

Open in app ↗

Sign up

Sign in

Medium

🔍 Search



Building iOS framework with dependencies.



Max Kalik · Follow

Published in ITNEXT

7 min read · Feb 23, 2023



Listen



Share



In my current iOS position, I'm deeply involved in building a framework for iOS games. I wouldn't write this article if this framework weren't unusual. What do I mean by that — unusual? Usually, frameworks encapsulate only the source code without dependencies and other libraries, and of course, I don't mean Foundation and other build frameworks in iOS. I mean the dependencies installed using cocoapods which for some business reason could be included in your framework manually.

Recently I wrote an interesting detailed article/analysis about how to create an iOS XCFramework with cocoapod dependencies, so if it's your case check this out:

<https://hackernoon.com/cocoapod-as-xcframework-with-dependencies>

In this article, we will build only a manual framework with other frameworks inside statically. I admire it's a pretty rare case but businesses sometimes dictate their terms where we — developers should implement a workable solution quickly.

Final product



The final product — MyFramework.framework

The task is to create a simple framework only for real devices with dependencies that can be installed via cocoapods. Where we could use this framework? In Unity! You can read my article about this [here](#). But let's move forward and start to create our final product.

Podfile

Assume, we created a new iOS Framework project. I named it MyFramework and in the terminal run `pod init` in the root of the project folder, so we have Podfile where we can start. Let's configure it:

```
platform :ios, '14.0'

source 'https://cdn.cocoapods.org/'
source 'https://github.com/passbase/cocoapods-specs.git'
source 'https://github.com/passbase/microblink-cocoapods-specs.git'

target 'YourFramework' do
  pod 'CropViewController', '2.6.1'
  pod 'Kingfisher', '7.5.0'
  pod 'Intercom', '14.0.6'
  pod 'Frames', '4.0.4'
  pod 'Passbase', '2.7.0'
```

end

I didn't use some abstract cocoapods, I took the real ones on purpose to show you the real process which you could repeat by yourself. At the first sight of the list of dependencies, you could say that they are picked randomly. And yes and no, because all these dependencies are different ones, for example:

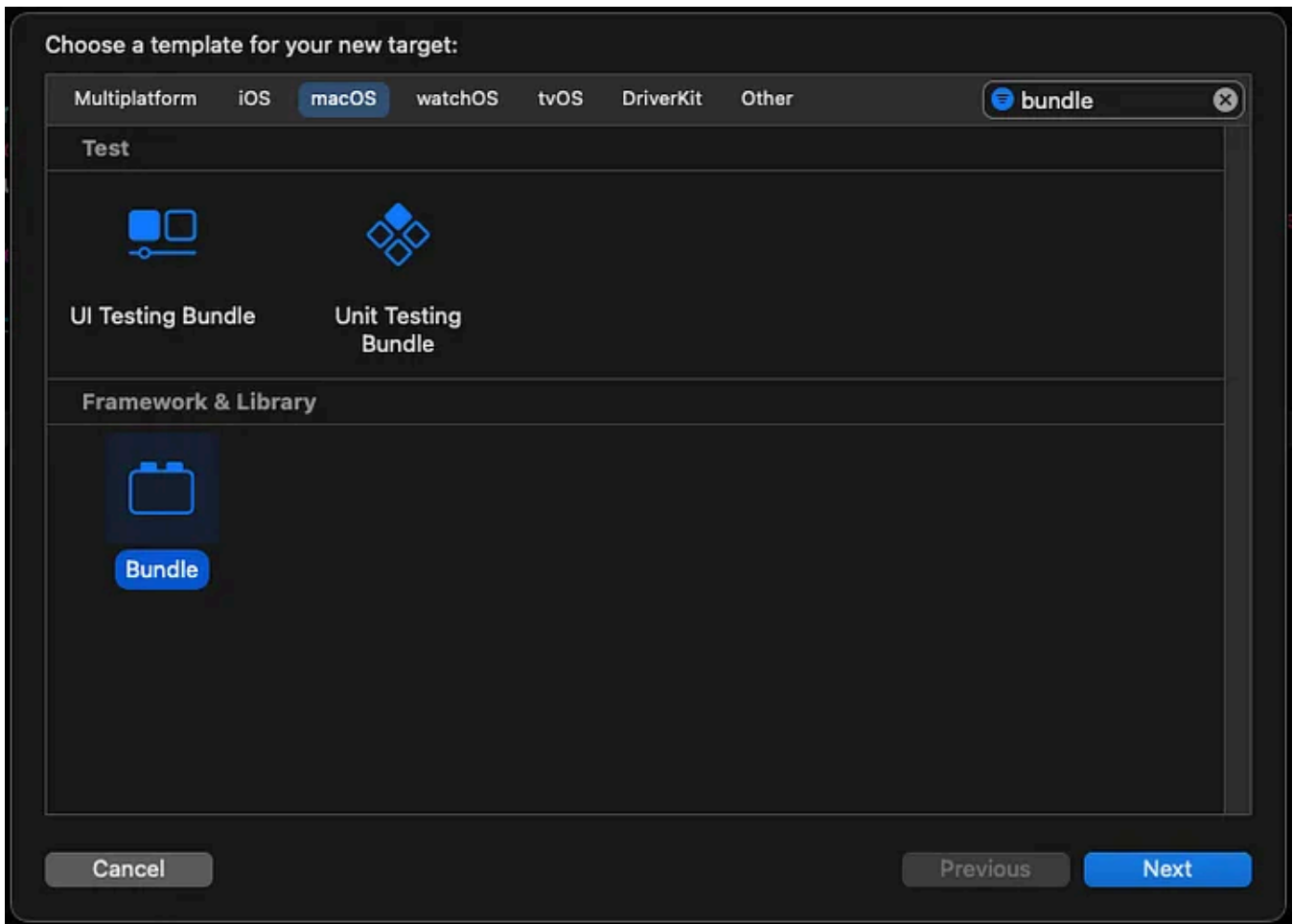
- `Kingfisher` — it's just a piece of source code
- `CropViewController` — also with source code but with a bundle of the resource
- `Intercom` is interested to be in this list because it's an `xcframework` itself
- `Frames` dependency is here because it's a combination of source code and some sub-dependencies inside including `xcframeworks`
- `Passbase` is one of the most interesting specimens because to install this guy we need to use specific source paths including an additional source path for `Passbase`'s dependency called `Microblink`. All of them are `xcframeworks`.

Testing all these various dependencies we will create a framework for manual installation.

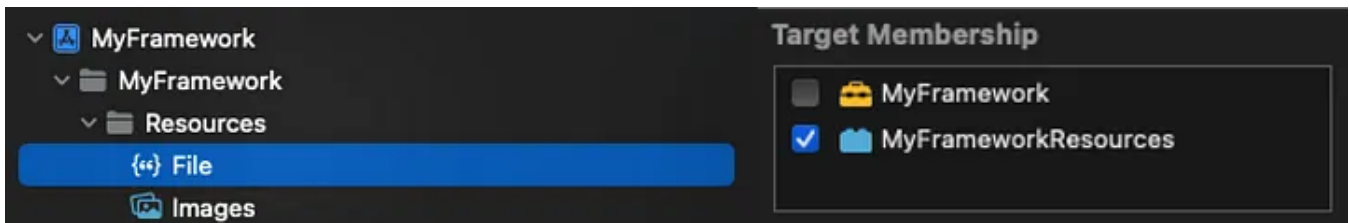
Resources

If a framework uses resources (images, video, or other files), then we need to prepare a separate target for it and this process looks a little bit like a workaround. Let me describe it step-by-step:

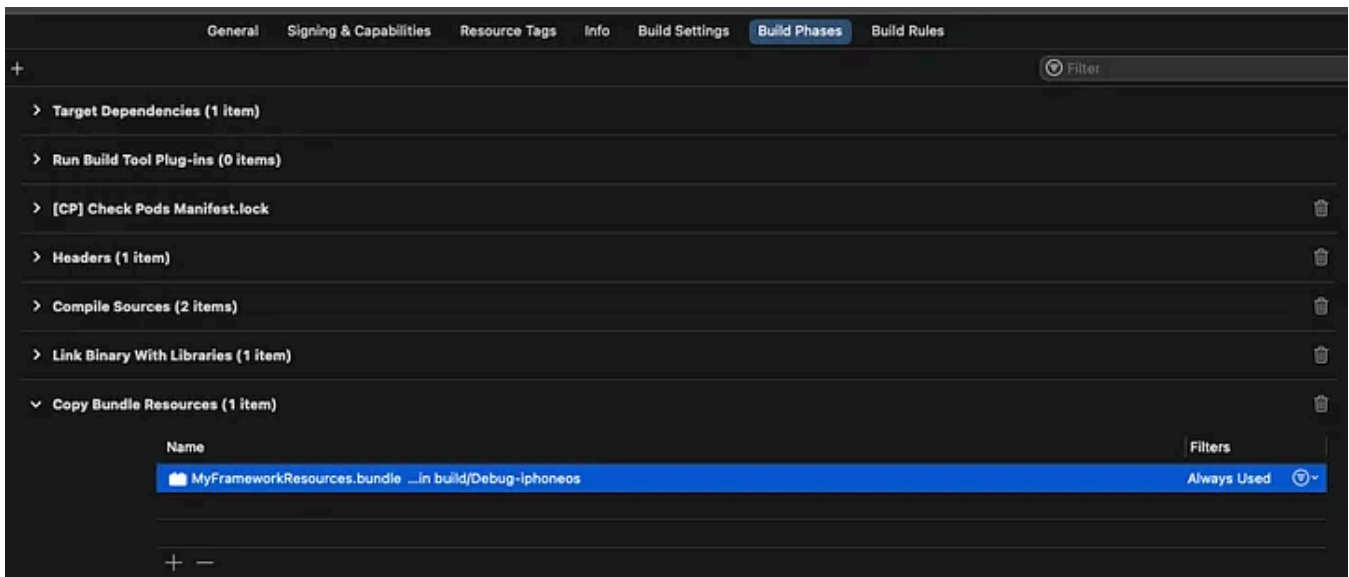
1. Open File / New / Target. Choose macOS and in the Framework & Library section choose Bundle.



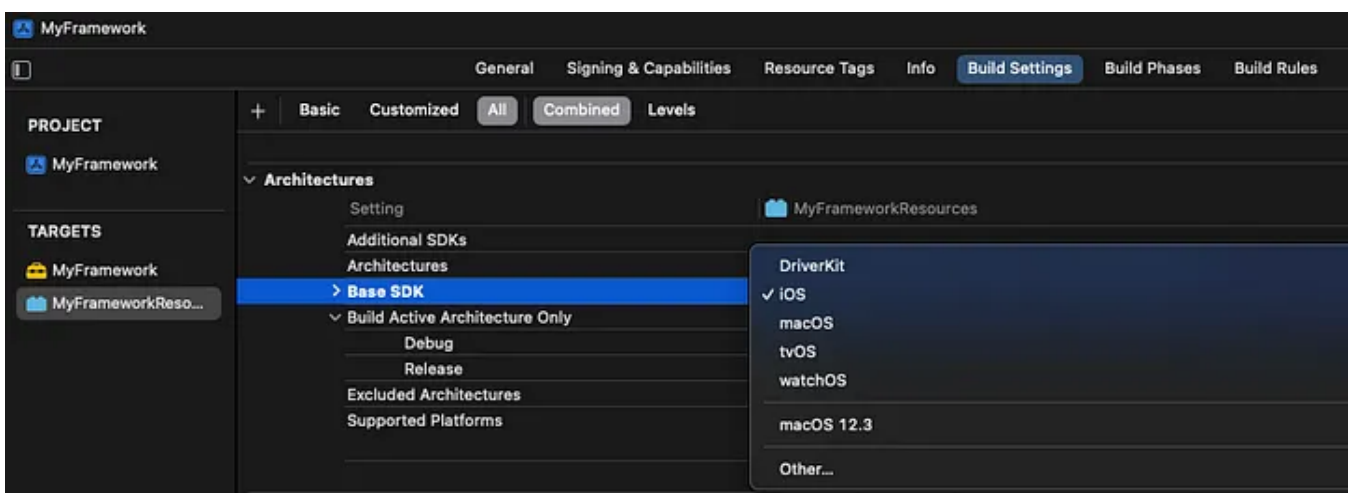
2. For now, make sure the resource files belong to this target.



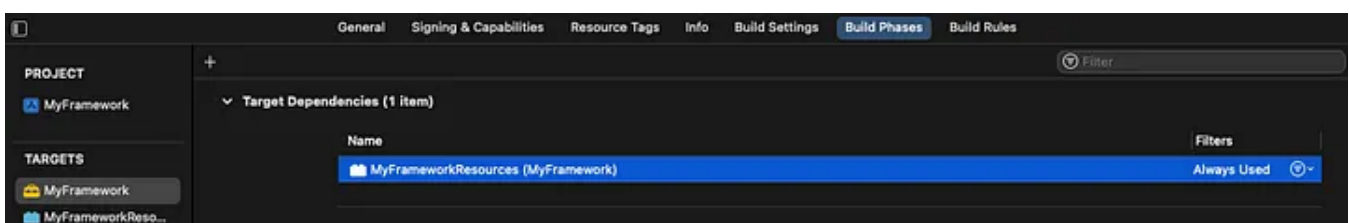
3. Go to Project target / Build phases / Copy Bundle Resources and remove and exclude everything there except the bundle.



4. The last step is not obvious but necessary. Do you remember we created this bundle for MacOS but we need it for iOS? So, go to Build Settings of the framework bundle and change Architectures / Base SDK to iOS.



5. Also in Framework target Build Phases we need to include our bundle as a target dependency.

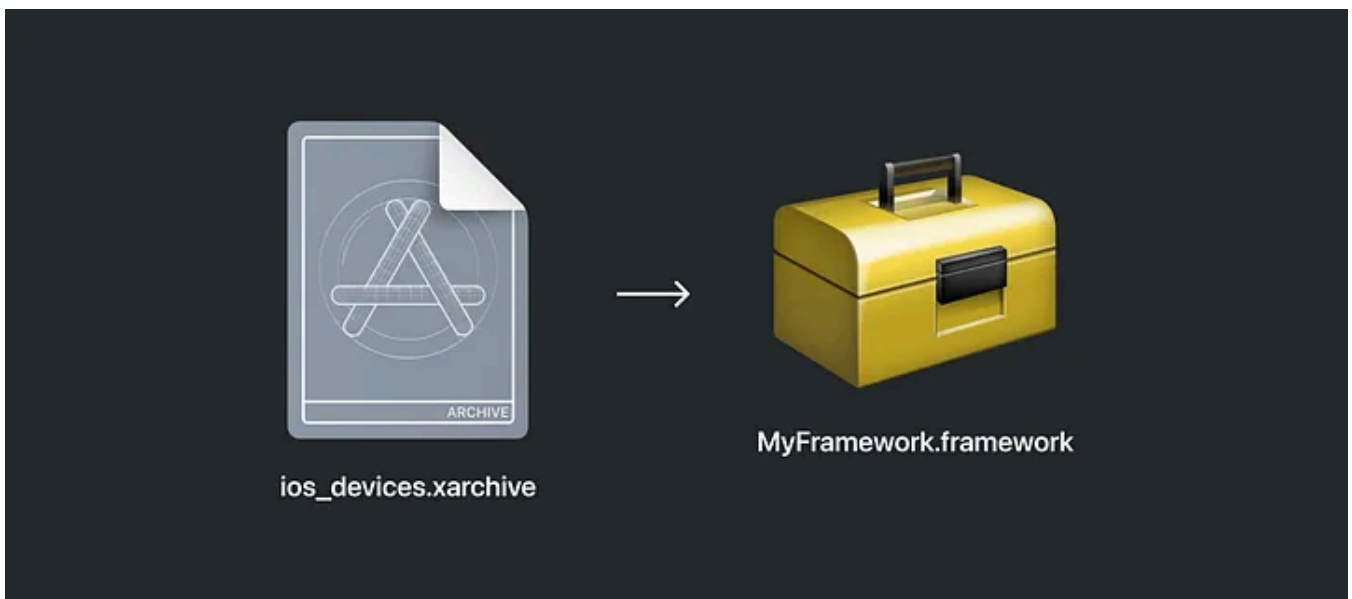


Build

Xcode has a powerful CLI to make an archive using `xcodebuild` with a set of flags:

```
xcodebuild archive \  
-workspace MyFramework.xcworkspace \  
-scheme MyFramework \  
-configuration Release \  
-sdk iphoneos \  
-archivePath archives/ios_devices.xcarchive \  
BUILD_LIBRARY_FOR_DISTRIBUTION=YES \  
SKIP_INSTALL=NO \
```

This simple script creates an archive. Open it with the right mouse button using “Show content” and find a fresh-backed framework in the products.



Now the framework is ready to be tested. You can just drag this framework to your test iOS project structure. Don't forget to Embed and Sign this framework. Choose a real device where we are going to test our app with this framework.

The result will be not so good. The project will give you an error — Build failed.



Build Failed

Dependencies not found

The reason is our dependencies are not found, they weren't included in our framework. So let's solve this problem.

Including dependencies to a framework

Since we install our dependencies using cocoapod we need to update Podfile. By default, the pods are installed with dynamic linking but we need to install them with static linkage.

```
platform :ios, '14.0'

source 'https://cdn.cocoapods.org/'
source 'https://github.com/passbase/cocoapods-specs.git'
source 'https://github.com/passbase/microblink-cocoapods-specs.git'

target 'YourFramework' do
  use_frameworks! :linkage => :static

  pod 'CropViewController', '2.6.1'
  pod 'Kingfisher', '7.5.0'
  pod 'Intercom', '14.0.6'
  pod 'Frames', '4.0.4'
  pod 'Passbase', '~> 2.7.0'

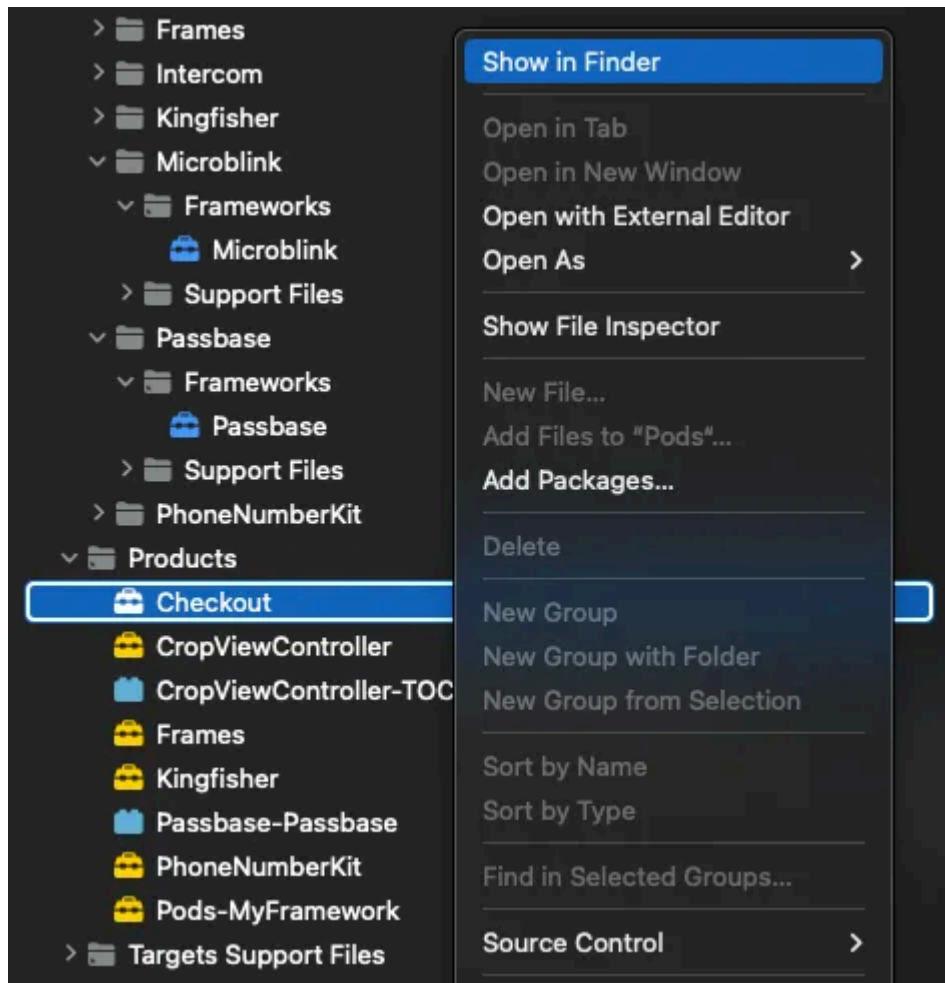
end
```

So the main update of our Podfile is updating this particular line: `use_frameworks!` to `use_frameworks! :linkage => :static`

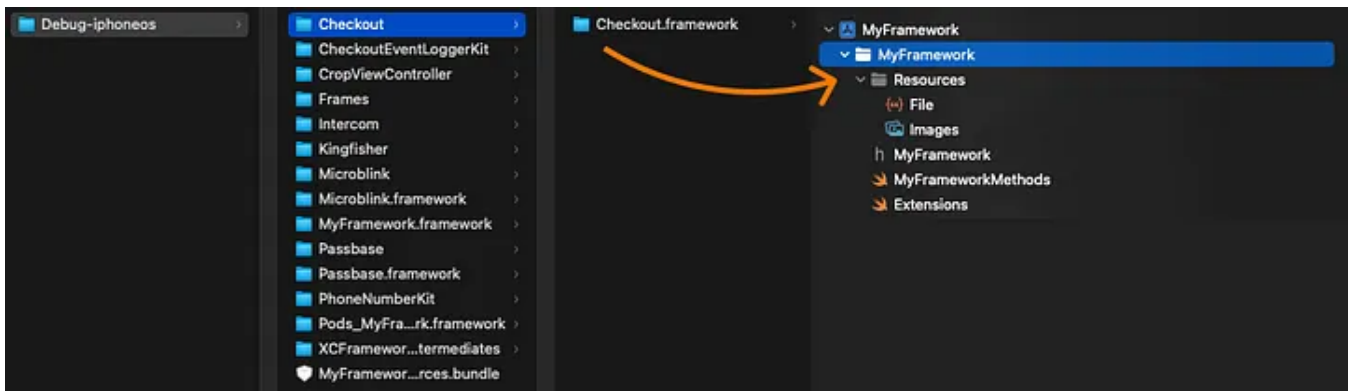
The next step is not very obvious (I would say even questionable) because we need to include all dependencies' frameworks to our framework literally by copying them from the Products folder of Pods into our project.

If doing this manually, you need to Run the framework project first, then you will get the Products folder with frameworks from each pod including their bundles.

To get there you can right-click on any framework in the Products folder and press Open in Finder.



From each dependency folder, you need to find a folder with a name something like `Name.framework`, and drag and drop it to your Framework project. By the way, we don't need to include frameworks' resource bundles in our project because they will be included automatically.

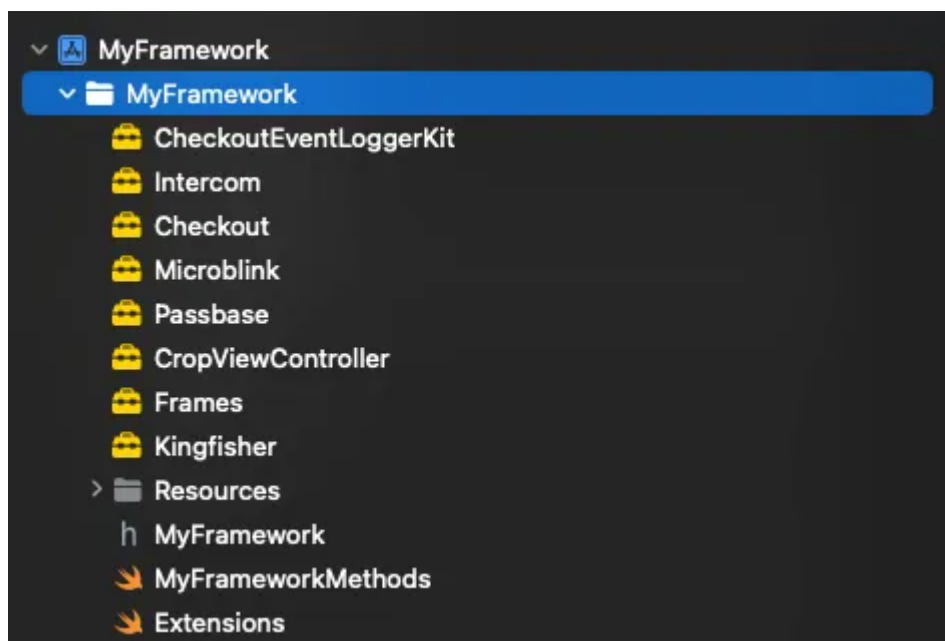


But, even here we can come across with small nuance. As you remember some of our dependencies are XCFrameworks, and the folder where supposed to be these frameworks are empty.

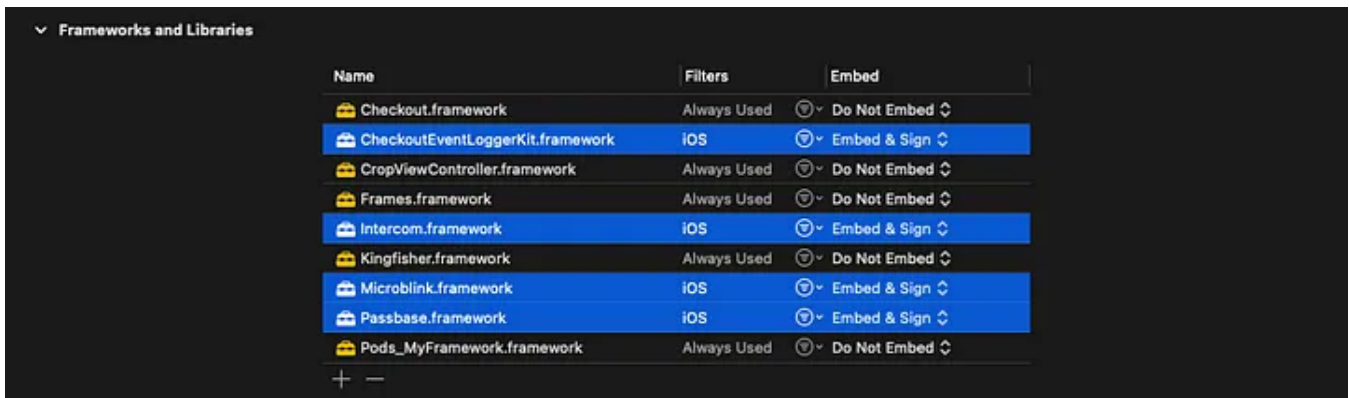
Among all the product folders you can find a specific one called:

`XCFrameworkIntermediates` where you can find all frameworks from XCFrameworks.

So, your project structure should look like this:



Another important thing about frameworks from XCFrameworks is — we need to embed and sign them in our Framework.



After all these manual manipulations, you can build your framework again and get the final product which you can test. If you did everything correctly, then your project with the new framework will run successfully and all methods from the framework will work without failures.

Potential problems

Traditionally, instead of a conclusion, which nobody cares about, I think it will be better to list some potential problems and how to solve them. Let's get started.

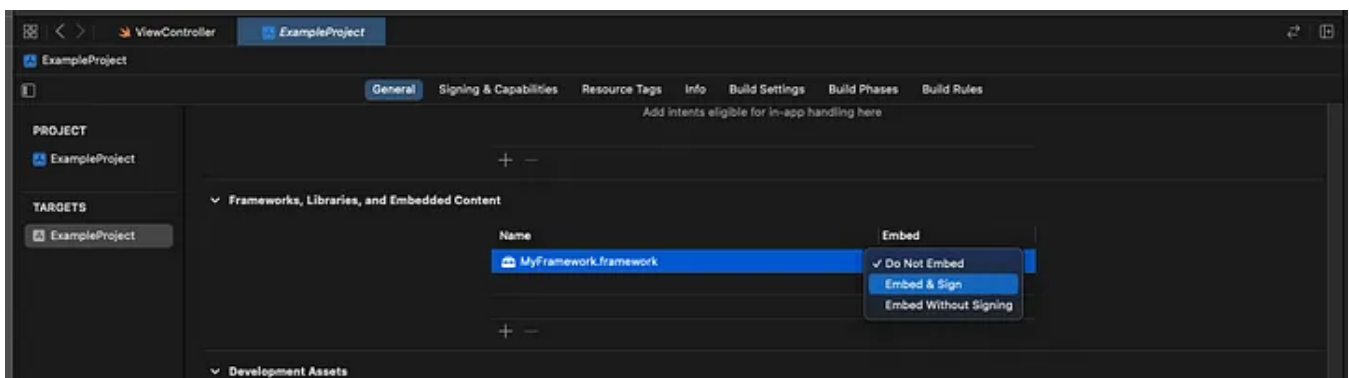
Problem 1: Library not loaded.

Basically, it is an error always together with this message:

```
dyld[17871]: Library not loaded.  
'@rpath/MyFramework.framework/MyFramework'
```

Solution:

One of the most often errors. We are all people and we can just forget to sign the framework.



Problem 2: Error: Compiled module was created by a newer version of the compiler.

In your build script of `xcodebuild`, you most likely forgot to add this flag:

```
BUILD_LIBRARY_FOR_DISTRIBUTION
```

Problem 3: Build failed. A client uses the same dependencies which your framework already uses.**Solution:**

Briefly: Versions of the dependencies in a framework and in an app should be the same.

Detailed: Let's say you build your framework with `cocoapods Alamofire 5.5.0` and a client app will use your framework and `Alamofire 6.0.0`, so this combination will cause a problem. In other words, you cannot build your app, and you won't understand what is going on, because the error will be about symbols, or something else.

```
pod 'Intercom', '2.0.0'  
pod 'Alamofire', '5.5.0'
```

Problem 4: The path to your product is not found (the product folder in the archive is empty)**Solution:**

Check flag: `SKIP_INSTALL=NO` in your build script — it will install your framework in the archive, in other words, your product folder won't be empty.

Problem 5: Property is not a member type of class MyFramework.MyFramework

```
.../MyFramework.swiftinterface:6:34: error: 'SomeProperty' is not a member type  
public var direction: MyFramework.SomeProperty { get set }
```

Solution:

The best way to avoid this is if the names of modules in a framework are different and don't coincide with the name of the framework itself. So, if you just started your framework from scratch — keep your public module name different. For example, our framework is called `MyFramework.xcframework` so the name of your general module could be `MyFrameworkMethods`

iOS

Xcode

Framework

Xcframework

Cocoapods




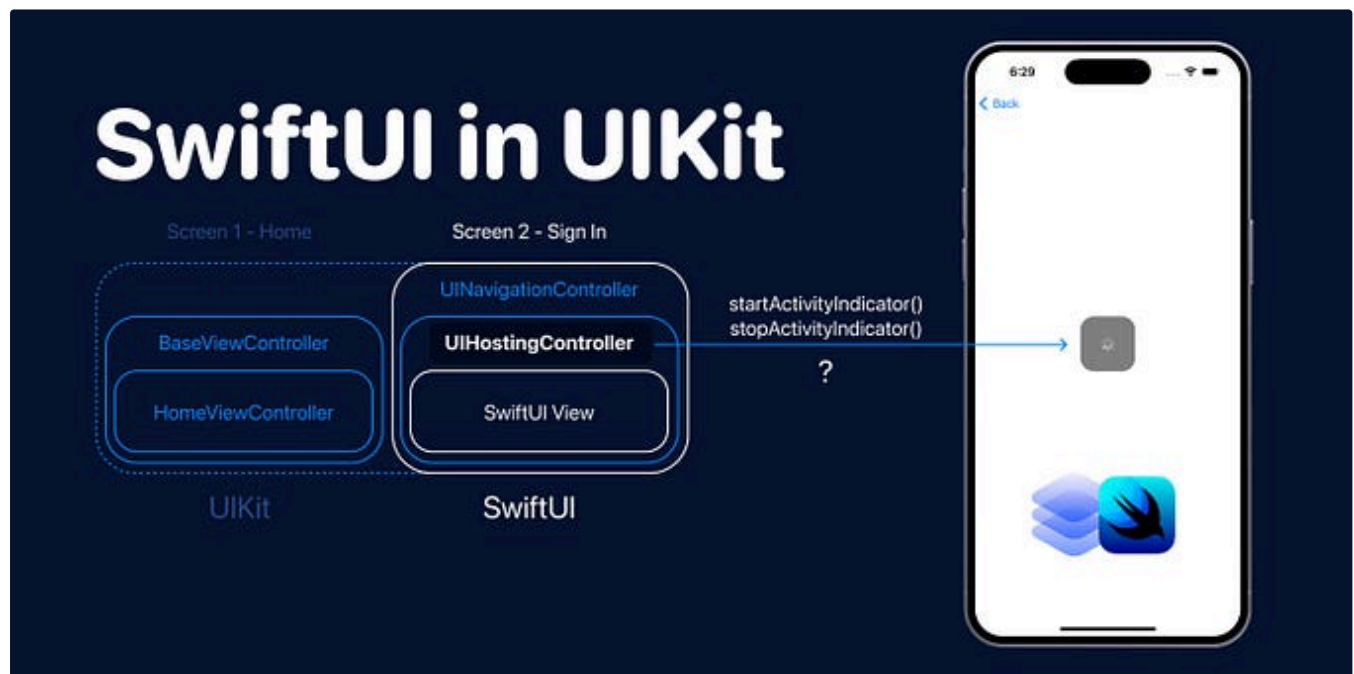
Follow

Written by Max Kalik

95 Followers · Writer for ITNEXT

iOS Developer

More from Max Kalik and ITNEXT

 Max Kalik in Geek Culture

Integrating SwiftUI into UIKit project

How to launch SwiftUI feature in UIKit MVVM+Coordinator design pattern without prejudice existing codebase.

Oct 17, 2022 🖱️ 271



Piotr in ITNEXT

Finally, a viable Helm Replacement

Fully backwards compatible

Jul 23 🖱️ 428 💬 5



Piotr in ITNEXT

5 Advanced Kubernetes Operators Every DevOps Engineer Should Know About

Simplify Infrastructure Management

Jul 16  520  5



Max Kalik in Level Up Coding

Implementing address autocomplete using SwiftUI and MapKit

Let's create a small iOS application with address autocompletion feature using the latest SwiftUI framework.

Dec 2, 2022  465



See all from Max Kalik

See all from ITNEXT

Recommended from Medium



Gizem Duman in Mobillium

Step-by-Step Guide to Creating a Local SPM Package in Xcode 15

Modularity and efficient dependency management are key in modern application development, and Apple provides a solid solution with the...

Feb 15  57



Zafar Ivaev in Level Up Coding

How to Modularize an iOS App

Leveraging XcodeGen and Clean Architecture principles

★ Sep 3, 2023 🖱 462 💬 2



Lists



Apple's Vision Pro

7 stories · 72 saves



Tech & Tools

17 stories · 279 saves



Icon Design

36 stories · 375 saves



Business

41 stories · 127 saves

 Riccardo Cipolleschi in Better Programming

Create Your First Swift Package Command Plugin

How to write and debug a Swift Package Command plugin

★ Sep 27, 2022 🖱 252 💬 1



 Steven Curtis

Understanding Intrinsic Content Size in iOS Development

Because it's tricky

 Jul 29  9




 Samuel Flender in Towards Data Science

How to become a command-line wizard

The most useful computer science class you've probably never taken

★ Nov 7, 2022  714  10



 Cihat Gündüz in Better Programming

Migrating to The Composable Architecture (TCA) 1.0

Sharing my learnings and my code structure after migrating my app to the vastly modernized APIs of TCA 1.0.

★ Apr 4, 2023 🖱 356



See more recommendations