Most Complete NUnit Unit Testing Framework Cheat Sheet

by Anton Angelov MVB · Jun. 17, 18 · DevOps Zone · Tutorial

An essential part of every UI test framework is the use of a unit testing framework. One of **the most popular ones in the .NET world is NUnit**. However, you cannot find a single place where you can get started with its syntax. **So, I decided that it would be great to create a complete cheat sheet.** I hope that you will find it useful. Enjoy!

Installation

```
Install-Package NUnit
Install-Package NUnit.TestAdapter
Install-Package Microsoft.NET.Test.Sdk
```

To discover or execute test cases, VSTest would call the test adapters based on your project configuration. (That is why NUnit/xUnit/MSTest all ask you to install a test adapter NuGet package to your unit testing projects). So **NUnit.TestAdapter** exists for that purposes.

NUnit itself implements the testing frameworks and its contracts. So you need to add a NuGet reference to it to write unit test cases and have them compiled. Only compiled projects along with the test adapter can then be consumed by Visual Studio.

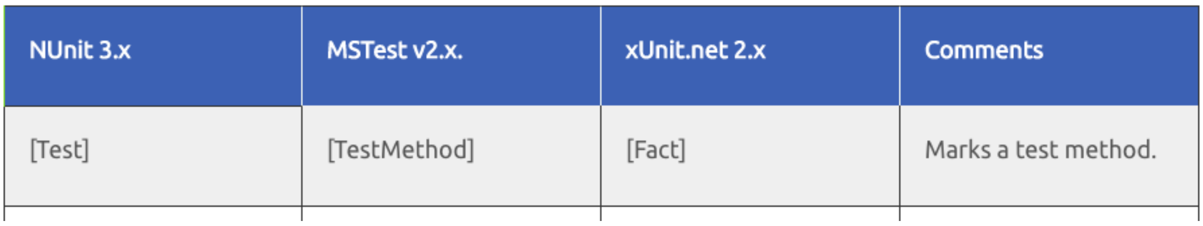
Test Execution Workflow

```
using NUnit.Framework;
namespace NUnitUnitTests
{
    // A class that contains NUnit unit tests. (Required)
    [TestFixture]
    public class NonBellatrixTests
    {
        [OneTimeSetUp]
        public void ClassInit()
        {
            // Executes once for the test class. (Optional)
        }
        [SetUp]
        public void TestInit()
        {
            // Runs before each test. (Optional)
        }
        [Test]
        public void TestMethod()
        {
        }
        [TearDown]
        public void TestCleanup()
        {
            // Runs after each test. (Optional)
        }
        [OneTimeTearDown]
        public void ClassCleanup()
        {
            // Runs once after all tests in this class are executed. (Optional)
            // Not guaranteed that it executes instantly after all tests from the class.
        }
    }
}
// A SetUpFixture outside of any namespace provides SetUp and TearDown for the entire assembly.
[SetUpFixture]
public class MySetUpClass
{
    [OneTimeSetUp]
    public void RunBeforeAnyTests()
    {
        // Executes once before the test run. (Optional)
    }
    [OneTimeTearDown]
    public void RunAfterAnyTests()
    {
        // Executes once after the test run. (Optional)
    }
}
```

OneTimeSetUp from SetUpFixture (once per assembly)
OneTimeSetUp from TestFixture (once per test class class)
SetUp (before each test of the class)
Test1
TearDown (after each test of the class)
SetUp
Test2
TearDown
...
OneTimeTearDown from TestFixture (once per test class)
OneTimeSetUp from TestFixture
...
OneTimeTearDown from TestFixture
OneTimeTearDown from SetUpFixture (once per assembly)

Attributes Comparison

Comparing NUnit to other frameworks.



NUnit 3.x	MSTest v2.x.	xUnit.net 2.x	Comments
[Test]	[TestMethod]	[Fact]	Marks a test method.

[TestFixture]	[TestClass]	n/a	Marks a test class.
[SetUp]	[TestInitialize]	Constructor	Triggered before every test case.
[TearDown]	[TestCleanup]	IDisposable.Dispose	Triggered after every test case.
[OneTimeSetUp]	[ClassInitialize]	IClassFixture<T>	One-time triggered method before test cases start.
[OneTimeTearDown]	[ClassCleanup]	IClassFixture<T>	One-time triggered method after test cases end.
[Ignore("reason")]	[Ignore]	[Fact(Skip="reason")]	Ignores a test case.
[Property]	[TestProperty]	[Trait]	Sets arbitrary metadata on a test.
[Theory]	[DataRow]	[Theory]	Configures a data-driven test.
[Category("")]	[TestCategory("")]	[Trait("Category", "")]	Categorizes the test cases or classes.

Assertions

Assertions — Classic Model

The classic Assert model uses a separate method to express each individual assertion of which it is capable.

```
Assert.AreEqual(28, _actualFuel); // Tests whether the specified values are equal.
Assert.AreNotEqual(28, _actualFuel); // Tests whether the specified values are unequal. Same as AreEqua
Assert.AreSame(_expectedRocket, _actualRocket); // Tests whether the specified objects both refer to th
Assert.AreNotSame(_expectedRocket, _actualRocket); // Tests whether the specified objects refer to diff
Assert.IsTrue(_isThereEnoughFuel); // Tests whether the specified condition is true
Assert.IsFalse(_isThereEnoughFuel); // Tests whether the specified condition is false
Assert.IsNull(_actualRocket); // Tests whether the specified object is null
Assert.IsNotNull(_actualRocket); // Tests whether the specified object is non-null
Assert.IsInstanceOf(_actualRocket, typeof(Falcon9Rocket)); // Tests whether the specified object is an
Assert.IsNotInstanceOf(_actualRocket, typeof(Falcon9Rocket)); // Tests whether the specified object is
```

```
StringAssert.AreEqualIgnoringCase(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specifiec
StringAssert.Contains(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specified string cont
StringAssert.DoesNotContain(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specified strir
StringAssert.StartsWith(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specified string be
StringAssert.StartsWith(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specified string be
StringAssert.IsMatch("(281)388-0388", @"(?d{3})?-? *d{3}-? *-?d{4}"); // Tests whether the specified st
StringAssert.DoesNotMatch("281)388-0388", @"(?d{3})?-? *d{3}-? *-?d{4}"); // Tests whether the specifie
CollectionAssert.AreEqual(_expectedRockets, _actualRockets); // Tests whether the specified collections
CollectionAssert.AreNotEqual(_expectedRockets, _actualRockets); // Tests whether the specified collecti
CollectionAssert.AreEquivalent(_expectedRockets, _actualRockets); // Tests whether two collections cont
CollectionAssert.AreNotEquivalent(_expectedRockets, _actualRockets); // Tests whether two collections c
CollectionAssert.AllItemsAreInstancesOfType(_expectedRockets, _actualRockets); // Tests whether all ele
CollectionAssert.AllItemsAreNotNull(_expectedRockets); // Tests whether all items in the specified coll
CollectionAssert.AllItemsAreUnique(_expectedRockets); // Tests whether all items in the specified colle
CollectionAssert.Contains(_actualRockets, falcon9); // Tests whether the specified collection contains
CollectionAssert.DoesNotContain(_actualRockets, falcon9); // Tests whether the specified collection doe
CollectionAssert.IsSubsetOf(_expectedRockets, _actualRockets); // Tests whether one collection is a sub
CollectionAssert.IsNotSubsetOf(_expectedRockets, _actualRockets); // Tests whether one collection is nc
Assert.Throws<ArgumentNullException>(() => new Regex(null)); // Tests whether the code specified by del
```

Assertions — Constraint Model

The constraint-based Assert model uses a single method of the Assert class for all assertions. The logic necessary to carry out each assertion is embedded in the constraint object passed as the second parameter to that method. The second argument in this assertion uses one of NUnit's **syntax helpers** to create an **EqualConstraint**.

```
Assert.That(28, Is.EqualTo(_actualFuel)); // Tests whether the specified values are equal.
Assert.That(28, Is.Not.EqualTo(_actualFuel)); // Tests whether the specified values are unequal. Same a
Assert.That(_expectedRocket, Is.SameAs(_actualRocket)); // Tests whether the specified objects both ref
Assert.That(_expectedRocket, Is.Not.SameAs(_actualRocket)); // Tests whether the specified objects refe
Assert.That(_isThereEnoughFuel, Is.True); // Tests whether the specified condition is true
Assert.That(_isThereEnoughFuel, Is.False); // Tests whether the specified condition is false
Assert.That(_actualRocket, Is.Null); // Tests whether the specified object is null
Assert.That(_actualRocket, Is.Null); // Tests whether the specified object is null
Assert.That(_actualRocket, Is.Not.Null); // Tests whether the specified object is non-null
Assert.That(_actualRocket, Is.InstanceOf<Falcon9Rocket>()); // Tests whether the specified object is ar
Assert.That(_actualRocket, Is.Not.InstanceOf<Falcon9Rocket>()); // Tests whether the specified object i
Assert.That(_actualFuel, Is.GreaterThan(20)); // Tests whether the specified object greater than the sp
```

Advanced Attributes

Author Attribute

The **Author** Attribute adds information about the author of the tests. It can be applied to test fixtures and to tests.

```
[TestFixture]
[Author("Joro Doev", "joro.doev@bellatrix.solutions")]
public class RocketFuelTests
{
   [Test]
   public void RocketFuelMeassuredCorrectly_When_Landing() { /* ... */ }
   [Test]
   [Author("Ivan Penchev")]
   public void RocketFuelMeassuredCorrectly_When_Flying() { /* ... */ }
}
```

Repeat Attribute

RepeatAttribute is used on a test method to specify that it should be executed multiple times. If any repetition fails, the remaining ones are not run and a failure is reported.

```
[Test]
[Repeat(10)]
public void RocketFuelMeassuredCorrectly_When_Flying() { /* ... */ }
```

Combinatorial Attribute

The **CombinatorialAttribute** is used on a test to specify that NUnit should generate test cases for all possible combinations of the individual data items provided for the parameters of a test.

```
[Test, Combinatorial]
public void CorrectFuelMeassured_When_X_Site([Values(1,2,3)] int x, [Values("A","B")] string s)
{
    ...
}
```

Generated tests:

CorrectFuelMeassured_When_X_Site(1, "A")
CorrectFuelMeassured_When_X_Site(1, "B")
CorrectFuelMeassured_When_X_Site(2, "A")
CorrectFuelMeassured_When_X_Site(2, "B")
CorrectFuelMeassured_When_X_Site(3, "A")

CorrectFuelMeassured_When_X_Site(3, "B")

Pairwise Attribute

The **PairwiseAttribute** is used on a test to specify that NUnit should generate test cases in such a way that all possible pairs of values are used.

```
[Test, Pairwise]
public void ValidateLandingSiteOfRover_When_GoingToMars
    ([Values("a", "b", "c")] string a, [Values("+", "-")] string b, [Values("x", "y")] string c)
{
    Debug.WriteLine("{0} {1} {2}", a, b, c);
}
```

Resulted pairs:

a + y

a - x

b - y

b + x

c - x

c + y

Random Attribute

The **RandomAttribute** is used to specify a set of random values to be provided for an individual numeric parameter of a parameterized test method.

The following test will be executed fifteen times, three times for each value of x, each combined with 5 random doubles from -1.0 to +1.0.

```
[Test]
public void GenerateRandomLandingSiteOnMoon([Values(1,2,3)] int x, [Random(-1.0, 1.0, 5)] double d)
{
    ...
}
```

Range Attribute

The **RangeAttribute** is used to specify a range of values to be provided for an individual parameter of a parameterized test method. NUnit creates test cases from all possible combinations of the provided on parameters - the combinatorial approach.

```
[Test]
public void CalculateJupiterBaseLandingPoint([Values(1,2,3)] int x, [Range(0.2,0.6)] double y)
{
    //...
}
```

Generated tests:

CalculateJupiterBaseLandingPoint(1, 0.2)
CalculateJupiterBaseLandingPoint(1, 0.4)

CalculateJupiterBaseLandingPoint(1, 0.6)
CalculateJupiterBaseLandingPoint(2, 0.2)
CalculateJupiterBaseLandingPoint(2, 0.4)
CalculateJupiterBaseLandingPoint(2, 0.6)
CalculateJupiterBaseLandingPoint(3, 0.2)
CalculateJupiterBaseLandingPoint(3, 0.4)
CalculateJupiterBaseLandingPoint(3, 0.6)

Retry Attribute

RetryAttribute is used on a test method to specify that it should be rerun if it fails, up to a maximum number of times.

```
[Test]
[Retry(3)]
public void CalculateJupiterBaseLandingPoint([Values(1,2,3)] int x, [Range(0.2,0.6)] double y)
{
    //...
}
```

Timeout Attribute

The **TimeoutAttribute** is used to specify a timeout value in milliseconds for a test case. If the test case runs longer than the time specified it is immediately cancelled and reported as a failure, with a message indicating that the timeout was exceeded.

```
[Test, Timeout(2000)]
public void FireRocketToProximaCentauri()
{
    ...
}
```

Execute Tests in Parallel

Parallel execution of methods within a class is supported starting with NUnit 3.7. In earlier releases, parallel execution only applies down to the TestFixture level, **ParallelScope.Childrenworks** as **ParallelScope.Fixtures** and any **ParallelizableAttribute** placed on a method is ignored.

```
[assembly: Parallelizable(ParallelScope.Fixtures)]
[assembly:LevelOfParallelism(3)]
```

The **ParallelizableAttribute** may be specified on multiple levels of the tests. Settings at a higher level may affect lower level tests, unless those lower-level tests override the inherited settings.

```
[TestFixture]
[Parallelizable(ParallelScope.Fixtures)]
public class TestFalcon9EngineLevels
{
    // ...
}
```

Like This Article? Read More From DZone

Unit Testing With .NET Core

How to Write Unit Tests for a .NET Core

Unit Testing With .NET Core

How to Write Unit Tests for a .NET Core Application

Unit Testing With C++: The How and the Why

Free DZone Refcard
Compliant DevOps

Topics: DEVOPS , TESTING , TUTORIAL , UI TESTING , UNIT TESTING , .NET , NUNIT