Building Hybrid Frameworks

My name is Cecilia, the past 4.5 years I have been working in Spotify for various projects. The area covers infrastructure, continuous integration, feature design and implementation, Swift upgrading etc. Today I'd like to focus on one thing, that is libraries and frameworks.

Cause the knowledge is useful in any circumstances during the development. I hope that you will be able to bring something back to your work after this workshop.

In our day to day work, we are using other people's code in our code base. Some of us are managing other people's code using third party dependency tools like cocoapods, swift package manager etc.

With or without these helpful and useful tools, it is interesting to know how to build a library yourself and how to use them manually. When you have a solid ground, it is much easier to understand on the big scale level. For example, Spotify has its own dependency management system, It is just a wrapper to stripe away the manual setup for developers. When you understand what is needed to be done, it is also easier to understand what the script is doing.

The big questions that are related to libraries and frameworks are:

- How to create your own libraries and frameworks in Swift and Objective-C
- How to add your libraries/frameworks into others' projects.

You can google these questions and internet has answers to them.

But if you dig one step deeper and there are more specific things that sometimes internet can not help. In this workshop, I will cover something that is basic, I will also try to show something that is hard to google. Therefore, the workshop will cover the following parts:

- How to create a framework with both Objective-C and Swift
- How to use Objective-C in a Swift framework?
- How to expose Objective-C APIs from your Swift framework?
- How to add a static library dependency to your framework?

Since this workshop is targeted to iOS developers at all levels, at any point, if you already have the knowledge of that section, feel free to jump to the next.

Version: Xcode 11, iOS 13.

Differences between static libraries and dynamic frameworks

First, a little theory, if you feel that you would like to read theory later, you can jump over to the hands-on section.

Static libraries	**Dynamic frameworks**
Code only, in binary form .a	Code, resources, header, source, binary.
Link statically	Link dynamically
Bigger binary size	smaller binary size
Slower launch time	Faster launch

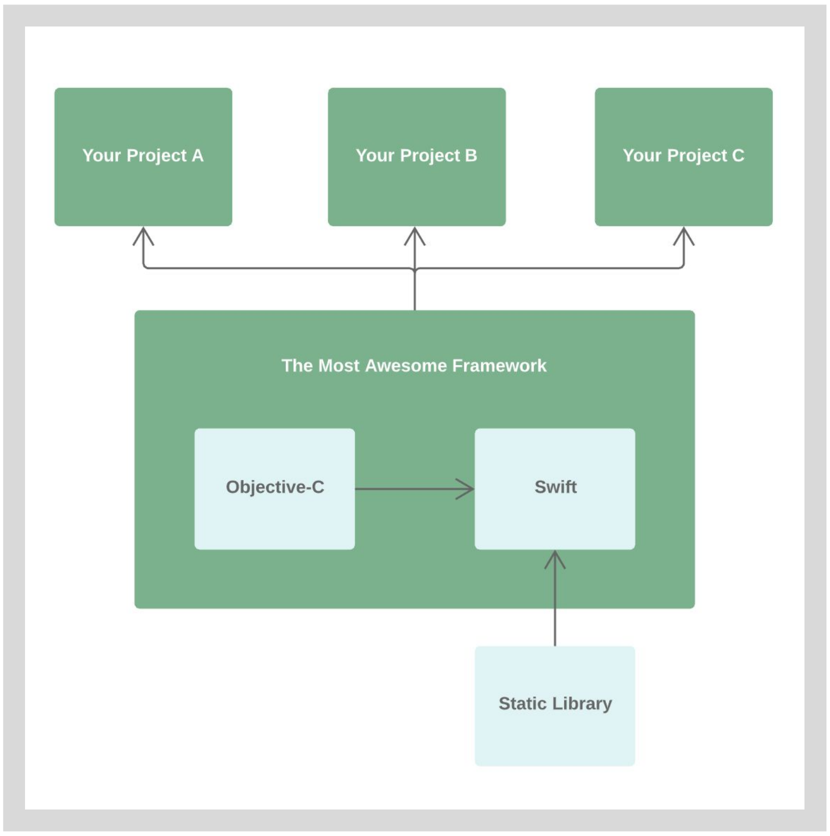
Example: static library vs dynamic framework hierarchy

Static Library
- lib*.a
- include /$(PROJECT_NAME)/.a

Dynamic Framework
- *.framework
    - Headers/ *.a
    - Info.plist
    - Modules/module.modulemap
    - * executive file

Getting Started - Creating a Framework

Let’s start by building this awesome framework that can be used in various projects.



1. The Most Awesome Framework is called Components.
2. In the Components, there is a TrackRowCell which is written in Swift.
3. The TrackRowCell has an image, the image implementation is written in Objective-C.
4. The Components frameWork uses ColorToken Static Library.

This section includes the following parts:

- Creating the framework **Components**
- Adding TrackRowCell implementations in Swift

Open Xcode, **Create a New Xcode Project** ▸ **iOS** ▸ **Framework**▸ **Next**, give *Components as* **Product name** and choose *Swift* as **Language** *and* **Create.**

Build and Run.

Go to **File** ▸ **New** ▸ **File** ▸ **iOS** ▸ **SwiftUI View** ▸ **Next**, name it **TrackRowCell** *and* **Create.**

In **TrackRowCell**, remove these boilerplate code.

```
  struct TrackRowCell: View {
      var body: some View {
          Text("Hello World!")
      }
 }
```

Add the following code.

```swift
public struct TrackRowCell: View {
    public init() {}
    public var body: some View {
        HStack {
            Image(systemName: "book")
            VStack {
                Text("三線の花").font(.title)
                Text("BEGIN").font(.subheadline)
            }
        }
    }
}
```

Basically, it implements a view that contains one image and two texts.
Now, build and Run.

Using the Framework into Your Project

In Xcode, go to **File** ▸ **New** ▸ **Project** ▸ **iOS** ▸ **Single View App** ▸ **Next**, give *Tracks as* **Product name** and choose *Swift* as **Language,** *SwiftUI* as **User Interface**, **Next** ▸ **Create.**

Build and Run the **Tracks** project.

Close down the **Components.xcodeproj,** Go to **where Components.xcodeproj** is located, drag **Components.xcodeproj** into the **Tracks Project navigator** lane**.**

Tap **Tracks Target,** build and run the **Tracks** project.

Go to build settings, Select **Tracks Target,** in **General** ▸ **Frameworks, Libraries, and Embedded Content** ▸ **+** **,** tap **Components.framework,** then **Add.**

Build and Run.

Go to **ContentView.swift,**

```swift
import Components
```

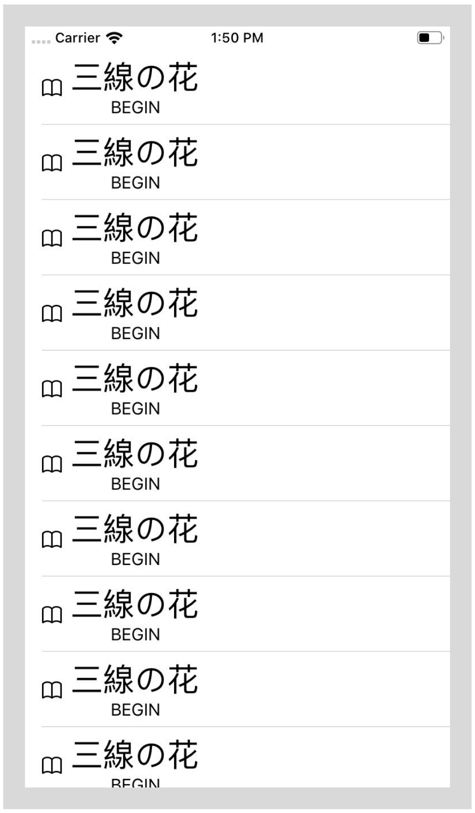
In the body implementation, remove this line

```swift
Text("Hello World!")
```

Add the following lines.

```
List(0..<10) { _ in
    TrackRowCell()
}
```

Build and Run.



Now you have successfully used the functions in the Swift framework for your project. Congratulations and the warm-up is done! Now let's start the real exercise!

Using Objective-C in the Swift Framework

Tap **Components** folder, go to **File ▸ New ▸ File ▸ Objective-C File ▸ Next,** File: ***Image*****,** File Type**: Category,** Class: **NSString.**

Tap **NSString+Image.h,** change the **Target Membership** from **Project** to **Public**.

Go to **NSString+Image.h,** add the following lines.

```
#import <UIKit/UIKit.h>

NS_ASSUME_NONNULL_BEGIN

/**
 The Image category on NSString
 */
@interface NSString (Image)

- (nullable UIImage
*)imageWithAttributes:(NSDictionary<NSAttributedStringKey, id>
*)attributes size:(CGSize)size;

@end

NS_ASSUME_NONNULL_END
```

Go to **NSString+Image.m,** add the following lines.

#import "NSString+Image.h"

@implementation NSString (Image)

```
- (nullable UIImage
*)imageWithAttributes:(NSDictionary<NSAttributedStringKey, id>
*)attributes size:(CGSize)size
{
    CGSize aSize = [self sizeWithAttributes:attributes];
    UIGraphicsImageRenderer *renderer = [[UIGraphicsImageRenderer alloc]
initWithSize:aSize];
    return [renderer imageWithActions:^(UIGraphicsImageRendererContext *
_Nonnull rendererContext) {
        [self drawInRect:CGRectMake(0, 0, size.width, size.height)
withAttributes:attributes];
```

```
    }];
}

@end
```

Go to **Components.h,** import the header.

```
#import <Components/NSString+Image.h>
```

Go to **TrackRowCell.swift**, add the following code just before the body closure.

```
let image = ("🎻" as NSString).image(attributes: [.font:
UIFont.systemFont(ofSize: 20)], size: CGSize(width: 44, height: 44))
```

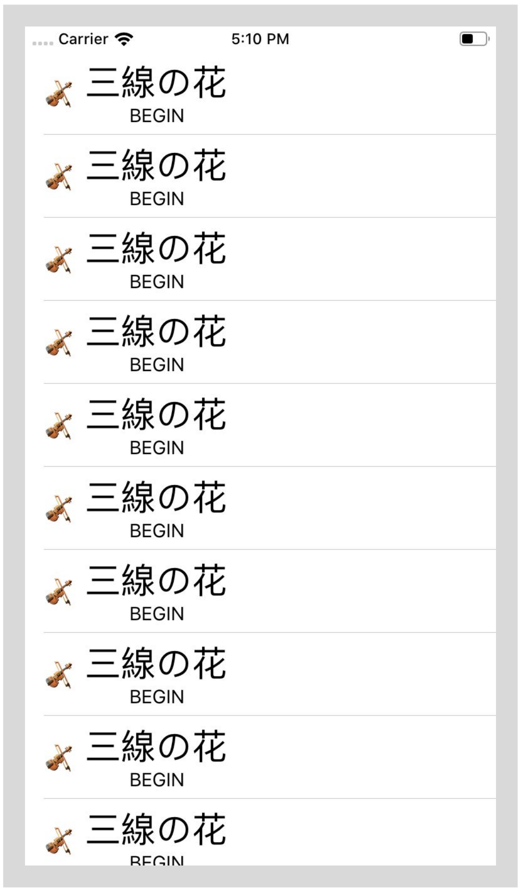
Remove this line.

```
Image(systemName: "book")
```

Add the following line in the same place.

```
image.map { Image(uiImage: $0) }
```

Build and Run.



Now you know how to use Objective-C in a Swift Framework and you have successfully built a hybrid framework.

Well done, but this is still very basic, so we're going to something that is even more awesome! So stay tight and embrace the next step!

Using Objective-C APIs From Your Framework

Remember these two things you did in the previous steps

- Making **NSString+Image.h** public.
- Importing **NSString+Image.h** in the umbrella header **Components.h**

These two things made it possible for Objective-C code to be reused in other projects. This is automatically supported by Apple if you are using a framework with bridging header.
We can try out by removing the following lines in **ContentView.**

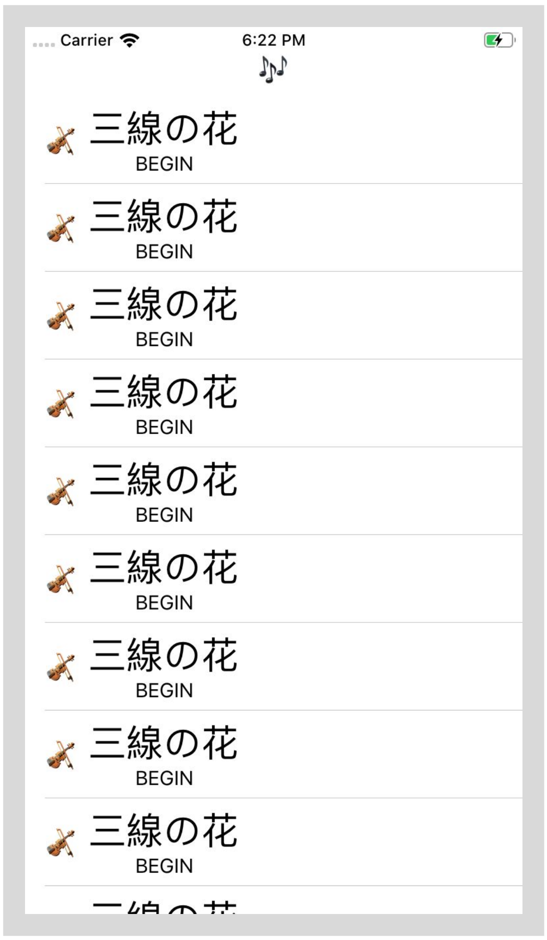
```
var body: some View {
    List(0..<10) { _ in
        TrackRowCell()
    }
}
```

Calling Objective-C functions in the project by adding the following code to **ContentView**.

```
let image = ("🎶").image(attributes: [.font: UIFont.systemFont(ofSize:
20)], size: CGSize(width: 44, height: 44))

var body: some View {
    Section(header: image.map { Image(uiImage: $0) }) {
        List(0..<10) {_ in
            TrackRowCell()
        }
    }
}
```

Build and Run.



Adding Swift Static Library Dependency to Your Framework

If you think that above is easy so far, that is great. Now we're going to spice up a little bit! What happens if you would like to add dependencies to your framework? What’s more important: it is a static library!

In Xcode, go to **File** ▸ **New** ▸ **Project** ▸ **iOS** ▸ **StaticLibrary** ▸ **Next**, give *ColorToken as* **Product name** and choose *Swift* as **Language** *then* **Next** ▸ **Create.**

Build and Run.

Add the following code to **ColorToken.**

```
import UIKit

public enum ColorToken {
  //https://www.colordic.org/w/
  public static let 小豆色: UIColor = UIColor(hex:0x96514d)
}

 extension UIColor {
  convenience init(hex: Int, alpha: Double = 1.0) {
    self.init(red: CGFloat((hex>>16)&0xFF)/255.0, green:
CGFloat((hex>>8)&0xFF)/255.0, blue: CGFloat((hex)&0xFF)/255.0, alpha:
CGFloat(255 * alpha) / 255)
  }

  convenience init(hexString: String, alpha: Double = 1.0) {
    let hex = hexString.trimmingCharacters(in:
CharacterSet.alphanumerics.inverted)
    var int = UInt64()
    Scanner(string: hex).scanHexInt64(&int)
    let r, g, b: UInt64
    switch hex.count {
    case 3: // RGB (12-bit)
      (r, g, b) = ((int >> 8) * 17, (int >> 4 & 0xF) * 17, (int & 0xF) *
17)
    case 6: // RGB (24-bit)
      (r, g, b) = (int >> 16, int >> 8 & 0xFF, int & 0xFF)
    default:
      (r, g, b) = (1, 1, 0)
    }
    self.init(red: CGFloat(r) / 255, green: CGFloat(g) / 255, blue:
CGFloat(b) / 255, alpha: CGFloat(255 * alpha) / 255)
  }
}
```

Build and Run.

Close the **ColorToken** Project, Open **Components.xcodeproj**, drag **ColorToken.xcodeproj** under the **Components** Project. Now we need to add ColorToken as dependency to the **Components** framework.

Tap **Components** Project, Select **Components** Target, **General** ▸ **Frameworks and Libraries** ▸ **+** , add **libColorToken.a.**

Build and Run.

Close **Components.xcodeproj** and open **Tracks.xcodeproj**, Go to **TrackRowCell,** add following lines.

```
import ColorToken
```

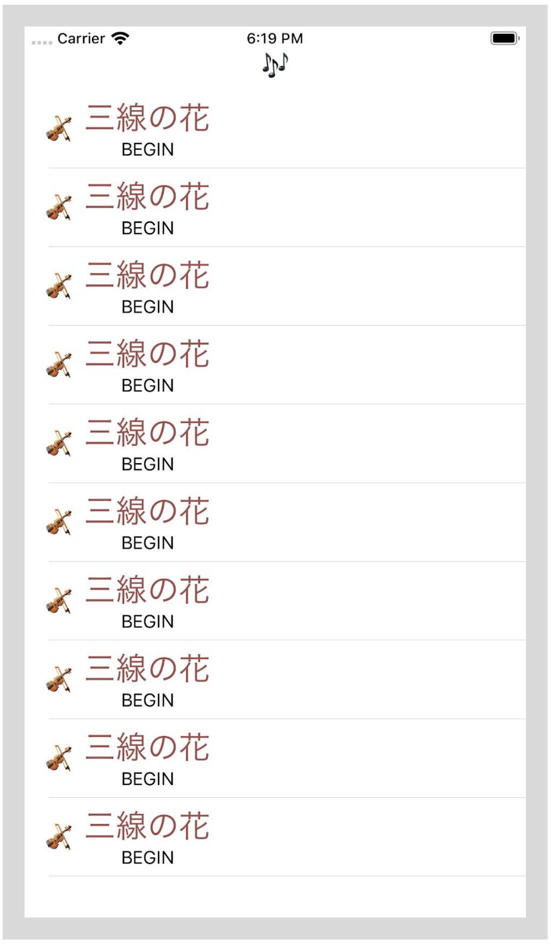
Remove this line:

```
Text("三線の花").font(.title)
```

Add this line:

```
Text("三線の花").font(.title).foregroundColor(Color(ColorToken.小豆色))
```

Build and Run Tracks.



So to add swift static library is fairly simple, you just need to add **libColorToken.a.** Everything else is fixed by Apple.

Adding Objective-C Static Library to Your Framework

Let’s see if adding an Objective-C static library would be as easy as Swift.

In Xcode, go to **File** ▸ **New** ▸ **Project** ▸ **iOS** ▸ **StaticLibrary** ▸ **Next**, give *FontToken as* **Product name** and choose Objective-C as **Language** then **Next** ▸ **Create.**

In the **Project Navigator, right-click FontToken, New Group** and name it **include.**
Build and Run.

Drag the **FontToken.h** under **include** folder.

In the **FontToken.h,** remove everything in this file and add the following lines:

```
#import <CoreText/CoreText.h>

@interface FontToken : NSObject
+(CTFontRef)heartFont;
+(CTFontRef)catFont;
@end
```

In the FontToken.m, remove everything in this file and write the following lines:

```
#import "FontToken.h"

@implementation FontToken
+(CTFontRef)heartFont
{
   CFStringRef strRef = (CFStringRef)@"dongrinuanxin";
   CTFontRef ref = CTFontCreateWithName(strRef, 20, NULL);
   return ref;
}

+(CTFontRef)catFont
{
   CFStringRef strRef = (CFStringRef)@"MaoMeiMei";
   CTFontRef ref = CTFontCreateWithName(strRef, 20, NULL);
   return ref;
}

@end
```

Build and run the **FontToken** target.

Open **Components.xcodeproj**, drag FontToken.xcodeproj under the **Components** Project. Now we need to add FontToken as dependency to the **Components** framework.

Tap **Components** Project, Select **Components** Target, **General** ▸ **Frameworks and Libraries** ▸ **+ ,** add **libFontToken.a.**

Compared to Swift static libraries, Objective-C classes have headers. We need to expose the headers for other projects to use. We can do that by using modulemaps.

Right-click **Components Project,** choose **New Group** and name it **Wrapper.** Right-click **Wrapper, New File** ▸ **Rich Text File**, name it *module.modulemap.*

Remove everything in the file and add the following lines.

```
module FontToken {
    umbrella "../../FontToken/include/"
    export *
}
```

In **Components** project, go to **Build Settings,** search **Import Paths,** click the **+** button, add the following line.

```
$(SRCROOT)/Wrapper/
```

Close **Components** project, and open **Tracks** project.

Now download these two font files that were mentioned in the above code. Drag both files to the **Tracks** project. In the popup menu, **copy items if needed** ▸ **Tick** Adds to targets: Tracks ▸ **Finish.**

You'll also need to register the fonts, in the **Tracks** project, go to **info.plist**, add the key "**Fonts provided by application**", add the following names of the font file as an item of the array.

```
dongrinuanxin.ttf
MaoMeiMei.ttf
```

In the **TrackRowCell**, import the module and instantiate the following variables.

```
import FontToken
import ColorToken

let heart: CTFont
let cat: CTFont
```

Inside the init method, adding the following lines:

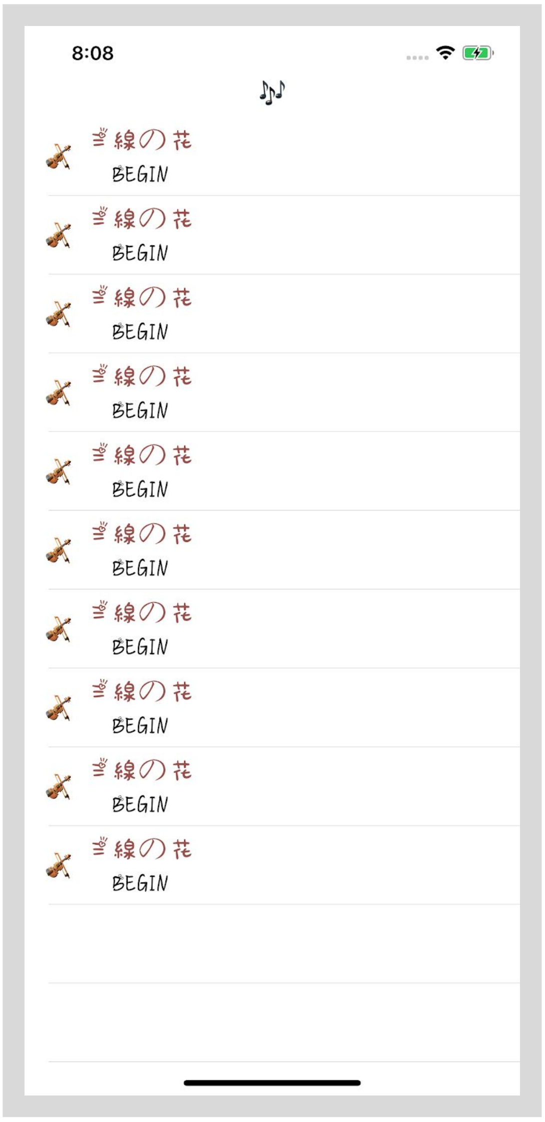
```
public init() {
    let heartFont = FontToken.heartFont()
    let catFont = FontToken.catFont()
    heart = unsafeBitCast(heartFont, to: CTFont.self)
    cat = unsafeBitCast(catFont, to: CTFont.self)
    heartFont?.release()
    catFont?.release()
}
```

We need to track the reference of the **heartFont** and **catFont** manually, that’s why we also need to release it manually. We are using **`unsafeBitCast`** because SwiftUI has a separate font library, it takes a CTFont, while Objective-C API returns a **unmanaged<CTFont>?.**

Use the font in the cell by adding the following lines.

Text("三線の花").foregroundColor(Color(ColorToken.小豆色)).font(Font(heart))
Text("BEGIN").font(Font(cat))

Build and Run.



Where to Go from Here?

That's it! You have finished the workshop. I hope that you have learned one thing or two. You can download the final project here. Feel free to share with me what you think of this workshop, what you have learned, what can be improved and so on.

My twitter is #humlelu and thank you for joining the workshop.