Testing iOS apps using XCTest in Swift

Overview

In this tutorial, we will "create" some tests for an iOS application using XCTest testing framework.

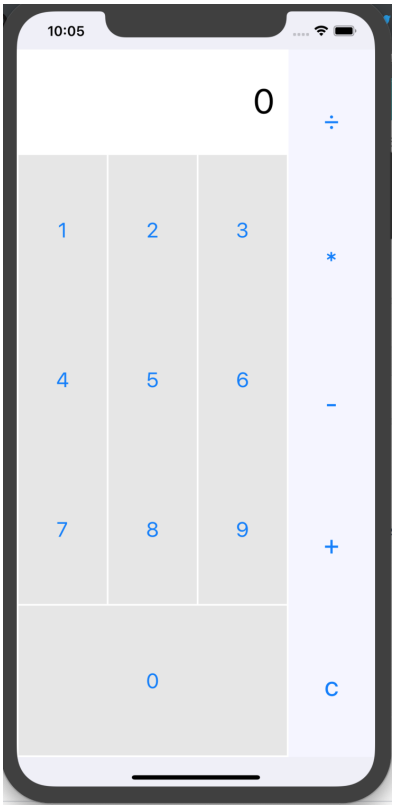
Requirements

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

Description

For this tutorial, we'll use a sample project composed of a calculator iOS app with unit tests by Ryan King, with minor updates as tracked in this fork.

The iOS application is quite simple: it contains one interactor and one controller.



The application has one interactor class.

CalculatorInteractor.swift

```
// MARK: Frameworks

import Foundation

// MARK: CalculatorInteractor

class CalculatorInteractor {
    func add(numberOne: Float, to numberTwo: Float) -> Float {
        return numberOne + numberTwo
    }

    func subtract(numberOne: Float, from numberTwo: Float) -> Float {
        return numberTwo - numberOne
    }

    func multiply(numberOne: Float, by numberTwo: Float) -> Float {
        return numberOne * numberTwo
    }

    func divide(numberOne: Float, by numberTwo: Float) -> Float {
        return numberOne / numberTwo
    }
}
```

The project contains tests for the interactor implemented in Swift, along with two dummy, empty unit tests implemented in a different class.

Tests use XCTest framework and thus extend XCTestCase. The following class validates the arithmetic operations.

UnitTest-CalculatorTests/CalculatorInteractorTests.swift

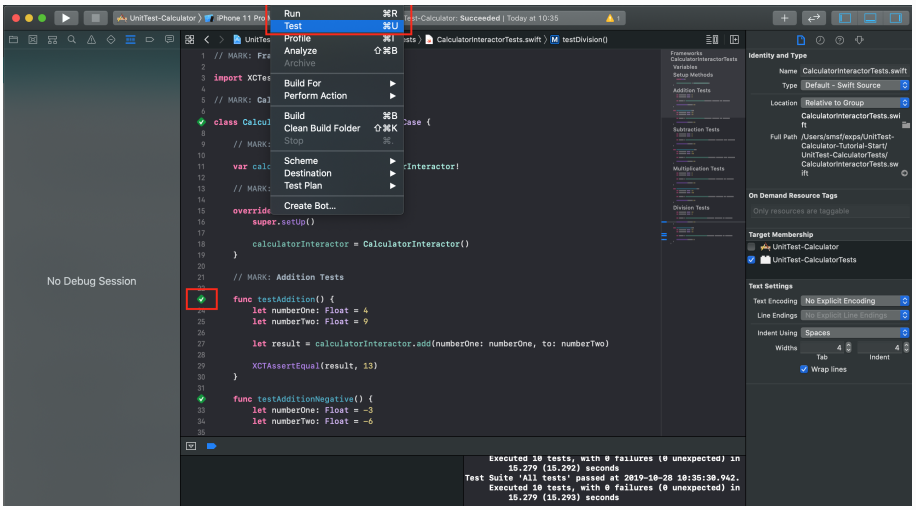
```
// MARK: Frameworks

import XCTest

// MARK: CalculatorInteractorTests

class CalculatorInteractorTests: XCTestCase {

    // MARK: Variables

    var calculatorInteractor: CalculatorInteractor!

    // MARK: Setup Methods

    override func setUp() {
        super.setUp()

        calculatorInteractor = CalculatorInteractor()
    }

    // MARK: Addition Tests

    func testAddition() {
        let numberOne: Float = 4
        let numberTwo: Float = 9

        let result = calculatorInteractor.add(numberOne: numberOne, to: numberTwo)

        XCTAssertEqual(result, 13)
    }
```

```
    func testAdditionNegative() {
        let numberOne: Float = -3
        let numberTwo: Float = -6

        let result = calculatorInteractor.add(numberOne: numberOne, to: numberTwo)

        XCTAssertEqual(result, -9)
    }

    // MARK: Subtraction Tests

    func testSubtraction() {
        let numberOne: Float = 9
        let numberTwo: Float = 4

        let result = calculatorInteractor.subtract(numberOne: numberOne, from: numberTwo)

        XCTAssertEqual(result, -5)
    }

    func testSubtractionNegative() {
        let numberOne: Float = -6
        let numberTwo: Float = -12

        let result = calculatorInteractor.subtract(numberOne: numberOne, from: numberTwo)

        XCTAssertEqual(result, -6)
    }

    // MARK: Multiplication Tests

    func testMultiplication() {
        let numberOne: Float = 9
        let numberTwo: Float = 4

        let result = calculatorInteractor.multiply(numberOne: numberOne, by: numberTwo)

        XCTAssertEqual(result, 36)
    }

    func testMultiplicationNegative() {
        let numberOne: Float = -2
        let numberTwo: Float = -12

        let result = calculatorInteractor.multiply(numberOne: numberOne, by: numberTwo)

        XCTAssertEqual(result, 24)
    }

    // MARK: Division Tests

    func testDivision() {
        let numberOne: Float = 28
        let numberTwo: Float = 2

        let result = calculatorInteractor.divide(numberOne: numberOne, by: numberTwo)

        XCTAssertEqual(result, 14)
    }

    func testDivisionNegative() {
        let numberOne: Float = -9
        let numberTwo: Float = -3

        let result = calculatorInteractor.divide(numberOne: numberOne, by: numberTwo)

        XCTAssertEqual(result, 3)
    }
}
```

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 UnitTest-Calculator.xcodeproj/ -scheme UnitTest-Calculator -destination 'platform=iOS Simulator,OS=13.1,name=iPhone 8' 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).

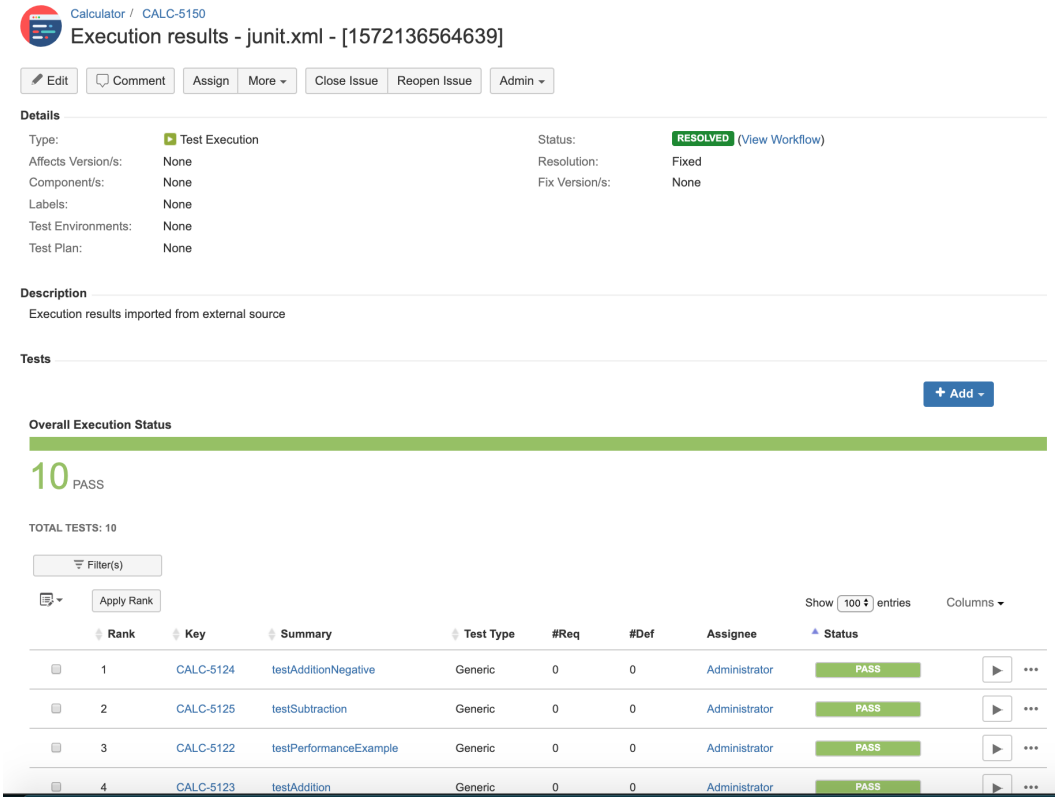


However, by default, this won't produce a JUnit XML report by itself which can, later on, be uploaded to Xray.

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).

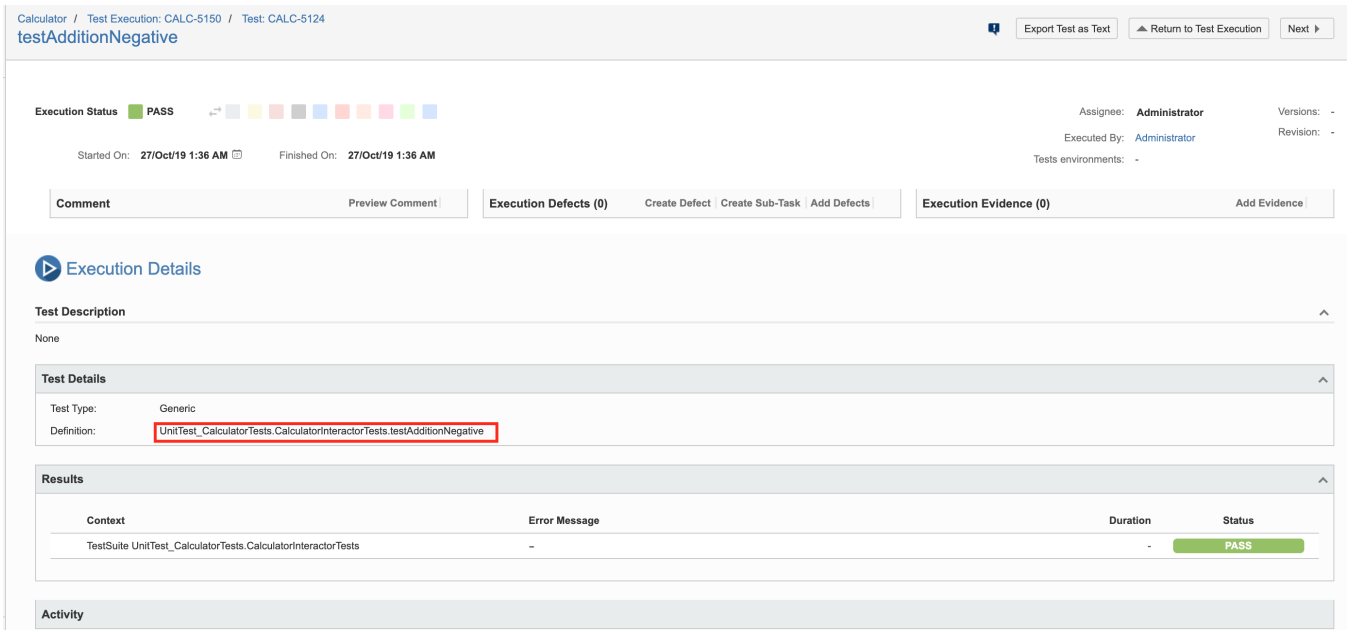
```
curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@build/reports/junit.xml" http://jiraserver.example.com/rest/raven/1.0/import/execution/junit?projectKey=CALC
```

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: "UnitTest-Calculator",
       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/01-introduction.html#//apple_ref/doc/uid/TP40014132-CH1-SW1
- https://github.com/rtking1993/UnitTest-Calculator-Tutorial-Start
- https://ios-cookbook.com/2018/03/28/writing-unit-tests-with-xctest/
- https://github.com/xcpretty/xcpretty
- https://www.slideshare.net/ShankarAnamalla/ios-app-testing-with-xctest-and-xcuitest
- https://github.com/zanizrules/fastlane-plugin-xcresult_to_junit