Wristband association App Developer’s Guide

Section 0: Overview 1

Project Credentials 1

User Story: 1

Restrictions and Requirements (MoSCoW method) 1

Acceptance Tests 2

http://arteksoftware.com/bdd-tests-with-xamarin-uitest-and-specflow/ 2

Section I Preparing to Develop the app 2

Install Development IDE 2

Install BDD Framework 2

Create Main Solution: 3

Setup the MVVMLight framework 5

Add BDD Project 6

Refactoring the Shared Project to MVVM model 9

Section 2- Developing the Application 10

Section 0: Overview

# Project Credentials

contact point: Reza Alemy (author), reza@alemy.net

# User Story:

**As a** Mobile App developer

**I want** to have a boilerplate example of using BDD and MVVM-Light in Xamarin framework

**So that** I can develop multiplatform apps that use the BDD technique for testing and MVVM-light for component management.

# Restrictions and Requirements (MoSCoW method)

* The boilerplate project MUST show how to use Visual Studio to set up the app codebase
* The boilerplate project MUST show how to add Specflow for BDD support to the app codebase
* The boilerplate project MUST show how to write Specflow steps in Xamarin.UITest framework
* The boilerplate project MUST show how to add pages and components using MVVM-Light framework

# Acceptance Tests

Given: The repo is cloned

When: I read the documentation included

Then: I have a step by step explanation on how to setup and develop Xamarin apps using Specflow and MVVM.

Reference and Credits:

Rob Gibbens article for Specflow:

<http://arteksoftware.com/bdd-tests-with-xamarin-uitest-and-specflow/>

Section I Preparing to Develop the app

# Install Development IDE

Selected Technology: Visual Studio, C# language on Xamarin framework

Presented Environment: Visual Studio for Mac

Downloaded from: <https://www.xamarin.com/download>

Version: 7.2.2 (build 11)

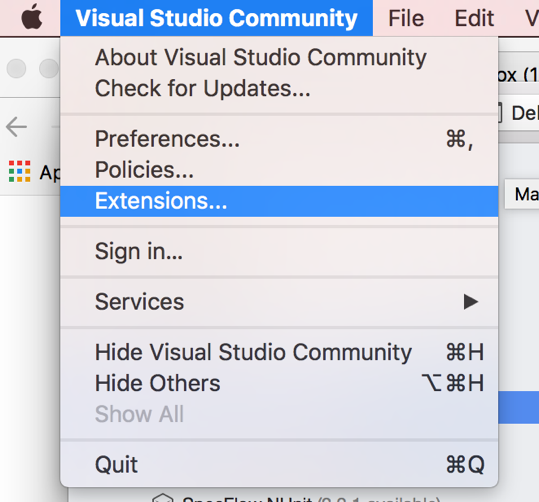
# Install BDD Framework

Selected Technology: Gherkin BDD language

Presented as Visual Studio for Mac Extension: Specflow for mac

Downloaded from: <https://github.com/straighteight/SpecFlow-VS-Mac-Integration/releases/tag/1.11.0.0>

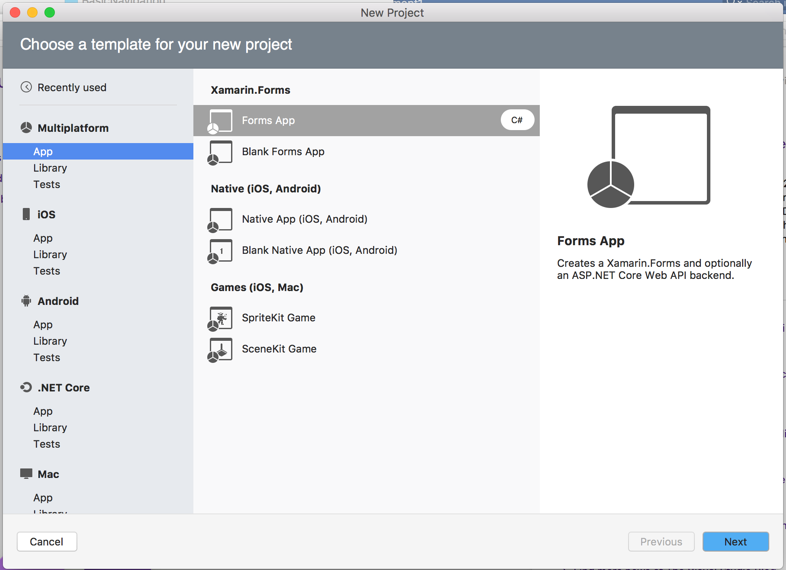
To install, go to 'Visual Studio.../Extensions...' click the button on the far left saying "Install from file..." and select the mpack file from your hard drive. The addin will install and you can now run and creat specflow tests

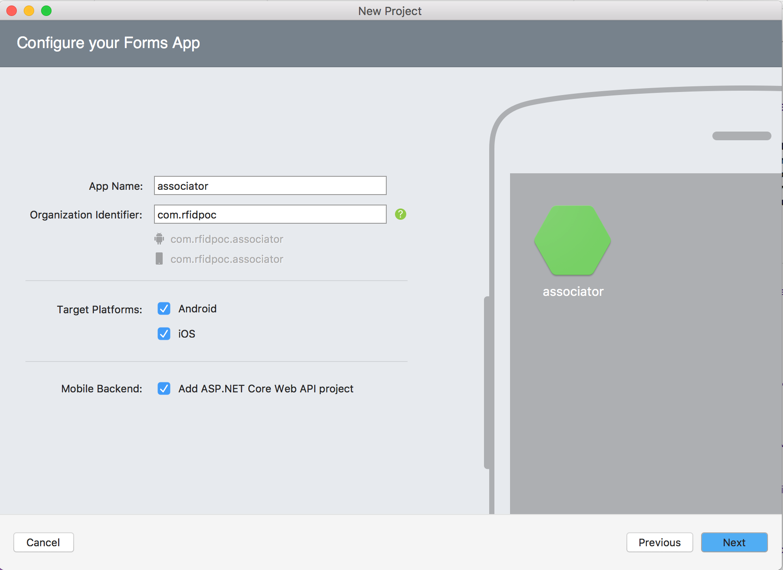


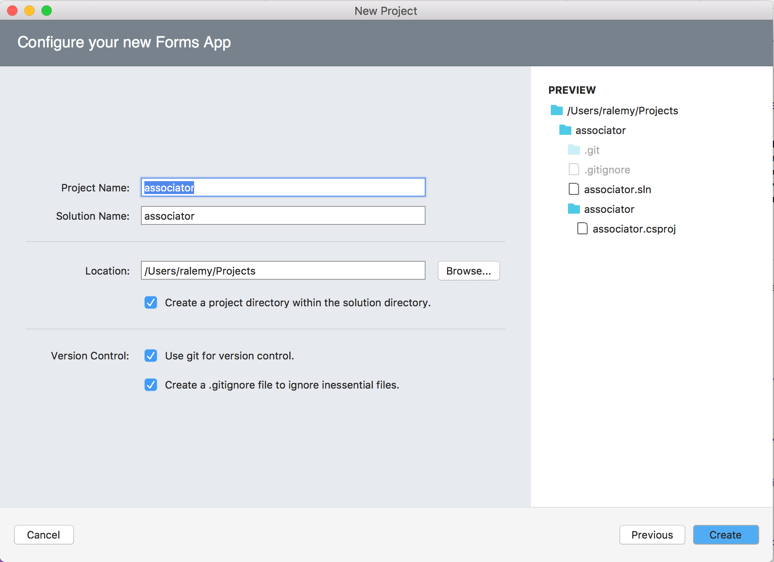
Restart Visual Studio for Mac

# Create Main Solution:

Create a solution with Android and iOS.

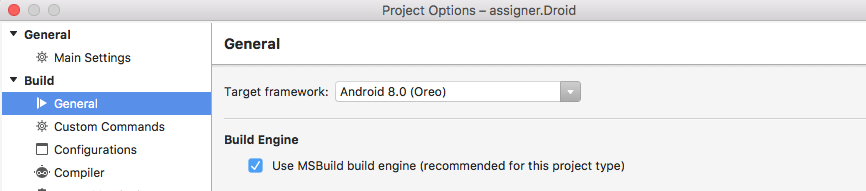






Target the latest Android framework for the Droid project (8.0 for this writing)

* Right click on Droid project and select options
* Go to Build->General->target framework



Install the latest Android SDK for emulator and solution in Tools->SDK Manager



Install the Lollipop Android SDK for emulator

Create a Moto G emulator profile in AVD Manager.

Update Packages for both Droid and iOS.

Add package MVVMLightLibs to both droid and ios frameworks.

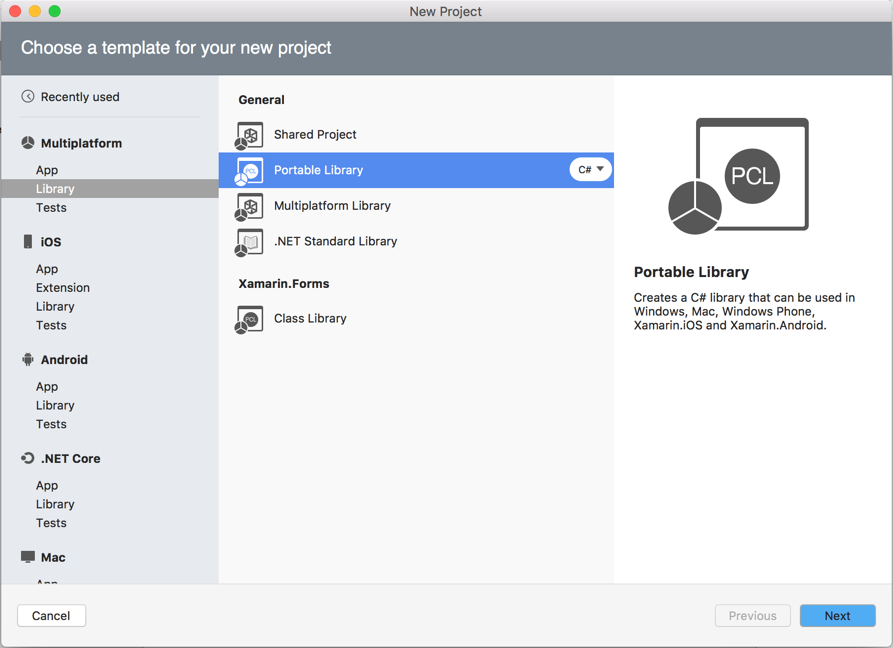
Add package Xamarin.TestCloud.Agent to iOS project

In the iOSProject, find the AppDelegate.cs, and start Calabash:



# Setup the MVVMLight framework

Add a new project for MVVM framework as a portable library

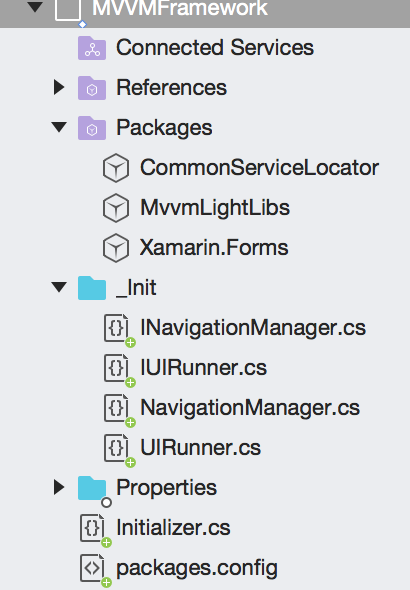


Add MVVMLightLibs and Xamarin.Forms package to this framework

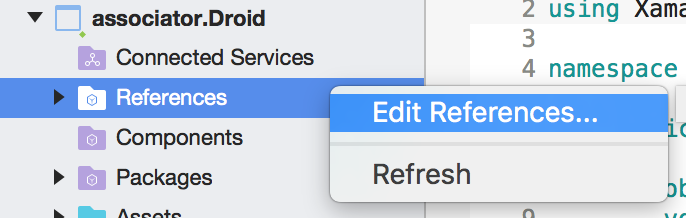
Copy the MVVMFramework/\_init directory from reference repo to this framework

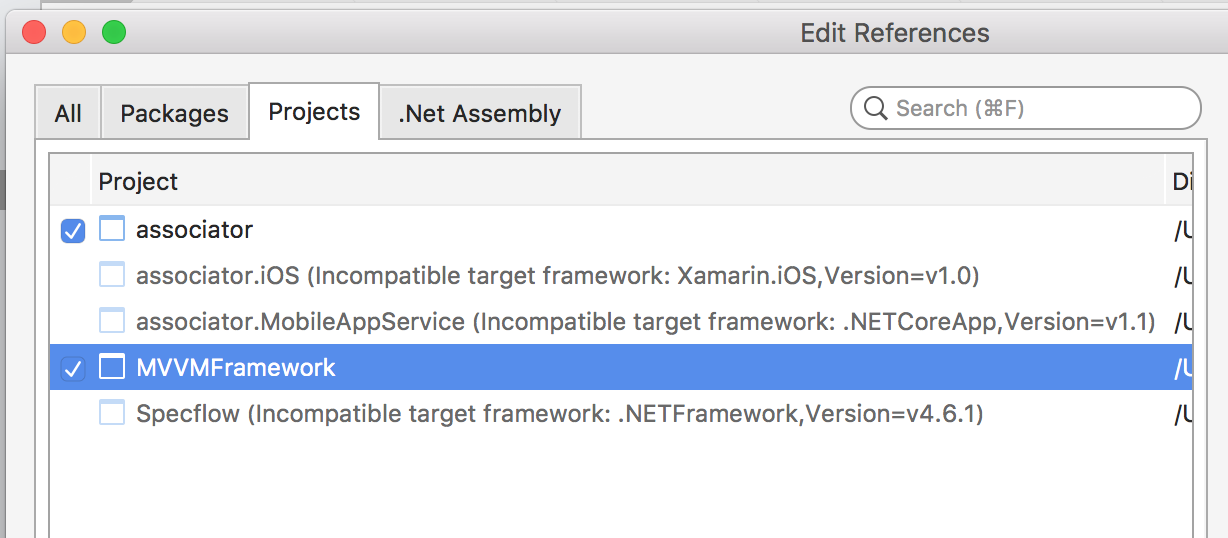
Copy the MVVMFramework/Initializer.cs from reference repo to this framework

Delete MyClass.cs file from the project:



Reference this framework in Droid and iOS projects:

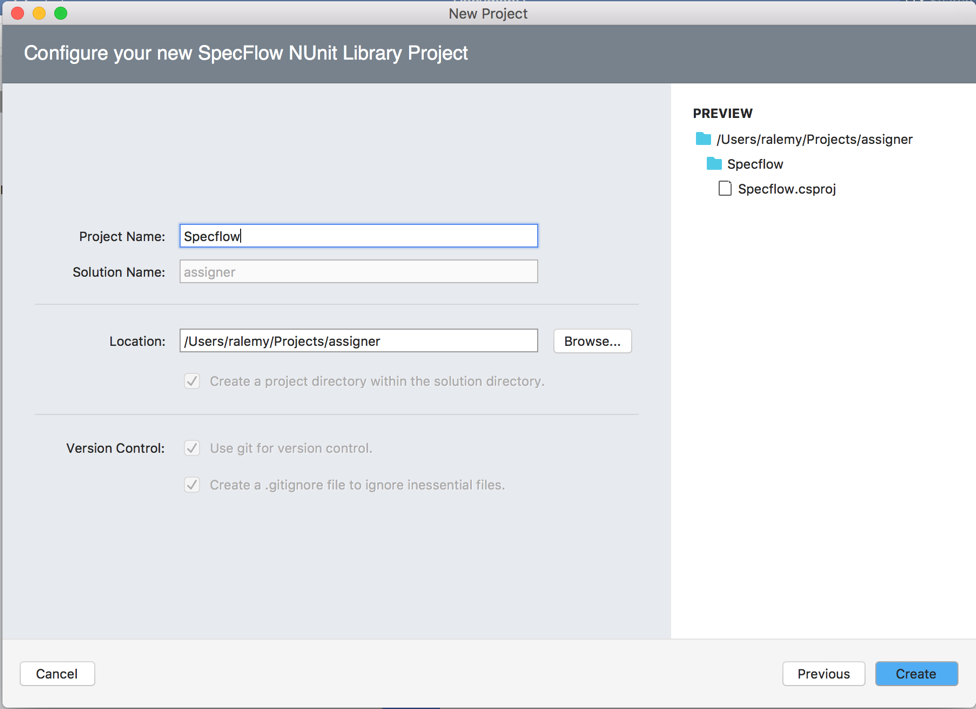




# Add BDD Project

Add a new Specflow project from miscellaneous section



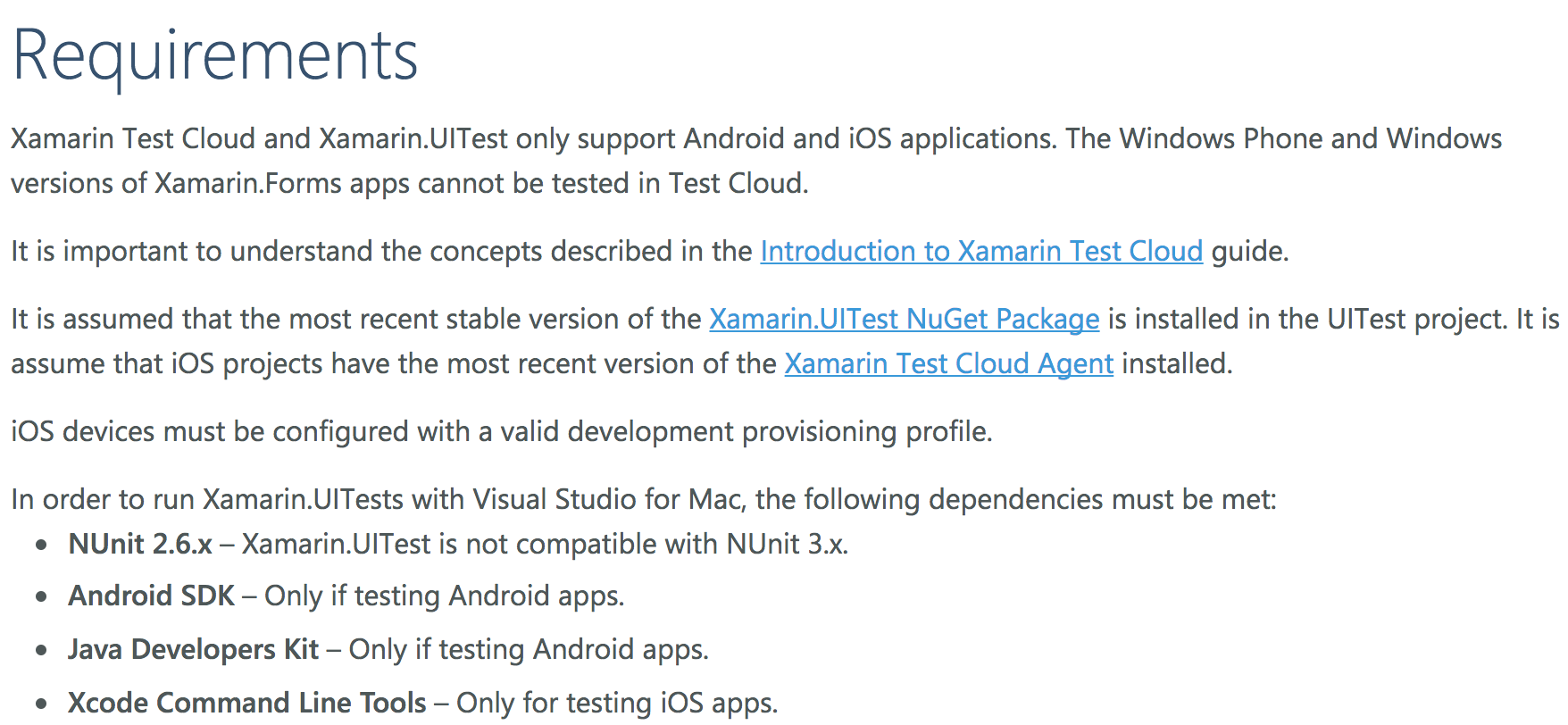


Update packages on Specflow project.

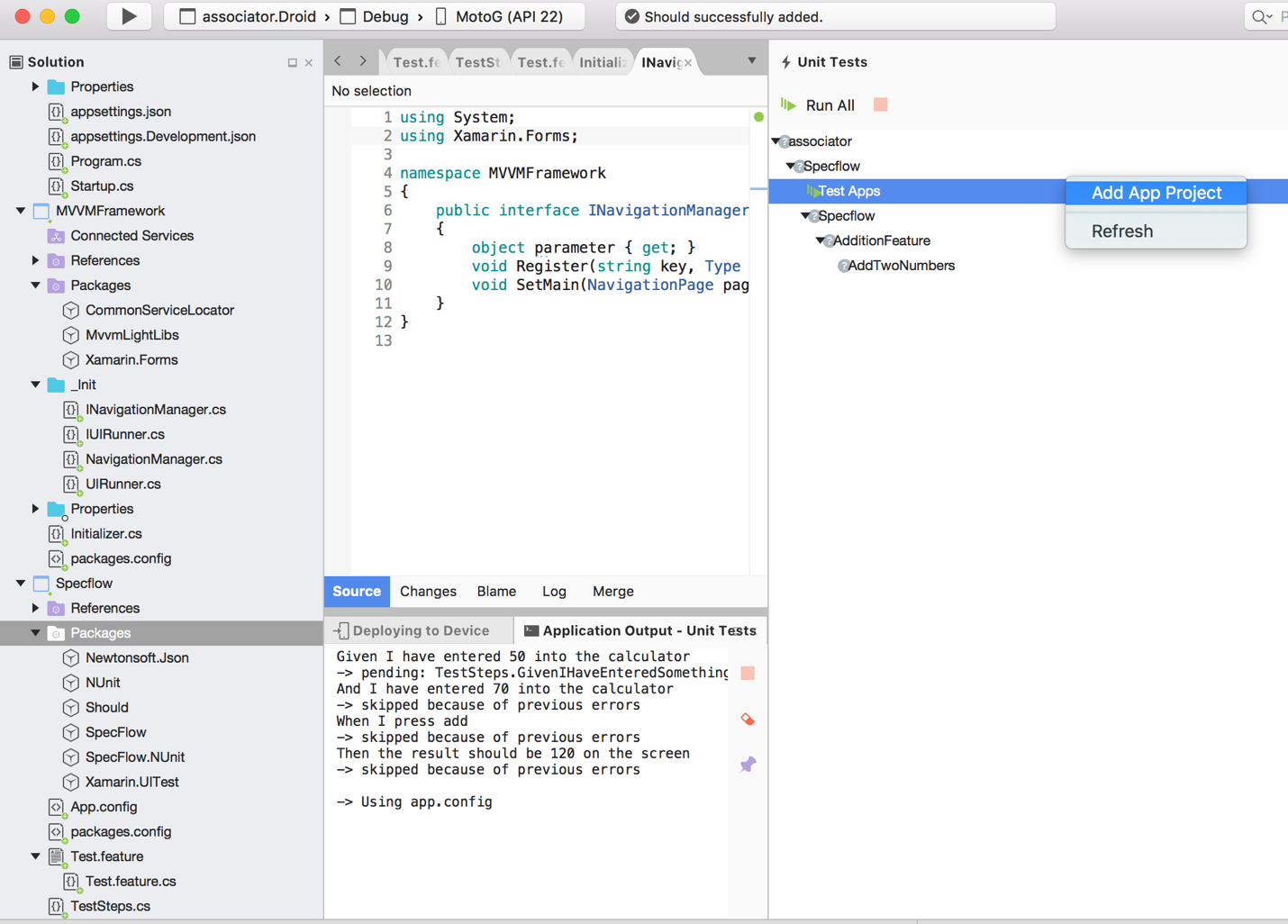
Remove Specflow.NUnit and NUnit packages

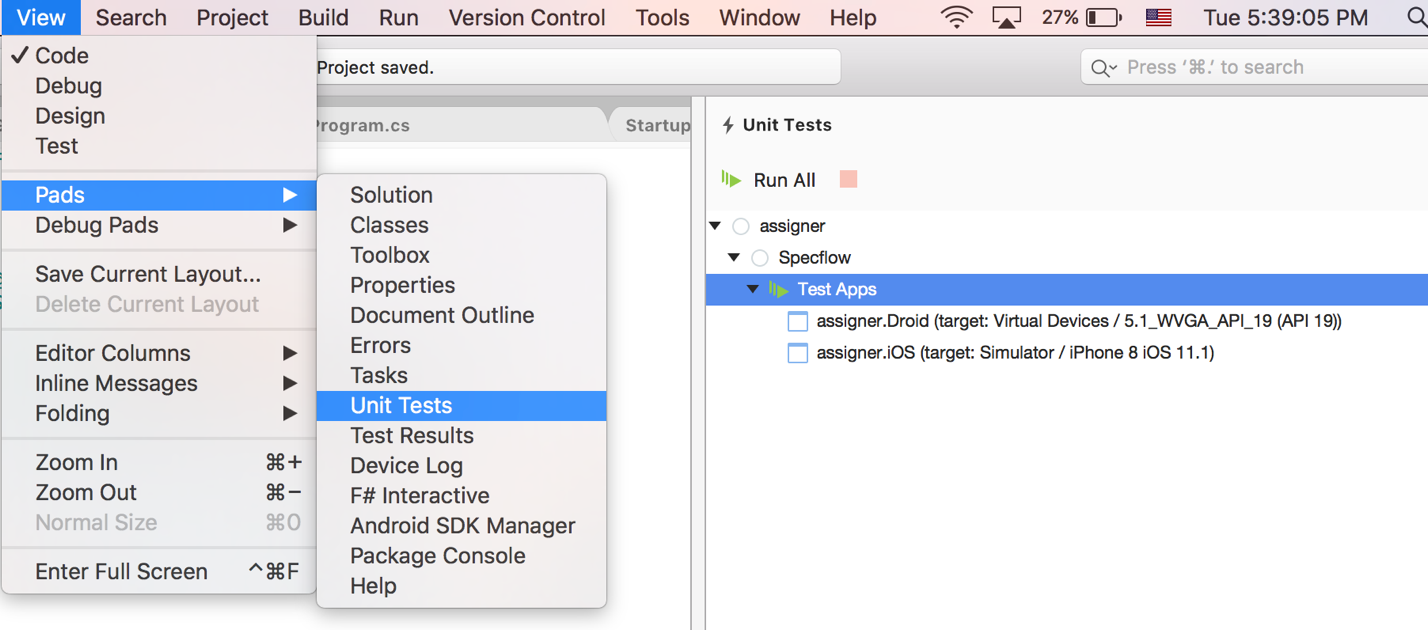
Add Xamarin.UiTest, NUnit (2.6.4), Docopt.Net and Should packages to Specflow project.

* Please note: **based on Xamarin.UITest documentation at the time of this writing, NUnit 2.6.x is the only supported version. Do not install higher versions**.



Open the UnitTest pad from the View menu and add the droid and iOS projects as Test Apps



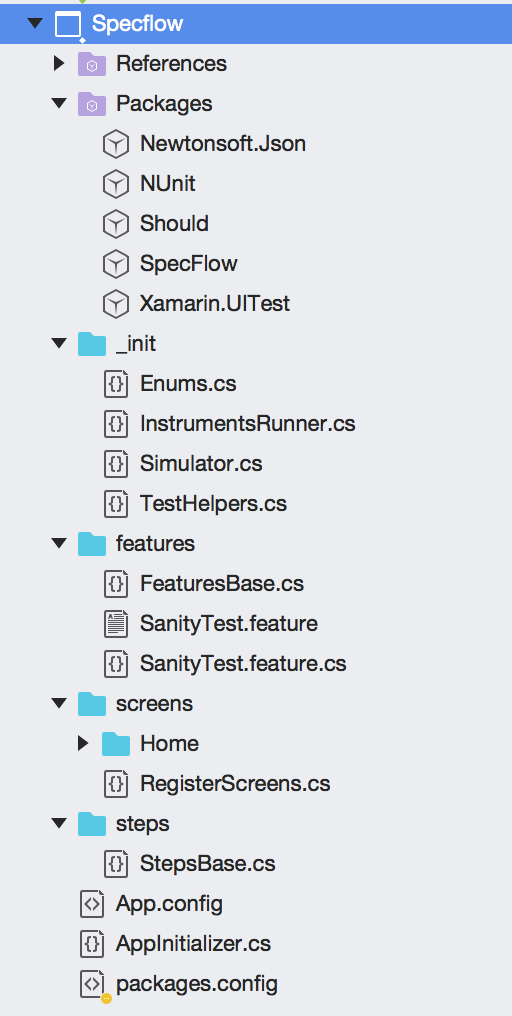


put the AppInitializer.cs in the Specflow project.

Copy Features directory from reference repo to Specflow.

Copy Screens directory from reference repo to Specflow.

Copy Steps directory from reference repo to Specflow.



# Refactoring the Shared Project to MVVM model

From the shared project, remove all folders (Model, View, ViewModel) and files and just keep the App.xaml and App.xaml.cs

Add a Pages and a ViewModels directory.

Add a Context Form page to Pages directory and call it MainPage

Add a Class to ViewModels directory and call it MainPageVM (it has to be a subclass of ViewModelBase)

Replace the contents of the App.xaml.cs to the following:



In the MainPage.xaml.cs, set the binding context to the View model object:

Put a simple component in MainPage.xaml

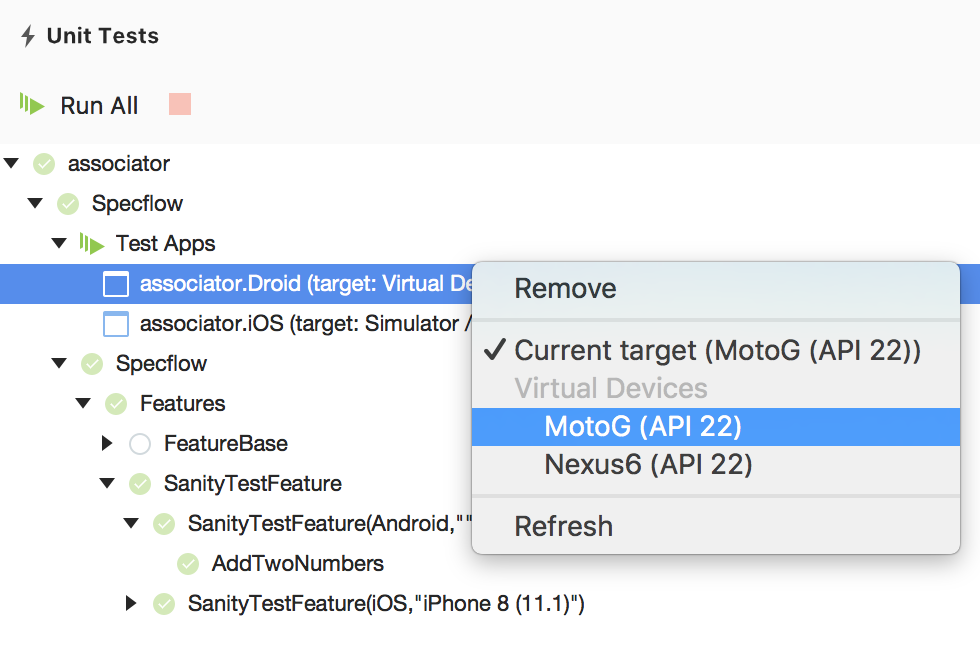


# Running The Sanity Tests

At this point, all configuration should be finished and Sanity tests should be able to run.

## Selecting Android Emulator to run

In the UnitTest pad, right click on the android test and select the simulator you want



The FeatureBase file then needs to have a TextFixture annotation for running Android apps:

## Selecting iOS Simulator to run

Selecting iOS Simulators are bit more complex. First, it depends on the Mac OSX machine and version. Assuming that XCode command line tools are installed (which they need to be), the list of available simulation devices can be obtained by executing the xcrun instruments command.

This command returns the name of the device followed by the GUID of the device and whether it is a device or a simulator. The output of this command should be used for selecting the device name, which is the second argument to the TextFixture annotation. For example, to run tests on iPhone X iOS 11.1 the command is executed as follows:



Thus, the simulator name will be “iPhone X (11.1)”. this can be put directly in the TextFixture annotation:



Alternatively, there is a Enum.cs file in the \_init directory that has constants defined for convenience. These were up to date as the time of this writing, but will certainly change in the future and readers should feel free to update the file for their own convenience.

For example, different iPhone 8 options are gathered in the same class:



and a fixture running on simulation for iPhone 8 plus and Apple watch 42 mm would look like the following:

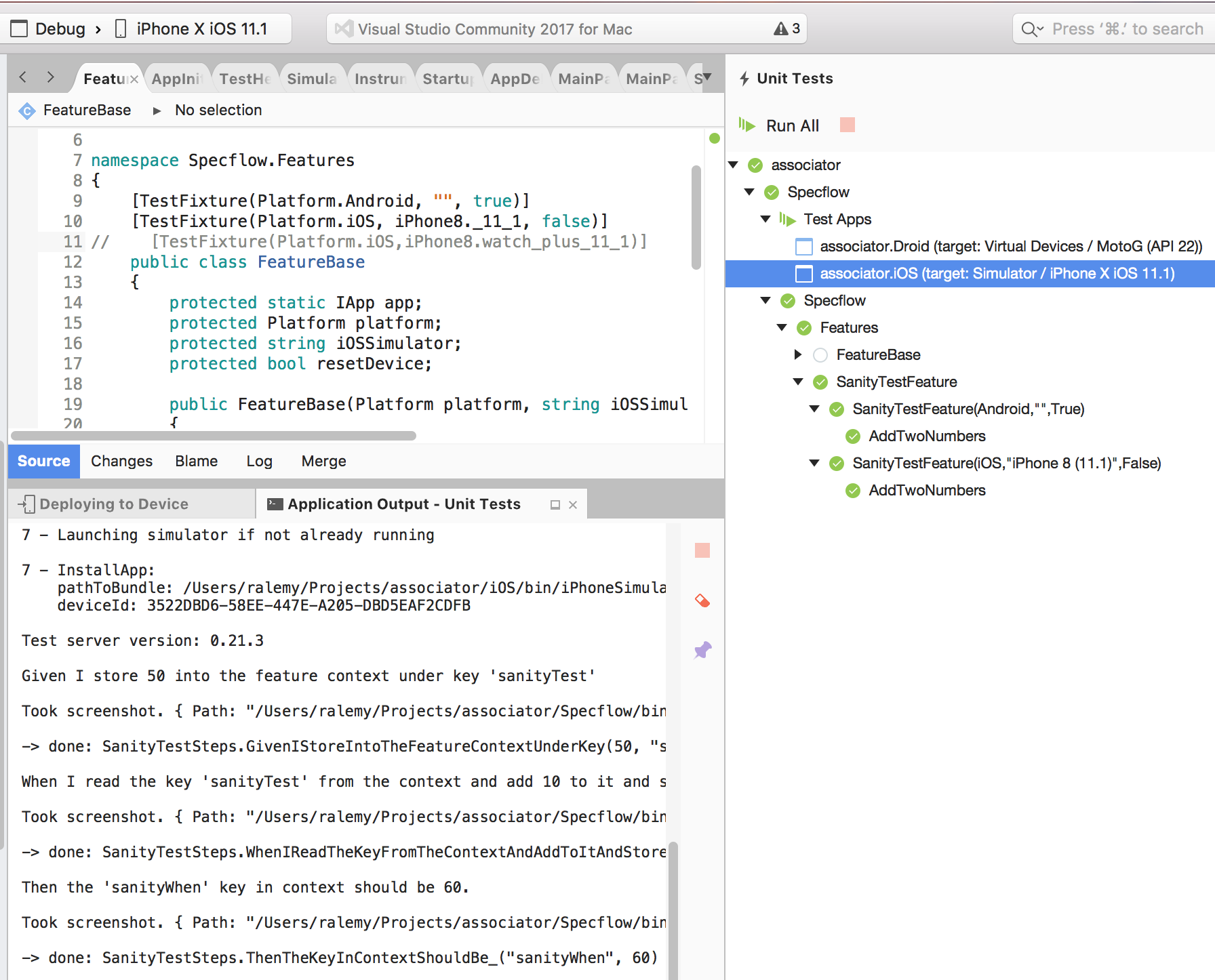


## Running tests on Xamarin Test Cloud

[To be added]

## Running tests locally

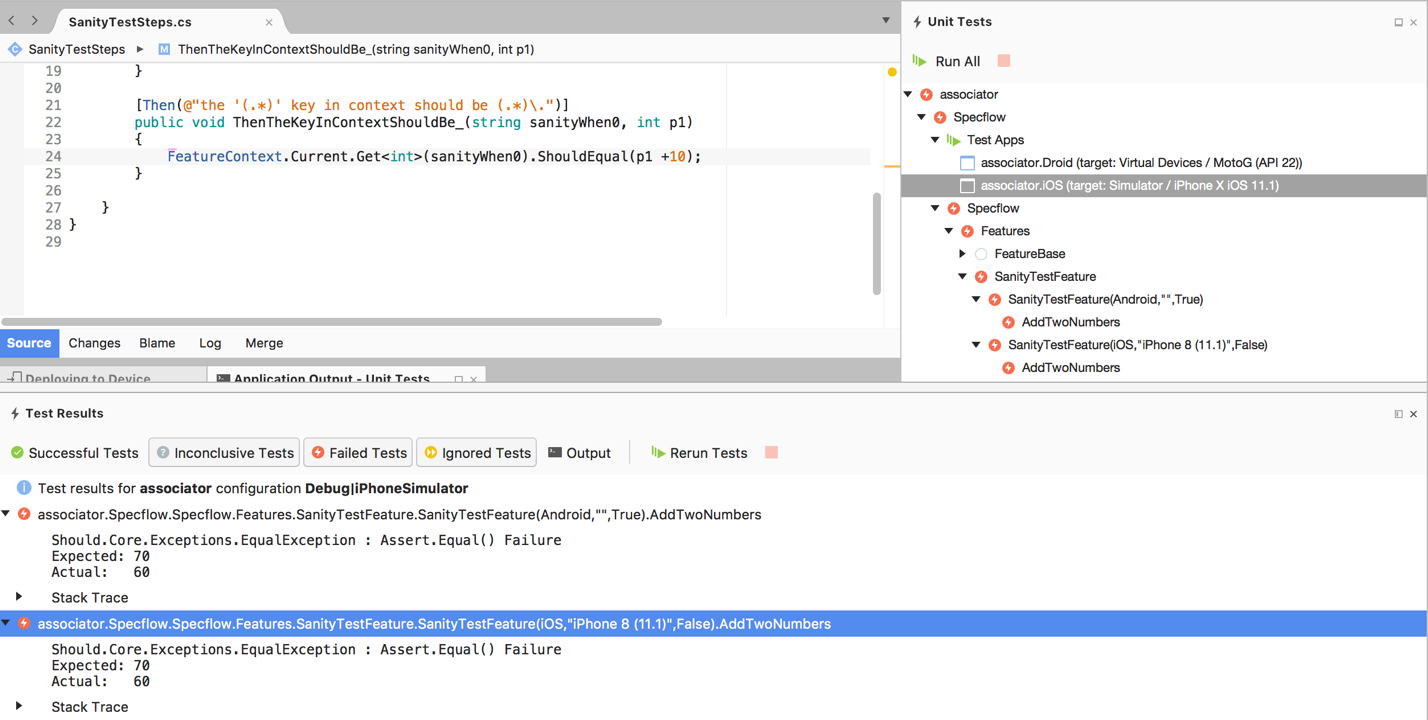
At this point, what is needed is to run the tests locally is to open the Unit Tests pad and click on Run All button:



to be sure, change the assertion in SanityTestSteps.cs file to look for something different than the expected value, for example, add 10 to it:



This should result in test failing

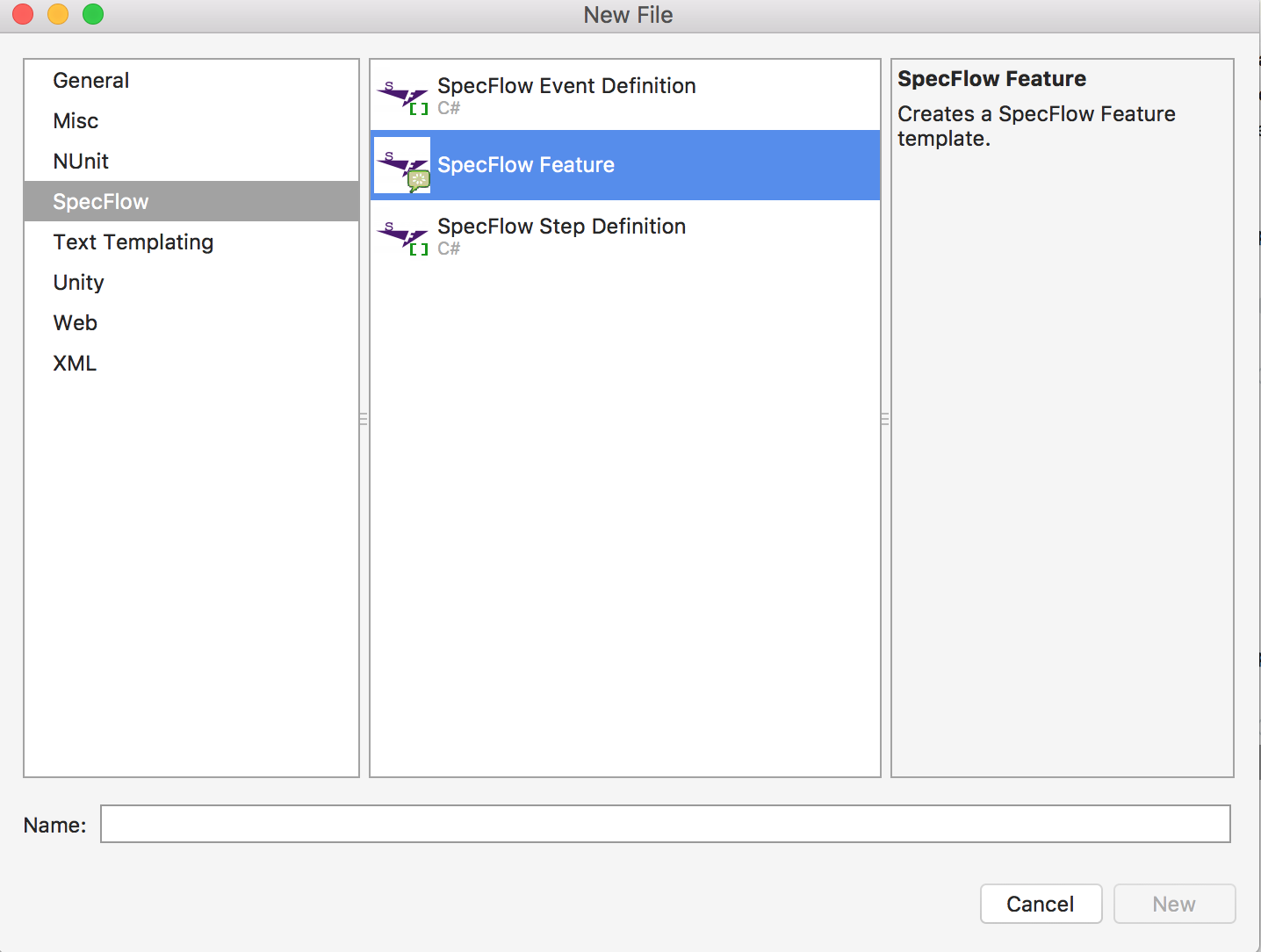


Section 2- General Techniques in Developing the Application

Application is developed following the TDD protocol, i.e. first a test is written against the required behavior, which will fail. Then enough code will be added until the test passes, then the code is refactored to comply with coding standards. The cycle is then repeated, with test followed by code until test passes followed by refactor, until all the requirements and features are implemented.

# Writing a failing test:

If you don’t already have a feature for this scenario, add a feature file to the features directory of the Specflow project.



Write the first scenario and assign a decorator to it.



Save and rebuild the Specflow project. This will generate the first part of partial class for feature.

Create the second part of partial class for the feature in the feature directory

* This second part is a subclass of FeaturesBase class.



In the FeaturesBase class, we can select the simulator(s) to run the test:



each [TextFixture()] line will result in one instance of FeatureBase being created. The first parameter is platform, the second parameter is name of iOS simulator (so, empty for Android) and the final one is a Boolean to decide whether to reset the simulator or not (recommended for Android, but too time consuming for iPhone).

Run the test

Copy and paste the skeleton code that is given in the error output

Write the test code

Run the test which will fail

git commit

Write code to pass the first test