xUnit Autofac

Use Autofac to resolve xUnit test cases.

The Test runners and discoverers are based on their xUnit counterparts. If `[UseAutofacTestFramework]` is missing, the tests in that class are run by the normal xUnit runners.

Originally a fork of [xunit.ioc.autofac] by @dennisroche

How to use

Install the Nuget package.

```
Install-Package xunit.frameworks.autofac
```

In your testing project, add the following framework

```
[assembly: TestFramework("Your.Test.Project.ConfigureTestFramework", "AssemblyName")]

namespace Your.Test.Project
{
    public class ConfigureTestFramework : AutofacTestFramework
    {
        public ConfigureTestFramework(IMessageSink diagnosticMessageSink)
```

```
            : base(diagnosticMessageSink)
        {
        }

        protected override void ConfigureContainer(ContainerBuilder builder)
        {
            builder.RegisterType<CurrentTestInfo>().As<ICurrentTestInfo>().InstanceP
            builder.RegisterType<CurrentTestClassInfo>().As<ICurrentTestClassInfo>()
            builder.RegisterType<CurrentTestCollectionInfo>().As<ICurrentTestCollect

            builder.RegisterSource(new NSubstituteRegistrationSource()); // https://

            builder.RegisterType<Foo>().As<IFoo>();

            // configure your container
            // e.g. builder.RegisterModule<TestOverrideModule>();
        }
    }
}
```

Example test `class`.

```
[UseAutofacTestFramework] // Without this attribute, the test class will be handled
public class MyAwesomeTests
{
    public MyAwesomeTests(IFoo foo)
    {
        _foo = foo;
    }

    [Fact]
    public void AssertThatWeDoStuff()
    {
        Console.WriteLine(_foo.Bar);
    }

    private readonly ITestOutputHelper _outputHelper;
}

public interface IFoo
{
    Guid Bar { get; }
}

public class Foo : IFoo
{
    public Guid Bar { get; } = Guid.NewGuid();
}
```

`ICollectionFixture<T>` and `IClassFixture<T>` are also supported, together with `INeedModule<T>`. (The latter specifies Autofac modules to be loaded when the lifetime scope is created.) This enables very elegant solutions:

```
[UseAutofacTestFramework]
public class MyEvenMoreAwesomeTests : IUseInMemoryDb
{
    public MyEvenMoreAwesomeTests(IDbConnectionFactory dbConnectionFactory)
    {
        _dbConnectionFactory = dbConnectionFactory;
    }

    [Fact]
    public void AssertThatWeDoEvenMoreStuff()
    {
        using (IDbConnection db = _dbConnectionFactory.Open())
        {
            db.CreateTableIfNotExists<Foo>();
            // ... and so on
        }
    }

    private readonly IDbConnectionFactory _dbConnectionFactory;
}

public interface IUseInMemoryDb : IClassFixture<MemoryDatabaseClassFixture>
{
}

public class MemoryDatabaseClassFixture : IDisposable, INeedModule<MemoryDatabaseCla
{
    private readonly IDbConnection _db;

    public MemoryDatabaseClassFixture(IDbConnectionFactory dbConnectionFactory)
    {
        // Keep the in-memory database alive
        _db = dbConnectionFactory.Open();
    }

    public void Dispose()
    {
        // Now it can rest in peace
        _db?.Dispose();
    }

    public class MemoryDatabaseFixtureModule : Module
    {
        protected override void Load(ContainerBuilder builder)
```

```
          {
              builder.Register(c => new OrmLiteConnectionFactory(":memory:", SqliteDia
          }
      }
  }
```

License

MIT

Releases

No releases published

Packages

No packages published

Contributors 5

Languages

● **C#** 100.0%