NSubstitute

========

[![Build status](https://ci.appveyor.com/api/projects/status/github/nsubstitute/NSubstitute?svg=true)](https://ci.appveyor.com/project/nsubstitute/NSubstitute/branch/master) [![Travis Build Status](https://travis-ci.org/nsubstitute/NSubstitute.svg?branch=master)](https://travis-ci.org/nsubstitute/NSubstitute)

Visit the [NSubstitute website](http://nsubstitute.github.com) for more information.

### What is it?

NSubstitute is designed as a friendly substitute for .NET mocking libraries.

It is an attempt to satisfy our craving for a mocking library with a succinct syntax that helps us keep the focus on the intention of our tests, rather than on the configuration of our test doubles. We've tried to make the most frequently required operations obvious and easy to use, keeping less usual scenarios discoverable and accessible, and all the while maintaining as much natural language as possible.

Perfect for those new to testing, and for others who would just like to to get their tests written with less noise and fewer lambdas.

### Getting help

If you have questions or feedback on NSubstitute, head on over to the [NSubstitute discussion group](http://groups.google.com/group/nsubstitute).

### Basic use

Let's say we have a basic calculator interface:

public interface ICalculator

{

int Add(int a, int b);

string Mode { get; set; }

event Action PoweringUp;

}

We can ask NSubstitute to create a substitute instance for this type. We could ask for a stub, mock, fake, spy, test double etc., but why bother when we just want to substitute an instance we have some control over?

\_calculator = Substitute.For<ICalculator>();

Now we can tell our substitute to return a value for a call:

\_calculator.Add(1, 2).Returns(3);

Assert.That(\_calculator.Add(1, 2), Is.EqualTo(3));

We can check that our substitute received a call, and did not receive others:

\_calculator.Add(1, 2);

\_calculator.Received().Add(1, 2);

\_calculator.DidNotReceive().Add(5, 7);

If our Received() assertion fails, NSubstitute tries to give us some help as to what the problem might be:

NSubstitute.Exceptions.ReceivedCallsException : Expected to receive a call matching:

Add(1, 2)

Actually received no matching calls.

Received 2 non-matching calls (non-matching arguments indicated with '\*' characters):

Add(1, \*5\*)

Add(\*4\*, \*7\*)

We can also work with properties using the Returns syntax we use for methods, or just stick with plain old property setters (for read/write properties):

\_calculator.Mode.Returns("DEC");

Assert.That(\_calculator.Mode, Is.EqualTo("DEC"));

\_calculator.Mode = "HEX";

Assert.That(\_calculator.Mode, Is.EqualTo("HEX"));

NSubstitute supports argument matching for setting return values and asserting a call was received:

\_calculator.Add(10, -5);

\_calculator.Received().Add(10, Arg.Any<int>());

\_calculator.Received().Add(10, Arg.Is<int>(x => x < 0));

We can use argument matching as well as passing a function to Returns() to get some more behaviour out of our substitute (possibly too much, but that's your call):

\_calculator

.Add(Arg.Any<int>(), Arg.Any<int>())

.Returns(x => (int)x[0] + (int)x[1]);

Assert.That(\_calculator.Add(5, 10), Is.EqualTo(15));

Returns() can also be called with multiple arguments to set up a sequence of return values.

\_calculator.Mode.Returns("HEX", "DEC", "BIN");

Assert.That(\_calculator.Mode, Is.EqualTo("HEX"));

Assert.That(\_calculator.Mode, Is.EqualTo("DEC"));

Assert.That(\_calculator.Mode, Is.EqualTo("BIN"));

Finally, we can raise events on our substitutes (unfortunately C# dramatically restricts the extent to which this syntax can be cleaned up):

bool eventWasRaised = false;

\_calculator.PoweringUp += () => eventWasRaised = true;

\_calculator.PoweringUp += Raise.Event<Action>();

Assert.That(eventWasRaised);

### Building

If you have Visual Studio 2008, 2010, 2012, 2013, or 2015 you should be able to compile NSubstitute and run the unit tests using the NUnit GUI or console test runner (see the ThirdParty directory). Note that some tests are marked `[Pending]` and are not meant to pass at present, so it is a good idea to exclude tests in the Pending category from test runs.

To do full builds you'll also need Ruby, as the jekyll gem is used to generate the website.