Using Swift in an Existing Objective-C Project

LA Swift #4 Meetup

Gustavo Barcena

Why use Swift at all?

- Type safety
- Performance
- You like concise code
- Modularize your code
- You want to be a polyglot
- It’s still super shiny

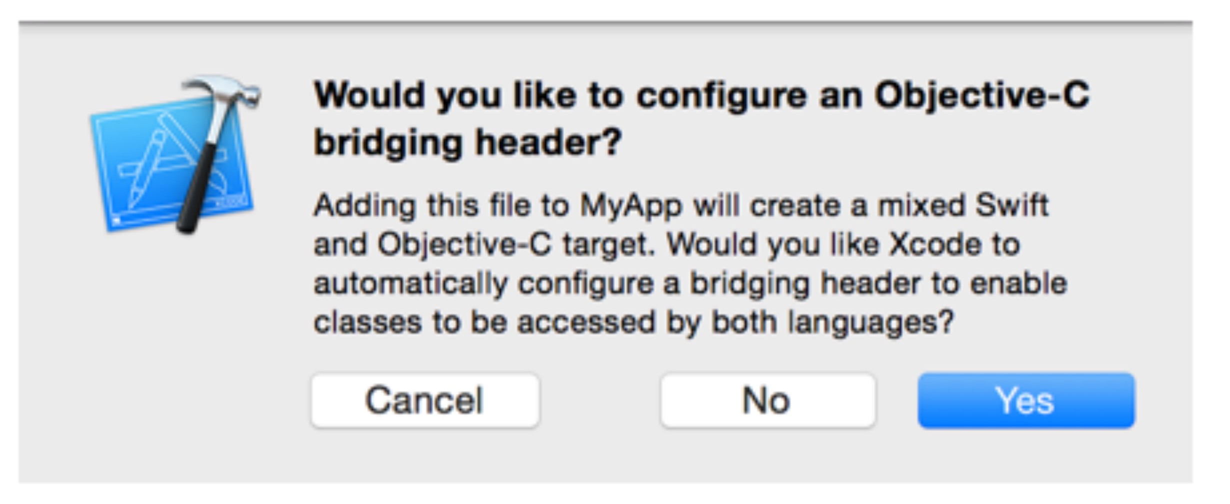
So I should rewrite everything in Swift then?

NO!

There is huge value in battle tested code.
Don't give it up

What you need to know

- **Xcode will do most of the work for you!**
  - Sometimes things go wrong but we can get to that later
- **The Bridging Header (Objective-C -> Swift)**
  - This is so you can manually expose code



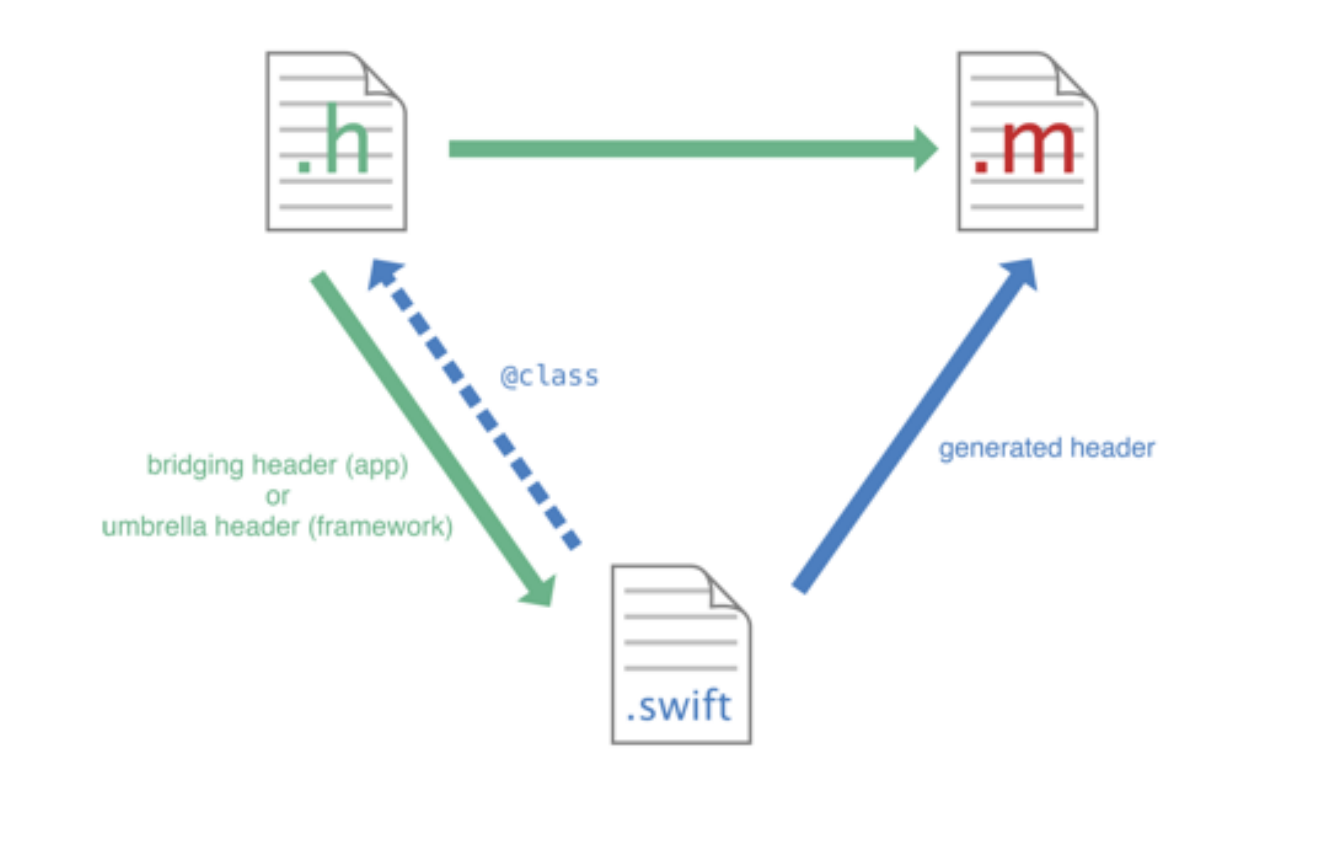
Would you like to configure an Objective-C bridging header?

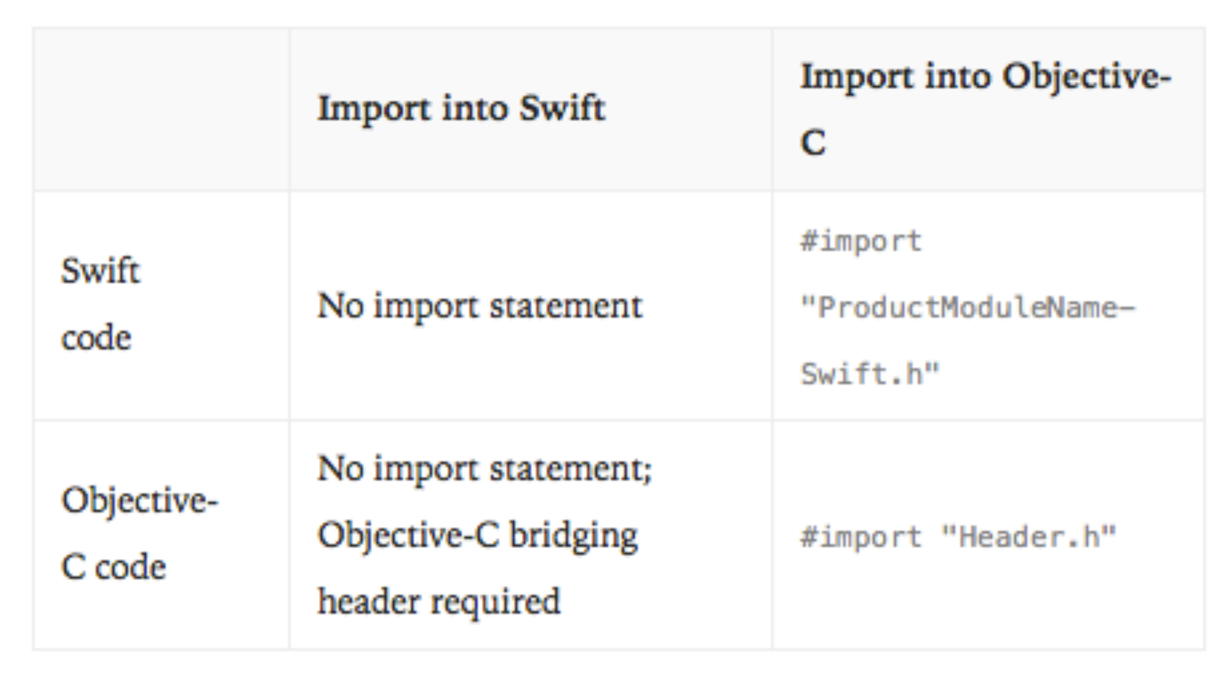
Adding this file to MyApp will create a mixed Swift and Objective-C target. Would you like Xcode to automatically configure a bridging header to enable classes to be accessed by both languages?

What you need to know

- **What about the other way around? (Swift -> Objective-C)**
  - Generated header
    - #import “\(productName)-Swift.h”
    - Xcode will manage it for you
- **Swift classes just show up in Interface builder**

Everything you just said but as an image.





	Import into Swift	Import into Objective-C
Swift code	No import statement	`#import "ProductModuleName-Swift.h"`
Objective-C code	No import statement; Objective-C bridging header required	`#import "Header.h"`

DEMO

Setting Up Interoperability

What do I get
(beyond the basics)

- **CF Interoperability**
  - using Core Foundation and Core Graphics is very natural in swift
  - CF can take advantage of ARC in swift
- **Blocks in Objective-C -> Optional Swift Closures**
  - Trailing closure syntax
  - Note: Variables are mutable by default not copied
    - Like using __block modifier

Demo

Objective-C-isms to Swift-isms

Gotchas

- **Swift doesn’t import complex macros**
- **Swift imports global constants and simple macros as Swift global constants**
- **NSSelectorFromString() of a Swift method**
  - Because of how namespacing, you need to use a fully qualified name of a swift method
    - MyApp.myMethod
- **NSError ** -> NSErrorPointer**
  - works pretty much the same way

Gotchas

- **Implicitly Unwrapped Optionals (e.g. text!)**
    - Objects have to exist or are never-nil
    - In order for something not to exists we have to make it Optional
        - WWDC: Optionals are a generalized notion of nil
    - In Objective-C, technically everything can not exist
    - If you know it will exist, you can just use the object.
    - From iBook: If the value is missing, a runtime error occurs. As a result, you should always check and unwrap an implicitly unwrapped optional yourself, unless you are sure that the value cannot be missing.

But I don’t like…

- **I can no longer use Key-Value Observing**
  - Use the dynamic keyword (it will also enable swizzling)
  - it is basically turning the method call into msgSend
- **What my Swift method is called in Objective C**
  - Swift naming the method differently from the parameter name
    - (rect:CGRect) -> (toRect rect:CGRect)
  - @objc() modifier
    - Remember to add a colon (:) wherever a parameter follows a selector piece. (e.g. @objc(fromRect:toRect:))

Why isn’t this sh*! working?!?!?!

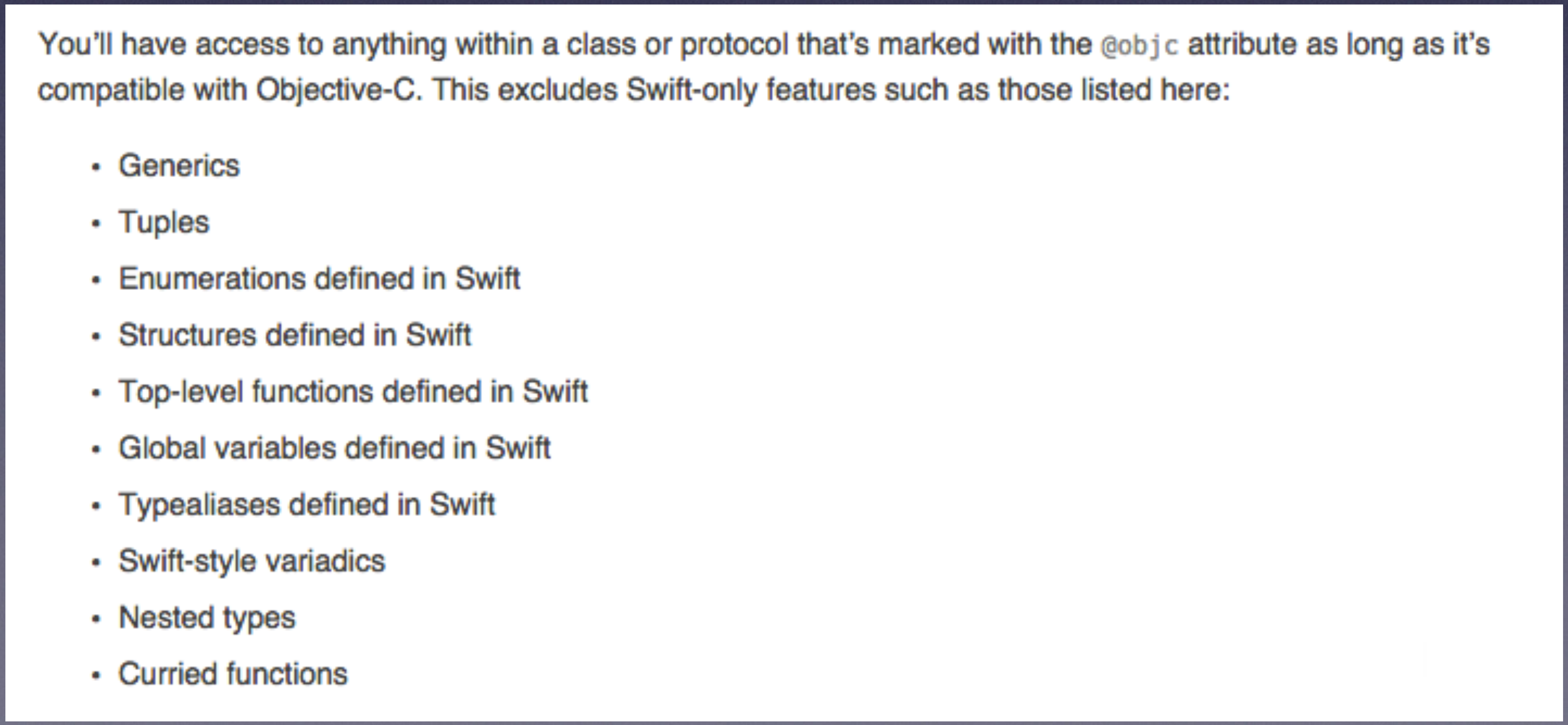
- **My Swift class isn’t showing up in Objective-C**
  - you are not inheriting from a objective c class
  - You can also add @objc modifier
- **I can’t find my -Swift.h file**
  - You don’t have a product name set or its not what you think it is
  - From Apple: “However, if your product name has any non-alphanumeric characters, such as a period (.), they are replaced with an underscore (_) in your product module name. If the name begins with a number, the first number is replaced with an underscore.

Demo

Transitioning Code

Why isn’t this sh*! working?!?!?!

- **My Objective C code is just not showing up in Swift**
  - From Apple: “Under Build Settings, make sure the Objective-C Bridging Header build setting under Swift Compiler - Code Generation has a path to the header. The path should be relative to your project, similar to the way your Info.plist path is specified in Build Settings. In most cases, you should not need to modify this setting”
- **Some things just don’t transfer**



You’ll have access to anything within a class or protocol that’s marked with the `@objc` attribute as long as it’s compatible with Objective-C. This excludes Swift-only features such as those listed here:

- Generics
- Tuples
- Enumerations defined in Swift
- Structures defined in Swift
- Top-level functions defined in Swift
- Global variables defined in Swift
- Typealiases defined in Swift
- Swift-style variadics
- Nested types
- Curried functions

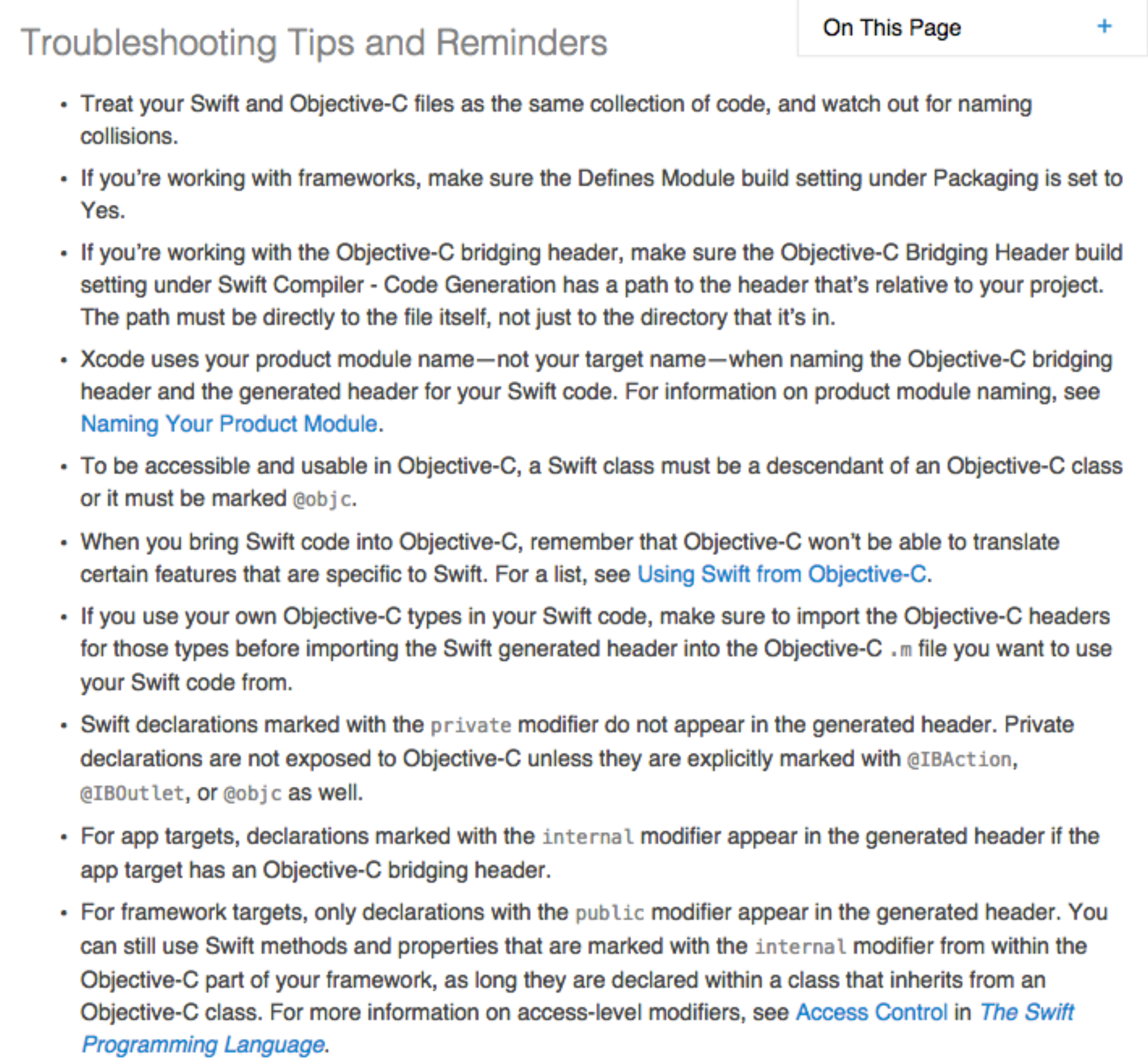
Why isn’t this sh*! working?!?!?!

- **dyld: Library not loaded: @rpath/libswift_stdlib_core.dylib**
- 1st Attempt
  - Embedded Content Contains Swift Code is 'Yes'
  - Runpath Search Paths is '$(inherited) @executable_path/Frameworks'
  - Cleaning & rebuilding
- 2nd Attempt
  - Restarting Xcode, iPhone, computer
  - Cleaning & rebuilding

Why isn’t this sh*! working?!?!?!

- **dyld: Library not loaded: @rpath/libswift_stdlib_core.dylib**
    - 3rd Attempt (This one is a real pain but it is what worked for me)
        - Creating new certificate and regenerating provision profile
        - Cleaning & rebuilding
    - Resources
        - http://stackoverflow.com/questions/24002836/dyld-library-not-loaded-rpath-libswift-stdlib-core-dylib

Why isn’t this sh*! working?!?!?!

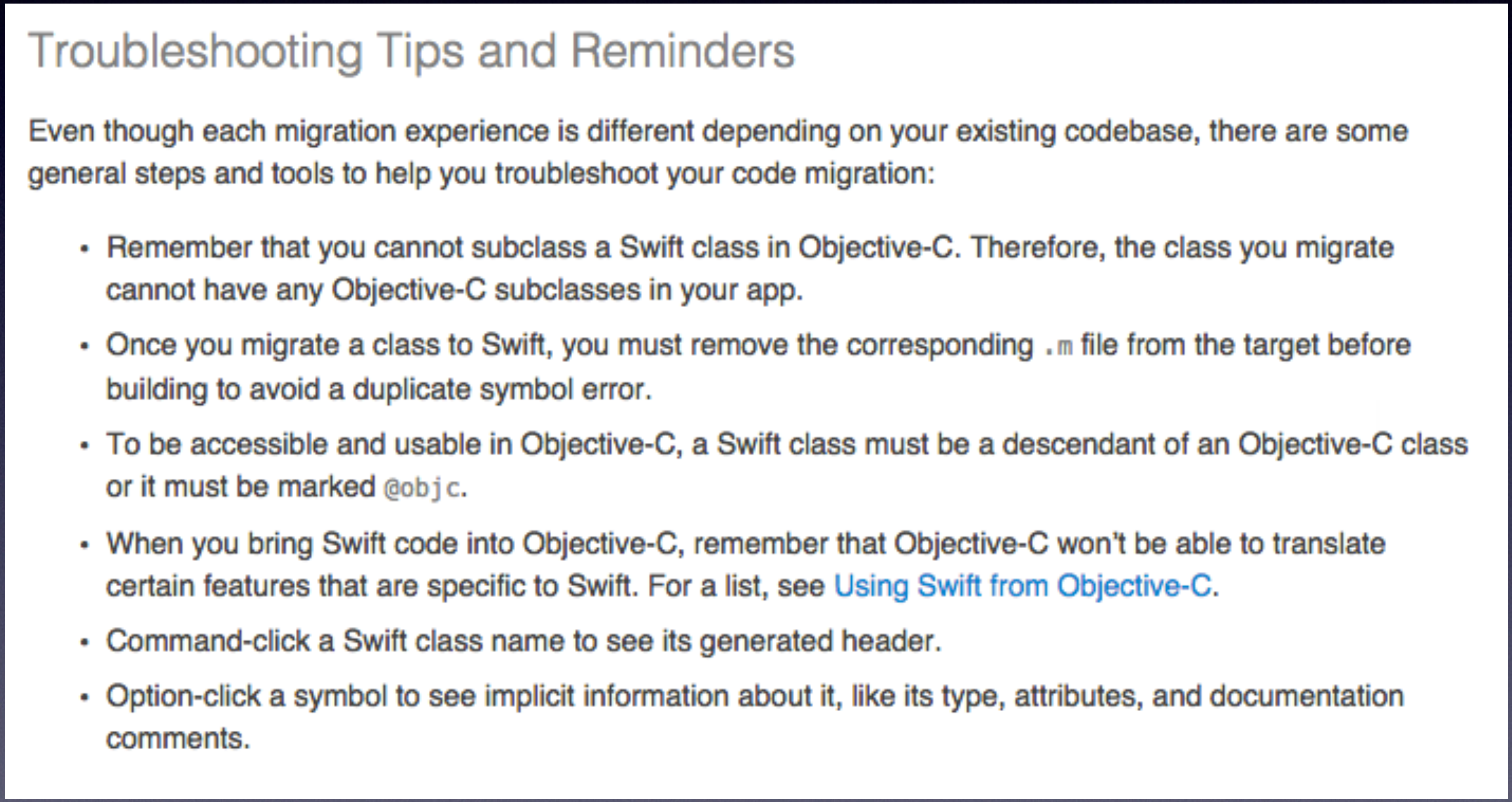


Troubleshooting Tips and Reminders

- Treat your Swift and Objective-C files as the same collection of code, and watch out for naming collisions.
- If you’re working with frameworks, make sure the Defines Module build setting under Packaging is set to Yes.
- If you’re working with the Objective-C bridging header, make sure the Objective-C Bridging Header build setting under Swift Compiler - Code Generation has a path to the header that’s relative to your project. The path must be directly to the file itself, not just to the directory that it’s in.
- Xcode uses your product module name—not your target name—when naming the Objective-C bridging header and the generated header for your Swift code. For information on product module naming, see Naming Your Product Module.
- To be accessible and usable in Objective-C, a Swift class must be a descendant of an Objective-C class or it must be marked `@objc`.
- When you bring Swift code into Objective-C, remember that Objective-C won’t be able to translate certain features that are specific to Swift. For a list, see Using Swift from Objective-C.
- If you use your own Objective-C types in your Swift code, make sure to import the Objective-C headers for those types before importing the Swift generated header into the Objective-C `.m` file you want to use your Swift code from.
- Swift declarations marked with the `private` modifier do not appear in the generated header. Private declarations are not exposed to Objective-C unless they are explicitly marked with `@IBAction`, `@IBOutlet`, or `@objc` as well.
- For app targets, declarations marked with the `internal` modifier appear in the generated header if the app target has an Objective-C bridging header.
- For framework targets, only declarations with the `public` modifier appear in the generated header. You can still use Swift methods and properties that are marked with the `internal` modifier from within the Objective-C part of your framework, as long they are declared within a class that inherits from an Objective-C class. For more information on access-level modifiers, see Access Control in *The Swift Programming Language*.

https://developer.apple.com/library/ios/documentation/swift/conceptual/buildingcocoaapps/MixandMatch.html#//apple_ref/doc/uid/TP40014216-CH10-XID_87

Why isn’t this sh*! working?!?!?!



Troubleshooting Tips and Reminders

Even though each migration experience is different depending on your existing codebase, there are some general steps and tools to help you troubleshoot your code migration:

- Remember that you cannot subclass a Swift class in Objective-C. Therefore, the class you migrate cannot have any Objective-C subclasses in your app.
- Once you migrate a class to Swift, you must remove the corresponding `.m` file from the target before building to avoid a duplicate symbol error.
- To be accessible and usable in Objective-C, a Swift class must be a descendant of an Objective-C class or it must be marked `@objc`.
- When you bring Swift code into Objective-C, remember that Objective-C won’t be able to translate certain features that are specific to Swift. For a list, see Using Swift from Objective-C.
- Command-click a Swift class name to see its generated header.
- Option-click a symbol to see implicit information about it, like its type, attributes, and documentation comments.

https://developer.apple.com/library/ios/documentation/swift/conceptual/buildingcocoaapps/Migration.html#//apple_ref/doc/uid/TP40014216-CH12-XID_74

Resources

- **WWDC Videos** (https://developer.apple.com/videos/wwdc/2014/)
  - Integrating Swift with Objective-C (http://asciiwwdc.com/2014/sessions/406)
  - Swift Interoperability In Depth (http://asciiwwdc.com/2014/sessions/407)
- **Using Swift with Cocoa and Objective-C**
  - Online Documentation
    - https://developer.apple.com/library/ios/documentation/swift/conceptual/buildingcocoaapps/index.html#//apple_ref/doc/uid/TP40014216-CH2-XID_0
  - iBook
    - https://itunes.apple.com/us/book/using-swift-cocoa-objective/id888894773?mt=11

Done