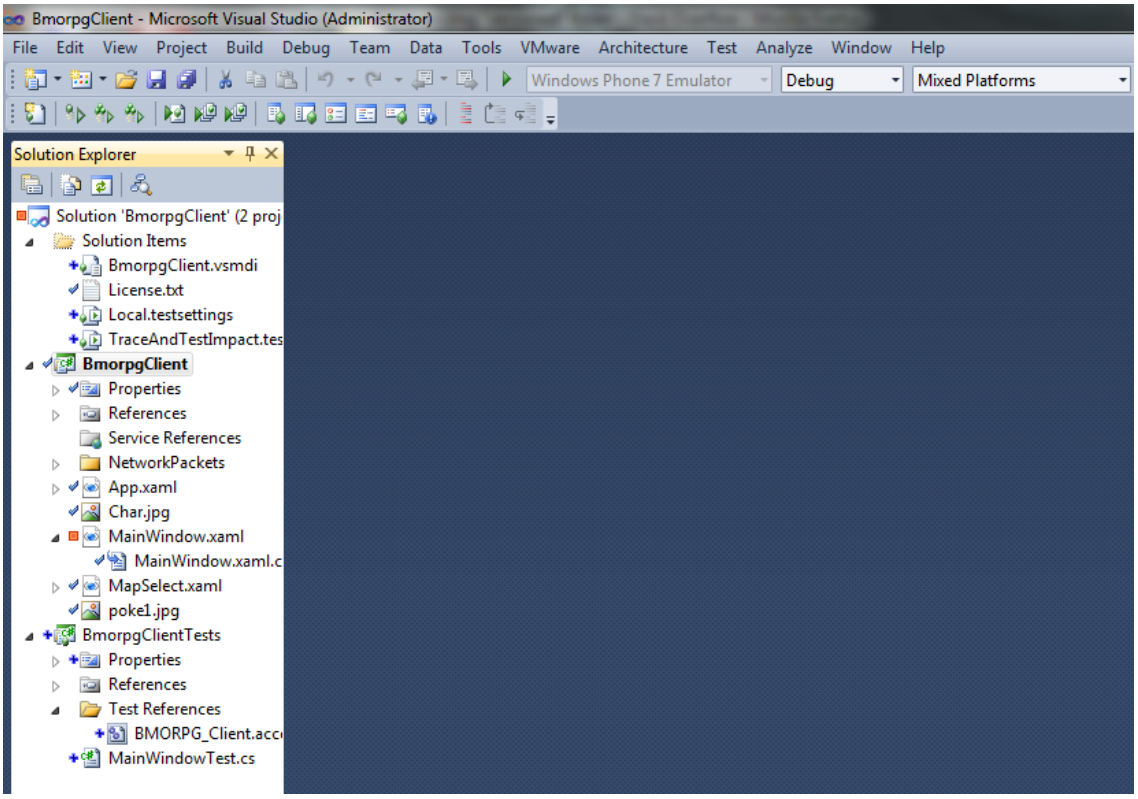
How to Run Automated Tests in VS2010

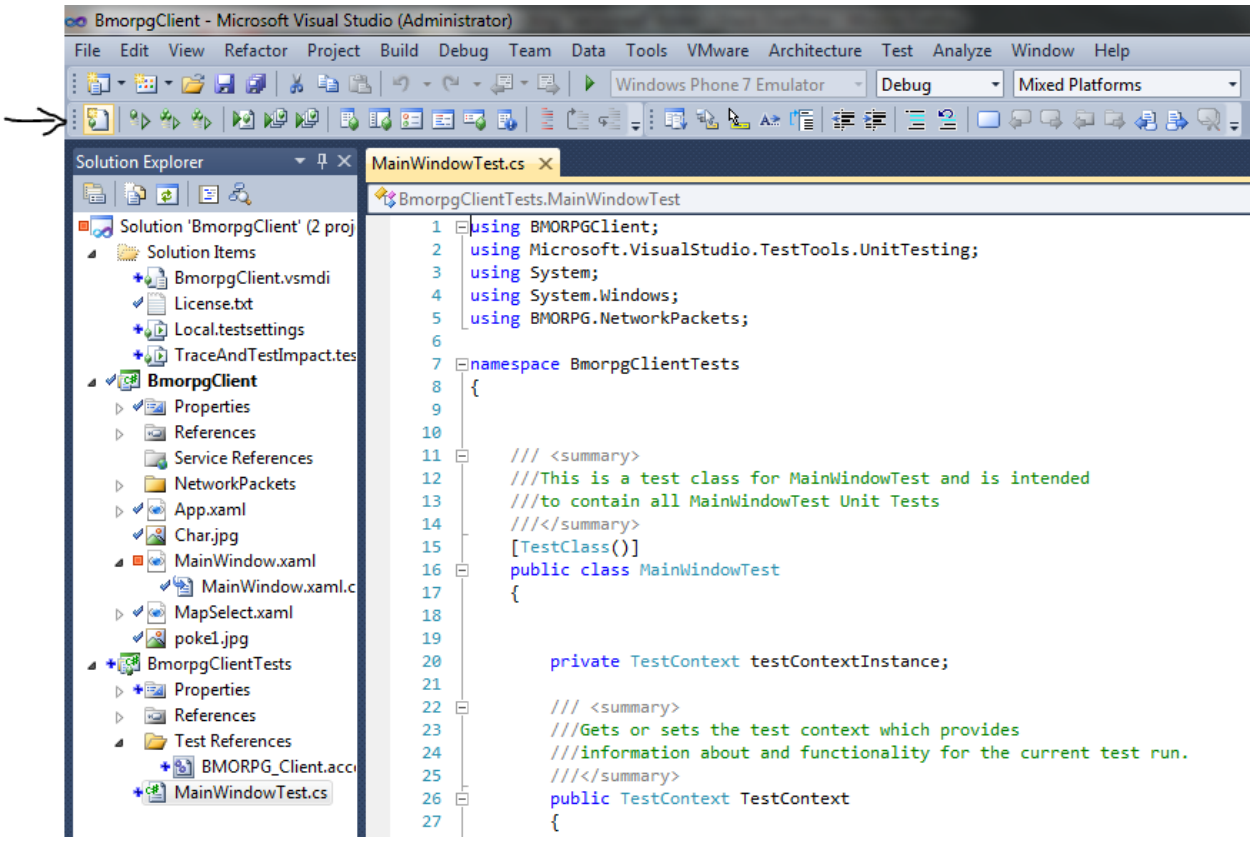
By: Josue Martinez

This tutorial is designed to help write, build, and run unit tests in VS2010.

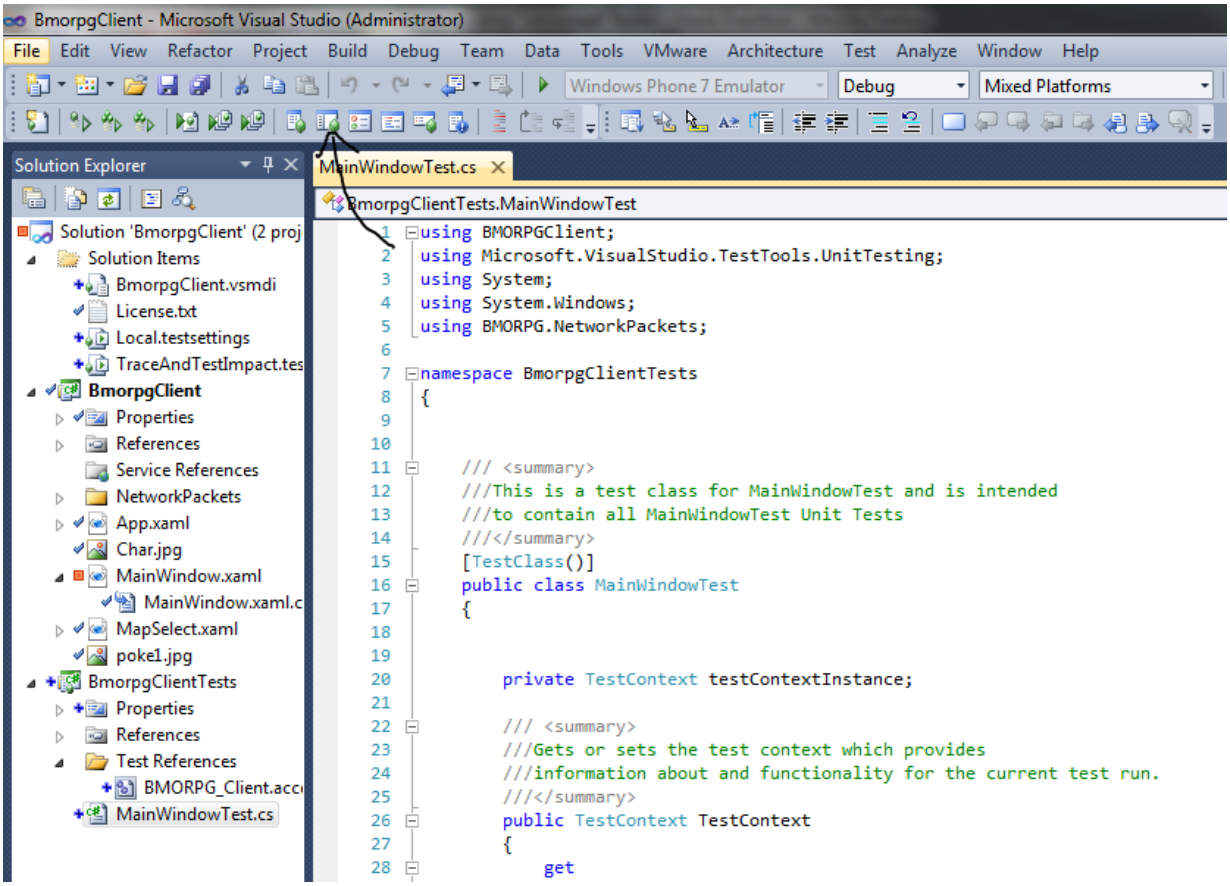
You should start out seeing this once you've loaded the solution into VS2010. (Ignore the checkmarks or red squares, those are AnkhSVN tags)



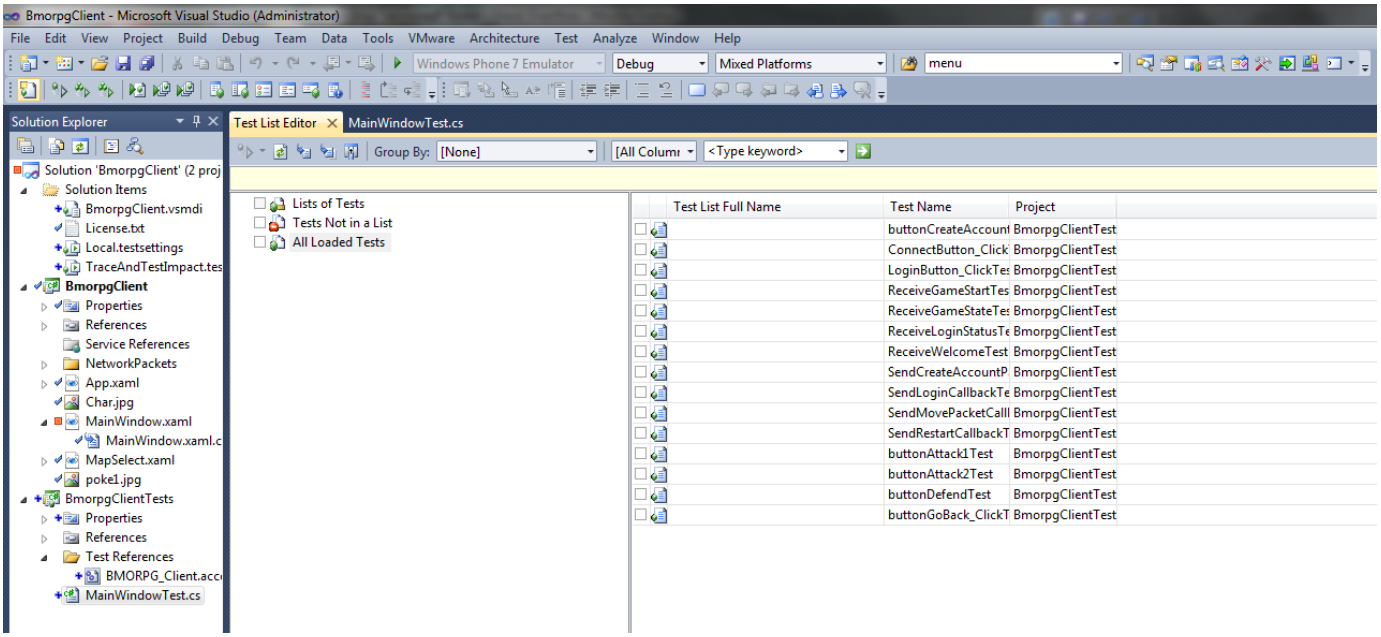
Double-click on MainWindowTest.cs, and you should see both the code window, as well as a new toolbar show up. This is the testing toolbar. It will serve as the primary access point to launch tests, as well as modify the tests to be run.



Click the Test List Editor button, shown in the screenshot below.



This will open the Test Editor window, shown below.

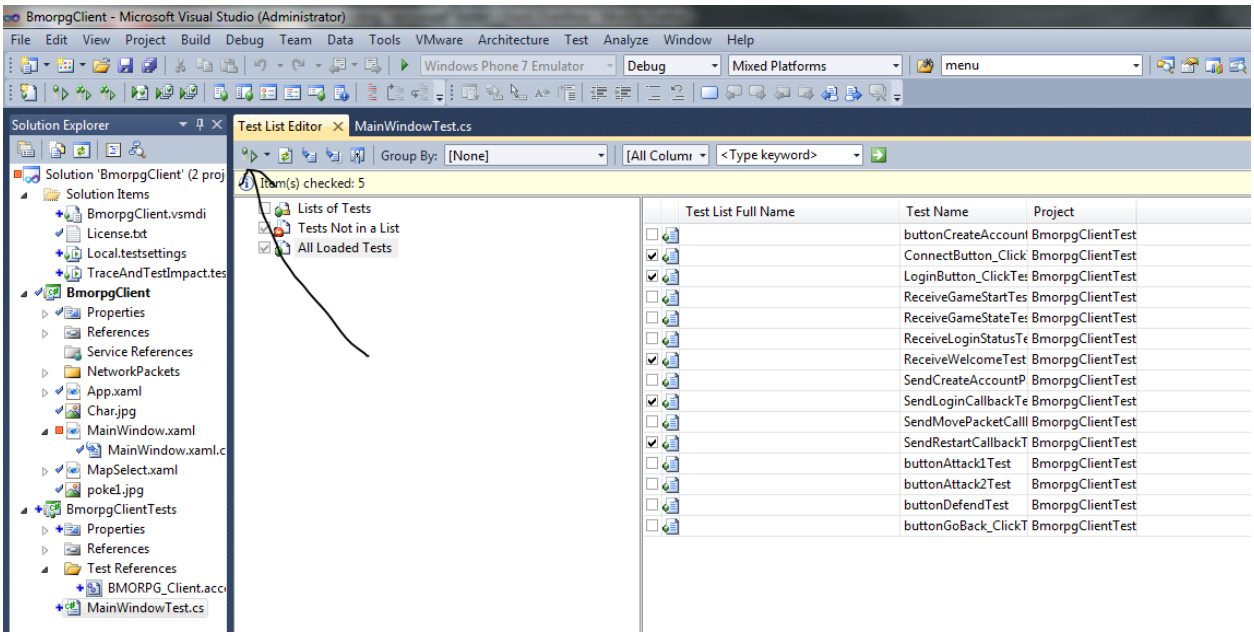


Currently, the only tests that are included are unit tests for the different functions. I only created stubbed tests for a handful of them, selecting the important ones first. The check marks next to each test signify whether or not they will be run on the next execution cycle. All tests are run independently, so as to minimize debugging effort. However, running the test will call the function as it is written. Thus, if there are message box popups that appear in a particular function call, they will appear during the execution of the test. There are ways of automating the clicks, or you can just sit there and wait for the popup to click it.

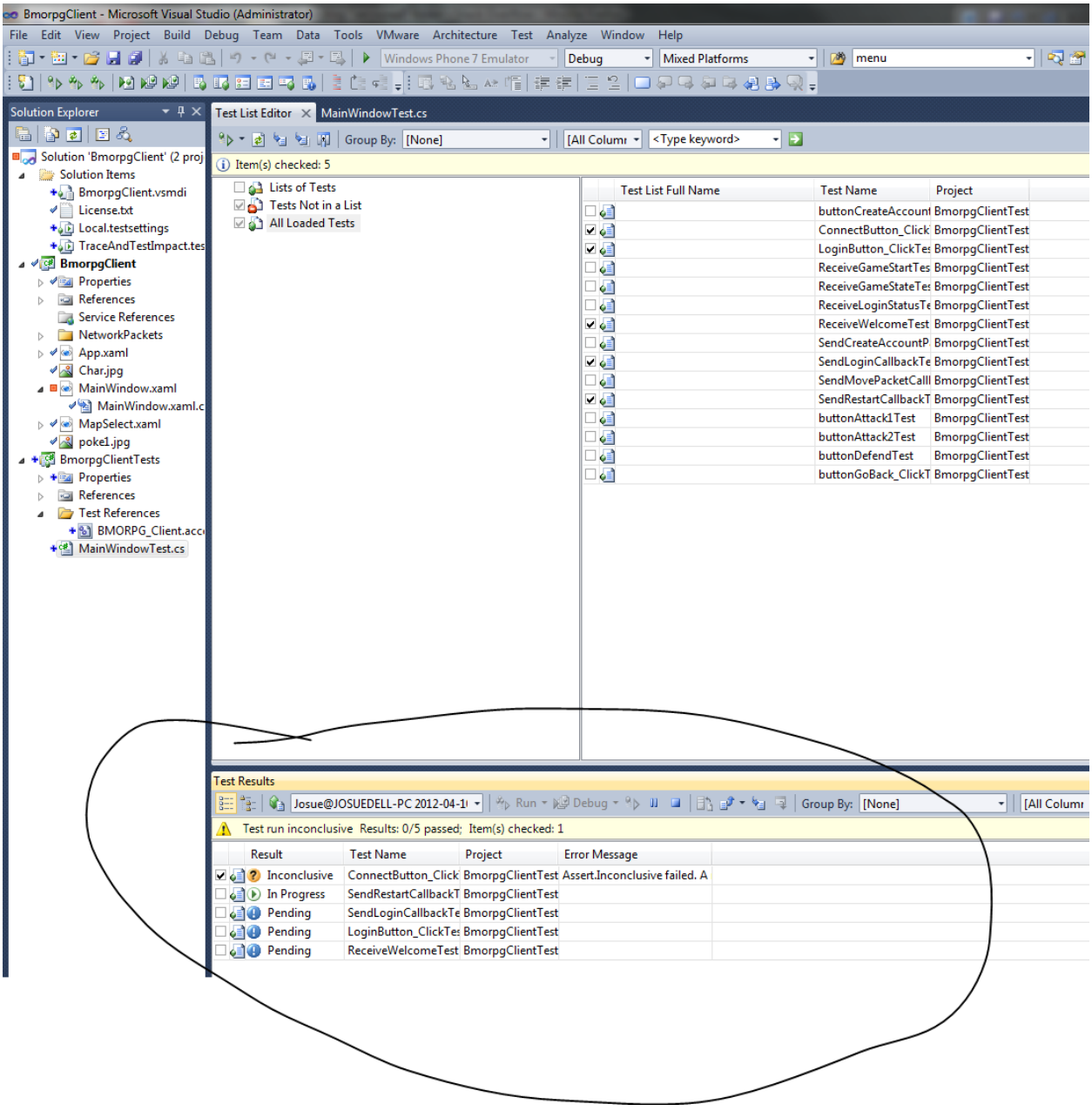
The TestInitialize function, located within the Additional Test Attributes region, can be used to set things up for any test. For example, if testing network connectivity, the TestInitialize function can initiate the connection to the server, thus enabling the test to check the specific functionality, rather than something that has already been tested.

```
53
54          //Use TestInitialize to run code before running each test
55          [TestInitialize()]
56          public void MyTestInitialize()
57          {
58              //Connect to server, or something like that
59          }
60
```

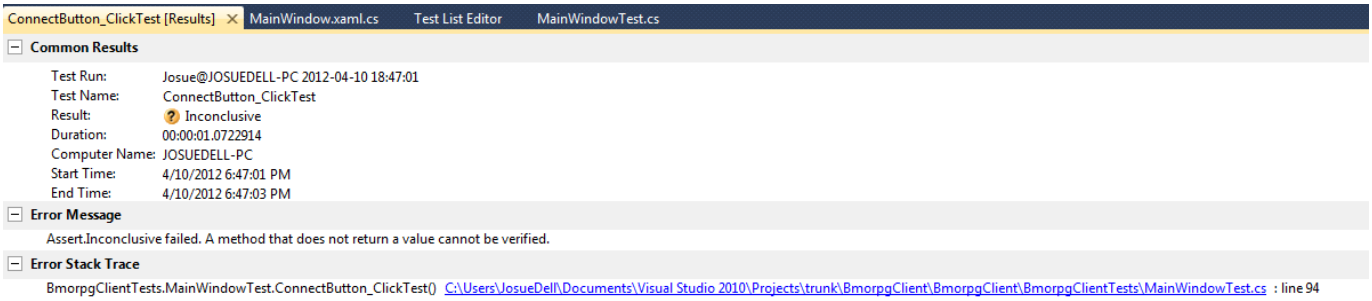
Now we will see how to debug test information. After selecting some tests, click the Run Tests button.



The result should look like this:

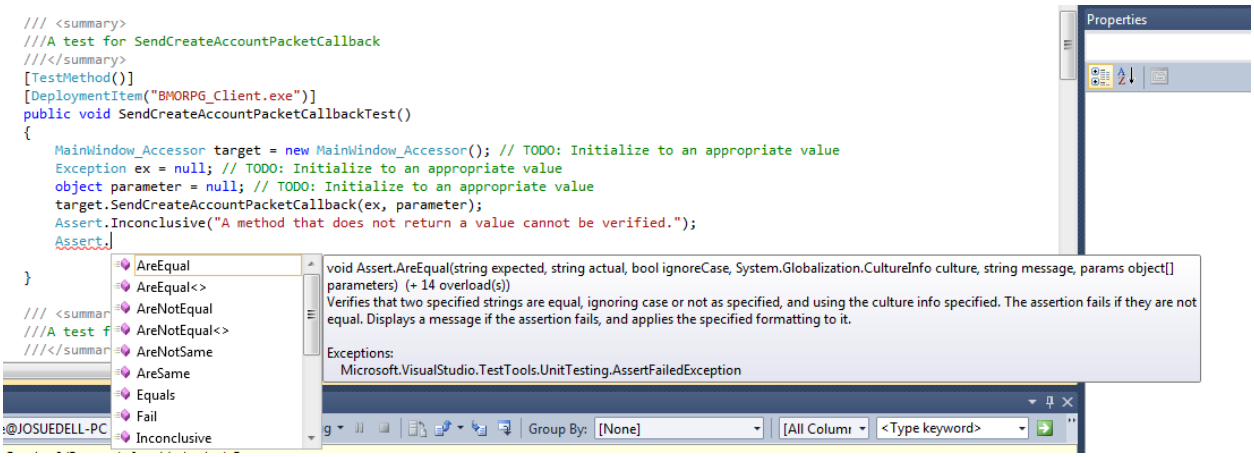


The test results can be double-clicked, leading to the following information window:



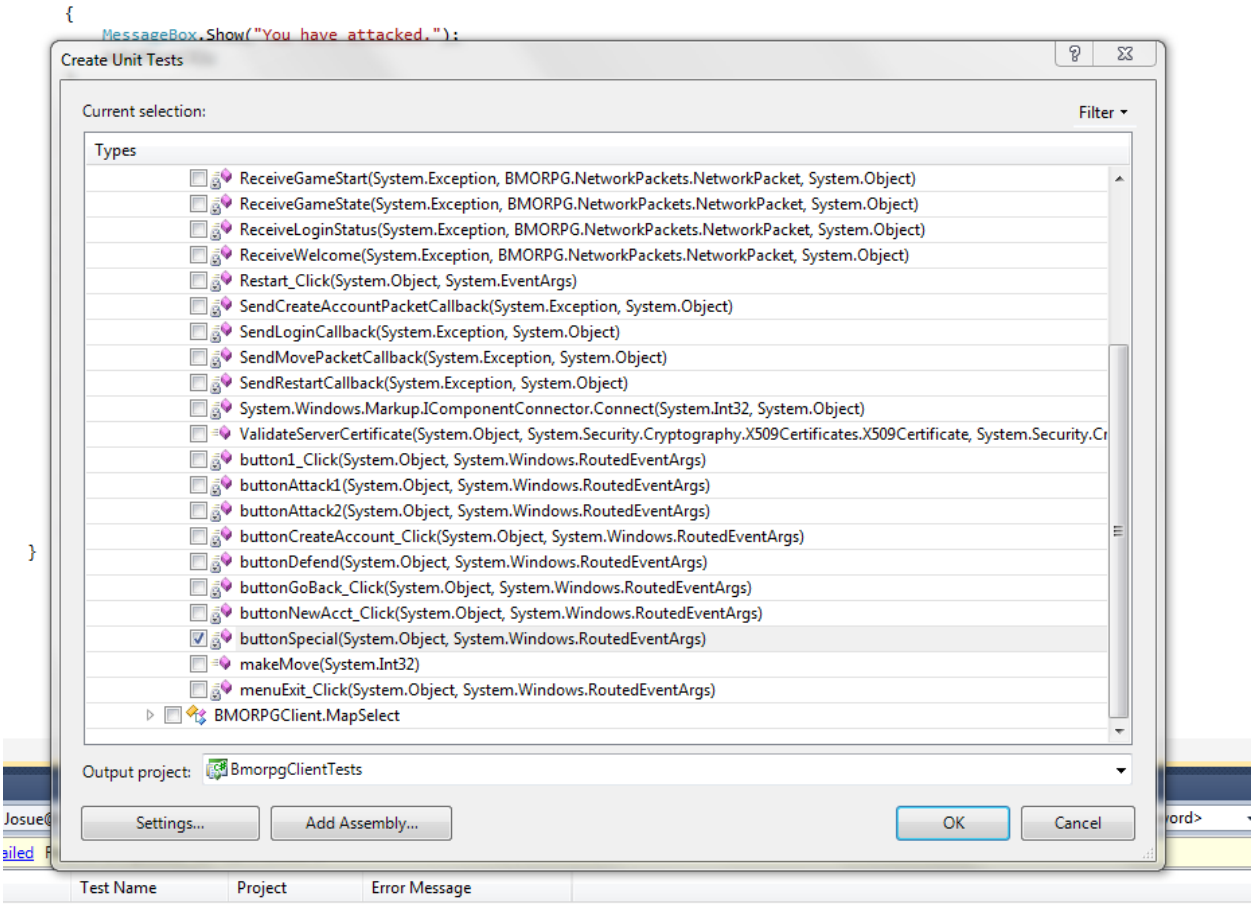
This window shows test information, the actual error code, as well as a stack trace that can be further used for debugging.

Currently, the tests are stubbed, meaning they will always fail. The holy grail of testing in C# is the Assert statement. Assert comes in a variety of flavors, such as:



In the case of this assertion, you place the condition you want to test as the first parameters, then a string message as the last parameter. Thus, when the tests execute, if an assertion fails, the test fails, and the message from the assert statement is displayed in the test result box, making debugging a particular statement even easier. The tests exist independently of the client, and are therefore suited to parallel development. For stubbed tests, the Assert.Inconclusive is used, however, Assert.Fail can also be used to force a test failure if a particular condition is met.

Adding unit tests are as easy as a right click. In the .cs file where the client/server code is located, right click on a function name and click “Create Unit Tests…”. If there is already a test project file created, it will automatically select that project as the output. A window will open listing all functions within the .cs file, and you may place checkmarks next to the functions you want unit tests created for. The screenshot below demonstrates this:



The test will then appear in the test .cs file, and can be modified as needed to perform the test. UI access is done through the MainWindow_Accessor object, which allows essentially unlimited access to all UI elements.

As of the writing of this tutorial, test projects will have been created and uploaded to the SVN, meaning the setup should be complete. Thus, all effort can be focused on writing the actual tests, and checking their completion. Additional tests can be added for each function as needed in order to test different scenarios (server not connected, server connected but wrong version, bad data received, etc). Also be aware that tests can be run virtually on VMWare if it is installed, or in the case of a client/server model, the server can be run in debug mode while the tests are executing, allowing the developer to step through the server code while a test is being executed. Tests can also be run simultaneously between a client and a server, allowing failures to be automatically detected on both ends.