An Essential Guide to XCTest Framework in iOS App Testing

a publication of bitbar.

Table of content

Introduction

There are very few test automation frameworks that are tightly coupled with the development tool itself. XCTest framework is one of those frameworks that enable its users to write basic unit, performance and some level of UI tests for iOS apps.

The thing with frameworks (like XCTest framework) that integrate tightly with development environment is that those frameworks typically provide easy-to-use and seamless workflow with other components and features of the development environment. And as always, frameworks that are tightly bundled with development tools and environment have their pros and cons that developers should be aware of.

As Apple has decided to deprecate UI Automation from Xcode, XCTest certainly is getting its popularity among iOS developers. In this case, we would like to share our experiences on this increasingly hot iOS test automation framework in this ebook. With this ebook, you will learn

- Pros and cons of using XCTest for iOS testing
- How to create .ipa files and XCTest package
- Getting Started with KIF for Functional iOS UI Testing
- The Basics of XCUITest and Using Xcode UI Test Recorder

Chapter

01

Pros and Cons of Using XCTest for iOS App Testing

Before we jump onto the upsides and downsides of testing iOS apps with XCTest, let's quickly go through a brief introduction of XCTest framework.

A Brief Introduction to XCTest

Like Android Studio, Xcode provides feature-rich software testing capabilities for developers that can significantly help enhancing the stability of software. Furthermore, Xcode provides XCTest and XCUITest that are extremely helpful for achieving and building better quality software. The rule of thumb here is that well tested mobile apps, regardless if done in unit level or UI level, improve user experience, which in turn accelerate their adoption with more downloads.

Frankly, XCTest is not a new framework but it has evolved quite a lot with Xcode releases. In fact, the XCTest framework was introduced with Xcode 5 couple of years ago. What XCTest basically does is to allow its users to do unit testing for Xcode projects (iOS apps among them) as it is currently considered as one of the top options for iOS app testing. And writing any tests with XCTest is a trivial task to iOS developers because XCTest is fully compatible with both Objective-C and Swift. In addition, all test classes that iOS developers create in Xcode project are basically the subclasses of XCTestCase.

XCTest tests can be executed on simulators or against real physical devices. If you use real devices locally, you just need to make sure all provisioning side of things (provisioning profile) is valid and set properly to test target. The test methods used in XCTest are instance methods so no parameters are passed nor they return a value. For this the name also begins with 'test' and all added tests are visible in Test Navigator of your Xcode project.

Things to Think About XCTest Framework

Many of test automation frameworks are based on the frameworks and layers that development tools provide with them. In case of iOS this is XCTest and with Android most of the frameworks are based on JUnit and Android instrumentation.

However, XCTest isn't perfect, though it provides some excellent basic functionality and capabilities to exercise tests on your iOS apps. As XCTest has evolved in each version of Xcode, there are a few cons you should take into account. Let's review the pros and cons of XCTest one by one.

Pros

Easy to Learn and No Additional Components Required

Xcode provides everything to get started with test automation with XCTest. Xcode's XCTest is an integral part of the tool so testing is fairly easy to start and convenient to work with.

Native iOS Language Support

Basically writing tests with the same programming language that your application is build with is not a requirement, but it gives some confidence for developers to create tests for their apps. Other way around, some developers may also think this differently and prefer other additional languages, frameworks and tools to be used in test creation. Nevertheless, there is no learning curve or language barriers to get started with XCTest test creation.

Xcode Test Recorder

This is mostly a feature for UI tests but as XCTest is closely related with XCUITest the UI recording is possible with Xcode environment. The UI testing capabilities with Xcode include UI recording, generating code out of those recordings, run tests identically as those were intended while UI test was recorded. Record-and-playback testing tools have plenty of benefits and can get testers do the job even without understanding of underlying software. In addition, recording tests again is possible or simply editing the generated piece of code with whatever changes have been done for app itself. In short, record and playback tools provide fast way to create tests, decent level of accuracy, and not being too sensitive for user-input errors.

Integrating with Continuous Integration

Integrating XCTest with continuous integration is also easy. Xcode allows XCTest tests to be executed using command-line scripts/shell and seamlessly integrated with Xcode's Continuous Integration Bots. However, there is a way to integrated XCTest scripts and the development environment with more widely used CI systems, such as Jenkins, but there are lots of things you need to know and sometimes – despite XCTest and Xcode have provided this capability for some time – tests end up failing just for no reason.

Faster than 'On-Top-of-It' Frameworks

Many other frameworks are relying on XCTest. And despite that those frameworks provide more features, a higher level of abstraction (or at least way to think so) for testing and better capabilities, they still rely on the foundation of XCTest. This naturally makes XCTest faster as you don't need to rely on abstraction APIs, albeit in many cases lightweight.

Cons

No Cross-Platform Support

Apple typically builds everything to be available only on their own tools, devices and environments. Although this is understandable in case of integral testing API and functionalities, many testers prefer to use cross-platform test automation frameworks and rely on one test script to work on both Android and iOS platforms.

Limited Programming Language Support

A coin has two sides. Basically using XCTest only needs you to learn and know either Objective-C or Swift for programming. However, you will be then limited to a small selection of languages with only two options.

As mentioned before, the abstraction framework on top of XCTest will expand the programming language selection and provide testers a freedom of choice with their tools and programming languages. For example, Appium provides great support for all major programming languages and several not-so-mainstream ones as well.

Flaky

XCTest is not new on the market but it still has some issues with basic usability and problems on the robustness of creating tests for iOS apps. Doing some of the basic unit tests and running those on simulators may work just fine but when you try to run the same tests on real devices, you'll see that there is some room for stability improvements with XCTest and Xcode in general.

CONCLUSION

XCTest is a valid choice for basic unit level testing of iOS apps and functionality. However, you need to add lots of those sort of basic tests to thoroughly cover the logic, and that makes functional UI testing more suitable for iOS apps.

Chapter

02

How to Create .ipa Files and XCTest Package

Despite its advantages and disadvantages, XCTest is still a great addition to iOS developers' toolbox. This chapter will cover a basic step-by-step tutorial to get you started with XCTest and how to create IPA properly for a test session.

Methods to Create iOS Package Archive

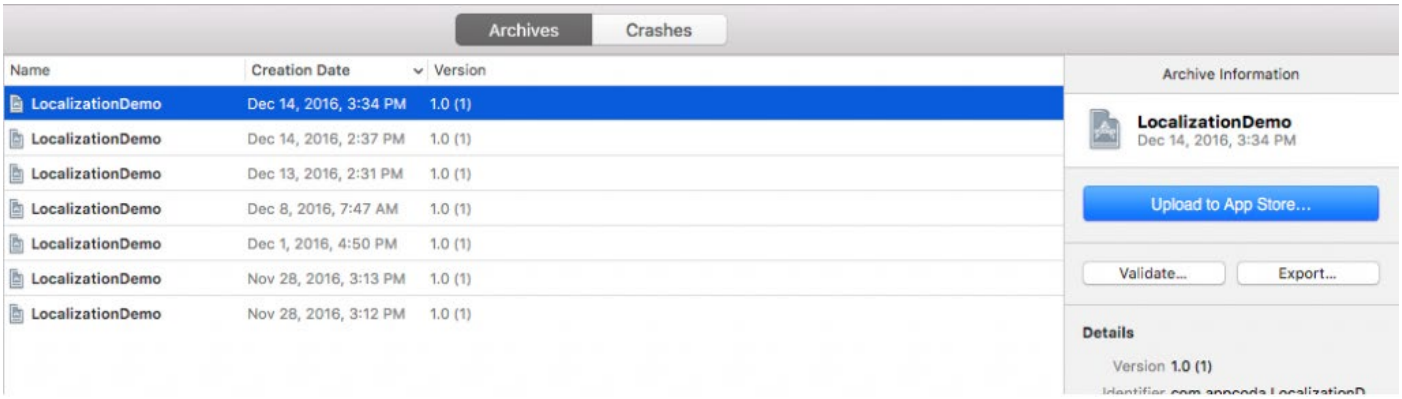
The iOS application and test package are both required for XCTest testing. The same thing applies for any functional UI test automation and then application (whether IPA or APK) can be used for target devices and test package for instrumentation or to be used as part of the device session. More specific information about iOS development certificates, provisioning profiles and identifiers and how to prepare IPA for Testdroid Cloud can be found at Bitbar blog.

› Working on Xcode to Create IPA for Testing

Let's first review how to working with Xcode to produce IPA compatible with a device session and the right configurations.

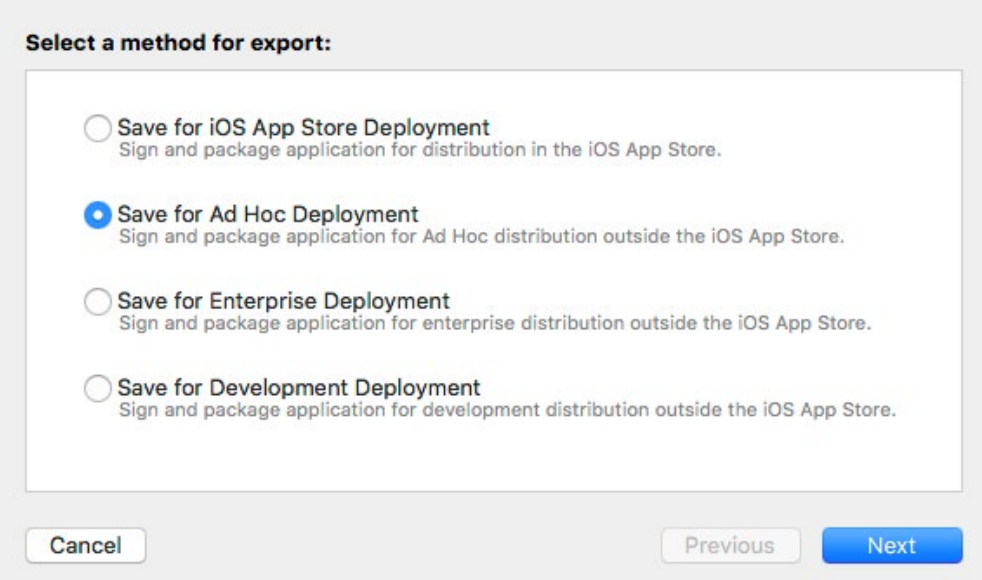
1. Archive Your Build

When you archive your build, the package will be compatible with an iOS device. When your build is done and archiving has finished, select the build from Archive list and click "Export...":



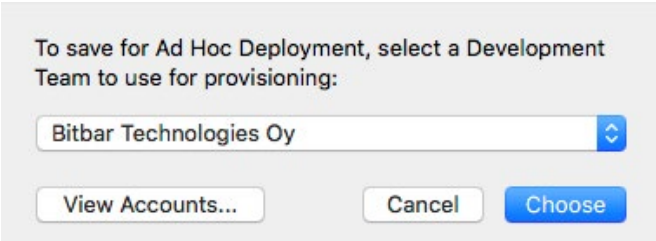
2. Select The Right Method for Export

When the following window is shown, simply select “Save for Ad Hoc Deployment” and click Next.



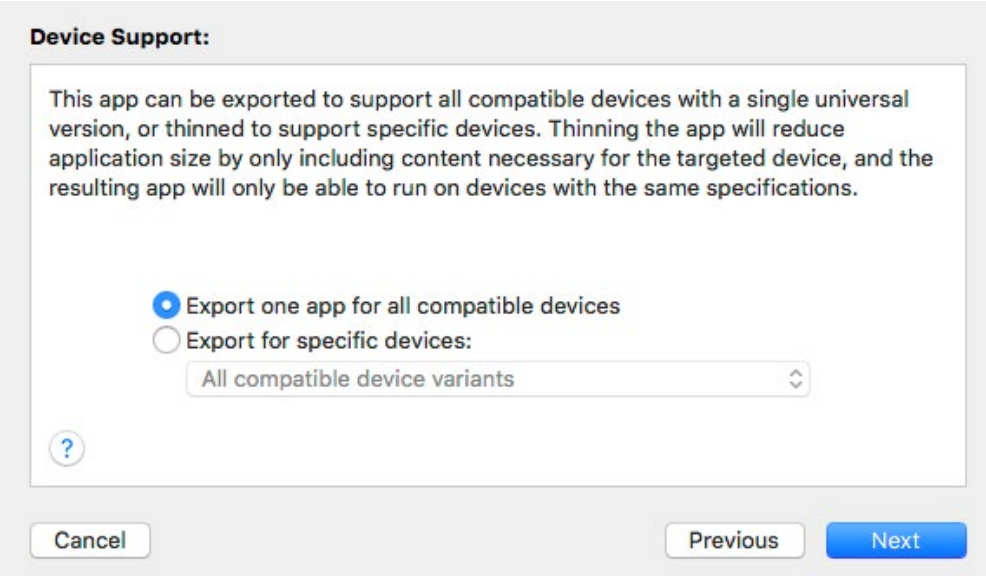
3. Identify Yourself (and the Build)

Use the same identify what you use in build settings for code signing. If your project and Xcode is properly configured you should see the following type of dialog proposing the first usable identifier:



4. Select Supported OS and Devices

It’s almost always recommended to include all possible device support for your app. However, if you want to reduce the size of your IPA you can shrink it by selecting to support only certain devices and OS versions. In that case you would select Export for specific devices and select devices from the dropdown. To support all of them (recommended), just select “Export one app for all compatible devices”:



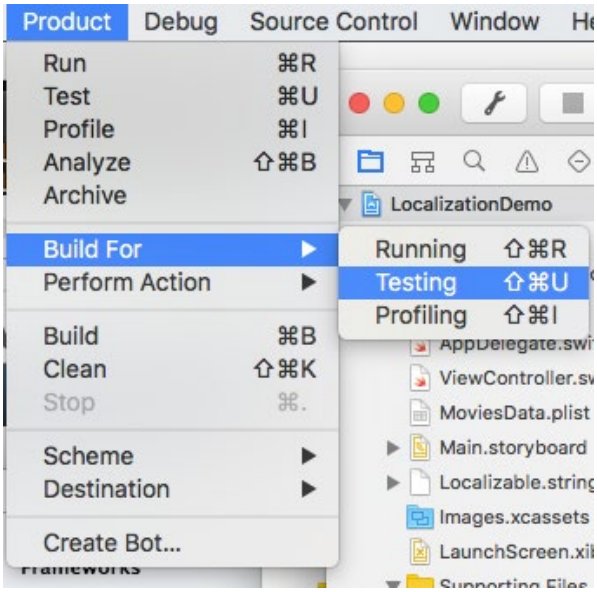
All done! Now just follow through the wizard to locate your .ipa file and everything will be ready for the test session.

› Working on Command Line to Create IPA for Testing

To get started with command line script you first need to make sure IPA file will be testable and it can be properly packaged.

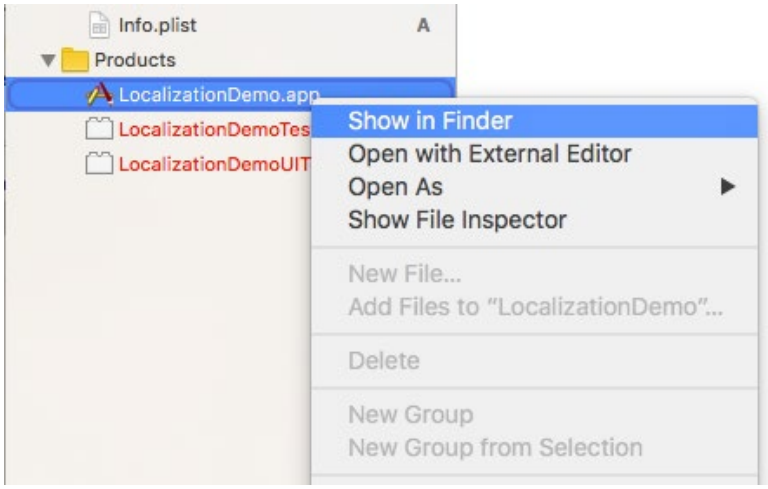
1. Select to Build for Testing

This can be done Product › Build for › Testing menu:



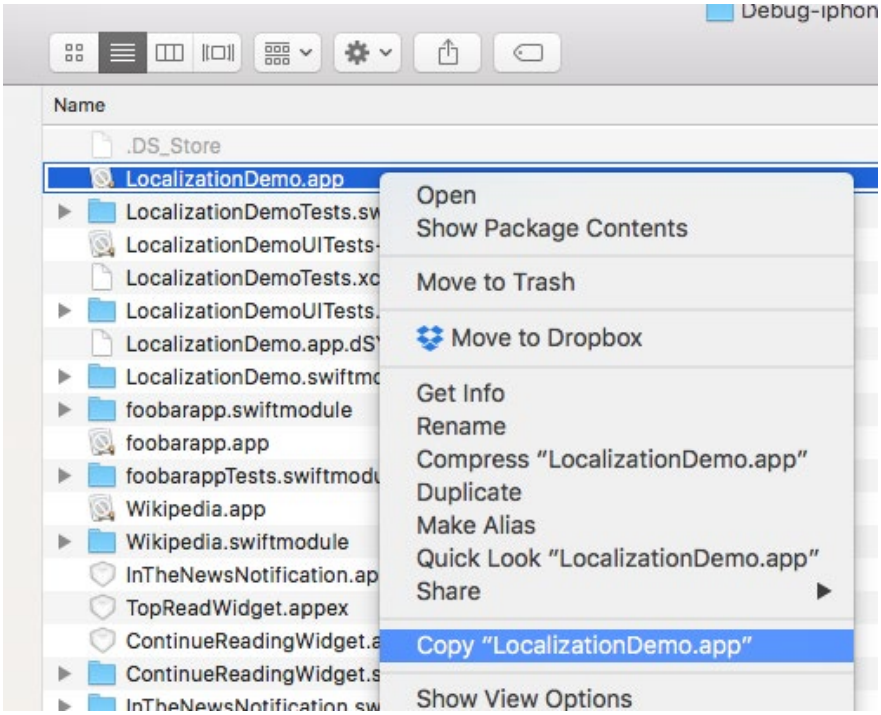
2. Locate the App File on Your Hard Disk

Next, select your project in Project Navigator and right click to “Show it in Finder”:



3. Copy the Application File

After Finder dialog is launched and files are shown, just highlight the .app file and right-click to see copying option, as follows:



4. Create IPA from Command Line

Finally, open the terminal, and create IPA package with following command line:

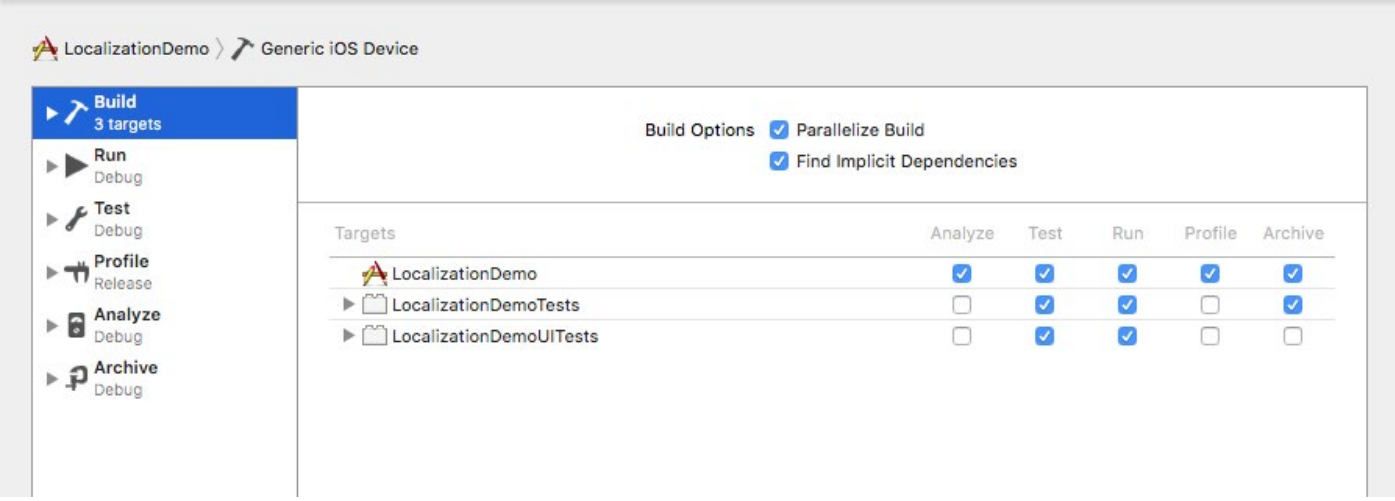
```
$ mkdir /ProjectName
$ cd ProjectName
$ cp -r /Path/To/Your/IPA/File/LocalizationDemo.app .
$ cd ..
$ zip --symlinks -qr "LocalizationDemo.ipa" ProjectName
```

Now you have an IPA file to be used with a test session on devices.

Create a Package for XCTest

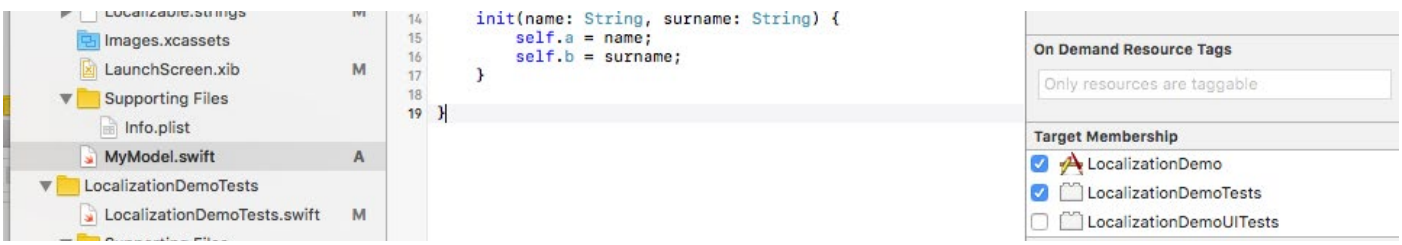
Things will start with compiling unit tests. It will actually happen automatically if you have checked target properly. This can be done or just verified that everything is properly set up under Product › Scheme › Edit scheme.

Select Target on the left-hand side menu and check all items on Run column:



This will make sure your tests will be compiled whenever your app is getting built.

The next thing you need to make sure of is that all used classes for testing are available in test bundle and memberships are appropriately tagged. This can be done on the right-hand side menu “Target Membership” for each file. In the following example, MyModel.swift file is the one under test:

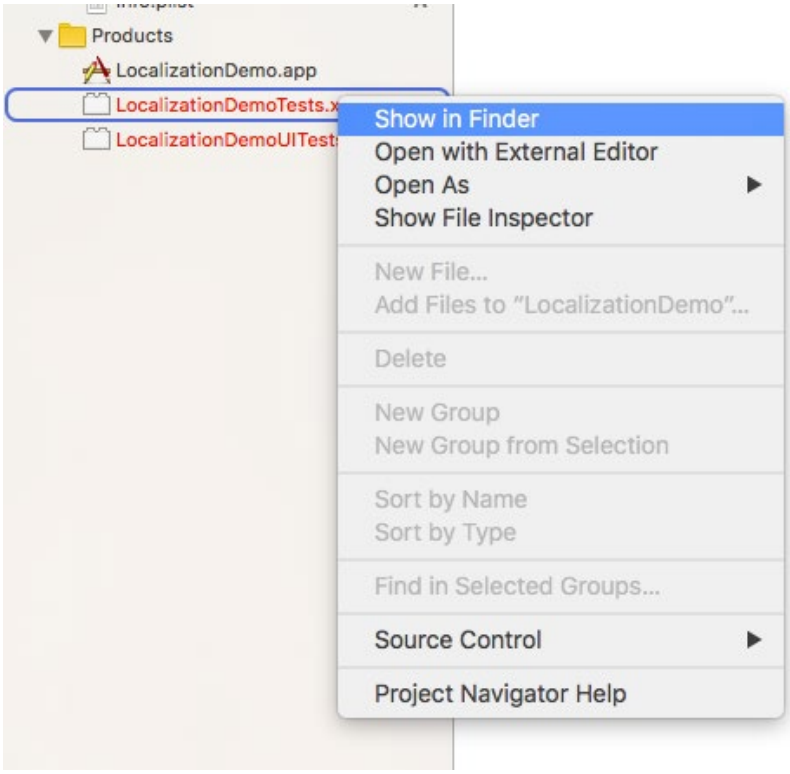


Another option is to use **@testable annotation** for importing different modules.

Now, to compile tests for real device, select device to be used from the menu and press Command + B (or Product › Build). Note that since Xcode has been changed quite a lot with the latest version, you have to deal with those tests differently based on which Xcode version you are using.

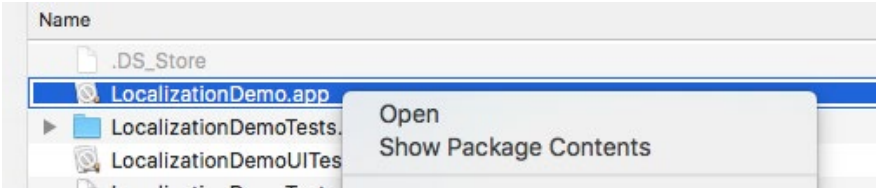
› Xcode 7

In Xcode 7 you can right click on .xctest under Product, and select Show in Finder:

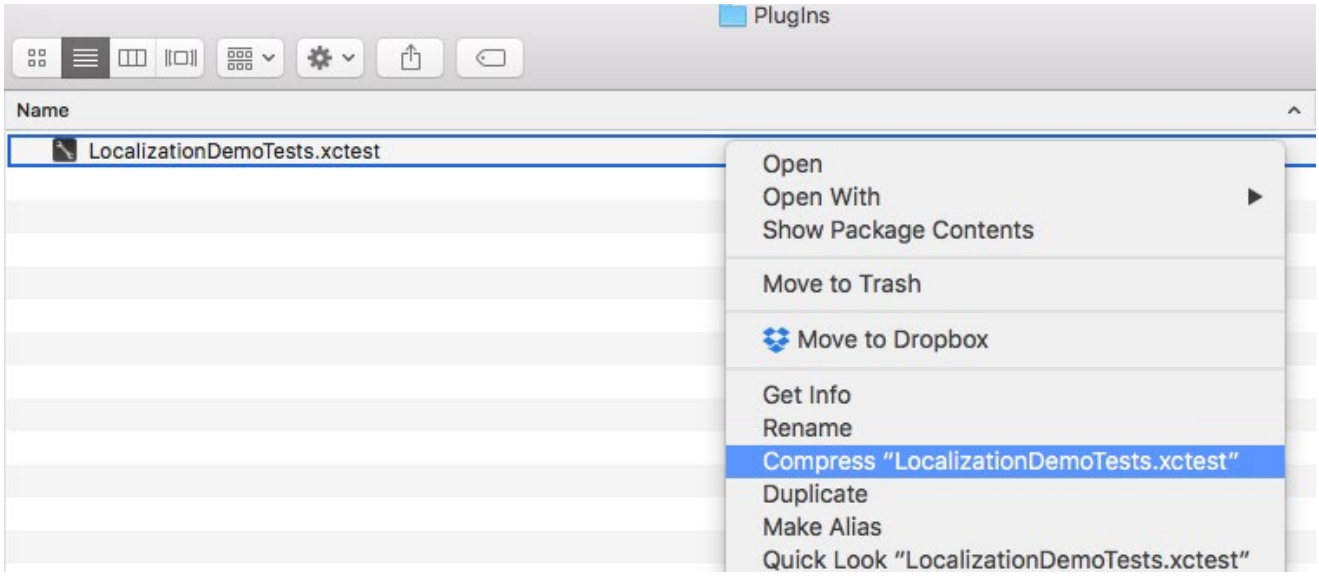


› Xcode 8

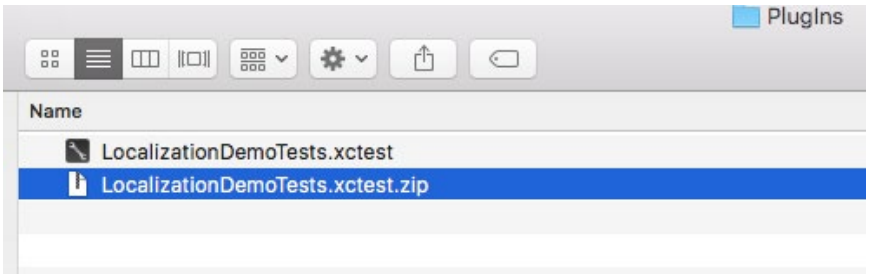
In Xcode 8, .xctest can be found inside of the .app. In order to locate you need to right click on top of the app, Show in Finder, right click again top of the app in Finder and select Show Package Contents:



Then go to the Plugins folder and right click top of the .xctest and select compress:



Now you have XCTest package as a zip file:



These files are now usable locally with your real physical iOS devices.

Chapter

03

Getting Started with KIF for Functional iOS UI Testing

As aforementioned, many of the open source iOS test automation frameworks are created on top of XCTest implementation. KIF - Keep It Functional is one of them.

Introduction to KIF iOS Integration Testing Framework

KIF (short for "keep it functional") is an iOS integration test framework that is closely related to and that uses XCTest test targets. KIF tests can be executed directly in XCTestCase or any subclass. KIF allows for easy automation of iOS applications by leveraging the accessibility attributes that the OS makes available to those with visual disabilities.

One of the reasons why KIF is popular today is that it performs tests using standard Xcode test targets and all tests can be written using Objective-C or Swift as normal XCTest subclasses.

Features and Benefits of KIF

Easy to Learn – No Real Learning Curve

As Swift and Objective-C are the options for iOS app developers to use, KIF is very easy to use in Swift and Objective-C based projects and it supports apps written either with Swift or Objective-C. This basically limits the complexity that typically comes from the test setups where several languages are used.

KIF integrates directly to Xcode project and there is no need to install any packages, set up additional servers, or integrate with any other instances.

Support for Various iOS Versions

Since XCTest was introduced in Xcode, iOS have seen several major (and minor) releases after that. However, XCTest has remained consistent test option for iOS app developers and the iOS version fragmentation hasn't had too much impact on how it can be used for test automation. Basically KIF supports even older iOS versions and from the user point of view changes required to get things running are minimalistic (all related to Xcode).

User-like Test Automation

KIF among the other frameworks provide user-like inputs and all events and interactions for automation are similar of how an actual user would use the app under test. In addition, the XCTest/XCUITest foundation provides great API's to variety of different user interactions, clicks and even gestures. The basic selection of gesture and interaction API includes:

```
func tap()
func doubleTap()
func twoFingerTap()
func tap(withNumberOfTaps: UInt, numberOfTouches: UInt)
func press(forDuration: TimeInterval)
func press(forDuration: TimeInterval, thenDragTo:
XCUIElement)
func swipeLeft()
func swipeRight()
func swipeUp()
func swipeDown()
func pinch(withScale: CGFloat, velocity: CGFloat)
func rotate(CGFloat, withVelocity: CGFloat)
```

You can find more information about XCTest elements from Apple's documentation.

Development Tool Compatibility

The only tool iOS developer needs to design, develop and test their iOS apps is the Xcode environment. KIF and XCTest are tightly integrated with it and for example Test Navigator, Bots and other Xcode features are readily available and handy to use with KIF tests.

How to Get Started with KIF

First of all, you need to download the KIF source assets from Github and place it somewhere where you can easily find it. Alternatively, you can use Git's submodule to get source code for your local use:

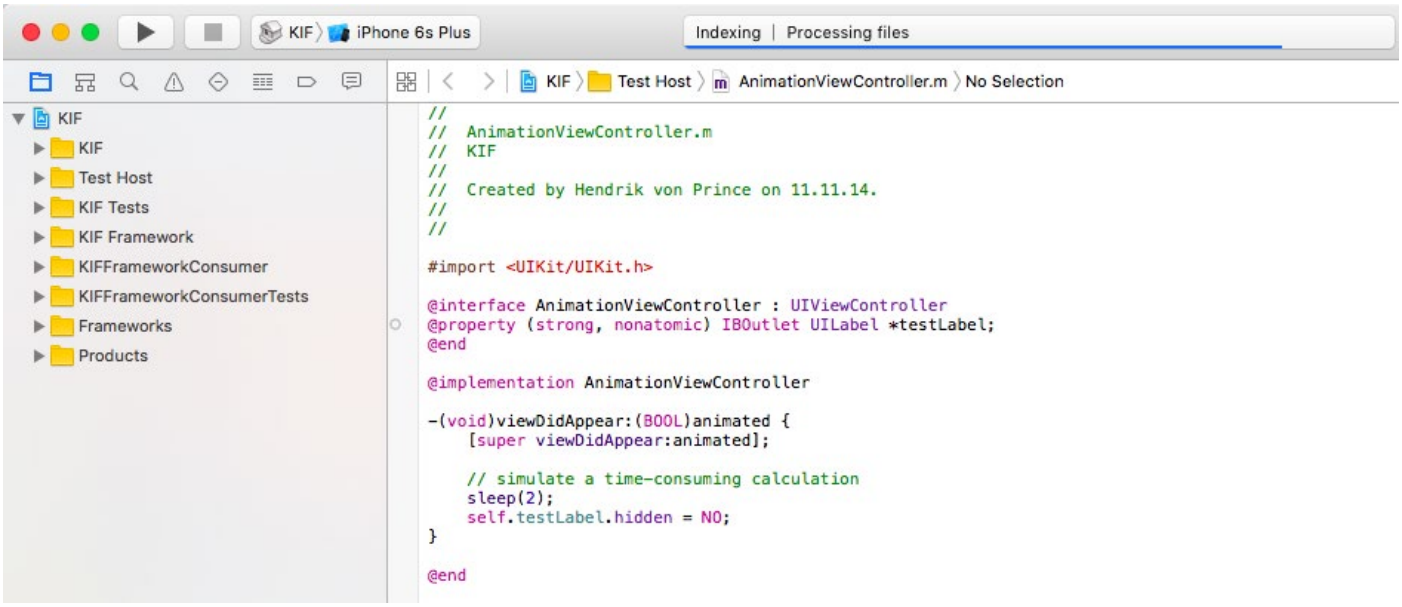
```
$ git init
$ git submodule add https://github.com/kif-framework/KIF.
git Folder/Here
```

NOTE! This needs to be added to your iOS project folder (=Folder/Here should be under your iOS project).

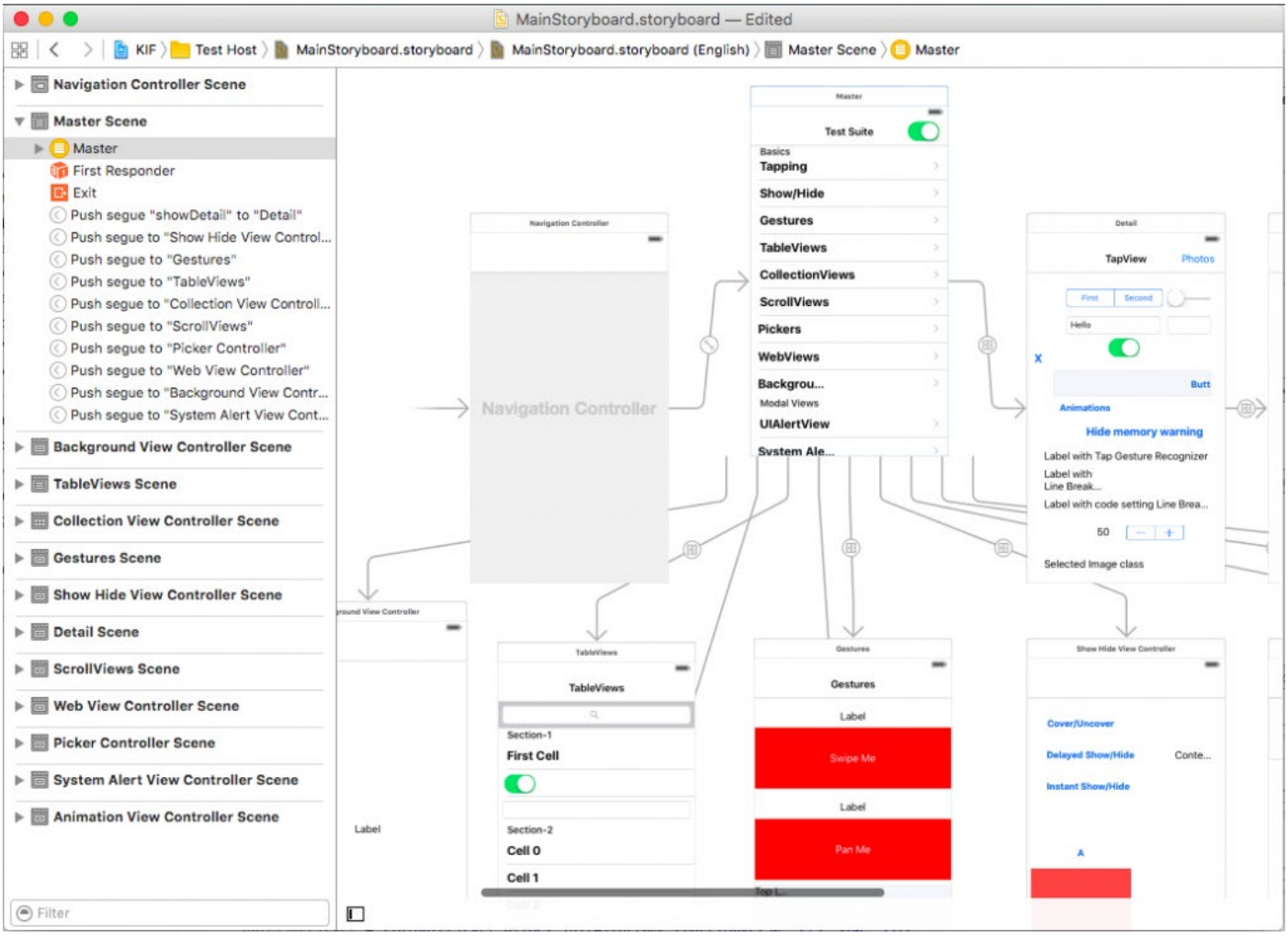
In order to install KIF, you'll need to link the `libKIF static library` directly to your app. Also, there is a readily available example for Swift under KIF repository.

Working on Xcode

To use KIF with your iOS app you have now placed its project, source and test templates under your iOS project. You can also open `KIF.xcodeproj` in Project Navigator and inspect the following files included with the installation:

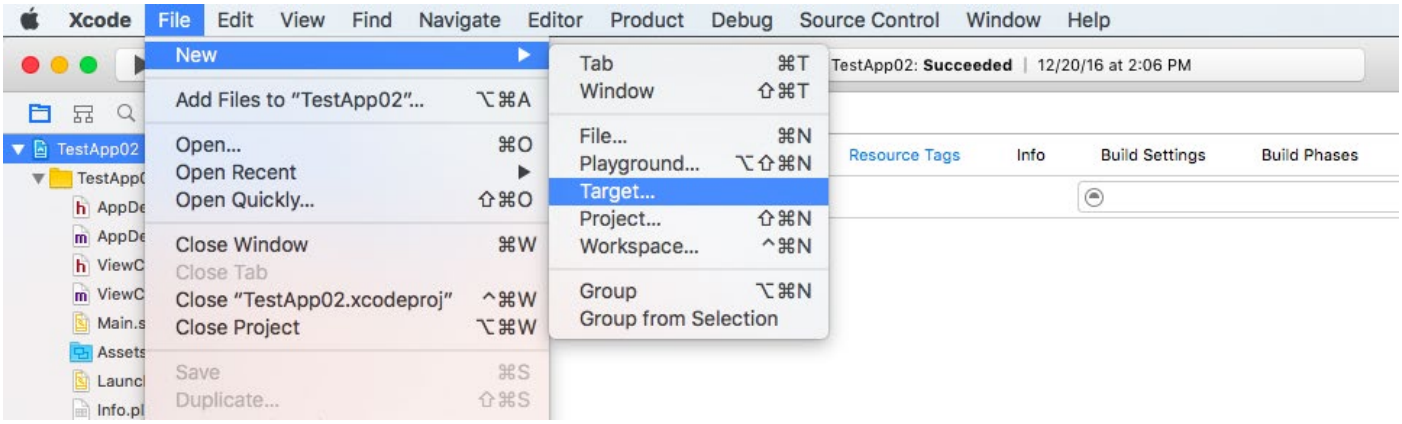


Now, click `MainStoryboard.storyboard` open (under Test Host folder) and the following scene should be open and editable:

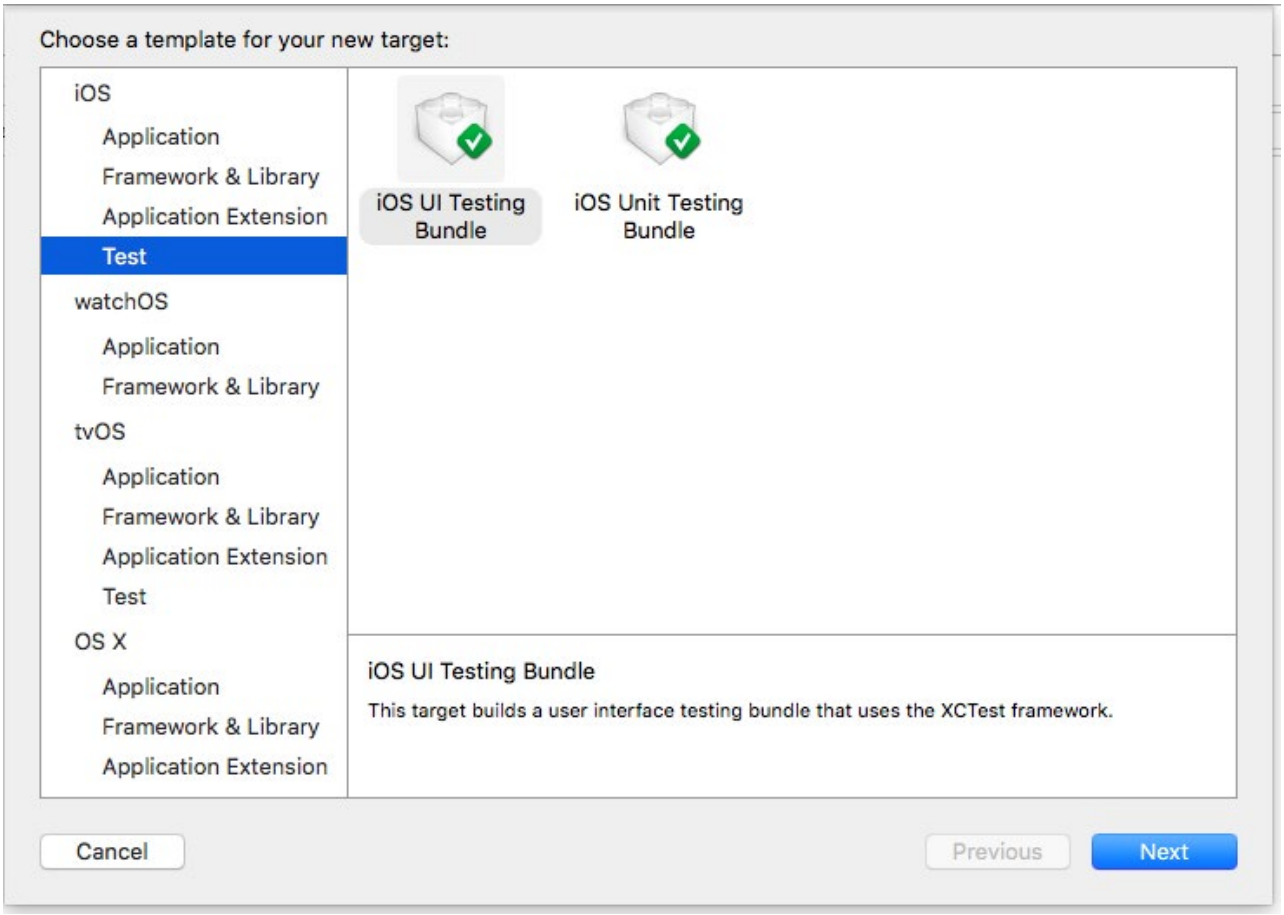


Adding a New Testing Target

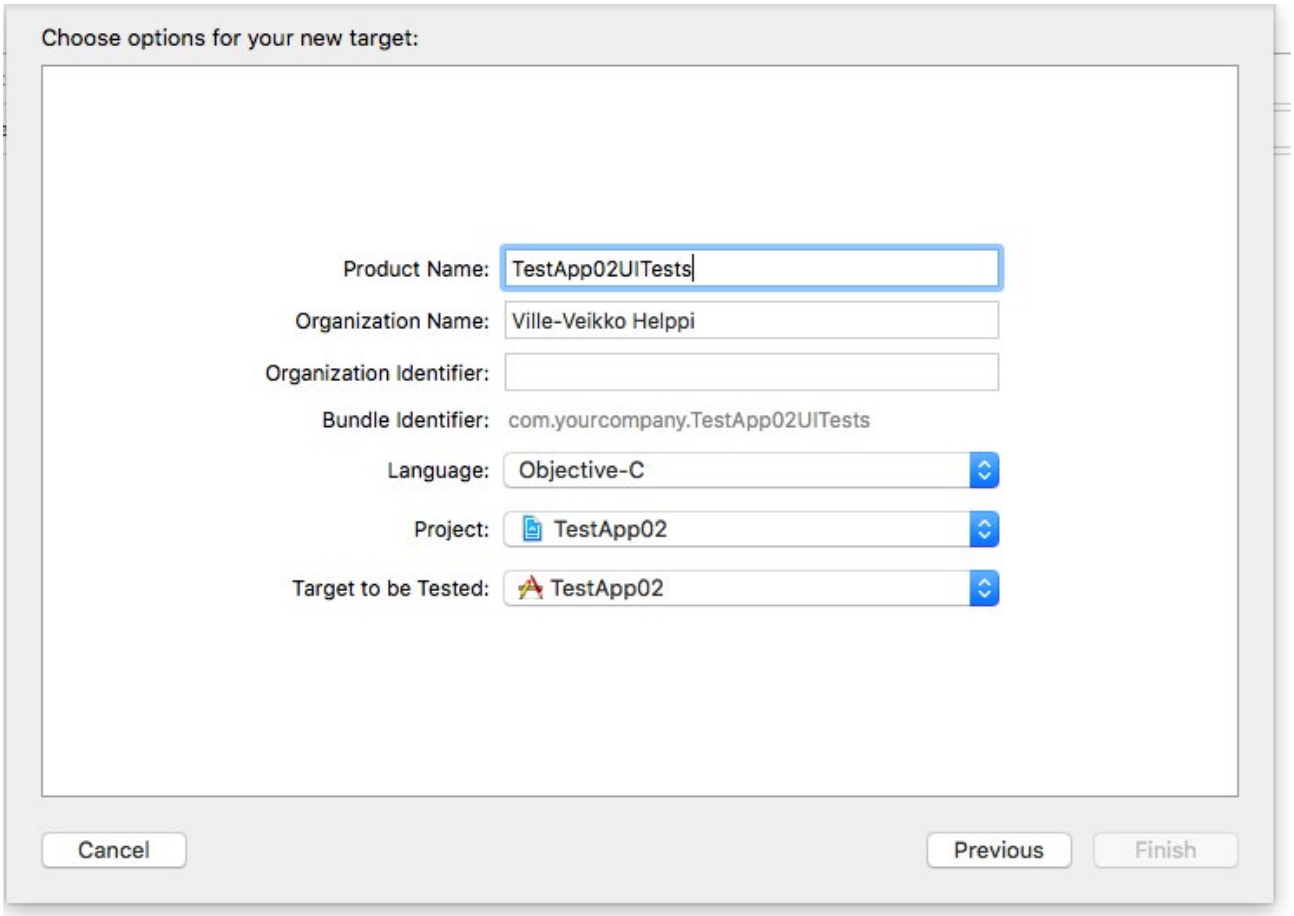
Depending on which Xcode you are using (and yes, we're encouraging people to use the latest and greatest) creating a new test target can be done via different routes.



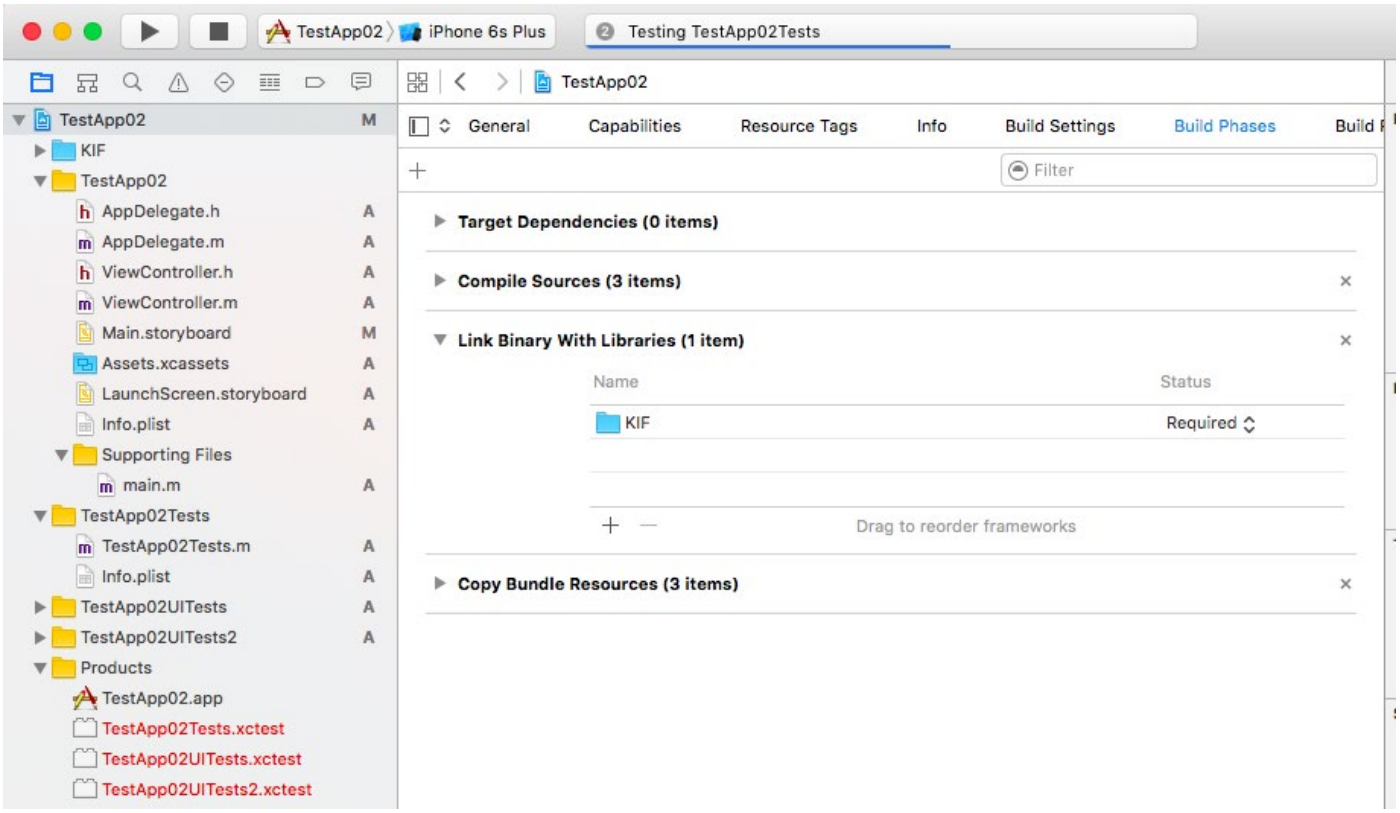
Under File › New › Target you can find the testing target wizard:



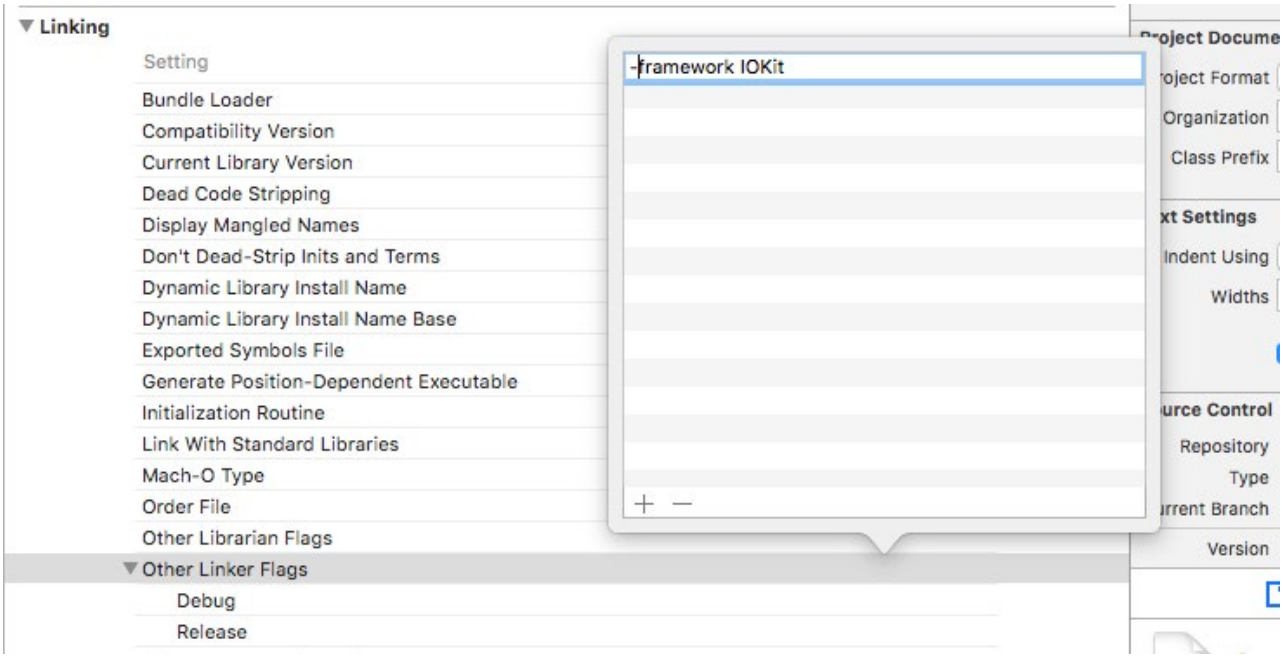
Configure the parameters and create a testing target (click Finish):



Now, you can configure your iOS project. First, click your project name on Project Navigator and you will see the “Build Phases” on the right side. Open “Link Binary With Libraries” and locate `libKIF.a` and click Add. KIF will be now added to your project and library is linked as a binary with your app.



Finally, KIF requires IOKit so this can be added to your project under “Build Settings” › Linking and “Other Linker Flags”. Just add “-framework IOKit” there as a parameter and all should be good:



Chapter

04

The Basics of XCUITest and Using Xcode UI Test Recorder

The User Interface testing for iOS apps provides the way to find and interact with the actual UI of the app, verify that all UI element characteristics are designed and implemented correctly, and navigate through the UI element hierarchy for an app. Now let's take a look at another XCTest derivative framework - XCUITest.

What is XCUITest and how are UI tests done with Xcode for iOS app testing?

The UI tests use XCTest framework as a basis, as well as UIAccessibility. UIAccessibility is integrated with UIKit and AppKit and provides an API for UI behavior tuning and that is used externally from the app/tests.

As XCTest and UI test recorder are integral part of Xcode, it is pretty straightforward to utilize it to set up and implement UI tests. Even using Xcode Server and xcodebuild is a good option.

Xcode UI Test Recorder and What It's Used for

The user interface testing provides lots of information about behavior, potential flaws in design, and especially user experience. Automating the testing effort on variety of different iOS devices is simple and straightforward and the UI test recording feature of Xcode makes it fast to create UI specific tests for iOS apps. You can use Xcode UI Test Recorder to:

› Inspect and insight UI elements and their hierarchy in app

The developer of an application typically knows what UI elements have been built into app, but for testers and users of record-and-playback tools this is not always clear. The whole point of using record-and-playback tools is that it makes creation of test script/UI-specific component interactions fast but also provides the higher level of abstraction of the whole UI. This way testers or other members of team do not need specific information about native UI components but all this is accessible and available through tool.

› Quickly Trace All UI details

The record-and-playback tools provide quickly all details about the application's UI layers, components and element-specific details. These details include name, description, value and even code-specific attributes. Take xpath locators as an

example.

› Record all manual interactions for the app

Record-playback tools allow accurate (as user means those) interactions to be recorded and the right test script generated out of those actions. Code itself in Xcode is quick to edit and user can do tweaks on recorded piece of code before running those on simulator, or preferably, on multiple devices simultaneously.

XCUITest – What Are XCUIApplication and XCUIElement

Implementing user interface tests for iOS apps with XCUITest goes the same way as unit tests are done with XCTest. There are basically no differences in the programming model and the methodology used is pretty much the same as with XCTestCase.

The `XCUIApplication` is basically a proxy for an app that can be launched and terminated. User can tell the application to run in testing mode by defining app as a "Target Application" in Xcode target settings.

```
// Objective-C
XCUIApplication *app = [[XCUIApplication alloc] init];

// Swift
let app = XCUIApplication()
```

The `XCUIElement` is the actual UI element in an iOS application. With iOS applications, XCUIElement provides all the basic symbols and functions for UI element interactions. For example, gestures with XCTest include clicking UI elements (normal, double, pressing) and interacting with the screen (swipe, pinch, zoom, rotate etc.).

The functions used with current XCUIElement are as follows:

```
// Click-based functions
tap()
doubleTap()
twoFingerTap()
tap(withNumberOfTaps: UInt, numberOfTouches: UInt)
press(forDuration: TimeInterval)
press(forDuration: TimeInterval, thenDragTo: XCUIElement)

// Generic UI interactions
swipeLeft()
swipeRight()
swipeUp()
swipeDown()
pinch(withScale: CGFloat, velocity: CGFloat)
rotate(CGFloat, withVelocity: CGFloat)
```

The XCUIElement is constructed using the actual user interface elements on the screen. XCUIElement inherits from NSObject and is used as follows:

```
XCUIApplication *app = [[XCUIApplication alloc] init];

XCUIElement *masterNavigationBar = app.
navigationBars[@"Master"];
XCUIElement *editButton = masterNavigationBar.
buttons[@"Edit"];
```

In order to perform any interaction (tap on this example) on a UI element the following UI interactions (as listed above) can be used:

```
// Objective-C
[masterNavigationBar.staticTexts[@”Master”] tap];
[editButton tap];
```

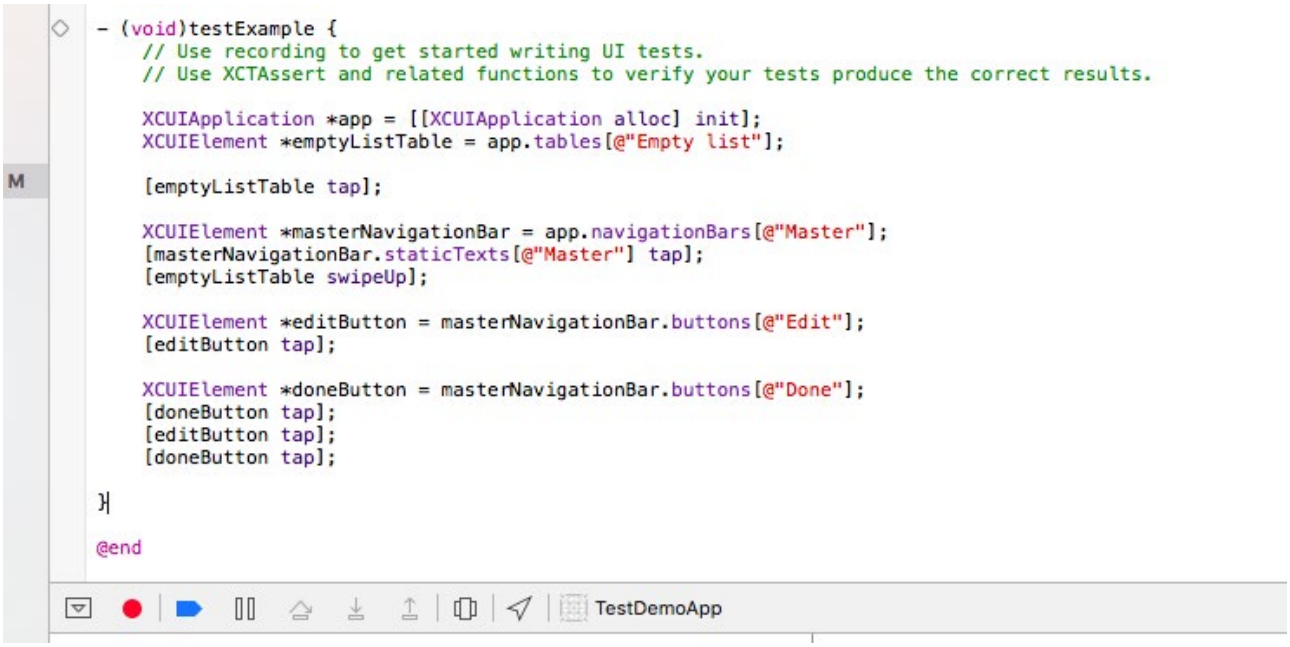
‘tap’ would perform a click on those elements.

The Basics of Using Xcode's UI Test Recorder

The user interface testing features in Xcode include the UI test recorder which allows one to generate test scripts and mocks all interactions that user does when the test script is getting recorded. Seeing this sort of UI test recorder in Xcode is definitely a big plus for Xcode environment and it – for sure – makes test creation much easier.

First, make sure you highlight a method that you want to record your test on. The record button can be found on bottom of the main editor screen and it will turn red (= test is recordable):

After you click the recording button the application will be compiled and started (on simulator or physical devices, based upon your preferences). Now, interact with the application's UI will get all steps recorded and generated as Objective-C test snippet. Below is one example of how it looks:



Note that if you haven't highlighted the test method you want to record, the record button stays gray.

To run your recorded UI test simply hit the Test (Product › Test) and your test will be executed against the application. After the test run, you will get some useful information and some details of potential failures. As Xcode itself doesn't provide much useful information about the UI test run on physical devices, it's always recommended to utilize a cloud-based mobile testing device farm to run specific tests on real iPhones, iPads and iPods at scale.

Conclusion

As many differences between the two operating systems, iOS app testing is quite different than Android app testing. Most iOS developers/testers are busy exploring their options and the tools available to decide what's right for their testing needs.

As for test automation frameworks, XCTest provides a strong base for you to start with. That being said, XCTest still has its limitations here and there. That's why you might want to take a look at other more versatile frameworks, especially those cross-platform frameworks like Appium and Calabash to ease your life, if you are also in charge of the quality of your Android version.

In terms of running the XCTest scripts, it depends on your ultimate goal and how you perceive the business objective of your iOS application. To start with, you can run scripted tests on your own iPhone or iPad to verify if the tests work perfectly and modify the scripts if needed. Once that's done, you probably would like to rely on an iOS device farm that contains various iOS models, from iPhones to iPads to iPods, to thoroughly check the performance and quality of your iOS app. This is what Bitbar Testing (previously known as Testdroid Cloud) has to offer.

Haven't tried Bitbar Testing for iOS app testing? Sign up today to scale up your iOS testing on various real iPhones and iPads.