# Set bash script to exit immediately if any commands fail.
set -e

# 2
# Setup some constants for use later on.
FRAMEWORK_NAME="BrandingFramework"

# 3
# If remnants from a previous build exist, delete them.
if [ -d "${SRCROOT}/build" ]; then
rm -rf "${SRCROOT}/build"
fi

# 4
# Build the framework for device and for simulator (using
# all needed architectures).
xcodebuild -target "${FRAMEWORK_NAME}" -configuration Release -arch arm64 -arch a
xcodebuild -target "${FRAMEWORK_NAME}" -configuration Release -arch x86_64 -arch

# 5
# Remove .framework file if exists on Desktop from previous run.
if [ -d "${HOME}/Desktop/${FRAMEWORK_NAME}.framework" ]; then
rm -rf "${HOME}/Desktop/${FRAMEWORK_NAME}.framework"
fi

# 6
# Copy the device version of framework to Desktop.
cp -r "${SRCROOT}/build/Release-iphoneos/${FRAMEWORK_NAME}.framework" "${HOME}/De

# 7
# Replace the framework executable within the framework with
# a new version created by merging the device and simulator
# frameworks' executables with lipo.
lipo -create -output "${HOME}/Desktop/${FRAMEWORK_NAME}.framework/${FRAMEWORK_NAM

# 8
# Copy the Swift module mappings for the simulator into the
# framework.  The device mappings already exist from step 6.
```

```
cp -r "${SRCROOT}/build/Release-iphonesimulator/${FRAMEWORK_NAME}.framework/Modul

# 9
# Delete the most recent build.
if [ -d "${SRCROOT}/build" ]; then
rm -rf "${SRCROOT}/build"
fi
```

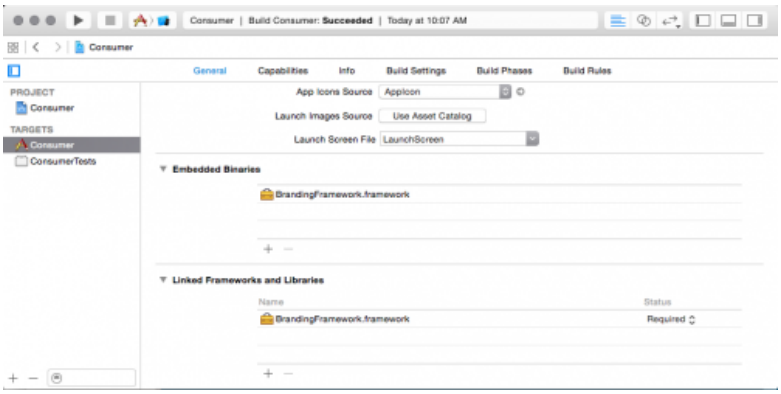
frameworks_blogpost_merge_script.sh hosted with by GitHub view raw

- Build the project using the target we just created. The BrandingFramework.framework file will appear on your Desktop. In practice you probably don't want to build the framework to the Desktop, but this makes things easy for our purposes. Next up, we'll add this newly created framework to our consumer project.

Consuming the Framework

Now that we have our fat framework ready, we can import and use it in a separate project.

- Create a new Single View Application project (File > New > Project then iOS > Application > Single View Application) in Xcode and name the project "Consumer."
- Drag the BrandingFramework.framework file from the Desktop into the Consumer target's "Embedded Binaries" section. In the resulting popup window ensure that "Copy items if needed" is checked. Xcode will automatically add the framework to the "Linked Frameworks and Libraries" section as well.



- Replace the contents of ViewController.Swift with the following:

```
import UIKit
import BrandingFramework

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = UIColor.brandedColor()
    }

}
```

frameworks_blogpost_consumer_view_controller.swift hosted with by GitHub view raw

- Run the app. The app should build and launch with a magenta background, which is coming from the framework. Hooray!

Working with Older Xcode Versions

If you're using Xcode 6.3.2 or newer, you can skip over this section. Versions of Xcode 6 prior to 6.3.2 suffered from a bug which caused validation to fail when an embedded framework contained the simulator architectures. Although this is no longer a problem, we'll cover an approach to solving this for posterity's sake. We again solve this by using lipo and some build scripts, but things get a little messy here. We'll use two run scripts, the first will create a backup copy of the fat framework and then replace the framework with a version that has had its simulator architectures removed. The second script will restore the backup of the fat framework after it has been used to the build the Consumer project.

- In the Consumer project, add a run script to the Consumer project and copy and paste the following script into the run script's editor area. This build script will strip out the simulator slices from the framework.

```
# Removal Script A

# 1
# Set bash script to exit immediately if any commands fail.
set -e

# 2
# Setup some constants for use later on.
FRAMEWORK_NAME="BrandingFramework"

# 3
# If there exists a backup version of the framework, restore it and remove backup
if [[ -d "${SRCROOT}/backup" ]]; then
    rm -rf "${SRCROOT}/${FRAMEWORK_NAME}.framework"
    cp -rf "${SRCROOT}/backup/${FRAMEWORK_NAME}.framework" "${SRCROOT}/${FRAMEWOR
    rm -rf "${SRCROOT}/backup"
fi

# 4
# Only perform the following steps when building for release on device
if [[ "${CONFIGURATION}" == "Release" && "${SDKROOT}" == *"iPhoneOS"* ]]; then

    # 5
    # Create backup copy of framework
        mkdir "${SRCROOT}/backup"
        cp -rf "${SRCROOT}/${FRAMEWORK_NAME}.framework" "${SRCROOT}/backup/${FRAM

    # 6
    # Strip out the unneeded architectures
    lipo "${SRCROOT}/${FRAMEWORK_NAME}.framework/${FRAMEWORK_NAME}" -remove "i386
    lipo "${SRCROOT}/${FRAMEWORK_NAME}.framework/${FRAMEWORK_NAME}" -remove "x86_
fi
```

frameworks_blogpost_removal_script_a.sh hosted with by GitHub view raw

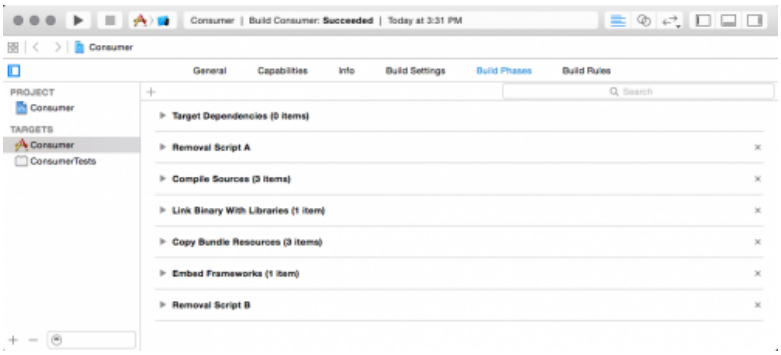
- Create a second run script and copy and paste the following script into the run script's editor area.

```
# Removal Script B

# 1
# Set bash script to exit immediately if any commands fail.
set -e

# 2
# Setup some constants for use later on.
FRAMEWORK_NAME="BrandingFramework"
```

```
# 3
# Restore backup copy of framework and remove backup
if [[ -d "${SRCROOT}/backup" ]]; then
    rm -rf "${SRCROOT}/${FRAMEWORK_NAME}.framework"
    cp -rf "${SRCROOT}/backup/${FRAMEWORK_NAME}.framework" "${SRCROOT}/${FRAMEWOR
    rm -rf "${SRCROOT}/backup"
fi
```

frameworks_blogpost_removal_script_b.sh hosted with by GitHub view raw

- Arrange the run scripts such that the first one is listed just below “Target Dependencies” and the second run script is listed just below “Embed Frameworks.:

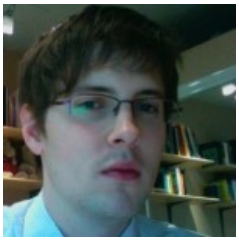


- Create an archive build of the Consumer Project (Product > Archive in the menu) and submit it for validation (Window > Organizer in the menu; select the project in the Organizer, press “Validate…” button and follow on screen instructions. The app should pass validation as the simulator architectures were removed from the framework during the build process.

Closing Thoughts

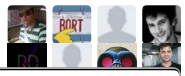
Today we covered how to create embedded frameworks, make them into fat frameworks and consume them from separate projects. We also demonstrated how to work around a related Xcode 6 bug in versions prior to Xcode 6.3.2. In practice, you would want to properly version the framework and have some sort of distribution mechanism in place (e.g., posting the framework on a web page or using git submodules instead of building and manually copying the framework file from the Desktop as we did). Although the DIY approach can be fun and a great way to learn, if it seems like a little too much work for you, Carthage and CocoaPods are both excellent tools that can do much of the heavy lifting. By using embedded frameworks when working on multiple closely related apps we can avoid much of the code duplication we might otherwise require.

About the Author Brett is an iOS developer. He spent 30 seconds trying to come up with an about the author blurb and then got distracted by something else. Follow him on Twitter @bstover.



Posted in: Code, iOS, Mobile · Tagged: frameworks, ios, swift

30 comments

Sign in **44 people listening** 

+ Follow | Share | **Post comment as...**

Newest | Oldest | Top Comments

angel rose May 19, 2016

I have created a dynamic framework in swift which consumes other external framework like Alamofire, AlamofireImage , XCGlogger etc. I used carthage to using this frameworks inside my framework project. Then i created a my Framework and used it in an application project. It works fine when i am using my framework in an app project in same workstation. But when i used handover this framework to my client he started getting error "Missing required module for Alamofire,Alamofireimage,XCGlogger".

Like Reply

angel rose May 19, 2016

I have gone deep into the structure of my newly created framework and found that it contains a folder named framework which list all the external frameworks which i have used. But the external framework structure is missing the module folder. Anybody have any idea that what i have missed in it? Thanks in advance.

Like Reply

angel rose May 21, 2016

@bstover do you have any notes which can be refered to make framework with external frameworks like alamofire reffered

Like Reply

Av1234 Mar 24, 2016

I've had problems when compiling frameworks with XCode 7.x and then consuming the .framework on XCode 7.y. I think that the swift "stuff" isn't compatible. Have you experienced this?

Like Reply

RavenApp Mar 9, 2016

Seems like the code snippets are no longer on the site.**@bstover**

Like Reply

RavenApp Mar 9, 2016

@RavenApp @bstover

Nevermind... My company intranet restricted it.

Like Reply

imtayaz Mar 3, 2016

Great Tutorial!

But I have an issue if you can help will be great help

cp: /Users/XXXX/Documents/Mobile Development/XXXXXX/XYZ Framework/V1.2/XYZFramework/build/Release-iphonesimulator/XYZFramework.framework/Modules/XYZFramework.swiftmodule/: No such file or directory

Like Reply

_luisvip Apr 21, 2016

@imtayaz have you been able to compile it?

Like Reply

VinceDavis Feb 15, 2016

I've looked at 4 tutorials on creating a universal framework and yours is by far the best. It actually works. Thanks for putting this together.

Like Reply

aravindnarothe Feb 2, 2016

Hey Brett, first of all a Big Thanks for this great tutorial.

I followed the steps in the tutorial to create a framework for m application. But I am facing an issue. My framework is an objective C only code. When I run **frameworks_blogpost_merge_script.sh**, I am getting an error. The error is in line #8 and it says {FRAMEWORK_NAME}.swiftmodule not found. Is it safe to remove this piece of code from the script?

Like Reply

uptscs Jan 21, 2016

With all the steps followed I am getting Bitcode enable issue: bitcode bundle could not be generated because '/Users/Test.framework/Test' was built without full bitcode. All frameworks and dylibs for bitcode must be generated from Xcode Archive or Install build for architecture armv7

clang: error: linker command failed with exit code 1 (use -v to see invocation)

I have changed the xcodebuild script a bit to enable Bitcode:

xcodebuild OTHER_CFLAGS="-fembed-bitcode" -target Test build

but still produced framework is not bitcode enabled.

Like Reply

vinodram487 Feb 3, 2016

@uptscs hi I am also facing with same issue if you find any solution let me know. vinodram.487@gmail.com this is my mail ID

Like Reply

dmvjs May 20, 2016

@vinodram487 @uptscs BITCODE_GENERATION_MODE needs to be set to bitcode as a User-defined Build Setting

Like Reply

alexzavatone Jan 20, 2016

What do we do now on Xcode 7.x where an aggregate target is no longer an option?

Like Reply

alexzavatone Jan 20, 2016

WOW. Apple moved the iOS > Other section to a top level platform below iOS, watchOS, tvOS and OS X.

In Xcode 7.x, you have to scroll down to the end of the left pane and there is Other as a top level item and inside it is Aggregate. : / Thanks Apple, for not documenting this.

Like Reply

bstover Jan 20, 2016

@alexzavatone Yeah Apple likes to rearrange things in those panels frequently. Thanks for posting the solution you found. :)

Like Reply

gautam987 Dec 11, 2015

Hi. So, I've been creating static frameworks in Objective C for sometime now. Now I wish to create an embedded or distributed framework in Swift. Part of the problem is that I keep some code exposed and some hidden. While it is very much possible with Objective C and static libs, is it possible with Swift and distributed frameworks ?

I'm unable to find any info on this. Thanks

Like Reply

rdspinz Nov 30, 2015

Hi, I am trying to replicate this using objective-c, and I managed to get the .framework file onto the desktop, and lipo-info yields "Architectures in the fat file: MyFramework are: armv7 armv7s i386 x86_64 arm64". However when I try to build on the simulator I get this message:

Undefined symbols for architecture x86_64:

"_OBJC_CLASS_$_MyFramework", referenced from: objc-class-ref in ViewController.o objc-class-ref in AppDelegate.o ld: symbol(s) not found for architecture x86_64 clang: error: linker command failed with exit code 1 (use -v to see invocation)

This also happens when I attempt to build to my device, except it says obviously says arm64. So I'm thinking this means it won't build anywhere. Any idea why this is?

Like Reply

rdspinz Nov 30, 2015

Ok, so the solution for this is to add the all the .m files to the compile sources in the target, however, no I get the same error for all of the AFNetworking classes I use. How do I get the cocoapod files in the compile sources?

Like Reply

VladimirTolstikov Nov 27, 2015

Hi there! Related question: my newly created Swift Framework depends on several pods (as Alamofire) which I installed with CocoaPods. So, question is: is it possible to distribute one framework (fat or not -- no matter) in the end without other programmers need to install those dependencies? (they also may use other versions of same libs, want easy install and don't want them to have incompatibility issues with their pods)

Like Reply

rdspinz Nov 30, 2015

@VladimirTolstikov How did you get the aggregate target to build with Cocoapods? I'm getting "lPods is not an object file (not allowed in a library)" for arm64, armv7 and armv7s. As far as using the cocoapods with dependencies, you can prefix the files as described here: **http://blog.sigmapoint.pl/avoiding-dependency-collisions-in-ios-static-library-managed-by-cocoapods/**

Like Reply

VladimirTolstikov Nov 30, 2015

@rdspinz sorry, I didn't try steps from that article above so can't help with aggregated target, but big thanks for article you dropped.

Btw, about my original question I found next possible solutions/digging ways:

- **https://github.com/CocoaPods/cocoapods-packager** (also it says class-names mangling is possible to remove possible dependencies collisions if needed)

- you can specify vendored_frameworks in pospec to bundle dependencies with your library/pod (not sure I got it fully right, may be wrong, not tried)

- old-fashion way: include all sources of dependencies libs into your sources and compile all in (most obvious way, if nothing else will help)

Like Reply

featherless Nov 4, 2015

Hey gang! Just want to give a heads up here that we shouldn't be encouraging OS X or iOS devs to be including Swift code in their .framework files while the language is still in flux. It can result in the .framework being unusable if a third party attempts to use the framework with a newer version of Swift.

My personal recommendation for now if you want to distribute a binary .framework with Swift code is to distribute the Swift parts alongside as code. Anything that you want to keep "secret" should be written in Objective-C for now.

Like Reply

bstover Nov 5, 2015

@featherless Hey featherless. Good point, but I'd note that this applies only under certain circumstances. In general, I'd say there are three use cases for writing/distributing frameworks: 1) As part of an open source project distributed to anyone (e.g. Alamofire), 2) as part of an internal project distributed internally, and 3) as part of a closed source project distributed to anyone.

In the first two cases, the standard approach would be to host the framework's code in a git repo and have the users of the framework build the framework themselves (either manually or by using a tool such as Carthage). In these cases the framework would/could always be built using the same versions of Xcode and Swift. I say "would/could" b/c if you're building the framework independently of each build of the consumer app you will encounter this problem at least once per update to Xcode and need to rebuild the framework. In these cases, there's no reason not to use Swift.

In the third case though, you're right that distributing the .framework with Swift code could quickly turn into a hassle, since the user of the framework cannot rebuild it as needed. Presumably the framework authors could post an updated version of the framework with every Xcode update, but that doesn't really seem like the right solution. I'll add a note in the article to call out this case.

Like Reply

JasonR1 Nov 4, 2015

First,

Thank you. This is THE most complete explanation of this I have seen. I have seen several posts that cover individual aspects of this issue (Fat slices, removing slices) but none that cover the process start to finish, and none that include the backing up and restoring of the framework.

The "bug" (I think it may be a feature) where you can't submit frameworks with simulator slices is back in Xcode 7.1 (7B91b).

Like Reply

bstover Nov 5, 2015

@JasonR1 Thanks Jason! Glad to help. :) I'll add a note to the article to reflect that this bug/feature is back. Interestingly, the Carthage guys/girls have had a radar out for ages on this issue that hasn't been updated: **http://www.openradar.me/radar?id=6409498411401216**. I hope it is a bug that gets fixed by Apple as it would simplify the process for everyone and there's no reason I'm aware of that they couldn't strip out the unneeded slices on their end.

Like Reply

amit0656 Oct 24, 2015

hi can we include storyboards,view controller in frameworks? if yes can you share an example?

Like Reply

bstover Oct 28, 2015

@amit0656 Yes you can. A view controller is a class like any other class and so can be included. Storyboards and xibs can be included, but you have to take care to load them from the correct bundle. Accessing the storyboard by

supplying nil as the bundle will cause a crash b/c the system will attempt to load the storyboard from the consumer app (e.g. "let storyboard = UIStoryboard(name: "MyStorboard", bundle: nil)" will crash). Instead you need to ensure the system knows to load the storyboard from the framework by supplying the bundle associated with the framework. An easy way to do this is: "let storyboard = UIStoryboard(name: "MyStorboard", bundle: NSBundle(forClass: MyClassContainedInTheFramework))" where "MyClassContainedInTheFramework" is the name of a class contained in the framework.

Like Reply

AmitDhawan Oct 18, 2015

Hi, When i submit one of the apps using the fat framework build for both simulator and device the itunes app store submission validation fails citing reaseons to remove "i386 architecture slices".

Like Reply

bstover Oct 20, 2015

@AmitDhawan Hi Amit: There's three options to solve this: (1) Ensure you've followed the steps in the "Working with Older Xcode Versions" section above. The scripts used in this section remove the i386 and x86_64 slices from the binary. (2) Ensure you're using the latest version of Xcode (currently 7.0.1). In my testing, the bug which causes the app store submission validation failure has been fixed as of Xcode 6.3.2, but it's possible it has come back. (3) Use Carthage [**https://github.com/Carthage/Carthage**] for importing the framework. Setup instructions for Carthage can be found on their github page. Carthage includes its own script for dealing with removal of the i386 and x86_64 slices. In general, Carthage is my preferred method for importing frameworks and is what we're using for this purpose in the Hootsuite app. It's good to know how to do these things manually (as the article describes), but for production code, it's nice to have the power of a more automated system (e.g., Carthage) with an active dev community on your side.

Like Reply

← Previous Post

Next Post →