How to create unit tests for Windows Phone with the Visual Studio Unit Testing Framework

This article shows how to use the *Visual Studio Unit Testing Framework* to test Windows Phone apps for Windows Phone 8 (and later).

Introduction

Unit testing divides apps into independent discrete units and validates them all against expected values for sample data. This sort of testing is particularly useful for detecting bugs that are introduced during maintenance ("regression tests") because they can be run automatically after each code change - and will break if any internal modification changes the behaviour at an interface. Unit tests are not typically effective for validating app user interfaces, so they are mainly used for testing app business logic.

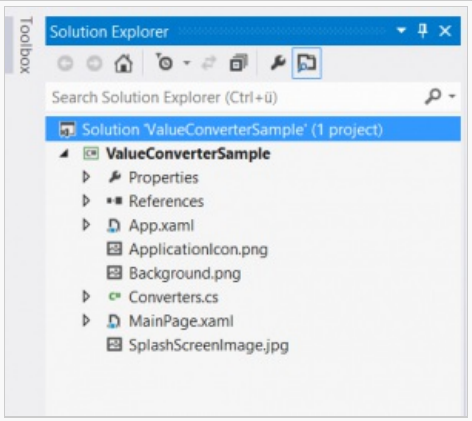
From *Visual Studio 2012 Update 2*, Windows Phone developers can use the *Visual Studio Unit Testing Framework* integrated in Visual Studio to test their Windows Phone 8 (and later) apps. When the tests are run, the Windows Phone test-app is generated and deployed automatically without further user interaction.

This article explains how to use *Visual Studio Unit Testing Framework* for testing Windows Phone 8 apps.

Note: Developers may instead use the Windows Phone Toolkit Test Framework for testing both Windows Phone 7 and 8 apps (see How to create unit test for Windows Phone applications for information). The *Visual Studio Unit Testing Framework* has some advantages - it does not require installation/referencing of additional tools and will be very familiar to desktop application developers migrating to Windows Phone.

Creating a unit test project

For this article we'll test the code example from the article Collection of Value converters for Windows Phone apps. As this is a Windows Phone 7 project and the test framework only works for Windows Phone 8, we first upgrade this project using the **Upgrade to Windows Phone 8** context menu entry on the project in *Solution View*.



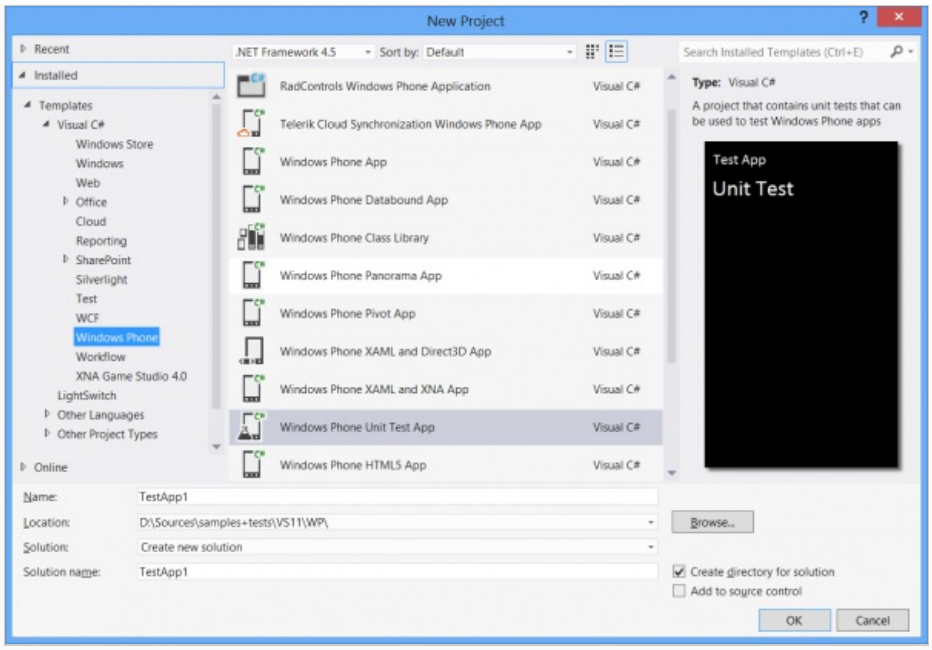
Value Converter Sample Solution - updating to Windows Phone 8

We then create a project for the Windows Phone unit tests using the 'Windows Phone Unit Test App' template.

Note: This is located under 'Windows Phone' and not under 'Test'

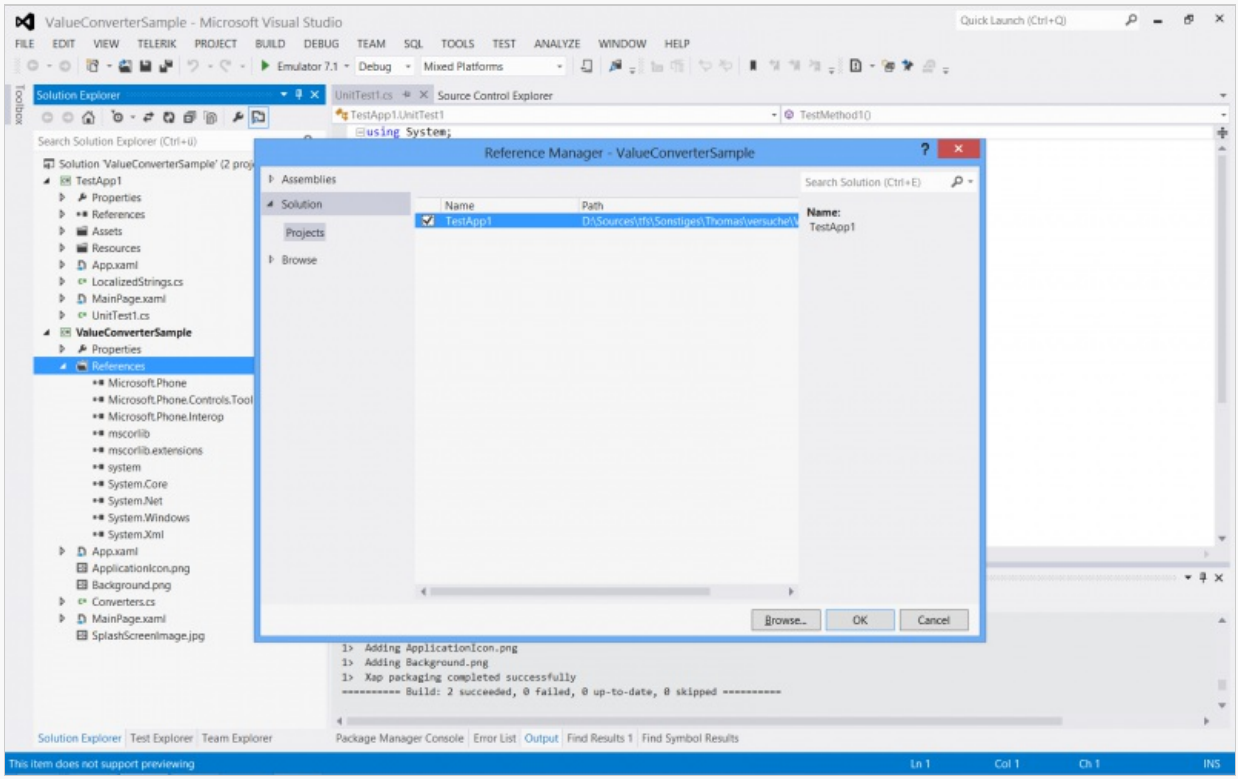
.

(C) Copyright Nokia 2014. All rights reserved.



New Project Dialog - create the test project

After the test project has been created, add a reference to the Windows Phone 8 app to be tested.



Adding the reference

At this point we have a test project with a reference to the project/app that we need to test. The test project contains a `UnitTest1` class that contains a `TestMethod1` function. The next step examines the test class.

Designing the test class

A unit test project can contain multiple test classes which themselves can contain multiple test functions. Each test function may have more than one test or more than one assertion. This is what the empty test class looks like:

```
using System;
using Microsoft.VisualStudio.TestPlatform.UnitTestFramework;

namespace TestApp1
{
```


(C) Copyright Nokia 2014. All rights reserved.

```
    [TestClass]
    public class UnitTest1
    {
       [TestMethod]
       public void TestMethod1()
       {
       }
    }
 }
```

We are going to test the `BooleanToVisibilityConverter`. This converter is expected to return `Visibility.Visible` if the input value is `true` and `Visibility.Collapsed` if the input value is `false`. We test both cases and the case of an invalid input for the `Convert()` and the `ConvertBack()` methods:

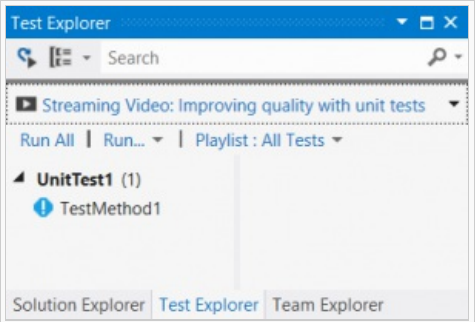
```
       [TestMethod]
       public void TestMethod1()
       {
       var conv = new BooleanToVisibilityConverter();
       Assert.AreEqual(Visibility.Visible, conv.Convert(true, typeof(Visibility), null,
null));
       Assert.AreEqual(Visibility.Collapsed, conv.Convert(false, typeof(Visibility),
null, null));
       Assert.AreEqual(Visibility.Collapsed, conv.Convert(null, typeof(Visibility), null,
null));

       Assert.IsTrue((bool)conv.ConvertBack(Visibility.Visible, typeof(bool), null,
null));
       Assert.IsFalse((bool)conv.ConvertBack(Visibility.Collapsed, typeof(bool), null,
null));
       Assert.IsFalse((bool)conv.ConvertBack(1234, typeof(bool), null, null));
       }
```

The Assert class has a number of member functions that can be used to check the outcome of a test. Most used are `Assert.AreEqual` and `Assert.IsTrue`.

After creating the test function we build the project and the test is recognized by *Test Explorer*.

The Test Explorer window

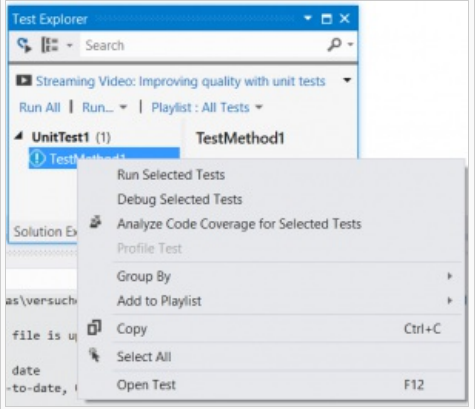
The *Test Explorer* window is the central dashboard for the tests. Here all the tests are listed and can be grouped by different criteria like name or the class they belong to or whether they are run and succeeded or failed. The tests can be started and the results are shown here.



Test Explorer

The context menu shows the available commands.

(C) Copyright Nokia 2014. All rights reserved.



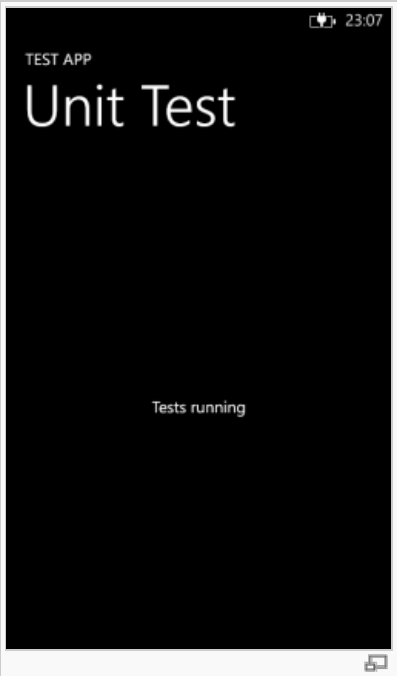
Test Explorer Context menu

We can now run our test by selecting **Run all** from the top row or **Run Selected Tests** from the context menu.

Tip: Be sure to configure the project configurations correctly for phone or emulator usage. Explicitly use AMD for the device or x86 for the emulator. Mixed or wrong configurations won't work.

Running the tests

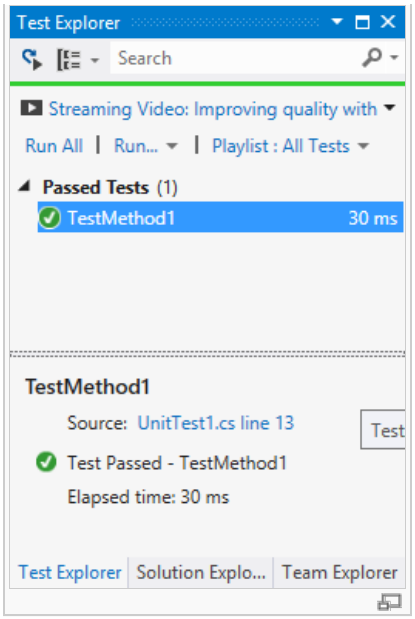
Running our test function deploys the test project to the phone or the emulator and runs it. The testing app has a dummy UI and looks as follows:



You don't have to do anything with the test app, it ends automatically when the test completes. The test results are then displayed in *Test Explorer*.

Tip: Don't forget to unlock you device screen before running the test. There won't be an error message.

(C) Copyright Nokia 2014. All rights reserved.



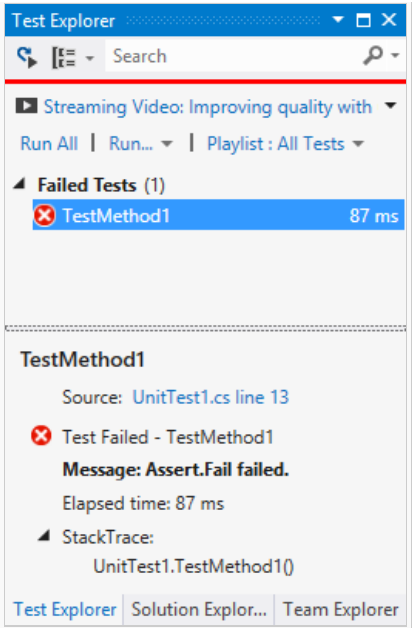
If the test succeeded, only the run-time is indicated. If there is an error the reason/exception text is given.

Let's create an assertion that will fail and review the result:

```
public void TestMethod1()
      {
      var conv = new BooleanToVisibilityConverter();
      Assert.AreEqual(Visibility.Visible, conv.Convert(true, typeof(Visibility), null,
null));
      Assert.AreEqual(Visibility.Collapsed, conv.Convert(false, typeof(Visibility),
null, null));
      Assert.AreEqual(Visibility.Collapsed, conv.Convert(null, typeof(Visibility), null,
null));

      Assert.IsTrue((bool)conv.ConvertBack(Visibility.Visible, typeof(bool), null,
null));
      Assert.IsFalse((bool)conv.ConvertBack(Visibility.Collapsed, typeof(bool), null,
null));
      Assert.IsFalse((bool)conv.ConvertBack(1234, typeof(bool), null, null));

      Assert.Fail();
      }
```

Compile and run and then *Test Explorer* displays:

(C) Copyright Nokia 2014. All rights reserved.



Test explorer with "failed" test

If the given error message and stack trace doesn't help enough, you can even debug the test run! Set a break point in the test routine and select **Debug Selected Tests** in the context menu.

Code coverage

Warning: At time of writing VS code coverage does not work on Windows Phone - you get the message: **Code coverage is not available for Windows Store or Windows Phone apps**. Help vote this issue up on UserVoice!

Code coverage tools can be used to find out which parts of your code are visited by the unit tests, in order to gain a quantitative assessment of how much of the code is validated. As a rule of thumb 80% code coverage is reasonable - higher is better, though in some cases even 80% may be difficult to achieve.

Code coverage works by inserting special "instrumentation calls" into the code during compilation. The tests are compiled, executed and each code line that's reached is marked. You can highlight the covered code lines by pressing a button in the code coverage results window.

Summary

The Unit Testing Framework for Windows Phone 8 is a helpful extension to Visual Studio that is well integrated and easy to use. Currently it omits the code coverage functions from the desktop versions, but otherwise it is an essential tool and fully functional tool for Windows Phone developers.

(C) Copyright Nokia 2014. All rights reserved.