Testing the UI of iOS apps using XCTest and XCUITest in Swift

Overview

In this tutorial, we will "create" some UI-based tests for an iOS application using XCTest testing framework along with XCUITest.

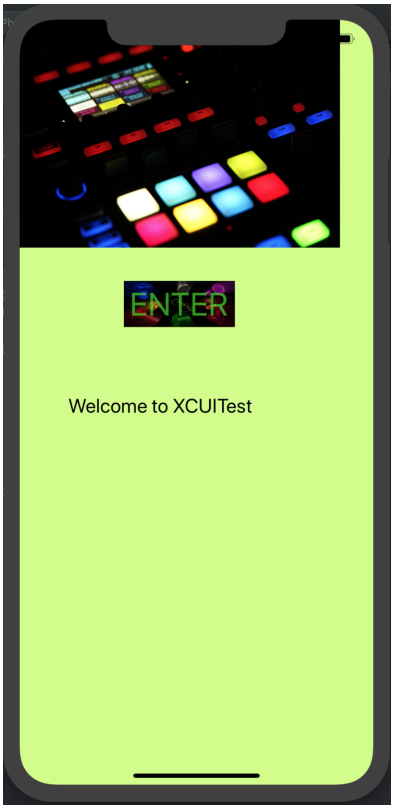
Requirements

- Xcode
- Xcode command line tools
  - `xcode-select --install`
- xcpretty
- (optional) fastlane and trainer plugin

Description

For this tutorial, we'll use a sample iOS app with UI tests by Shashikant, with minor updates as tracked in this fork.

The iOS application is quite simple: it has just a button and a text that appears whenever clicking on it.



The application has one View Controller class.

XCUITest101/ViewController.swift

```
//
//  ViewController.swift
//  XCUITest101
//
//  Created by Shashikant Jagtap on 24/09/2018.
//  Copyright © 2018 Shashikant Jagtap. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var welcomeText: UILabel!
    @IBAction func enterPressed(_ sender: Any) {

        welcomeText.text = "Welcome to XCUITest"
        welcomeText.isHidden = false

    }
    override func viewDidLoad() {
        super.viewDidLoad()
        welcomeText.isHidden = true
        // Do any additional setup after loading the view, typically from a nib.
    }

}
```

The project contains two tests implemented in Swift: one named "testRecorded" (which is not an actual test case) and another, the real one, named "testRefactored".

Tests use XCTest framework and XCUITest to load the application and execute the UI-baed tests.

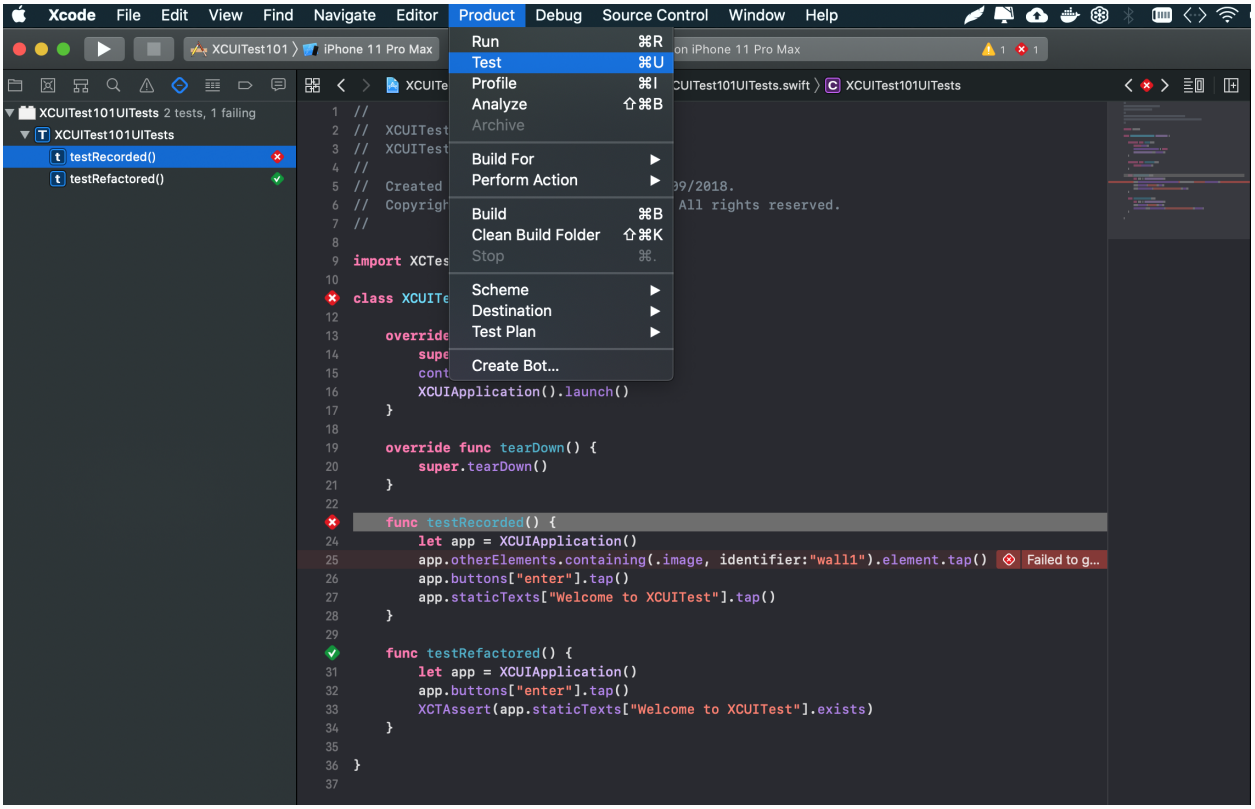
XCUITest101UITests/XCUITest101UITests.swift

```
//
//  XCUITest101UITests.swift
//  XCUITest101UITests
//
//  Created by Shashikant Jagtap on 24/09/2018.
//  Copyright © 2018 Shashikant Jagtap. All rights reserved.
//

import XCTest

class XCUITest101UITests: XCTestCase {

    override func setUp() {
        super.setUp()
        continueAfterFailure = false
        XCUIApplication().launch()
    }

    override func tearDown() {
        super.tearDown()
    }

    func testRecorded() {
        // this is not an actual test...
        let app = XCUIApplication()
        app.otherElements.containing(.image, identifier:"wall1").element.tap()
        app.buttons["enter"].tap()
        app.staticTexts["Welcome to XCUITest"].tap()
    }

    func testRefactored() {
        let app = XCUIApplication()
        app.buttons["enter"].tap()
        XCTAssert(app.staticTexts["Welcome to XCUITest"].exists)
    }

}
```

In order to run the tests from the command line or during CI, you can use `xcodebuild`. By processing its output using `xcpretty`, a JUnit XML can be generated (`build/reports/junit.xml`, by default).

```
xcodebuild -project XCUITest101.xcodeproj/ -scheme XCUITest101 -destination 'platform=iOS Simulator,OS=13.1,
name=iPhone 11 Pro Max' clean build test CODE_SIGN_IDENTITY="" CODE_SIGNING_REQUIRED=NO | xcpretty -r junit
```

If you're using Xcode, then you can also use it to write and run your tests (either all of them or just a specific one).

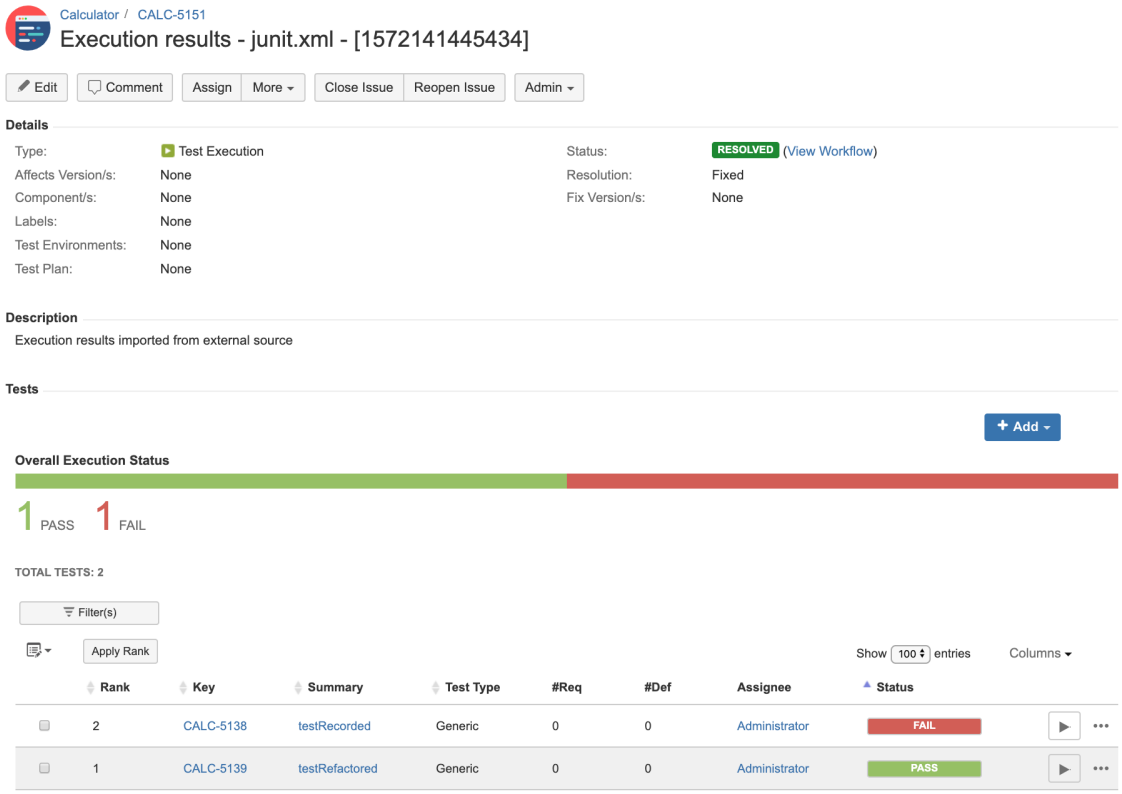


However, by default, this won't produce a JUnit XML report by itself which can, later on, be uploaded to Xray.

After running the tests and generating the JUnit XML report (e.g., junit.xml), it can be imported to Xray (either by the REST API or by using one of the CI plugins or through **Import Execution Results** action within the Test Execution).

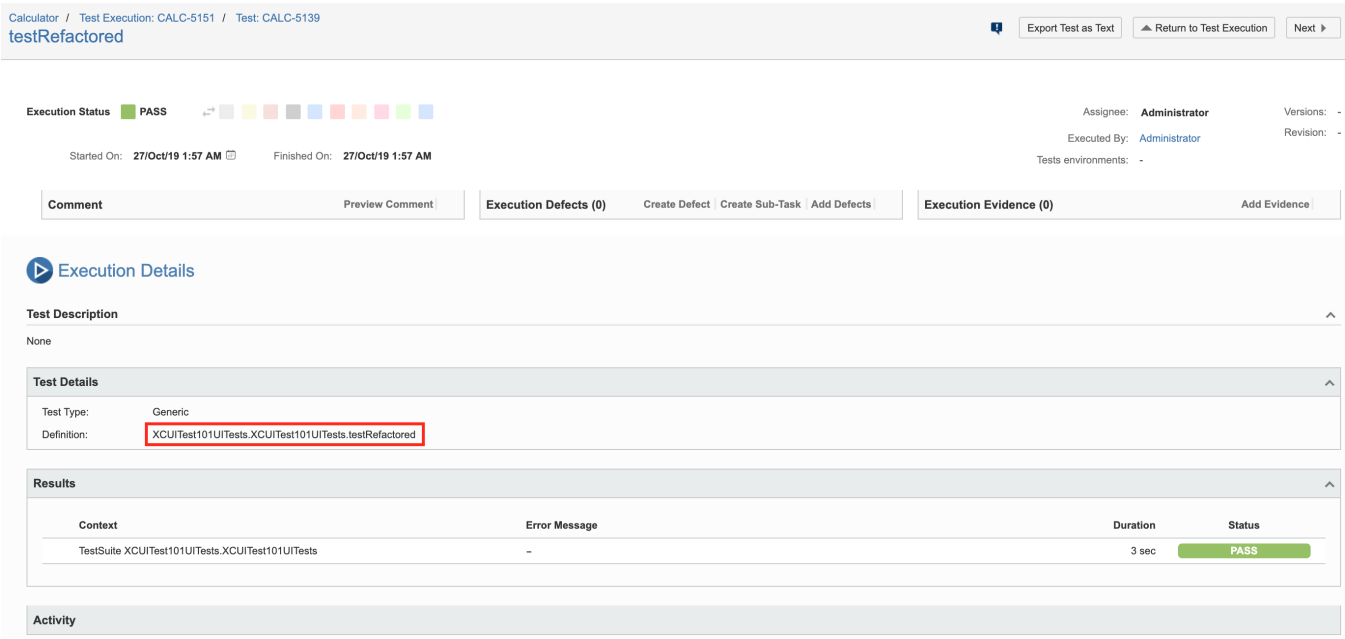
```
curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@build/reports/junit.xml" http://jiraserver.
example.com/rest/raven/1.0/import/execution/junit?projectKey=CALC
```

A Test Execution will be created containing information about the executed scenarios.



Each test is mapped to a Generic Test in Jira, and the **Generic Test Definition** field contains the name of the class concatenated with the method name of the corresponding automated test.

The Execution Details of the Generic Test contains information about the "Test Suite" (as per JUnit format), which in this case corresponds to the fully-qualified name of the class holding the test.



Notes

You should be able to use fastlane (docs here) to build and run your tests by using the trainer plugin.

In that case, you need to define a *lane* to run the tests and invoke the *trainer* plugin.

fastlane/Fastfile

```
default_platform(:ios)

platform :ios do
  desc "Run tests"
  lane :test do
    scan(scheme: "XCUITest101",
       output_types: "",
       fail_build: false)

    trainer(output_directory: "build/reports/")
  end

end
```

Notes

- xcpretty project seems to be not very active
- xcpretty has a known limitation that inhibits the processing of multiline error descriptions (only the header is imported as the log/output of the assertion)

References

- https://developer.apple.com/documentation/xctest
  https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/09-ui_testing.html
- https://github.com/xcpretty/xcpretty
- https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/08-automation.html#//apple_ref/doc/uid/TP40014132-CH7-SW3
- https://www.slideshare.net/ShankarAnamalla/ios-app-testing-with-xctest-and-xcuitest