# Step by step guide for running UI Automation Tests in Parallel Using Xcode 9 Beta

With the recent launch of Xcode-9 at WWDC 17, Apple has provided an official support for multiple simulators on a single host, thereby accommodating the possibility of running multiple UI tests in parallel on same machine. UI testing is an integral and yet time consuming part of any application testing. Apple, in the past, has provided many frameworks to facilitate UI testing, but due to lack of backward compatibility and limited functionality, developers have always been refrained to fully utilize the iOS frameworks to write UI Tests. Also, third party open source BDD frameworks like Cucumber, in integration with Appium, Frank or calabash, can provide many different features and documentation/support to work with.

Over the time, many different attempts have been made by various developers to run the tests in parallel, and after a long research (link to doc), I reached to a conclusion that running cucumber tests in parallel using single host machine is difficult, requires a lot of monkey patches and migration to calabash, and most importantly, not supported anymore. So, what’s the solution? **XCTests.**

With features like multiple simulators and UI recording, Apple has made it easier (In other word, left no choice) for developers to migrate to XCTests. Since, there are not much online support or attempts to run UI Tests in parallel, this article gives you a walkthrough of my attempt to do the same using XCode-9.

**Step-1) Install XCode-9 beta:**

If you already have Xcode installed in your machine, you can install the latest version by renaming the old XCode.app in the Application to Xcode\_<version>. app and then download the latest beta from <https://developer.apple.com/download/> using your apple developer account.

**Step-2) Configuration:**

Apple has stopped providing support for older version of many known functionalities in XCode 9 (For exp. cocoapod 0.39 is no longer supported by XCode-9). If you want to use such things, you need to use XCode-8 as a target. You can set the Xcode version for command line using the following command:

Xcode-select –p : To see the current version of XCode

Sudo xcode-select –s /Application/Xcode-<version>/Contents/Developer – To select the version of Xcode

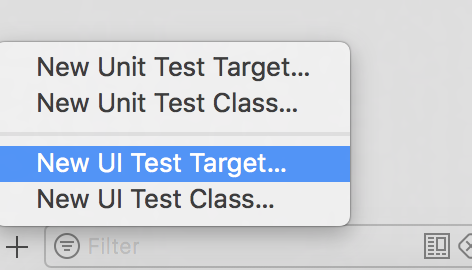
**Step-3) Dummy application:**

If you don’t already have an application you want to test, create a dummy single page application and write/ record simple UI tests. For this article, let’s say you have 6 separate test files under your UI Test target. **The key point here is that you need to divide your tests into n different targets and create one scheme for each target if you want to run n simulators.**

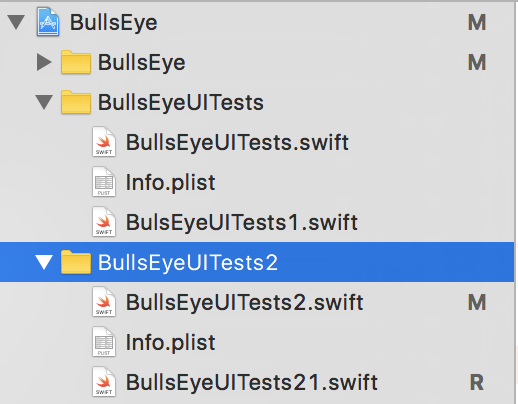
So, for example, to run 2 simulators simultaneously, 2 targets are required and 6 files can be distributed in 2 targets, 3 for each. However, it is recommended that the test files are split based on the total tome required to run the tests, and not based on the number of tests. But for our application, it doesn’t matter as all the dummy tests are similar.

**Step-4) Split the tests:**

Start by going to the Test Navigator, click the + button at the bottom left corner and add new UI Target.

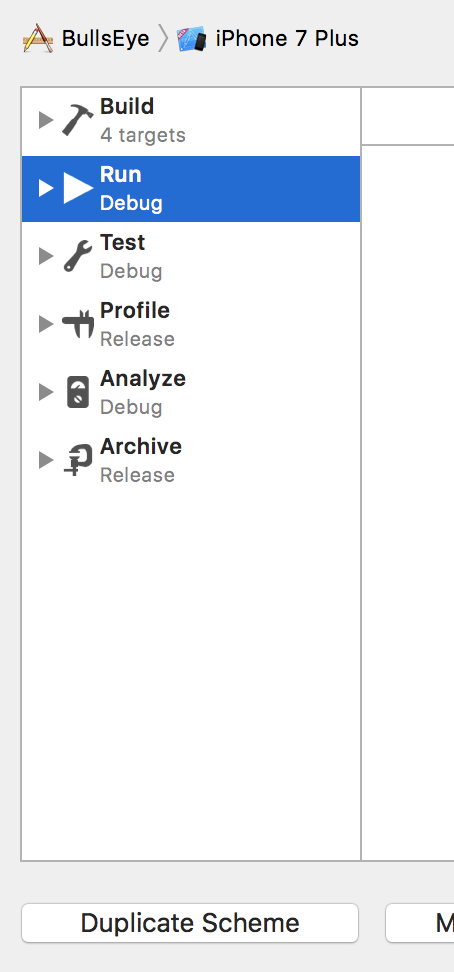


In the dialog box, save the new target with appropriate name. Next you need to shift some of the test cases to the newly create target. You can do that by copying the test files from target1 to target2. After you split the files equally, your new workspace should look something like this:

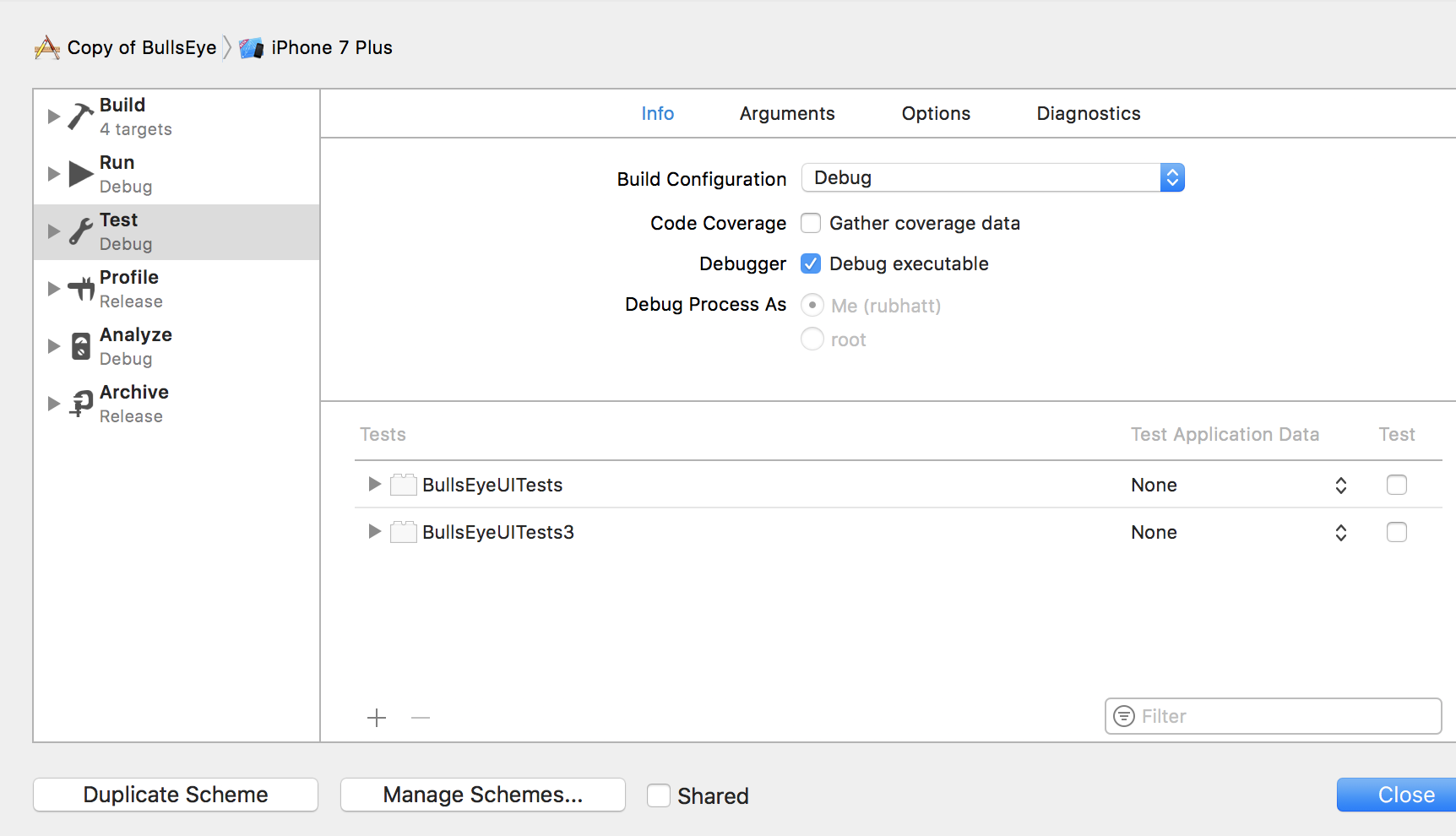


**Step-5) Create new schemes to support those target:**

Now that we have split all our test files between 2 test targets, we need to create 2 schemes to support those targets and run 2 simulators. On the top of your XCode, select the current scheme and go to manage scheme. In the dialog box, double tap on current scheme and you will end up at this dialog box.



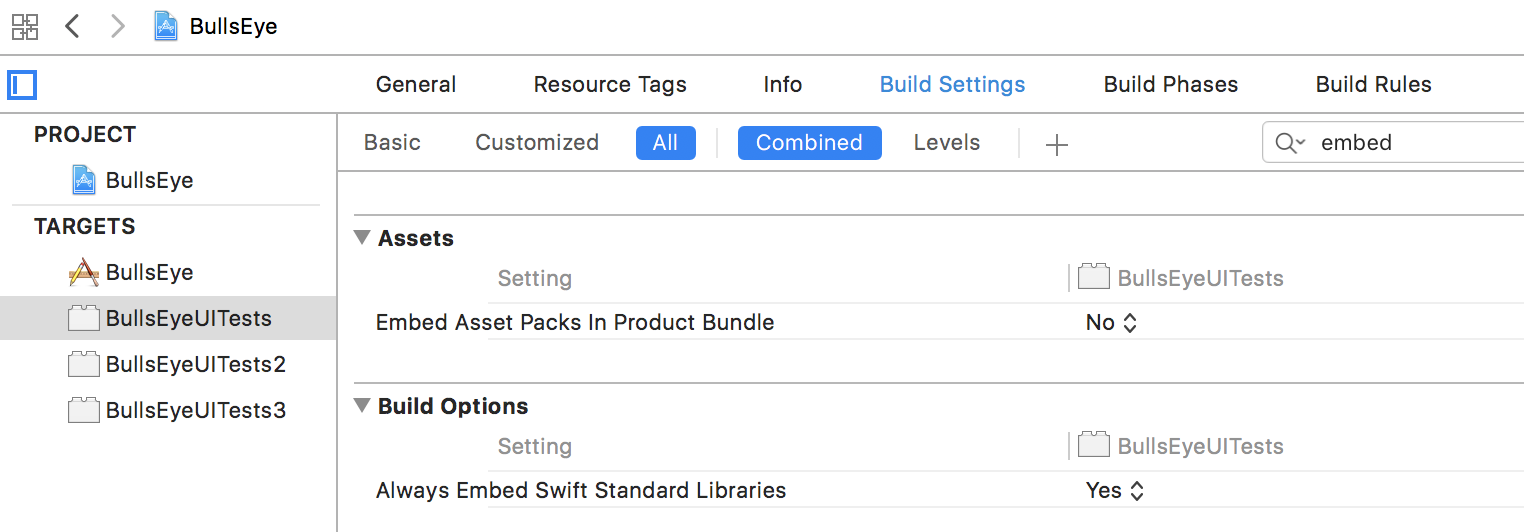
Select Duplicate scheme, enter an appropriate name for the scheme, and in the test configuration, select only the target for which you want to run the tests using this scheme.



So, for scheme1 -> Select only Target1UITests. Repeat the same procedure for both the targets, and for scheme 2-> Select only Target2UITests in the Test configuration.

Now that we have created 2 separate schemes, next step is to execute them on different simulators.

There is one more thing required for each target which facilitates the smooth execution of parallel simulator builds. Select the xcode project and in target navigator, select all the test targets that you are using for UI tests.



Go to Build settings - > search for embed, and you will find “Always Embed Swift Standard Library” in Build options, Set it to YES. That’s all for now.

**Step-6) Execution:**

Even though Xcode 9 supports multiple simulator, you can’t select 2 targets from Xcode UI for any specific scheme. So, you need to use command line to build the application using xcode-build. Since the launch of XCode-9, not many people have tried running the tests In parallel and I only found one tutorial on Medium, However the command used by that tutorial was not complete and after some trial and error, I found the following command works perfectly.

Pre-requisite: considering 2 simulators, you need 2 schemes built with 2 targets specified above. And you will also require 2 simulators to run them. Here, I have created 2 iPhone 6 with same configurations and got their device id from the simulator manager window.

Next step is to build and run the schemes in those simulators. Select scheme1 and Run it in device1. Wait till the application launches and then stop the execution. Do the same for all schemes for all the devices (scheme n on device n). Stop the execution and go to terminal.

Command: In terminal, go to the project folder and go into the directory <ProjectName>.xcproj. So that you can access project.xcworkspace directly. Else you will have to provide the relative path to this workspace. And then run the following command with your values.

xcodebuild clean

-workspace project.xcworkspace

-scheme <Sceheme1>

-sdk iphonesimulator

-destination 'platform=iOS Simulator,id=<Your simulator1 id>' test

&

xcodebuild clean

-workspace project.xcworkspace –

-scheme <Sceheme2>

-sdk iphonesimulator

-destination 'platform=iOS Simulator,id=<Your simulator2 id> ' test

.

.

[& xcodebuild … for nth scheme]

It will try and build both the schemes simultaneously and at the end, you will see the tests running on both the devices at the same time. Hurray...!!

**Step-7) Issues:**

I have tried and added as much configuration possible to avoid errors as I could. However, I must say that this approach sometimes gives error while in build.

Some of the common errors I encountered:

1. The following build command failed:

**ProcessInfoPListFile** /Users/rubhatt….. Info.plist

1. Assertion Failure <unknown>
2. Following build command failed:

CompileSwift normal x86\_64 com.apple.xcode …

These errors can be resolved with one of the following:

1. Clean the product
2. Clean the build folder
3. Remove the Derived Data folder of xCode
4. Reset the simulator on every run
5. Quite the xCode if needed and reopen.

**Therefore, It’s always a good practice to Erase all the content after each run, clean the project and build folder, build the application on each simulator again and then run the command.**

**Resources:**

I have added the links for working demo and my entire copy of application here. Hope you like it.

[**https://github.com/Rushi-Bhatt/IOS\_Development-Testing/tree/master/IOS\_Testing/Parallel\_UITesting\_XCTest/BullsEye**](https://github.com/Rushi-Bhatt/IOS_Development-Testing/tree/master/IOS_Testing/Parallel_UITesting_XCTest/BullsEye)

[**https://github.com/Rushi-Bhatt/IOS\_Development-Testing/blob/master/IOS\_Testing/Parallel\_UITesting\_XCTest/workind\_demo2\_720p.mov**](https://github.com/Rushi-Bhatt/IOS_Development-Testing/blob/master/IOS_Testing/Parallel_UITesting_XCTest/workind_demo2_720p.mov)

**References:**

[**https://medium.com/@t.camin/parallelizing-ui-tests-28c16000f141**](https://medium.com/@t.camin/parallelizing-ui-tests-28c16000f141)