Isolating Code Under Test with Microsoft Fakes - Visual Studio (Windows)

learn.microsoft.com/en-us/visualstudio/test/isolating-code-under-test-with-microsoft-fakes

Mikejo5000

Isolate code under test with Microsoft Fakes

Code isolation is a testing strategy often implemented with tools like Microsoft Fakes, where the code you're testing is separated from the rest of the application. This separation is achieved by replacing parts of the application that interact with the code under test with stubs or shims. These are small pieces of code controlled by your tests, which simulate the behavior of the actual parts they're replacing.

The benefit of this approach is that it allows you to focus on testing the specific functionality of the code in isolation. If a test fails, you know the cause is within the isolated code and not somewhere else. Additionally, the use of stubs and shims, provided by Microsoft Fakes, enables you to test your code even if other parts of your application aren't functioning yet.

Requirements

- Visual Studio Enterprise
- A .NET Framework project
- .NET Core, .NET 5.0 or later, and SDK-style project support in Visual Studio 2019 and later. For more information, see Microsoft Fakes for .NET Core and SDK-style projects.

Note

Profiling with Visual Studio isn't available for tests that use Microsoft Fakes.

The Role of Microsoft Fakes in Code Isolation

Microsoft Fakes plays a key role in code isolation by providing two mechanisms - stubs and shims.

- **Stubs**: These are used to replace a class with a small substitute that implements the same interface. This requires your application to be designed such that each component depends only on interfaces, not on other components.

- **Shims**: These are used to modify the compiled code of your application at runtime. Instead of making a specified method call, the application runs the shim code that your test provides. Shims can replace calls to assemblies that you can't modify, such as .NET assemblies.

Typically, stubs are used for calls within your Visual Studio solution, and shims for calls to other referenced assemblies. This is because within your solution, it's good practice to decouple the components by defining interfaces in the way that stubbing requires. However, external assemblies often don't come with separate interface definitions, so shims are used instead.

 Diagram that show Fakes replacing other components.

Recommendations on When to Use Stubs

Stubs are typically used for calls within your Visual Studio solution because it's a good practice to decouple the components by defining interfaces in the way that stubbing requires. However, external assemblies, such as System.dll, typically aren't provided with separate interface definitions, so shims would be used in these cases instead.

Using stubs involves designing your application so that the different components are not dependent on each other, but only on interface definitions. This decoupling makes the application more robust and flexible, and allows you to connect the component under test to stub implementations of the interfaces for testing purposes.

In practice, you can generate stub types from the interface definitions in Visual Studio, then replace the real component with the stub in your test.

Recommendations on When to Use Shims

While stubs are used for calls within your Visual Studio solution, shims are typically used for calls to other referenced assemblies. This is because external assemblies such as System.dll usually aren't provided with separate interface definitions, so shims must be used instead.

However, there are some factors to consider when using shims:

Performance: Shims run slower because they rewrite your code at runtime. Stubs don't have this performance overhead and are as fast as virtual methods can run.

Static methods, sealed types: You can only use stubs to implement interfaces. Therefore, stub types can't be used for static methods, non-virtual methods, sealed virtual methods,

methods in sealed types, and so on.

Internal types: Both stubs and shims can be used with internal types that are made accessible by using the assembly attribute InternalsVisibleToAttribute.

Private methods: Shims can replace calls to private methods if all the types on the method signature are visible. Stubs can only replace visible methods.

Interfaces and abstract methods: Stubs provide implementations of interfaces and abstract methods that can be used in testing. Shims can't instrument interfaces and abstract methods, because they don't have method bodies.

Transitioning Microsoft Fakes in .NET Framework to SDK-Style Projects

Transitioning your .NET Framework test projects that use Microsoft Fakes to SDK-style .NET Framework, .NET Core, or .NET 5+ projects.

You'll need minimal changes in your .NET Framework set up for Microsoft Fakes to transition to .NET Core or .NET 5.0. The cases that you would have to consider are:

- If you're using a custom project template, you need to ensure that it's SDK-style and builds for a compatible target framework.
- Certain types exist in different assemblies in .NET Framework and .NET Core/.NET 5.0 (for example, `System.DateTime` exists in `System/mscorlib` in .NET Framework, and in `System.Runtime` in .NET Core and .NET 5.0), and in these scenarios you need to change the assembly being faked.
- If you have an assembly reference to a fakes assembly and the test project, you might see a build warning about a missing reference similar to:

  ```
  (ResolveAssemblyReferences target) ->
  warning MSB3245: Could not resolve this reference. Could not locate the assembly "
  If this reference is required by your code, you may get compilation errors.
  ```

  This warning is because of necessary changes made in Fakes generation and can be ignored. It can be avoided by removing the assembly reference from the project file, because we now implicitly add them during the build.

Running Microsoft Fakes tests

As long as Microsoft Fakes assemblies are present in the configured `FakesAssemblies` directory (The default being `$(ProjectDir)FakesAssemblies`), you can run tests using the vstest task.

Distributed testing with the vstest task .NET Core and .NET 5+ projects using Microsoft Fakes requires Visual Studio 2019 Update 9 Preview `20201020-06` and higher.

Compatibility and Support for Microsoft Fakes in Different .NET and Visual Studio Versions

Microsoft Fakes in older projects targeting .NET Framework (non-SDK style).

- Microsoft Fakes assembly generation is supported in Visual Studio Enterprise 2015 and higher.
- Microsoft Fakes tests can run with all available Microsoft.TestPlatform NuGet packages.
- Code coverage is supported for test projects using Microsoft Fakes in Visual Studio Enterprise 2015 and higher.

Microsoft Fakes in SDK-style .NET Framework, .NET Core, and .NET 5.0 or later projects

- Microsoft Fakes assembly generation previewed in Visual Studio Enterprise 2019 Update 6 and is enabled by default in Update 8.
- Microsoft Fakes tests for projects that target .NET Framework can run with all available Microsoft.TestPlatform NuGet packages.
- Microsoft Fakes tests for projects that target .NET Core and .NET 5.0 or later can run with Microsoft.TestPlatform NuGet packages with versions 16.9.0-preview-20210106-01 and higher.
- Code coverage is supported for test projects targeting .NET Framework using Microsoft Fakes in Visual Studio Enterprise version 2015 and higher.
- Code coverage support for test projects targeting .NET Core and .NET 5.0 or later using Microsoft Fakes is available in Visual Studio 2019 update 9 and higher.

Related content

- Use stubs to isolate parts of your application from each other for unit testing
- Use shims to isolate your application from other assemblies for unit testing
- Code generation, compilation, and naming conventions in Microsoft Fakes

Additional resources

Documentation

- Isolate Your App with Shims (Unit Testing) - Visual Studio (Windows)

  Learn how to use shim types to divert calls to specific methods in code that you write as part of your test. A shim can return consistent results at every call.

- Use stubs to isolate parts of your app for testing - Visual Studio (Windows)

  Learn about a stub, which is a small piece of code that takes the place of another component during testing. Using a stub returns consistent results.

- Configure Microsoft Fakes code generation - Visual Studio (Windows)

  Learn about options and issues in Fakes code generation and compilation, including the naming conventions for Fakes-generated types, members, and parameters.

Training

Certification

Microsoft Certified: Azure AI Fundamentals - Certifications

Demonstrate fundamental AI concepts related to the development of software and services of Microsoft Azure to create AI solutions.