Tutorials

Justin Crites
Rice University
22 January 2005

NUnit Overview and Tutorial

NUnit vs. JUnit

NUnit is very similar to JUnit in that all test cases are built directly into the code of the project. However, NUnit uses a very different mechanism than JUnit to specify test stubs, cases, and properties.

NUnit uses .NET Attributes to specify testing information. If you are unfamiliar with C# attributes, please review their usage, syntax, and semantics, as we are likely to use them in this project beyond simply NUnit.

Some Attribute Resources & Guides

- MSDN Introduction to Attributes (this is a formal introduction, with links to the C# Standard, etc.)
- MSDN Attributes Tutorial
- The Code Project – Attributes in C#

From this point forward this document assumes you have a good understanding of .NET attributes.

The best place to start is with the NUnit sample source. I will be referring to parts of it in this tutorial. If it is installed to its default location, you can find the source in `C:\Program Files\NUnit 2.2\src\samples\csharp`. You can download NUnit from their website, http://www.nunit.com.

Before attempting to compile the NUnit example code, please read the `ReadMe.txt` file provided with the source, it contains important information necessary for compilation.

NUnit tests are grouped into a number of classes, called test fixtures. You denote a test fixture class by the `TestFixture` attribute. Any class that contains tests must be marked with this attribute so that NUnit can find the tests. In the example code, the test fixture looks like this:

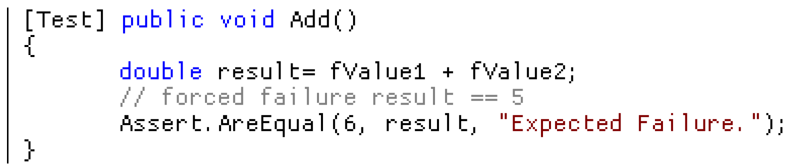
```
namespace NUnit.Samples
{
	using System;
	using NUnit.Framework;

	/// <summary>Some simple Tests.</summary>
	[TestFixture]
	public class SimpleCSharpTest
	{
		...
```

Another point to note is that all NUnit attributes you will be using reside in the `NUnit.Framework` namespace. It must be included via keyword `using` in any file you work with (or, alternatively, all attributes must be fully qualified, e.g., `[NUnit.Framework.TextFixture]`, which looks ugly, so don't do it).

Test Cases

Rather than having to begin each test case with "test" as in JUnit, you merely mark a test function with the attribute `[Test]`. An example test case that fails is below:



```
[Test] public void Add()
{
        double result= fValue1 + fValue2;
        // forced failure result == 5
        Assert.AreEqual(6, result, "Expected Failure.");
}
```

The Assert class contains many of the methods you will use to construct test cases. Some methods it has are `AreEqual()`, `AreSame()`, `IsTrue()`, and `Fail()`. If you wish to read more into detail about this class, please visit the NUnit Assertions page: http://www.nunit.org/assertions.html.

It is important to note the distinction between the `AreSame()` and `AreEqual()` methods; quote the page: "The `AreSame` method tests that the same objects are referenced by both arguments. All the variants of `AreEqual` test for equality."

SetUp and TearDown

Sometimes test cases have a complicated shared set up process (e.g., creating an Internet or database connection). In these instances it is awkward to duplicate the same code across every test case. NUnit provides the attributes `[SetUp]` and `[TearDown]`. These attributes are applied to functions, like the `[Test]` attribute. They cause NUnit to execute the functions before every `[Test]` in the test fixture in the case of `[SetUp]`, and after in the case of `[TearDown]`. These functions allow you to initialize private members of the test fixture before tests, and clean up after tests.

In newer versions of NUnit, there are two new attributes `[TestFixtureSetUp]` and `[TestFixtureTearDown]`. They do what you would expect, provide initialization and cleanup code for the entire test fixture.

Other Attributes: `ExpectedException, Ignore`

Sometimes it is expected in testing that a given method will throw an exception. When this is the case, you should declare the test method with the `[ExpectedException]` attribute. Some example code is below:

```
[Test]
[ExpectedException(typeof(InvalidOperationException))]
public void ExpectAnException()
{
	throw new InvalidCastException();
}
```

This method will fail as a test since the thrown exception is not the expected exception.

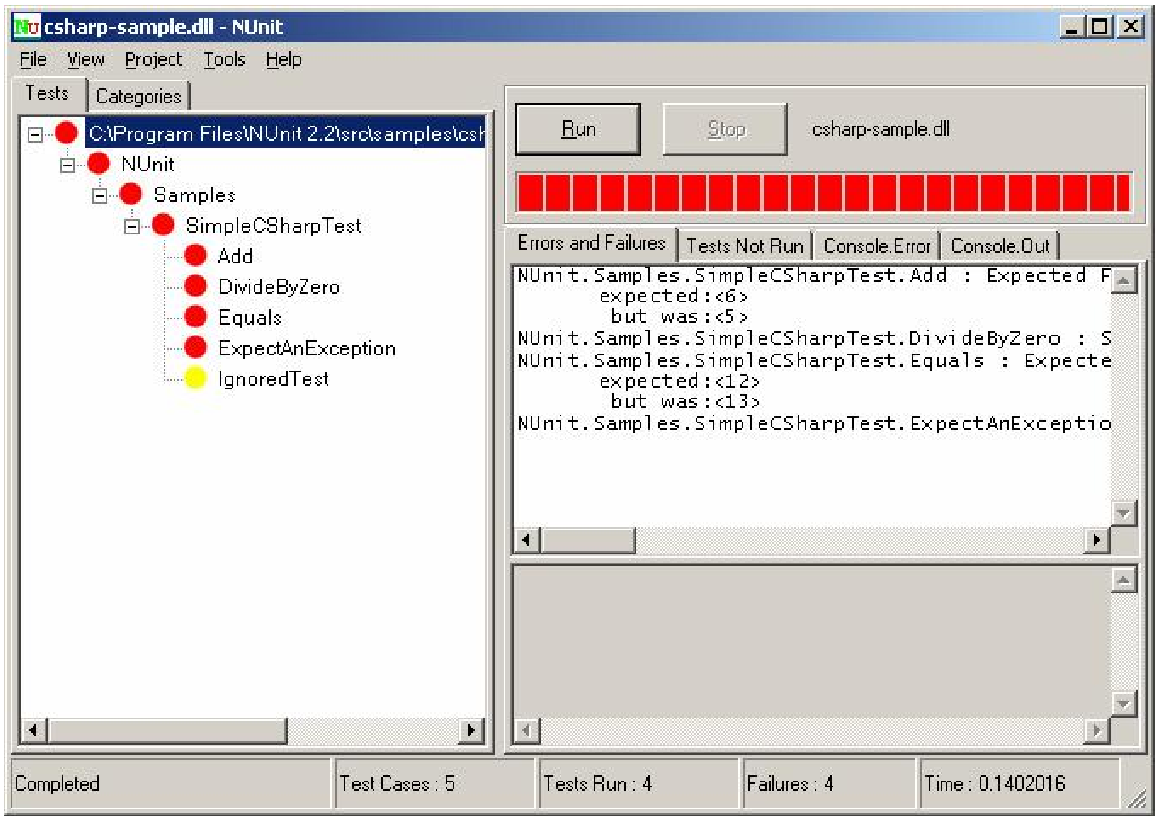
Ignore is an attribute that can be applied to either tests or test fixtures, and will cause NUnit to skip over that test fixture or case. It is preferred over commenting out the test or renaming methods, since the tests will be compiled with the rest of the code and there is an indication at run time that the test is not being run. This ensures that the tests will not be forgotten. Example:

```
[Test]
[Ignore("ignored test")]
public void IgnoredTest()
{
	throw new Exception();
}
```

There are more attributes available to NUnit. All the attributes covered here and more are documented online at http://www.nunit.org/attributes.html.

Running Test Cases

NUnit is not currently built into Visual Studio, but the provided GUI program is easy to use. Run the GUI program, load the assembly you wish to test, and hit the Run button. If you wish, you can also drag the assembly file into the Tests tree of the GUI (on the left).



```
NUnit.Samples.SimpleCSharpTest.Add : Expected F
       expected:<6>
        but was:<5>
NUnit.Samples.SimpleCSharpTest.DivideByZero : S
NUnit.Samples.SimpleCSharpTest.Equals : Expecte
       expected:<12>
        but was:<13>
NUnit.Samples.SimpleCSharpTest.ExpectAnExceptio
```

```
Completed    Test Cases : 5    Tests Run : 4    Failures : 4    Time : 0.1402016
```