# How to do MSTest (VS Unit Test) for a MEFed class?

Y. Tosa (July 8th, 2012)

## Introduction

Unit Test in Visual Studio is so convenient that the wizard generated most of the tests you want to perform for your class. Unfortunately the test framework has no idea about MEF (Microsoft Extensibility Framework) which was introduced in .NET 4. I this article I will describe how to modify the wizard generated unit test for your MEFed class. The example give below consists of an exported MEFed class which imports logger. I hope that **I made everything as simple as possible but not simpler**. I found that there are articles discusses on unit tests for a MEFed class, but my method is the simplest. My method applies to Shared lifestyle (Singleton) which is the default MEF behavior.

The example exported MEFed class which is made as a class library is the following class:

 [Export(typeof(MEFImport))]

 public class MEFImport

 {

 [Import(typeof(MEFInterface.ILogger))]

 public MEFInterface.ILogger myLogger = null;

 public MEFImport()

 {

 }

 public bool Test(string msg)

 {

 if (myLogger != null)

 {

 myLogger.ShowLog(msg);

 return true;

 }

 else // when Import fails, the following is executed

 {

 Console.WriteLine("ILogger not resolved");

 return false;

 }

 }

 }

Where ILogger is defined as

 namespace MEFInterface

 {

 public interface ILogger

 {

 void ShowLog(string msg);

 }

 }

If you use Visual Studio 2010 Unit Test Wizard (Test->New Test … and pick **Unit Test Wizard**), then you get the following tests generated:

 [TestMethod()]

 public void MEFImportConstructorTest()

 {

 MEFImport target = new MEFImport();

 Assert.Inconclusive("TODO: Implement code to verify target");

 }

 [TestMethod()]

 public void TestTest()

 {

 MEFImport target = new MEFImport(); // TODO: Initialize to an appropriate value

 string msg = string.Empty; // TODO: Initialize to an appropriate value

 bool res = target.Test(msg);

 Assert.Inconclusive("A method that does not return a value cannot be verified.");

 }

Note that a MEFed class is instantiated in each test, since MSTEST does not understand MEF. How can we make a singleton MEFed object to be tested? My first thought was to introduce an imported variable

 [TestClass()]

 public class MEFImportTest

 {

 [Import(typeof(MEFImport)]

 private MEFImport target = null;

This does not work due to the following reason. In order to test a MEFed class, we have to “compose” assemblies to resolve exports. Usually this is done in the instance of the unit test class MEFImportTest. However, I don’t have the accessor to this instance. In MS Unit Test, the only place I can perform this “initialization” is at the static MyClassInitialize function:

 [ClassInitialize()]

 public static void MyClassInitialize(TestContext testContext)

Because this is “**static**”, you **cannot** “resolve” the instance variable we defined above inside this initializer. You could do the composition in every single test method. This is not testing a “Singleton” MEFed object. I really need a static MEF Import object.

## Solution

It turned out that everything fits together with a static variable:

 [TestClass()]

 public class MEFImportTest

 {

 // introduce static object

 private static MEFImport target = null;

and add the dependency injection “static” function:

 [ClassInitialize()]

 public static void MyClassInitialize(TestContext testContext)

 {

 // Dependency Injection

 DI();

 }

 private static void DI()

 {

 AggregateCatalog agg = new AggregateCatalog();

 var exeCatalog = new AssemblyCatalog(Assembly.GetExecutingAssembly());

 agg.Catalogs.Add(exeCatalog);

 var libCatalog = new AssemblyCatalog(typeof(MEFLib.MEFImport).Assembly);

 agg.Catalogs.Add(libCatalog);

 var container = new CompositionContainer(agg);

 // assign MEFed object

 target = container.GetExportedValue<MEFImport>();

 }

The trick is to assign the static target to GetExportedValue. That is all. Note that I also introduced the concrete ILogger class which is included in the unit test:

 [Export(typeof(MEFInterface.ILogger))]

 public class ConsoleLogger : MEFInterface.ILogger

 {

 public void ShowLog(string msg)

 {

 Console.WriteLine(msg);

 }

 }

That is why we need to have exeCatalog in AggregationCatalog. Now it is easy to modify unit tests to use this MEFed object. Instead of using the instantialed object, just use the MEFed object.

 [TestMethod()]

 public void MEFImportConstructorTest()

 {

 // MEFImport target = new MEFImport();

 // Assert.Inconclusive("TODO: Implement code to verify target");

 Assert.IsNotNull(target);

 }

 [TestMethod()]

 public void TestTest()

 {

 // MEFImport target = new MEFImport(); // TODO: Initialize to an appropriate value

 string msg = “Testing MEFed object”; // TODO: Initialize to an appropriate value

 bool res = target.Test(msg);

 // Assert.Inconclusive("A method that does not return a value cannot be verified.");

 Assert.IsTrue(res);

 }

As you can see, my method is applicable to any singleton MEFed class object.