Setup async Task callback in Moq Framework

I've got an interface which declares

```
Task DoSomethingAsync();
```

I'm using MoqFramework for my tests:

```
[TestMethod()]
public async Task MyAsyncTest()
{
   Mock<ISomeInterface> mock = new Mock<ISomeInterface>();
   mock.Setup(arg => arg.DoSomethingAsync()).Callback(() => { <my code here> });
   ...
}
```

Then in my test I execute the code which invokes `await DoSomethingAsync()`. And the test just fails on that line. What am I doing wrong?

c# unit-testing task-parallel-library moq

edited Jan 23 '14 at 0:41 — svick — 137k 27 224 336

asked Jan 21 '14 at 9:05 — Waldemar — 987 2 6 18

3 When you say the test errors on that line, what error does it produce? – AlSki Jan 21 '14 at 9:07

3 Answers

Your method doesn't have any callbacks so there is no reason to use `.CallBack()`. You can simply return a Task with the desired values using `.Returns()` and Task.FromResult, e.g.:

```
MyType someValue=...;
mock.Setup(arg=>arg.DoSomethingAsync())
    .Returns(Task.FromResult(someValue));
```

Update 2014-06-22

Moq 4.2 has two new extension methods to assist with this.

```
mock.Setup(arg=>arg.DoSomethingAsync())
    .ReturnsAsync(someValue);

mock.Setup(arg=>arg.DoSomethingAsync())
    .ThrowsAsync(new InvalidOperationException());
```

Update 2016-05-05

As Seth Flowers mentions in the other answer, `ReturnsAsync` is only available for methods that return a `Task<T>`. For methods that return only a Task,

```
.Returns(Task.FromResult(default(object)))
```

can be used.

As shown in this answer, in .NET 4.6 this is simplified to `.Returns(Task.CompletedTask);`, e.g.:

```
mock.Setup(arg=>arg.DoSomethingAsync())
    .Returns(Task.CompletedTask);
```

edited Mar 23 at 9:42

answered Jan 21 '14 at 11:04

Panagiotis Kanavos
34.9k 4 51 76

It solved the problem. Thank you! – Waldemar Jan 21 '14 at 12:04

.Returns(Task.CompletedTask); that was my answer – Todd Vance Jun 7 '16 at 15:19

ReturnsAsync works great. – Sully Mar 21 at 20:22

Similar Issue

I have an interface that looked roughly like:

```
Task DoSomething(int arg);
```

Symptoms

My unit test failed when my service under test `awaited` the call to `DoSomething`.

Fix

Unlike the accepted answer, you are unable to call `.ReturnsAsync()` on your `Setup()` of this method in this scenario, because the method returns the non-generic `Task`, rather than `Task<T>`.

However, you are still able to use `.Returns(Task.FromResult(default(object)))` on the setup, allowing the test to pass.

answered Nov 16 '15 at 21:32

seth flowers
6,228 2 12 27

1 Just a thought on this, if you need to return a non-generic task (non .net 4.6), I would consider returning Task.Delay(1) as an easy way to return a Task. You can also mimic work too by increasing the time argument. – stevethethread Apr 18 '16 at 11:10

You only need to add `.Returns(Task.FromResult(0));` after the Callback.

Example:

```
mock.Setup(arg => arg.DoSomethingAsync())
    .Callback(() => { <my code here> })
    .Returns(Task.FromResult(0));
```

answered Aug 18 '16 at 19:39

Diego Torres
185 2 2