rstropek / **Samples**

Branch: master Samples / CSharpUnitTestingWorkshop / **readme.md** Find file Copy path

rstropek Fixes in readme 9ea5300 on 18 Nov 2015

1 contributor

455 lines (366 sloc) 15.1 KB

CSharp Unit Testing in Visual Studio

Introduction

I created this sample for the Microsoft Technical Summit 2015. My session was about CSharp unit testing. Here is the abstract (German):

> Jeder weiß, dass man es tun sollte, oft ist es aber das erste, das unter Termindruck gestrichen wird: Unit Testing. Visual Studio macht automatisierte Unit Tests so einfach, dass es eigentlich keine Ausrede mehr gibt, warum man keine solchen Tests in seinem C#-Projekt hat. In dieser Livecoding-Session zeigt MVP und MS Regional Director Rainer Stropek, was Visual Studio in Sachen Unit Tests alles kann. Es wird nicht nur an der Oberfläche gekratzt. Rainer zeigt auch, wie man scheinbar schwierig zu testenden Code mit Microsoft Fakes in Unit Tests packen kann. Neben Tool-Wissen wird Rainer Tipps geben, wie man C# Code richtig aufbaut, damit automatisierte Tests effizienter umgesetzt werden können. 60 Minuten Code und keine Slides - eine Session für echte C#-Entwicklerinnen und -Entwickler.

Part 1: The Basics

Don't worry, we will not cover very basics like creating a unit test project, handling Visual Studio's Test Explorer, etc. You can easily find information about that on MSDN. However, when doing C# trainings, I regularly find that people don't know some quite helpful details about Visual Studio's unit test library.

> In UnitTestBasics.cs you will find samples for many important things to know about Visual Studio's unit testing framework.

Let me point you to some things that might be new.

Inconclusive

Did you know the difference between `Assert.Fail` and `Assert.Inconclusive` ?

```
// Manual asserts with fail and inconclusive
if (DateTime.Today.DayOfWeek == DayOfWeek.Sunday)
{
    Assert.Fail("This test will fail on Sunday.");
} else if (DateTime.Today.DayOfWeek == DayOfWeek.Saturday)
{
    // Note that inconclusive does not lead to a failing test.
    // The test will be marked as "skipped".
    Assert.Inconclusive("Result of test cannot be determined on Saturday.");
}
```

CollectionAssert

Did you know there is a dedicated class for asserts related to collections?

```
private class Person { }
private class VipPerson : Person { }

[TestMethod]
public void CollectionAsserts()
{
    var somePersons = new Person[] { new Person(), new VipPerson() };

    // Some trivial collection asserts
    CollectionAssert.AllItemsAreNotNull(somePersons);
    CollectionAssert.Contains(somePersons, somePersons[0]);

    // Some more interesting checks
    CollectionAssert.AllItemsAreInstancesOfType(somePersons, typeof(Person));

    CollectionAssert.AllItemsAreUnique(somePersons);
    // CollectionAssert.AllItemsAreUnique(new [] { "A", "A" }); --> this would fail.

    var someStrings = new[] { "AB", "CD" };
    var someOtherStrings = new[] { $"{"A"}B", "CD" };
    var evenMoreStrings = new[] { "CD", $"{"A"}B" };
    CollectionAssert.AreEqual(someStrings, someOtherStrings);
    // CollectionAssert.AreEqual(someStrings, evenMoreStrings); -> this would fail

    // Note that AreEquivalent ignores order, so this works:
    CollectionAssert.AreEquivalent(someStrings, evenMoreStrings);

    CollectionAssert.IsSubsetOf(new[] { "AB" }, someStrings);
}
```

StringAssert

Did you know there is a dedicated class for asserts related to strings?

```
[TestMethod]
public void StringAsserts()
{
    // Trivial string asserts
    StringAssert.Contains("ABC", "B");
    StringAssert.EndsWith("ABC", "C");
    StringAssert.StartsWith("ABC", "A");

    // Pattern matching with Regex
    StringAssert.Matches("-10.0", new Regex(@"[+-]?\d+\.\d*$"));
}
```

Testing private members

Did you know that there are two dedicated classes for testing private members?

```
[TestMethod]
public void TestPrivates()
{
    // Note how we test private instance method with PrivateObject
    // https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.privateobject.aspx
    var po = new PrivateObject(new SomeBusinessLogicClass());
    Assert.AreEqual(3, po.Invoke("SomeInternalLogic", 1, 2));

    // Note how we test private static methods with PrivateType
    // https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.privatetype.aspx
    var pt = new PrivateType(typeof(SomeBusinessLogicClass));
    Assert.AreEqual(42, pt.InvokeStatic("SomeInternalStaticLogic"));

    // For PrivateObject and PrivateType, be VERY CAREFUL with parameters.
    // If you use the wrong type, you will get strange erros as methods
    //   cannot be found.
}
```

Data-Driven Tests

Ever built a data-driven test?

```
// Note that the framework will automatically fill this property
public TestContext TestContext { get; set; }

[DataSource("System.Data.SqlServerCe.4.0", "Data Source=TestData.sdf;", "TestValues", DataAccessMethod.Sequential)]
[DeploymentItem("TestData.sdf")]
```

```
[TestMethod]
public void DataDrivenTest()
{
    // Don't forget to set a breakpoint in this method and
    //   inspect this.TestContext and watch how the test method
    //   is called multiple times depending on the content
    //   of the data source.

    var objectToTest = new SomeBusinessLogicClass();

    var number = Convert.ToInt32(this.TestContext.DataRow["Value"]);
    Assert.AreEqual(number * number, objectToTest.Square(number));
}
```

Test Attribute

There are many handy attributes you can use for your tests.

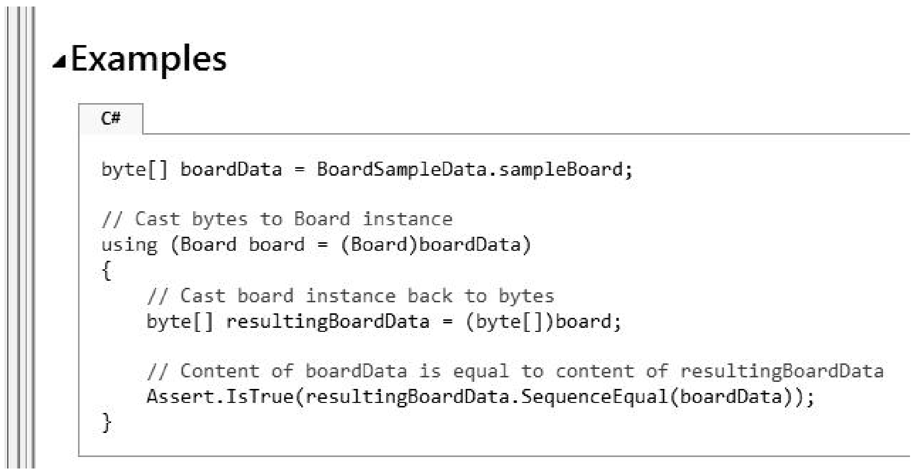
```
// Note how you can add descriptive information using attributes
[Description("This is a test that demonstrates various attributes.")]
[Owner("Rainer")]
[Priority(1)]
[TestCategory("Demotests")]
[TestMethod]
[TestProperty("DB", "SQL Server")]
public void TestWithManyAttributes()
{
}

// Note that this test expects a specific exception
[ExpectedException(typeof(DivideByZeroException))]
[TestMethod]
public void TestWithException()
{
    var x = 0;
    var y = 5;
    Trace.WriteLine(y / x);
}

// Note that you can mark a test as ignorable.
[Ignore]
[TestMethod]
public void FailingTest()
{
    Assert.Fail();
}

// Note that you can specify timeouts to check perf limits.
[Timeout(150)]
[TestMethod]
public void LongRunningTest()
{
    Thread.Sleep(100);
}

// You can also create your own test attributes.
// This is out of scope for this example.
// See e.g. http://blogs.msdn.com/b/vstsqualitytools/archive/2009/09/04/extending-the-visual-studio-unit-test-type-pa
```

Unit Tests in Docs

Did you know that you can reference unit tests in the `example` section of your C# XML Code Comments?

```
/// <summary>
/// Reads some data from the database.
/// </summary>
/// <returns>Result from the database.</returns>
/// <example>
/// <code source="../UnitTests/UnitTestBasics.cs" region="Async test" language="C#" />
/// </example>
public static async Task<int> ReadDataAsync() { ... }
```

Note how we use the unit test as a code example for our API. See also Sudoku sample for larger example including *Sandcastle Help File Builder*.



```
byte[] boardData = BoardSampleData.sampleBoard;

// Cast bytes to Board instance
using (Board board = (Board)boardData)
{
    // Cast board instance back to bytes
    byte[] resultingBoardData = (byte[])board;

    // Content of boardData is equal to content of resultingBoardData
    Assert.IsTrue(resultingBoardData.SequenceEqual(boardData));
}
```

Part 2: Async Unit Tests

Today, many APIs are async. Visual Studio unit testing can handle async test methods including `async` / `await` .

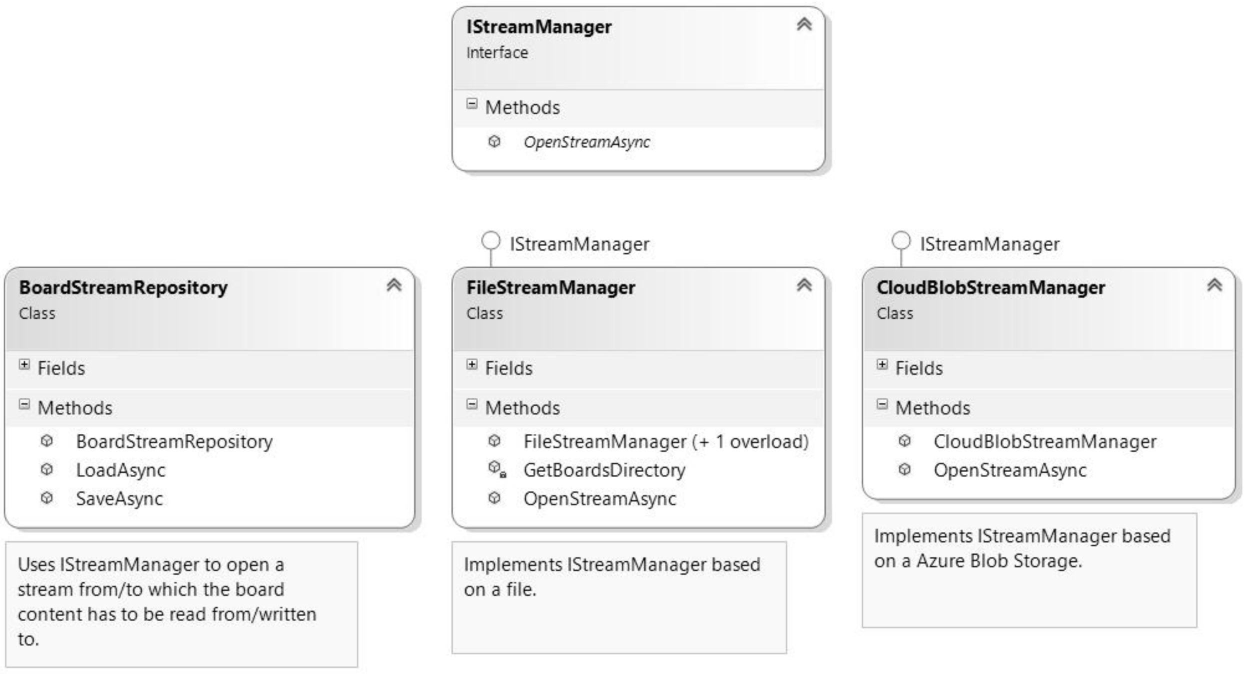
```
// Note that the following test is WRONG!
[Ignore]
[TestMethod]
public void TestAsyncDBAccess()
{
    // Note that this test will FAIL as ReadDataAsync
    // is an async method.
    Assert.AreEqual(42, SomeDatabaseAccess.ReadDataAsync());
}

// Async tests MUST return Task.
// Note that you can use async await in async tests.
[TestMethod]
public async Task SimpleAsyncTest()
{
    Assert.AreEqual(42, await SomeDatabaseAccess.ReadDataAsync());
}

// Note that you can still use ExpectedException
// when writing async tests.
[ExpectedException(typeof(Exception))]
[TestMethod]
public async Task AsyncExceptionTest()
{
    await Task.Delay(100);
    await SomeDatabaseAccess.FailingDataAccessAsync();
}
```

Part 3: Microsoft Fakes

Microsoft Fakes is a great tool for unit testing. It can save you tons of code you would have to write without it. I will describe Microsoft Fakes based on my Sudoku Sample.

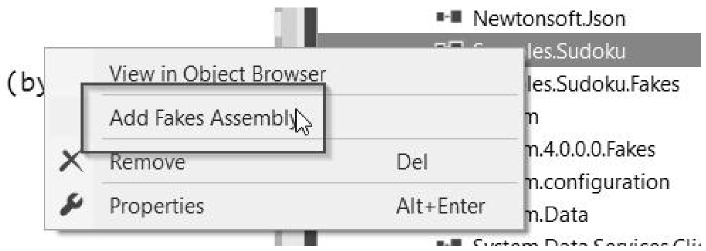
Shims

Shims are helpful if you need a (partly) implementation of an interface or an abstract base class. Take a look at the following example. We have a class `BoardStreamRepository` . It can load an save content of a Sudoku board from/to a stream. Our class library defined an interface `IStreamManager` that `BoardStreamRepository` uses to get the `Stream` it should use. The class library contains two implementation. One for files (`FileStreamManager`) and one for Azure Blob Storage (`CloudBlobStreamManager`).



```
/// <summary>
/// Contains methods to read/write <see cref="Board"/> instances from/to a <see cref="System.IO.Stream"/>.
/// </summary>
public class BoardStreamRepository
{
	private IStreamManager streamManager;

	/// <summary>
	/// Initializes a new instance of the <see cref="BoardStreamRepository"/> class.
	/// </summary>
	/// <param name="streamManager">Underlying stream manager.</param>
	public BoardStreamRepository(IStreamManager streamManager)
	{
		...
		this.streamManager = streamManager;
	}

	/// <summary>
	/// Saves the board using the specified name
	/// </summary>
	/// <param name="boardName">Name of the board.</param>
	/// <param name="board">The board to save.</param>
	/// <returns>A task that represents the asynchronous operation.</returns>
	public async Task SaveAsync(string boardName, Board board)
	{
		...
		// Open underlying stream for writing
		using (var stream = await this.streamManager.OpenStreamAsync(boardName, AccessMode.Write))
		{
			...
		}
	}
	...
}
```

In order to test `BoardStreamRepository` with a mockup version of `IStreamManager`, we add a fake assembly:



```
<Fakes xmlns="http://schemas.microsoft.com/fakes/2011/">
  <Assembly Name="Samples.Sudoku"/>
  <StubGeneration>
    <Clear/>
    <Add FullName="Samples.Sudoku.IStreamManager!"/>
  </StubGeneration>
  <ShimGeneration>
    <Clear/>
  </ShimGeneration>
</Fakes>
```

Based on that, it is simple to create a mock object:

```
/// <summary>
/// Tests for <see cref="Samples.Sudoku.BoardStreamRepository"/>
/// </summary>
[TestClass]
public class BoardStreamRepositoryTest
{
	[TestMethod]
	[TestCategory("With fakes")]
	public async Task TestLoadBoard()
	{
		// A BoardStreamRepository needs an IStreamManager. Note that we use a
		// stub generated by Microsoft Fakes here.

		// Prepare
		var repository = BoardStreamRepositoryTest.SetupBoardStreamRepository(BoardSampleData.sampleBoard);

		// Execute
		var board = await repository.LoadAsync("DummyBoardName");

		// Assert
	CollectionAssert.AreEqual(BoardSampleData.sampleBoard, (byte[])board);
	}
	...
	private static BoardStreamRepository SetupBoardStreamRepository(byte[] buffer)
	{
		var stub = new StubIStreamManager();
		stub.OpenStreamAsyncStringAccessMode = (_, __) =>
			Task.FromResult(new MemoryStream(buffer) as Stream);
		return new BoardStreamRepository(stub);
	}
}
```

Stubs

Stubs are useful if you have to inject mock implementation in existing code without changing it. Let us look at an example. We want to test the board managers `FileStreamManager` and `CloudBlobStreamManager`.

```
/// <summary>
/// Stream manager implementation for local files.
/// </summary>
public class FileStreamManager : IStreamManager
{
	// Note the use of Lazy<T> here.
	private Lazy<string> boardsDirectory = null;

	/// <summary>
	/// Initializes a new instance of the <see cref="FileStreamManager"/> class.
	/// </summary>
	/// <remarks>
	/// Boards will be stored in the subdirectory "Boards" in the roaming application data
	/// directory.
	/// </remarks>
	public FileStreamManager()
	{
		Contract.Ensures(this.boardsDirectory != null);

		this.boardsDirectory = new Lazy<string>(FileStreamManager.GetBoardsDirectory);
	}
```

```
        /// <summary>
        /// Initializes a new instance of the <see cref="FileStreamManager"/> class.
        /// </summary>
        /// <param name="boardsDirectory">Directory where the boards should be stored.</param>
        public FileStreamManager(string boardsDirectory)
        {
                Contract.Ensures(this.boardsDirectory != null);

                this.boardsDirectory = new Lazy<string>(() => boardsDirectory);
        }

        // Note the use of inheritdoc here. Documentation is inherited from interface.
        /// <inheritdoc />
        [SuppressMessage("Microsoft.Reliability", "CA2000:Dispose objects before losing scope", Justification = "Call
        public Task<Stream> OpenStreamAsync(string boardName, AccessMode accessMode)
        {
                // Note that we do not need any preconditions here. They are inherited from
                // the base interface.

                return Task.FromResult(new FileStream(
                        Path.Combine(this.boardsDirectory.Value, boardName),
                        accessMode == AccessMode.Read ? FileMode.Open : FileMode.OpenOrCreate) as Stream);
        }

        private static string GetBoardsDirectory()
        {
                Contract.Ensures(Contract.Result<string>() != null);
                Contract.Ensures(Contract.Result<string>().Length > 0);

                var appDataPath = Environment.GetFolderPath(System.Environment.SpecialFolder.ApplicationData);

                // Note: Try this method in the debugger and check out the new features of the
                // Autos window.
                return Path.Combine(appDataPath, "Boards");
        }
}
```

In order to test `OpenStreamAsync`, we need to replace File API with a mock object. For that, we create a fake assembly as shown above. Here is the MS Fakes configuration. Note the shim generation for `FileStream`.

```
<Fakes xmlns="http://schemas.microsoft.com/fakes/2011/">
  <Assembly Name="mscorlib" Version="4.0.0.0"/>
  <StubGeneration>
    <Clear />
    <Add TypeName="IAsyncResult!"/>
  </StubGeneration>
  <ShimGeneration>
    <Clear />
    <Add TypeName="FileStream!" />
  </ShimGeneration>
</Fakes>
```

With that, we can write the test:

```
[TestMethod]
[TestCategory("With fakes")]
public async Task FileStreamManagerShimmedLoadTest()
{
        using (ShimsContext.Create())
        {
                // Note how we use a shimmed constructor here.
                ShimFileStream.ConstructorStringFileMode = (@this, fileName, __) =>
                        {
                                Assert.IsTrue(fileName.EndsWith("\\AppData\\Roaming\\Boards\\" + dummyBoardName));
                                new ShimFileStream(@this)
                                        {
                                                ReadAsyncByteArrayInt32Int32CancellationToken = (buffer, ___, ____, _
                                                {
                                                        BoardSampleData.sampleBoard.CopyTo(buffer, 0);
                                                        return Task.FromResult(BoardSampleData.sampleBoard.Length);
                                                }
                                        };
```

```
                    };

                var repository = new BoardStreamRepository(new FileStreamManager());
                var result = await repository.LoadAsync(dummyBoardName);

                CollectionAssert.AreEqual(BoardSampleData.sampleBoard, (byte[])result);
            }
        }
```