# **Using a .NET 4 Based DLL From a .NET 2 Based Application**

## This sample show how you can call methods from a .NET 4 DLL from a .NET 2 executable.

# Introduction

There are times when you ask yourself the question: How can you use a .NET 4 DLL from a .NET 2 application? and vice versa?

Personally I’ve encountered two scenarios when I had to solve this problem:

1. I had a 3rd-party control that would load only in a .NET 3.5 application, but I had to use it in a .NET 4 application.
2. I wanted to write a plug-in for Windows Live Writer, which must use .NET 2.0, but I needed to use in my plug-in a .NET 4 DLL.

The official answer is you can’t. Even with the [In-Process Side by Side execution](http://msdn.microsoft.com/en-us/library/ee518876.aspx) (SxS) feature, introduced in .NET 4.

The SxS feature was intended to be used when [COM](http://en.wikipedia.org/wiki/Component_Object_Model) is involved.   
For example, if you got an application that loads plugins, like outlook, and it loads 2 COM plugins, one is using .NET 4 and the other is using .NET 2.0 then it will load two versions of the CLR into the process using the new SxS feature.

**So, what can we do if no COM is involved?**

Well, simply add **COM** to the mixture..

The idea is that you can expose the required classes from your DLL (which uses .NET Framework X) as COM classes (using [COM Interop](http://msdn.microsoft.com/en-us/library/aa645736(v=vs.71).aspx)), and then use those classes from your other DLL (which uses .NET Framework Y). Since you are crossing a COM interface, in-process SxS will kick in and work its magic.

# 

# Building the Sample

Simply compile and run the solution. It contains the following three projects:

1. Net2Assembly - host application written with .NET 2
2. Net4Assembly - DLL written with .NET 4
3. Net4ToNet2Adapter - DLL which helps exposing the Net4Assembly to Net2Assembly

# Steps to work around the problem

## Create a .NET 4 DLL

Suppose we have a .NET 4 DLL which does some .NET 4 functionality. In the attached sample our .NET 4 class prints the CLR version, which should be 4. This DLL is compiled with .NET Framework 4.

## *C#*

***using****System;*

***namespace****Net4Assembly*

*{*

***public******class****MyClass*

*{*

***public******void****DoNet4Action()*

*{*

*Console.WriteLine("CLR version from DLL: {0}", Environment.Version);*

*}*

*}*

*}*

## *Create a .NET 2 EXE*

Here we create a .NET 2 EXE which will eventually call the .NET 4 DLL, currently all it does is write it’s own CLR version.

**C#**

**using** System;

**namespace** Net2Assembly

{

**class** Program

    {

**static** **void** Main(**string**[] args)

        {

            Console.WriteLine("CLR version from EXE: {0}", Environment.Version);

        }

    }

}

## Create a .NET 4 to .NET 2 adapter

Here we create a .NET 4 DLL that exposes the same functionality we need from our original .NET 4 DLL only it exposes it in a COM-friendly way. In this sample, it only needs to delegate the call to the original implementation, but in more advanced scenarios it should translate the parameters to something more COM friendly. In addition to changing the parameters the classes also implement interfaces (as required by COM) and are marked with **[ComVisible](http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.comvisibleattribute.aspx)**and **[Guid](http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.guidattribute.aspx)**attributes to allow access using COM.

Here is our COM visible interface:

**C#**

**using** System;

**using** System.Runtime.InteropServices;

**namespace** Net4ToNet2Adapter

{

    [ComVisible(**true**)]

    [Guid("E36BBF07-591E-4959-97AE-D439CBA392FB")]

**public** **interface** IMyClassAdapter

    {

**void** DoNet4Action();

    }

}

And our COM visible class, which delegates its calls to the original class.

**C#**

**using** System;

**using** System.Runtime.InteropServices;

**using** Net4Assembly;

**namespace** Net4ToNet2Adapter

{

    [ComVisible(**true**)]

    [Guid("A6574755-925A-4E41-A01B-B6A0EEF72DF0")]

**public** **class** MyClassAdapter : IMyClassAdapter

    {

**private** MyClass \_myClass = **new** MyClass();

**public** **void** DoNet4Action()

        {

            \_myClass.DoNet4Action();

        }

    }

}

Note: we could have combined **Net4Assembly.MyClass**and **Net4ToNet2Adapter.MyClassAdapter** into the same class but I wanted to keep the example general. In real life application you often can’t change the original object and thus you are forced to create a wrapper.

## Add support to the adapter for registration-free COM activation

Important note: this part is not really necessary for the .NET 4 to .NET 2 interop to work. But without it you will need to start using he registry for registering your .NET COM components and most projects would rather to avoid it if possible. If this is not a problem just register your objects in the registry and move to the next step.

To add support for registration-free COM we need to create two application manifest files.

The first application manifest specifies dependent assemblies for the client executable. Note that since this manifest replaces the default .NET manifest I’ve added some extra standard manifest stuff (trustinfo), but only the first part is really needed for the registration-free COM to work. To add it, add a file to the client project named **app.manifest**(“Add new item” –> “Application Manifest”) and change the project properties to use this file.

Following is the content of **app.manifest**for the **Net2Aseembly.exe** client in our example:

**XML**

**<?xml** version="1.0" encoding="UTF-8" standalone="yes"**?>**

**<assembly**

  xmlns="urn:schemas-microsoft-com:asm.v1"

  manifestVersion="1.0"**>**

**<assemblyIdentity**

    type = "win32"

    name = "Net2Assembly"

    version = "1.0.0.0"

**/>**

**<dependency>**

**<dependentAssembly>**

**<assemblyIdentity**

        type="win32"

        name="Net4ToNet2Adapter"

        version="1.0.0.0" **/>**

**</dependentAssembly>**

**</dependency>**

**<trustInfo** xmlns="urn:schemas-microsoft-com:asm.v2"**>**

**<security>**

**<requestedPrivileges** xmlns="urn:schemas-microsoft-com:asm.v3"**>**

        <!-- UAC Manifest Options

            If you want to change the Windows User Account Control level replace the

            requestedExecutionLevel node with one of the following.

        <requestedExecutionLevel  level="asInvoker" uiAccess="false" />

        <requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />

        <requestedExecutionLevel  level="highestAvailable" uiAccess="false" />

            If you want to utilize File and Registry Virtualization for backward

            compatibility then delete the requestedExecutionLevel node.

        -->

**<requestedExecutionLevel** level="asInvoker" uiAccess="false" **/>**

**</requestedPrivileges>**

**</security>**

**</trustInfo>**

**</assembly>**

The second application manifest, describes the COM components which are exposed in the assembly. It needs to be set as the application manifest which resides as a native Win32 resource inside the DLL.

Unfortunately, this can’t be done as easily as the previous manifest. In **Visual Studio 2010**, the relevant field in the project   
properties is disabled when the project is of type **Class Library**. So we must go to the **Net4ToNet2Adapter.csproj**file and change it ourselves. The change is easy, just add the following lines in the relevant place:

**XML**

**<PropertyGroup>**

**<ApplicationManifest>**app.manifest**</ApplicationManifest>**

**</PropertyGroup>**

Following is the content of **app.manifest** for the **Net4ToNet2Adapter.dll** in our example:

**XML**

**<?xml** version="1.0" encoding="UTF-8" standalone="yes"**?>**

**<assembly** xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0"**>**

**<assemblyIdentity**

    type="win32"

    name="Net4ToNet2Adapter"

    version="1.0.0.0" **/>**

**<clrClass**

    clsid="{A6574755-925A-4E41-A01B-B6A0EEF72DF0}"

    progid="Net4ToNet2Adapter.MyClassAdapter"

    threadingModel="Both"

    name="Net4ToNet2Adapter.MyClassAdapter"

    runtimeVersion="v4.0.30319"

**/>**

**</assembly>**

## Use our .NET 4 DLL via COM

Now all you need to do is create an instance of your .NET 4 class from your .NET 2 executable using COM:

**C#**

**using** System;

**using** Net4ToNet2Adapter;

**namespace** Net2Assembly

{

**class** Program

    {

**static** **void** Main(**string**[] args)

        {

            Console.WriteLine("CLR version from EXE: {0}", Environment.Version);

            Type myClassAdapterType = Type.GetTypeFromProgID("Net4ToNet2Adapter.MyClassAdapter");

**object** myClassAdapterInstance = Activator.CreateInstance(myClassAdapterType);

            IMyClassAdapter myClassAdapter = (IMyClassAdapter)myClassAdapterInstance;

            myClassAdapter.DoNet4Action();

        }

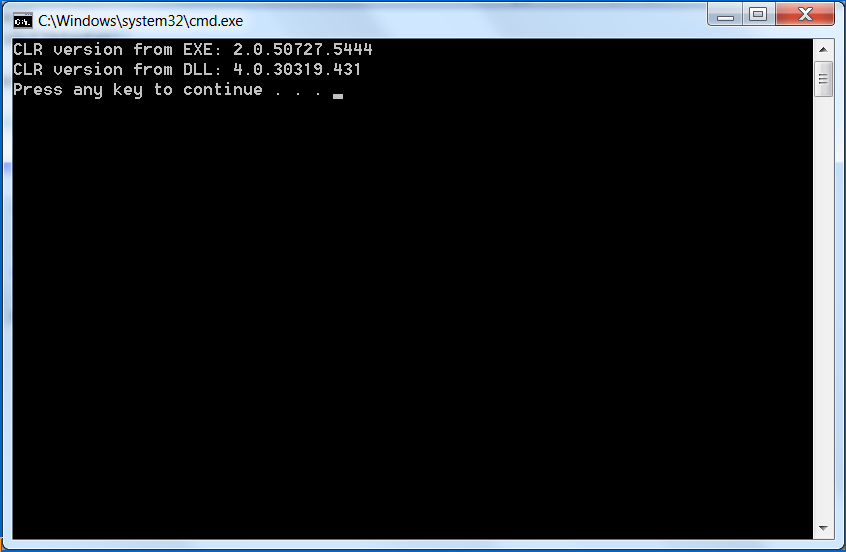
    }

}

Note: Since the interface **IMyClassAdapter** should be duplicated in the client, I’ve added the source file **IMyClassAdapter.cs**as a link to the client project.

## The result

The result of running this simple console application is:



There you go, two CLR versions in the same process!

# More Information

The relevant post on my blog:[*http://blogs.microsoft.co.il/blogs/arik*](http://blogs.microsoft.co.il/blogs/arik)

[Versions of the Common Language Runtime (CLR)](javascript:void(0))

The version number of the .NET Framework doesn't necessarily correspond to the version number of the CLR it includes. The following table shows how the two version numbers correlate.

|  |  |
| --- | --- |
| **.NET Framework version** | **Includes CLR version** |
| 1.0 | 1.0 |
| 1.1 | 1.1 |
| 2.0 | 2.0 |
| 3.0 | 2.0 |
| 3.5 | 2.0 |
| 4 | 4 |
| 4.5 (including 4.5.1 and 4.5.2) | 4 |

The CLR converts CIL ([Common Intermediate Language](http://en.wikipedia.org/wiki/Common_Intermediate_Language)) to native code:

