OpenCV for iOS Development

1. Steps to build OpenCV for iOS

OpenCV with extra modules needs to be built for iOS development in Xcode. Following are the steps to do that.

Tools used:
OpenCV - 4.7.0
Xcode - 14.2
CMake - 3.20.1

Step 1 - Download the latest OpenCV package from the GitHub repo, along with the extra modules:

In Terminal,

```
git clone https://github.com/opencv/opencv.git
```

Contrib modules

```
git clone https://github.com/opencv/opencv_contrib.git
```

Step 2 - Download CMake on the device.

Note - There are some dependency issues with the latest version of CMake. So, using HomeBrew to install may not be useful. Instead use the following link to get an older version:

Link - **https://github.com/Kitware/CMake/releases**
(v3.20.1 and v3.19.6 seem to work for this build)

Step 3 - Download Xcode CLI tools

In Terminal,

```
xcode-select install
```

Check if it is downloaded:

```
sudo xcodebuild -license
```

Step 4 - Create a new directory

opencv-build/

Step 5 - Perform the build operation

Make sure the downloaded **opencv/** and **opencv_contrib/**, and new directory **opencv-build/** are in the same directory.

In Terminal,

Case 1 - Build only for physical iOS devices

```
python3 opencv/platforms/iOS/build_framework.py ios --contrib opencv_contrib —iphoneos_deployment_target 12.0 --iphoneos_archs arm64 --build_only_specified_archs
```

[Note - Here, I have used iOS 12.0 as the minimum deployment target. For Xcode 14.2, CLI tools allow only iOS >11.0]

Case 2 - Build for iPhone simulators (Currently iPhone simulator throws a compilation error due to some CMake build issues)

2.1 Build iPhone simulator with x86_64 architecture

```
python3 opencv/platforms/iOS/build_framework.py ios --contrib opencv_contrib —iphoneos_deployment_target 12.0 --iphoneos_archs arm64 --iphonesimulator_archs x86_64 --build_only_specified_archs
```

2.2 Build iPhone simulator with arm64 architecture

```
python3 opencv/platforms/apple/build_xcframework.py ios --contrib opencv_contrib —iphoneos_deployment_target 12.0 --iphoneos_archs arm64 --iphonesimulator_archs arm64 --build_only_specified_archs
```

[Note - OpenCV cannot be built with the .framework extension if we are building for two same architectures (here, arm64). Hence, we need to use a variant of the extension .xcframework. All other steps are the same, only the building script and extension differ]

If no issues occur, you will find opencv.framework or opencv.xcframework (Case 2.2) in opencv-build/ directory. This is the package that can be directly imported in an Xcode project.

— —

2. Steps to add OpenCV for iOS in a project

Step 1 - Create a new Xcode project -> iOS application -> Language (Objective-C or Swift)

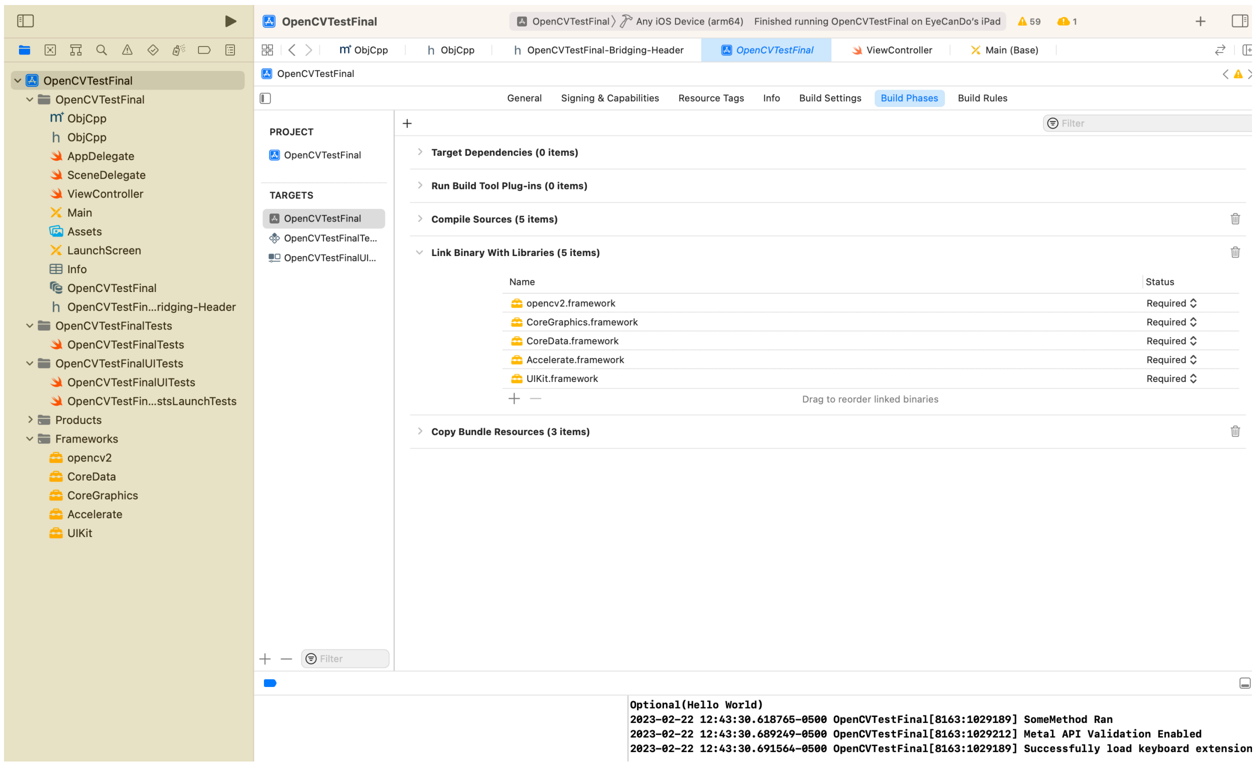
For Objective-C project, the library can be used directly.
For Swift, a bridging header is needed to use Objective-C code along with Swift.

Step 2 - Copy the opencv.(xc)framework (as it is) into the project directory

Step 3 - In Project Settings -> Build Phases,

Go to “Link Binary with Libraries”

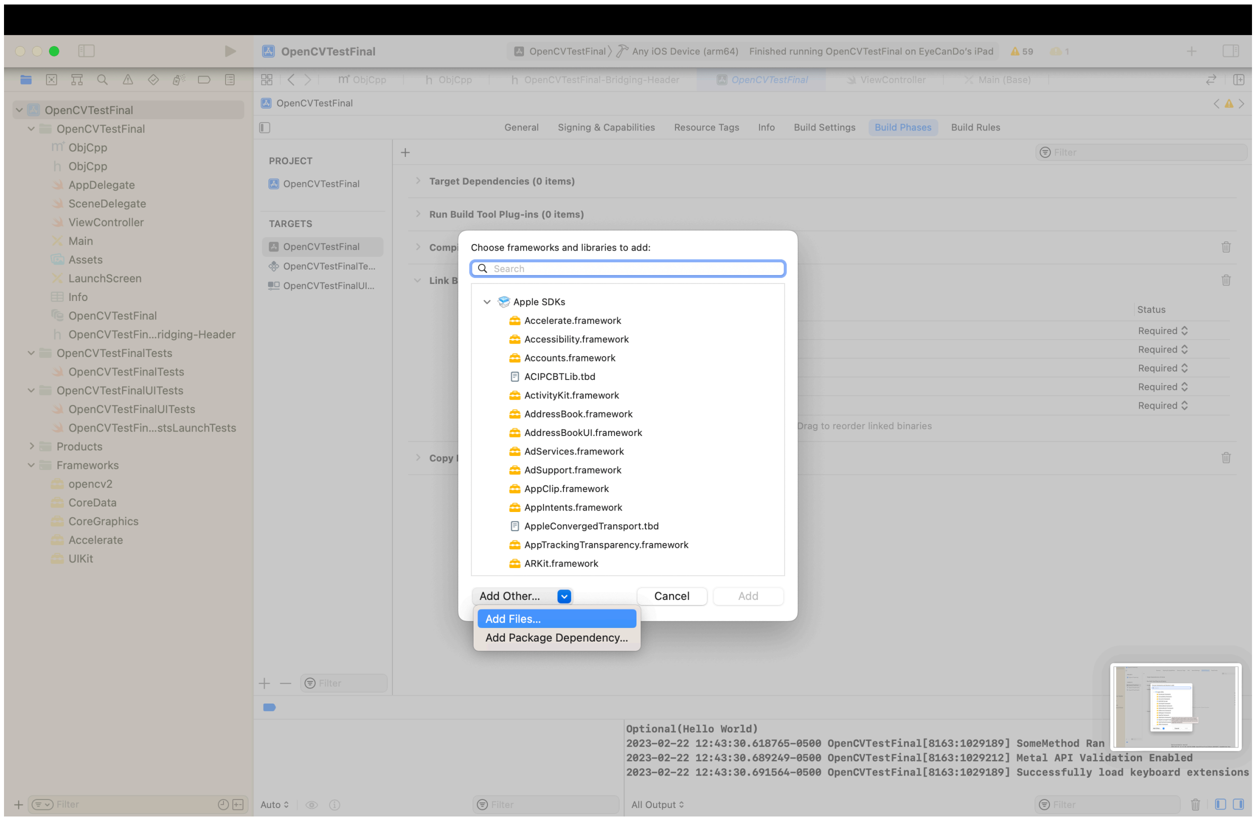
Add libraries, Go to “Add Other” at the bottom, and locate the **opencv.(xc)framework** and import the directory.



Name	Status
opencv2.framework	Required
CoreGraphics.framework	Required
CoreData.framework	Required
Accelerate.framework	Required
UIKit.framework	Required

```
Optional(Hello World)
2023-02-22 12:43:30.618765-0500 OpenCVTestFinal[8163:1029189] SomeMethod Ran
2023-02-22 12:43:30.689249-0500 OpenCVTestFinal[8163:1029212] Metal API Validation Enabled
2023-02-22 12:43:30.691564-0500 OpenCVTestFinal[8163:1029189] Successfully load keyboard extension
```

For Swift, you may need the other frameworks (as per use case) for the build.



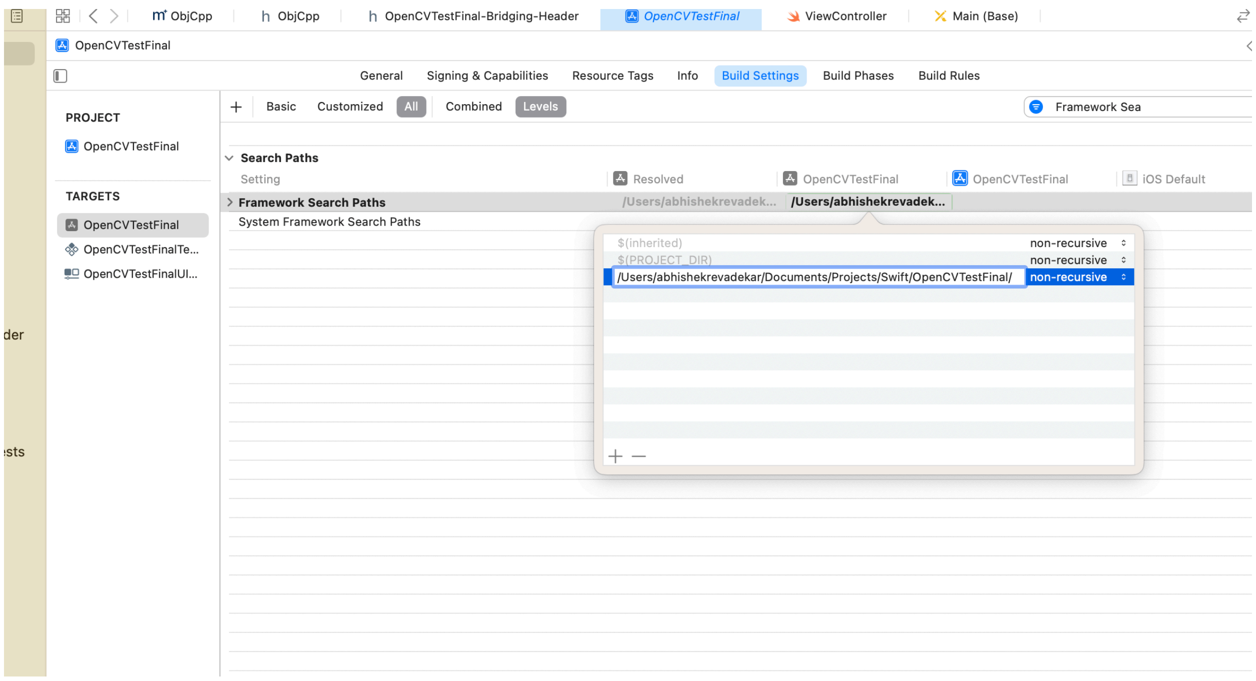
```
Optional(Hello World)
2023-02-22 12:43:30.618765-0500 OpenCVTestFinal[8163:1029189] SomeMethod Ran
2023-02-22 12:43:30.689249-0500 OpenCVTestFinal[8163:1029212] Metal API Validation Enabled
2023-02-22 12:43:30.691564-0500 OpenCVTestFinal[8163:1029189] Successfully load keyboard extensions
```

You should see the OpenCV library added under the Frameworks directory in the project structure

Step 4 - Once done, go to the Build Settings tab,

Search for “Framework Search Paths”

Click on it and check if the path contains the project directory, else dd the directory path of the **opencv.(xc)framework**



Framework Search Paths /Users/abhishekrevadek... /Users/abhishekrevadek...
System Framework Search Paths

```
$(inherited)                                                        non-recursive
$(PROJECT_DIR)                                                      non-recursive
/Users/abhishekrevadekar/Documents/Projects/Swift/OpenCVTestFinal/  non-recursive
```

Ideally, under the project directory, Xcode should be able to detect the framework in the project directory, so in most cases this will not be needed. But, this can be used if the framework is not copied in the project directory.

---

3. Steps to import OpenCV library

OpenCV in Objective-C project

Since OpenCV is built in C++, direct import in Objective-C (.m) file will not be detected.

Simply, change the extension of the Objective-C file to (.mm). Xcode will identify this as an Objective-C++ file. Make sure the file with new extension is detected in Xcode project structure. (No need to modify the header files)

You can now import OpenCV modules in Objective-C files.

```
#import <opencv2/core.hpp>
```

OpenCV in Swift project

To use OpenCV modules in a Swift project, we need create Objective-C++ files for the OpenCV utilities and use a “Bridging Header” to import this file in the Swift file.

Step 1 - Add a new Objective-C file to the project

Step 2 - Create a bridging header

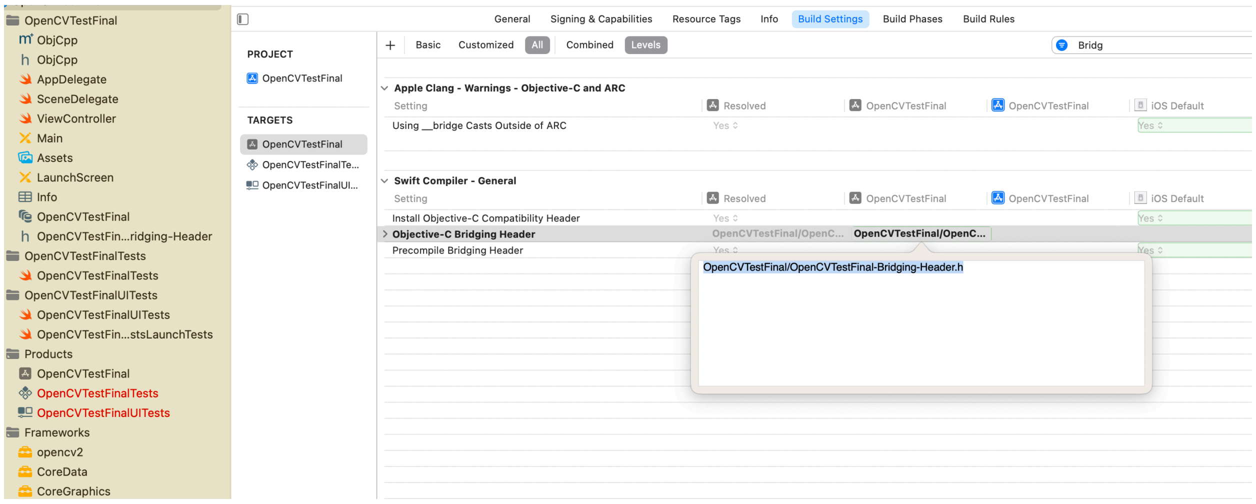
Ideally, Xcode will prompt whether you would like to create a Bridging Header for this file

If it doesn’t ,check if the header is already present,

Go to Project Build Settings, Look for “Objective-C Bridging Header”,

If a file path is present, a bridging header exists, you can just locate that in the project settings.

If not, create a header file (.h) for bridging, and add the relative path to the “Objective-C Bridging Header” tab.



Apple Clang - Warnings - Objective-C and ARC

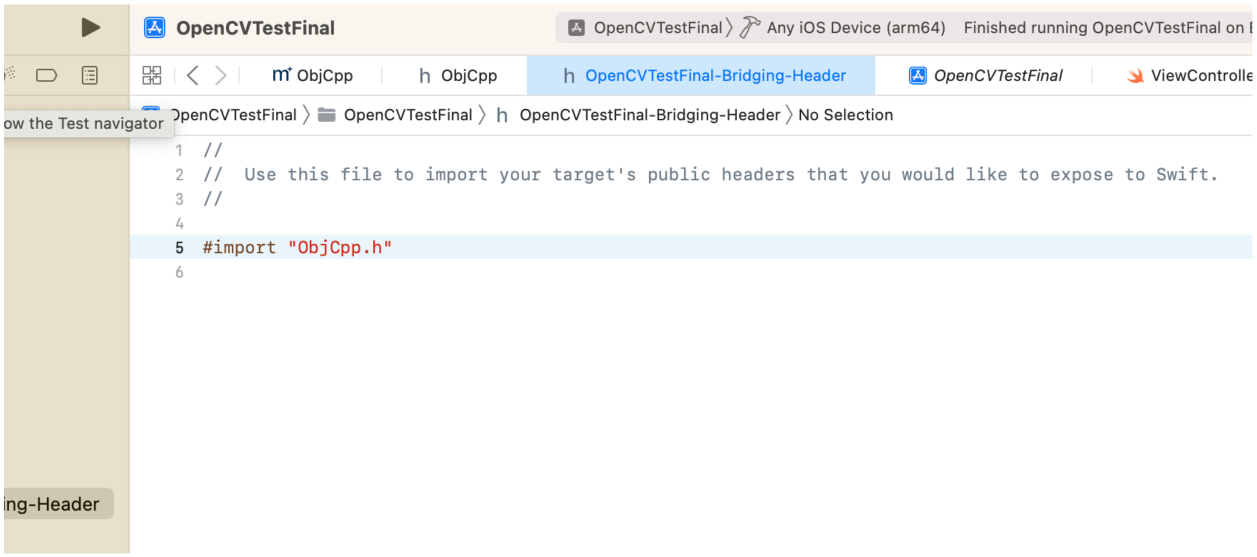
Setting	Resolved	OpenCVTestFinal	OpenCVTestFinal	iOS Default
Using __bridge Casts Outside of ARC	Yes			Yes

Swift Compiler - General

Setting	Resolved	OpenCVTestFinal	OpenCVTestFinal	iOS Default
Install Objective-C Compatibility Header	Yes			Yes
Objective-C Bridging Header	OpenCVTestFinal/OpenC...	OpenCVTestFinal/OpenC...		
Precompile Bridging Header	Yes			Yes

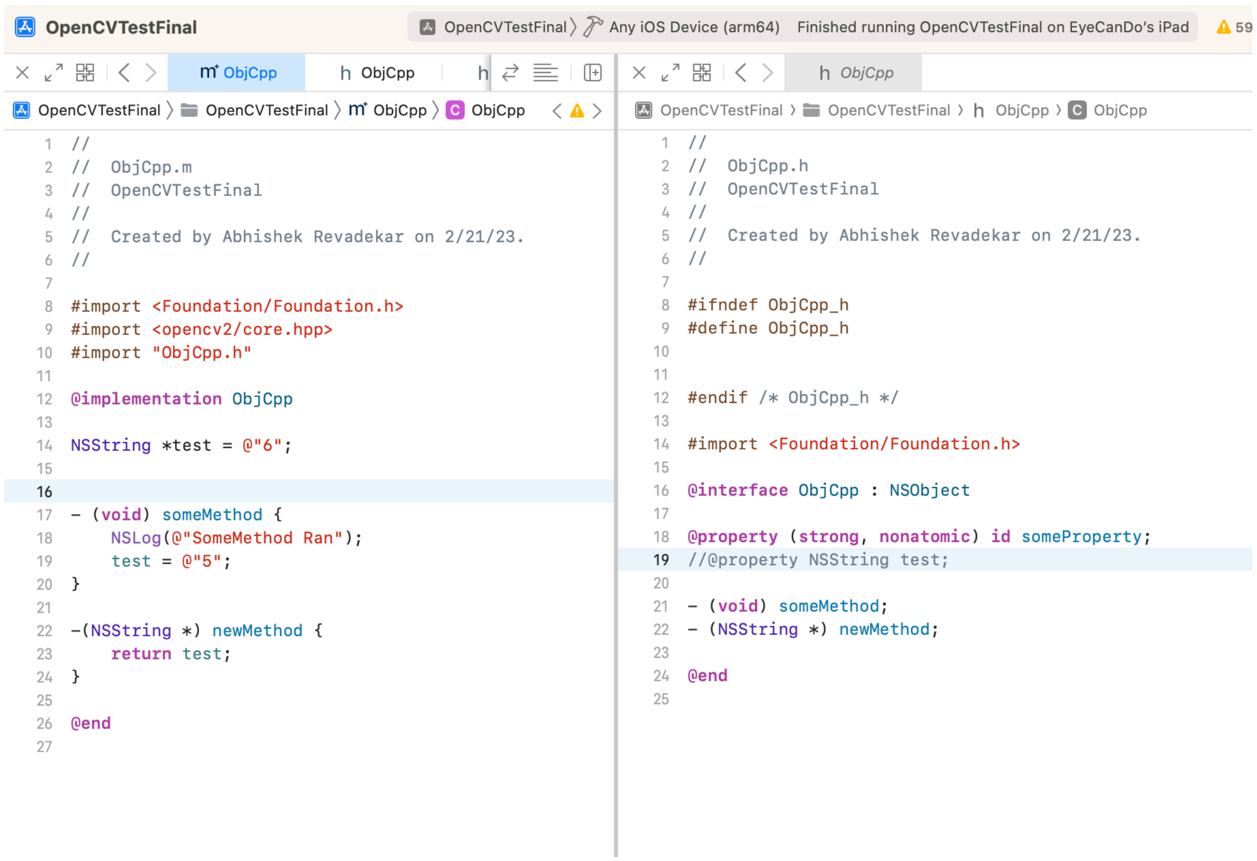
OpenCVTestFinal/OpenCVTestFinal-Bridging-Header.h

This Header file needs to be created only once. There is no need to create separate headers for each Objective-C file. Essentially, all Objective-C files to be imported in Swift, need to be imported in the header file. No other code is necessary here.


OpenCVTestFinal
OpenCVTestFinal 〉 Any iOS Device (arm64) Finished running OpenCVTestFinal on
ObjCpp
ObjCpp
OpenCVTestFinal-Bridging-Header
OpenCVTestFinal
ViewControlle
ow the Test navigator
OpenCVTestFinal 〉 OpenCVTestFinal 〉 OpenCVTestFinal-Bridging-Header 〉 No Selection
ing-Header

```
//
//  Use this file to import your target's public headers that you would like to expose to Swift.
//

#import "ObjCpp.h"

```

Step 3 - Work on the Objective-C file as you would for an Objective-C Xcode project. Add an interface for the classes and functions in the header file associated with it.

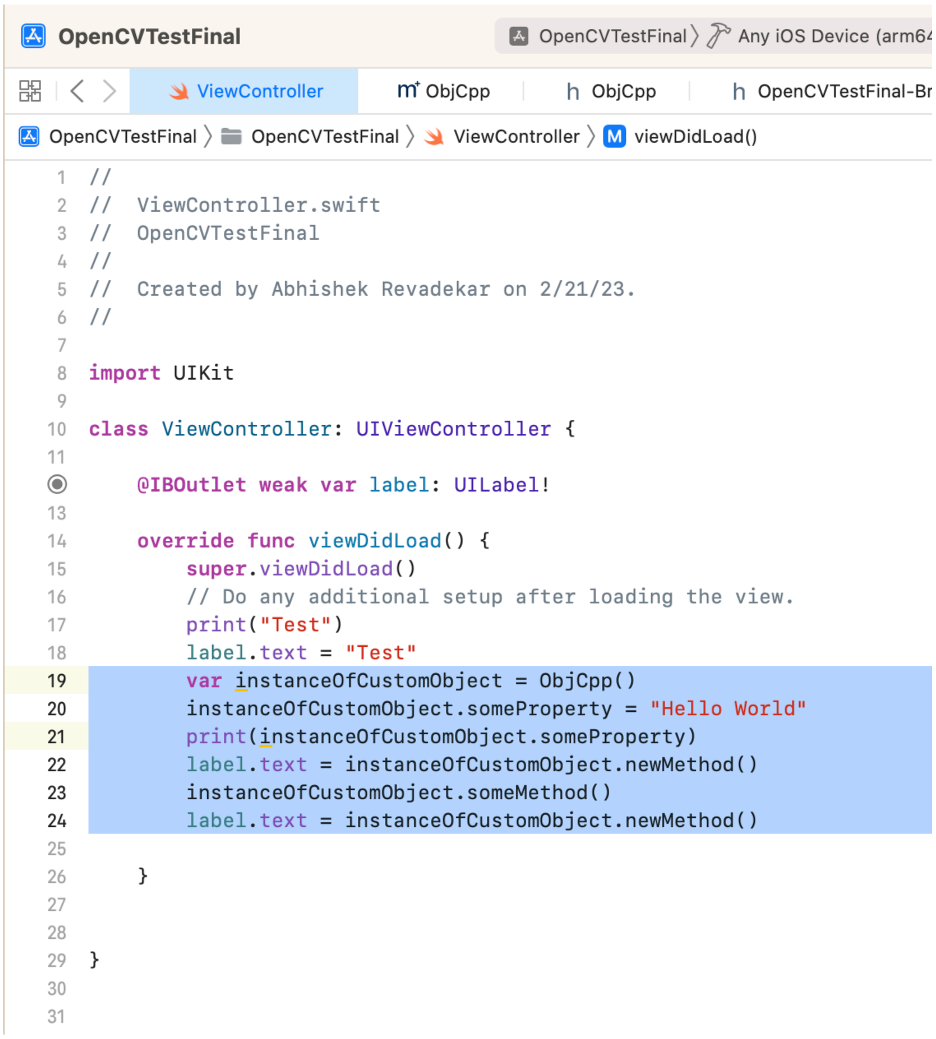

OpenCVTestFinal
OpenCVTestFinal 〉 Any iOS Device (arm64) Finished running OpenCVTestFinal on EyeCanDo's iPad
59
ObjCpp
ObjCpp
ObjCpp
OpenCVTestFinal 〉 OpenCVTestFinal 〉 ObjCpp 〉 ObjCpp
OpenCVTestFinal 〉 OpenCVTestFinal 〉 ObjCpp 〉 ObjCpp

```
//
//  ObjCpp.m
//  OpenCVTestFinal
//
//  Created by Abhishek Revadekar on 2/21/23.
//

#import <Foundation/Foundation.h>
#import <opencv2/core.hpp>
#import "ObjCpp.h"

@implementation ObjCpp

NSString *test = @"6";


- (void) someMethod {
    NSLog(@"SomeMethod Ran");
    test = @"5";
}

-(NSString *) newMethod {
    return test;
}

@end

```

```
//
//  ObjCpp.h
//  OpenCVTestFinal
//
//  Created by Abhishek Revadekar on 2/21/23.
//

#ifndef ObjCpp_h
#define ObjCpp_h


#endif /* ObjCpp_h */

#import <Foundation/Foundation.h>

@interface ObjCpp : NSObject

@property (strong, nonatomic) id someProperty;
//@property NSString test;

- (void) someMethod;
- (NSString *) newMethod;

@end

```

[Note - You may want to use **#import <Foundation/Foundation.h>** in the Objective-C file to use the standard development classes for iOS projects.]

Step 4 - Now you can import the functions from the Objective-C interface in Swift.



OpenCVTestFinal
OpenCVTestFinal 〉 Any iOS Device (arm64
ViewController
ObjCpp
ObjCpp
OpenCVTestFinal-Br
OpenCVTestFinal 〉 OpenCVTestFinal 〉 ViewController 〉 viewDidLoad()

```swift
//
//  ViewController.swift
//  OpenCVTestFinal
//
//  Created by Abhishek Revadekar on 2/21/23.
//

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
        print("Test")
        label.text = "Test"
        var instanceOfCustomObject = ObjCpp()
        instanceOfCustomObject.someProperty = "Hello World"
        print(instanceOfCustomObject.someProperty)
        label.text = instanceOfCustomObject.newMethod()
        instanceOfCustomObject.someMethod()
        label.text = instanceOfCustomObject.newMethod()

    }


}

```