Visual Studio 2010

**Walkthrough: Embedding Types from Managed Assemblies (C# and Visual Basic)**

If you embed type information from a strong-named managed assembly, you can loosely couple types in an application to achieve version independence. That is, your program can be written to use types from multiple versions of a managed library without having to be recompiled for each version.

Type embedding is frequently used with COM interop, such as an application that uses automation objects from Microsoft Office. Embedding type information enables the same build of a program to work with different versions of Microsoft Office on different computers. However, you can also use type embedding with a fully managed solution.

Type information can be embedded from an assembly that has the following characteristics:

* The assembly exposes at least one public interface.
* The embedded interfaces are annotated with a **ComImport** attribute and a **Guid** attribute (and a unique GUID).
* The assembly is annotated with the **ImportedFromTypeLib** attribute or the **PrimaryInteropAssembly** attribute, and an assembly-level **Guid** attribute. (By default, Visual Basic and Visual C# project templates include an assembly-level **Guid** attribute.)

After you have specified the public interfaces that can be embedded, you can create runtime classes that implement those interfaces. A client program can then embed the type information for those interfaces at design time by referencing the assembly that contains the public interfaces and setting the **Embed Interop Types** property of the reference to **True**. This is equivalent to using the command line compiler and referencing the assembly by using the **/link** compiler option. The client program can then load instances of your runtime objects typed as those interfaces. If you create a new version of your strong-named runtime assembly, the client program does not have to be recompiled with the updated runtime assembly. Instead, the client program continues to use whichever version of the runtime assembly is available to it, using the embedded type information for the public interfaces.

Because the primary function of type embedding is to support embedding of type information from COM interop assemblies, the following limitations apply when you embed type information in a fully managed solution:

* Only attributes specific to COM interop are embedded; other attributes are ignored.
* If a type uses generic parameters and the type of the generic parameter is an embedded type, that type cannot be used across an assembly boundary. Examples of crossing an assembly boundary include calling a method from another assembly or a deriving a type from a type defined in another assembly.
* Constants are not embedded.
* The [System.Collections.Generic..::.Dictionary<(Of <(TKey, TValue>)>)](http://msdn.microsoft.com/en-us/library/xfhwa508.aspx) class does not support an embedded type as a key. You can implement your own dictionary type to support an embedded type as a key.

In this walkthrough, you will do the following:

* Create a strong-named assembly that has a public interface that contains type information that can be embedded.
* Create a strong-named runtime assembly that implements that public interface.
* Create a client program that embeds the type information from the public interface and creates an instance of the class from the runtime assembly.
* Modify and rebuild the runtime assembly.
* Run the client program to see that the new version of the runtime assembly is being used without having to recompile the client program.

|  |
| --- |
| **Description: NoteNote** |
| Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Visual Studio Settings](http://msdn.microsoft.com/en-us/library/2ewd52wf.aspx). |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating an Interface

**To create the type equivalence interface project**

1. In Visual Studio, on the **File** menu, point to **New** and then click **Project**.
2. In the **New Project** dialog box, in the **Project Types** pane, make sure that **Windows** is selected. Select **Class Library** in the **Templates** pane. In the **Name** box, type **TypeEquivalenceInterface**, and then click **OK**. The new project is created.
3. In **Solution Explorer**, right-click the Class1.vb or Class1.cs file and click **Rename**. Rename the file to **ISampleInterface.vb** or **ISampleInterface.cs** and press ENTER. Renaming the file will also rename the class to ISampleInterface. This class will represent the public interface for the class.
4. Right-click the TypeEquivalenceInterface project and click **Properties**. Click the **Compile** tab in Visual Basic or the **Build** tab in Visual C#. Set the output path to a valid location on your development computer, such as **C:\TypeEquivalenceSample**. This location will also be used in a later step in this walkthrough.
5. While still editing the project properties, click the **Signing** tab. Select the **Sign the assembly** option. In the **Choose a strong name key file** list, click **<New...>**. In the **Key file name** box, type **key.snk**. Clear the **Protect my key file with a password** check box. Click **OK**.
6. Open the ISampleInterface.vb or ISampleInterface.cs file. Add the following code to the ISampleInterface class file to create the ISampleInterface interface.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl32_ctl00_ctl00_code');" \o "Copy Code)

Imports System.Runtime.InteropServices

<ComImport()>

<Guid("8DA56996-A151-4136-B474-32784559F6DF")>

Public Interface ISampleInterface

Sub GetUserInput()

ReadOnly Property UserInput As String

...

End Interface

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl32_ctl00_ctl01_code');" \o "Copy Code)

using System;

using System.Runtime.InteropServices;

namespace TypeEquivalenceInterface

{

[ComImport]

[Guid("8DA56996-A151-4136-B474-32784559F6DF")]

public interface ISampleInterface

{

void GetUserInput();

string UserInput { get; }

...

}

}

1. On the **Tools** menu, click **Create Guid**. In the **Create GUID** dialog box, click **Registry Format** and then click **Copy**. Click **Exit**.
2. In the **Guid** attribute, delete the sample GUID and paste in the GUID that you copied from the **Create GUID** dialog box. Remove the braces ({}) from the copied GUID.
3. In Visual Basic, on the **Project** menu, click **Show All Files**. Skip this step if you are using Visual C#.
4. In **Solution Explorer**, expand the **My Project** folder if you are using Visual Basic. Expand the **Properties** folder if you are using Visual C#. Double-click the AssemblyInfo.vb or AssemblyInfo.cs file. Add the following attribute to the file.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl32_ctl00_ctl02_code');" \o "Copy Code)

<Assembly: ImportedFromTypeLib("")>

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl32_ctl00_ctl03_code');" \o "Copy Code)

[assembly: ImportedFromTypeLib("")]

Save the file.

1. Save the project.
2. Right-click the TypeEquivalenceInterface project and click **Build**. The class library .dll file is compiled and saved to the specified build output path (for example, C:\TypeEquivalenceSample).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating a Runtime Class

**To create the type equivalence runtime project**

1. In Visual Studio, on the **File** menu, point to **New** and then click **Project**.
2. In the **New Project** dialog box, in the **Project Types** pane, make sure that **Windows** is selected. Select **Class Library** in the **Templates** pane. In the **Name** box, type **TypeEquivalenceRuntime**, and then click **OK**. The new project is created.
3. In **Solution Explorer**, right-click the Class1.vb or Class1.cs file and click **Rename**. Rename the file to **SampleClass.vb** or **SampleClass.cs** and press ENTER. Renaming the file also renames the class to SampleClass. This class will implement the ISampleInterface interface.
4. Right-click the TypeEquivalenceRuntime project and click **Properties**. Click the **Compile** tab in Visual Basic or the **Build** tab in Visual C#. Set the output path to the same location you used in the TypeEquivalenceInterface project, for example, **C:\TypeEquivalenceSample**.
5. While still editing the project properties, click the **Signing** tab. Select the **Sign the assembly** option. In the **Choose a strong name key file** list, click **<New...>**. In the **Key file name** box, type **key.snk**. Clear the **Protect my key file with a password** check box. Click **OK**.
6. Right-click the TypeEquivalenceRuntime project and click **Add Reference**. Click the **Browse** tab and browse to the output path folder. Select the TypeEquivalenceInterface.dll file and click **OK**.
7. In Visual Basic, on the **Project** menu, click **Show All Files**. Skip this step if you are using Visual C#.
8. In **Solution Explorer**, expand the **References** folder. Select the TypeEquivalenceInterface reference. In the Properties window for the TypeEquivalenceInterface reference, set the **Specific Version** property to **False**.
9. Add the following code to the SampleClass class file to create the SampleClass class.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl33_ctl00_ctl00_code');" \o "Copy Code)

Imports TypeEquivalenceInterface

Public Class SampleClass

Implements ISampleInterface

Private p\_UserInput As String

Public ReadOnly Property UserInput() As String Implements ISampleInterface.UserInput

Get

Return p\_UserInput

End Get

End Property

Public Sub GetUserInput() Implements ISampleInterface.GetUserInput

Console.WriteLine("Please enter a value:")

p\_UserInput = Console.ReadLine()

End Sub

...

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl33_ctl00_ctl01_code');" \o "Copy Code)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using TypeEquivalenceInterface;

namespace TypeEquivalenceRuntime

{

public class SampleClass : ISampleInterface

{

private string p\_UserInput;

public string UserInput { get { return p\_UserInput; } }

public void GetUserInput()

{

Console.WriteLine("Please enter a value:");

p\_UserInput = Console.ReadLine();

}

...

}

}

1. Save the project.
2. Right-click the TypeEquivalenceRuntime project and click **Build**. The class library .dll file is compiled and saved to the specified build output path (for example, C:\TypeEquivalenceSample).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating a Client Project

**To create the type equivalence client project**

1. In Visual Studio, on the **File** menu, point to **New** and then click **Project**.
2. In the **New Project** dialog box, in the **Project Types** pane, make sure that **Windows** is selected. Select **Console Application** in the **Templates** pane. In the **Name** box, type **TypeEquivalenceClient**, and then click **OK**. The new project is created.
3. Right-click the TypeEquivalenceClient project and click **Properties**. Click the **Compile** tab in Visual Basic or the **Build** tab in Visual C#. Set the output path to the same location you used in the TypeEquivalenceInterface project, for example, **C:\TypeEquivalenceSample**.
4. Right-click the TypeEquivalenceClient project and click **Add Reference**. Click the **Browse** tab and browse to the output path folder. Select the TypeEquivalenceInterface.dll file (not the TypeEquivalenceRuntime.dll) and click **OK**.
5. In Visual Basic, on the **Project** menu, click **Show All Files**. Skip this step if you are using Visual C#.
6. In **Solution Explorer**, expand the **References** folder. Select the TypeEquivalenceInterface reference. In the Properties window for the TypeEquivalenceInterface reference, set the **Embed Interop Types** property to **True**.
7. Add the following code to the Module1.vb or Program.cs file to create the client program.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl34_ctl00_ctl00_code');" \o "Copy Code)

Imports TypeEquivalenceInterface

Imports System.Reflection

Module Module1

Sub Main()

Dim sampleAssembly = Assembly.Load("TypeEquivalenceRuntime")

Dim sampleClass As ISampleInterface = CType( \_

sampleAssembly.CreateInstance("TypeEquivalenceRuntime.SampleClass"), ISampleInterface)

sampleClass.GetUserInput()

Console.WriteLine(sampleClass.UserInput)

Console.WriteLine(sampleAssembly.GetName().Version)

Console.ReadLine()

End Sub

End Module

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl34_ctl00_ctl01_code');" \o "Copy Code)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using TypeEquivalenceInterface;

using System.Reflection;

namespace TypeEquivalenceClient

{

class Program

{

static void Main(string[] args)

{

Assembly sampleAssembly = Assembly.Load("TypeEquivalenceRuntime");

ISampleInterface sampleClass =

(ISampleInterface)sampleAssembly.CreateInstance("TypeEquivalenceRuntime.SampleClass");

sampleClass.GetUserInput();

Console.WriteLine(sampleClass.UserInput);

Console.WriteLine(sampleAssembly.GetName().Version.ToString());

Console.ReadLine();

}

}

}

1. Press CTRL+F5 to build and run the program.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifModifying the Interface

**To modify the interface**

1. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, right-click the TypeEquivalenceInterface project, and then click **Properties**. Click the **Application** tab. Click the **Assembly Information** button. Change the **Assembly Version** and **File Version** values to **2.0.0.0**.
3. Open the ISampleInterface.vb or ISampleInterface.cs file. Add the following line of code to the ISampleInterface interface.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl35_ctl00_ctl00_code');" \o "Copy Code)

Function GetDate() As Date

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl35_ctl00_ctl01_code');" \o "Copy Code)

DateTime GetDate();

Save the file.

1. Save the project.
2. Right-click the TypeEquivalenceInterface project and click **Build**. A new version of the class library .dll file is compiled and saved in the specified build output path (for example, C:\TypeEquivalenceSample).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifModifying the Runtime Class

**To modify the runtime class**

1. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, right-click the TypeEquivalenceRuntime project and click **Properties**. Click the **Application** tab. Click the **Assembly Information** button. Change the **Assembly Version** and **File Version** values to **2.0.0.0**.
3. Open the SampleClass.vb or SampleClass.cs file. Add the following lines of code to the SampleClass class.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl36_ctl00_ctl00_code');" \o "Copy Code)

Public Function GetDate() As DateTime Implements ISampleInterface.GetDate

Return Now

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl36_ctl00_ctl01_code');" \o "Copy Code)

public DateTime GetDate()

{

return DateTime.Now;

}

Save the file.

1. Save the project.
2. Right-click the TypeEquivalenceRuntime project and click **Build**. An updated version of the class library .dll file is compiled and saved in the previously specified build output path (for example, C:\TypeEquivalenceSample).
3. In Windows Explorer, open the output path folder (for example, C:\TypeEquivalenceSample). Double-click the TypeEquivalenceClient.exe to run the program. The program will reflect the new version of the TypeEquivalenceRuntime assembly without having been recompiled.