QuickStart

Featured

Updated Jun 24, 2010 by kzu.net

Introduction to Moq

Moq is intended to be simple to use, strong typed (no magic strings!, and therefore full compiler-verified and refactoring-friendly) and minimalistic (while still fully functional!).

Methods

```
var mock = new Mock<IFoo>();
mock.Setup(foo => foo.DoSomething("ping")).Returns(true);


// out arguments
var outString = "ack";
// TryParse will return true, and the out argument will return "ack", lazy evaluated
mock.Setup(foo => foo.TryParse("ping", out outString)).Returns(true);


// ref arguments
var instance = new Bar();
// Only matches if the ref argument to the invocation is the same instance
mock.Setup(foo => foo.Submit(ref instance)).Returns(true);


// access invocation arguments when returning a value
mock.Setup(x => x.DoSomething(It.IsAny<string>()))
                 .Returns((string s) => s.ToLower());
// Multiple parameters overloads available


// throwing when invoked
mock.Setup(foo => foo.DoSomething("reset")).Throws<InvalidOperationException>();
mock.Setup(foo => foo.DoSomething("")).Throws(new ArgumentException("command");


// lazy evaluating return value
mock.Setup(foo => foo.GetCount()).Returns(() => count);


// returning different values on each invocation
var mock = new Mock<IFoo>();
var calls = 0;
mock.Setup(foo => foo.GetCountThing())
    .Returns(() => calls)
    .Callback(() => calls++);
// returns 0 on first invocation, 1 on the next, and so on
Console.WriteLine(mock.Object.GetCountThing());
```

Matching Arguments

```
// any value
```

```
mock.Setup(foo => foo.DoSomething(It.IsAny<string>())).Returns(true);


// matching Func<int>, lazy evaluated
mock.Setup(foo => foo.Add(It.Is<int>(i => i % 2 == 0))).Returns(true);


// matching ranges
mock.Setup(foo => foo.Add(It.IsInRange<int>(0, 10, Range.Inclusive))).Returns(true);


// matching regex
mock.Setup(x => x.DoSomething(It.IsRegex("[a-d]+", RegexOptions.IgnoreCase))).Returns("foo");
```

Properties

```
mock.Setup(foo => foo.Name).Returns("bar");


// auto-mocking hierarchies (a.k.a. recursive mocks)
mock.Setup(foo => foo.Bar.Baz.Name).Returns("baz");

// expects an invocation to set the value to "foo"
mock.SetupSet(foo => foo.Name = "foo");

// or verify the setter directly
mock.VerifySet(foo => foo.Name = "foo");
```

- Setup a property so that it will automatically start tracking its value (also known as Stub):

  ```
  // start "tracking" sets/gets to this property
  mock.SetupProperty(f => f.Name);

  // alternatively, provide a default value for the stubbed property
  mock.SetupProperty(f => f.Name, "foo");


  // Now you can do:

  IFoo foo = mock.Object;
  // Initial value was stored
  Assert.Equal("foo", foo.Name);

  // New value set which changes the initial value
  foo.Name = "bar";
  Assert.Equal("bar", foo.Name);
  ```

- Stub all properties on a mock (not available on Silverlight):

  ```
  mock.SetupAllProperties();
  ```

Events

```
// Raising an event on the mock
mock.Raise(m => m.FooEvent += null, new FooEventArgs(fooValue));

// Raising an event on a descendant down the hierarchy
mock.Raise(m => m.Child.First.FooEvent += null, new FooEventArgs(fooValue));

// Causing an event to raise automatically when Submit is invoked
mock.Setup(foo => foo.Submit()).Raises(f => f.Sent += null, EventArgs.Empty);
// The raised event would trigger behavior on the object under test, which
// you would make assertions about later (how its state changed as a consequence, typically)
```

```
// Raising a custom event which does not adhere to the EventHandler pattern
public delegate void MyEventHandler(int i, bool b);
public interface IFoo
{
  event MyEventHandler MyEvent;
}

var mock = new Mock<IFoo>();
...
// Raise passing the custom arguments expected by the event delegate
mock.Raise(foo => foo.MyEvent += null, 25, true);
```

Callbacks

```
var mock = new Mock<IFoo>();
mock.Setup(foo => foo.Execute("ping"))
    .Returns(true)
    .Callback(() => calls++);


// access invocation arguments
mock.Setup(foo => foo.Execute(It.IsAny<string>()))
    .Returns(true)
    .Callback((string s) => calls.Add(s));

// alternate equivalent generic method syntax
mock.Setup(foo => foo.Execute(It.IsAny<string>()))
    .Returns(true)
    .Callback<string>(s => calls.Add(s));

// access arguments for methods with multiple parameters
mock.Setup(foo => foo.Execute(It.IsAny<int>(), It.IsAny<string>()))
    .Returns(true)
    .Callback<int, string>((i, s) => calls.Add(s));

// callbacks can be specified before and after invocation
mock.Setup(foo => foo.Execute("ping"))
    .Callback(() => Console.WriteLine("Before returns"))
    .Returns(true)
    .Callback(() => Console.WriteLine("After returns"));
```

Verification

```
mock.Verify(foo => foo.Execute("ping"));

// Verify with custom error message for failure
mock.Verify(foo => foo.Execute("ping"), "When doing operation X, the service should be pinged always");

// Method should never be called
mock.Verify(foo => foo.Execute("ping"), Times.Never());

// Called at least once
mock.Verify(foo => foo.Execute("ping"), Times.AtLeastOnce());

mock.VerifyGet(foo => foo.Name);

// Verify setter invocation, regardless of value.
mock.VerifySet(foo => foo.Name);

// Verify setter called with specific value
mock.VerifySet(foo => foo.Name ="foo");

// Verify setter with an argument matcher
mock.VerifySet(foo => foo.Value = It.IsInRange(1, 5, Range.Inclusive));
```

Customizing Mock Behavior

- Make mock behave like a "true Mock", raising exceptions for anything that doesn't have a corresponding expectation: in Moq slang a "Strict" mock; default behavior is "Loose" mock, which never throws and returns default values or empty arrays, enumerables, etc. if no expectation is set for a member

```
var mock = new Mock<IFoo>(MockBehavior.Strict);
```

- Invoke base class implementation if no expectation overrides the member (a.k.a. "Partial Mocks" in Rhino Mocks): default is false. (**this is required if you are mocking Web/Html controls in System.Web!**)

```
var mock = new Mock<IFoo> { CallBase = true };
```

- Make an automatic recursive mock: a mock that will return a new mock for every member that doesn't have an expectation and whose return value can be mocked (i.e. it is not a value type)

```
var mock = new Mock<IFoo> { DefaultValue = DefaultValue.Mock };
// default is DefaultValue.Empty

// this property access would return a new mock of IBar as it's "mock-able"
IBar value = mock.Object.Bar;

// the returned mock is reused, so further accesses to the property return
// the same mock instance. this allows us to also use this instance to
// set further expectations on it if we want
var barMock = Mock.Get(value);
barMock.Setup(b => b.Submit()).Returns(true);
```

- Centralizing mock instance creation and management: you can create and verify all mocks in a single place by using a MockFactory, which allows setting the MockBehavior, its CallBase and DefaultValue consistently

```
var factory = new MockFactory(MockBehavior.Strict) { DefaultValue = DefaultValue.Mock };

// Create a mock using the factory settings
var fooMock = factory.Create<IFoo>();

// Create a mock overriding the factory settings
var barMock = factory.Create<IBar>(MockBehavior.Loose);

// Verify all verifiable expectations on all mocks created through the factory
factory.Verify();
```

Miscellaneous

- Setting expectations for protected members (you can't get intellisense for these, so you access them using the member name as a string):

```
// at the top of the test fixture
using Moq.Protected()

// in the test
var mock = new Mock<CommandBase>();
mock.Protected()
      .Setup<int>("Execute")
      .Returns(5);

// if you need argument matching, you MUST use ItExpr rather than It
// planning on improving this for vNext
mock.Protected()
     .Setup<string>("Execute",
         ItExpr.IsAny<string>())
     .Returns(true);
```

Advanced Features

```
// get mock from a mocked instance
```

```
IFoo foo = // get mock instance somehow
var fooMock = Mock.Get(foo);
fooMock.Setup(f => f.Submit()).Returns(true);


// implementing multiple interfaces in mock
var foo = new Mock<IFoo>();
var disposableFoo = foo.As<IDisposable>();
// now the IFoo mock also implements IDisposable :)
disposableFoo.Setup(df => df.Dispose());


// custom matchers
mock.Setup(foo => foo.Submit(IsLarge())).Throws<ArgumentException>();
...
public string IsLarge()
{
  return Match<string>.Create(s => !String.IsNullOrEmpty(s) && s.Length > 100);
}
```

- Mocking internal types of another project: add the following assembly attribute (typically to the AssemblyInfo.cs) to the project containing the internal types:

```
// This assembly is the default dynamic assembly generated Castle DynamicProxy,
// used by Moq. Paste in a single line.
[assembly:InternalsVisibleTo("DynamicProxyGenAssembly2,PublicKey=0024000004800000940000000602000000240000525
```

Read more

- TDD : Introduction to Moq
- Beginning Mocking With Moq 3 – Part 1
- Beginning Mocking With Moq 3 - Part 2
- Beginning Mocking With Moq 3 - Part 3
- Beginning Mocking With Moq 3 - Part 4
- Unit Testing Revisited - The evolution unit test with C-Sharp 3.0 (Google-translated)
- The automated testing continuum
- Unit Test Linq to Sql in ASP.Net MVC with Moq
- Introduction to Mocking with Moq (Video)
- Comparing Moq to Rhino Mocks
- Moq: Linq, Lambdas and Predicates applied to Mock Objects
- [http://www.goneeded.com/javas/articles/20080324/a1811942466.html
- Improved argument matchers in Moq
- Moq Trigs - Successive Expectations
- Basic Mocking with Moq

Note: when you need to pass the mock for consumption, you must use the `mock.Object` accessor as a consequence of a C# compiler restriction (vote to get it removed at Microsoft Connect)

Head on to the API documentation, download it and have fun! Engage in the discussion group to give us feedback, share your experiences or wishes for vNext!

Comment by david.ka...@gmail.com, Apr 6, 2009

Is there a way to view the old 2.6 QuickStart? We haven't quite ported over and it's a nice reference.

Comment by jcss...@gmail.com, Apr 9, 2009

It looks like you can see the old wiki pages in svn, http://code.google.com/p/moq/source/browse/wiki/QuickStart.wiki?r=477

Comment by danny.er...@gmail.com, Apr 23, 2009