Team Developer and .NET

Unify Corporation

Table of Contents

Abstract

Within the next few years, most shops will run a lot of Team Developer code along with a growing amount of Microsoft .NET code. You can make these two code bases interoperate as well as possible, using the .NET runtime's interoperability services. These let you write .NET (like VB.NET) applications that use existing Team Developer COM Objects, and they allow you write Team Developer apps that use DLLs created with VB.NET. You can create .NET applications that leverage your existing investment in COM components, and you can migrate a Team Developer DLL to VB.NET—and still access its functionality from Team Developer apps.

Performance Implications

I will introduce some key concepts about how the system marshals data between COM and .NET platforms, and how that can impact performance. When code on either platform calls a method on an object running on the other platform, the method call requires marshalling. In marshalling, the .NET Interop services package up the method call, transfer it to the other platform, invoke the method, package up the results - then transfer them back to your original platform (see Figure 1). This causes a performance hit as the marshalling process moves the parameters from one platform to the other.

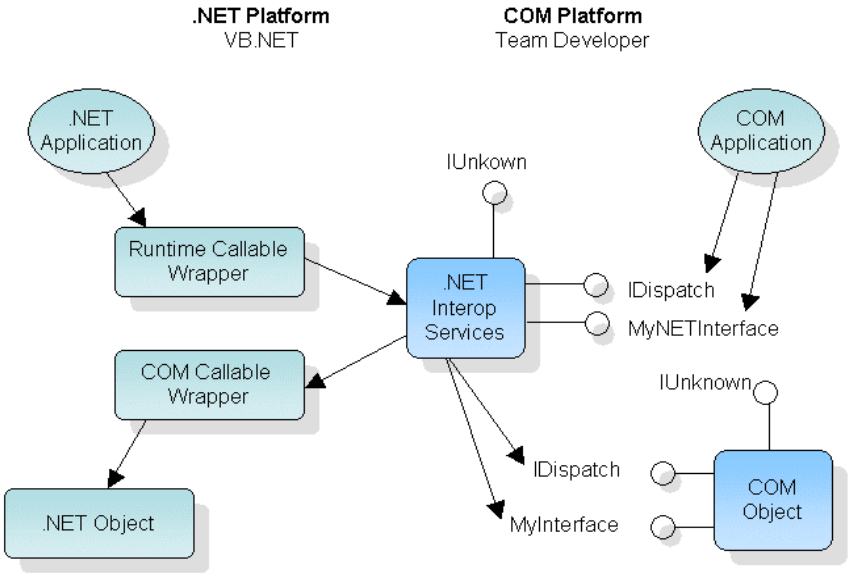

Figure 1

This performance hit will vary depending on the data types of your parameters and result types. Performance is best when you're moving around simple numeric data, such as 32-bit integers. These are *isomorphic* data types, which are represented in memory the same way on both platforms. Consequently, Interop with them proceeds almost as quickly as regular method calls. You can simply copy, isomorphic data from COM memory to .NET memory (or vice versa).

The Team Developer isomorphic data types are the Number and String. One-dimensional arrays of isomorphic types are isomorphic, as long as they comprise only isomorphic types.

Other types of data require more work during the marshalling process because they aren't represented the same in .NET and COM. Data types such as Boolean are *non-isomorphic* and might require conversion from one format to another—and some extra cycles—as they're copied from one platform to another.

Marshalling

The .NET Interop services provide four types of marshalling. Of these, only Type I marshalling is supported automatically by Interop services and Visual Studio .NET (VS.NET). If you need to convert from one data type to another, pass complex data as a single value (such as a Variant) or represent one type of object as a different object. You'll need to implement one of the other types of marshalling by hand by editing the Microsoft Intermediate Language (MSIL) code in Notepad, among other things.

As long as you use data types supported by Type I marshalling, calling a Team Developer COM Object from a VB.NET application and vice versa is speedy and virtually transparent. So avoid using the Variant data type if you can, because that's typically where you run into issues. I'll show you how to use Type I marshalling because you'll most likely use this type in your applications.

Differences Between .NET Framework and COM Framework

The .NET framework object model and its workings are different from Component Object Model (COM) and its workings. For example, clients of .NET components don't have to worry about the lifetime of the object. Common Language Runtime (CLR) manages things for them. In contrast, clients of COM objects must take care of the lifetime of the object. Similarly, .NET objects live in the memory space that is managed by CLR. CLR can move objects around in the memory for performance reasons and update the references of objects accordingly, but COM object clients have the actual address of the object and depend on the object to stay on the same memory location.

.NET runtime provides many new features and constructs to managed components. For example, .NET components can have parameterized constructors, functions of the components can have accessibility attributes (like public, protected, internal, and others) associated with them, and components can also have static methods. Apart from these features, there are many others. These include those that are not accessible to COM clients because standard implementation of COM does not recognize these features. Therefore .NET runtime must put something in between the two, .NET server and COM client, to act as mediator.

The .NET And COM Mediator

The .NET runtime provides COM Interoperability wrappers for overcoming the differences between .NET and COM environments. For example, runtime creates an instance of COM Callable Wrapper (CCW) when a COM client accesses a .NET component. In the same way, an instance of Runtime Callable Wrapper (RCW) is created when a .NET client accesses a COM component. These wrappers abstract the differences and provide the seamless integration between the two environments. **Figure 2** illustrates CCW and RCW.

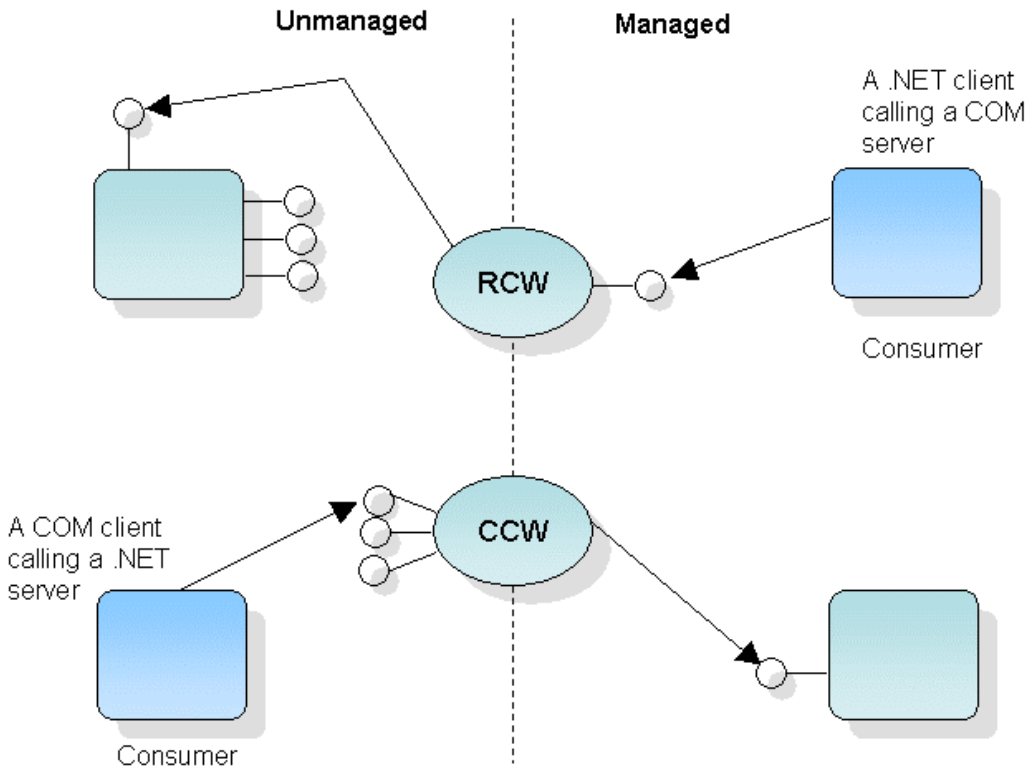

Figure 2

PART I – Consuming Team Developer Code from VB .NET

VS.NET lets you reference and use COM DLLs much as you reference and use .NET DLLs. Start by creating a simple Team Developer COM DLL, as illustrated in **Figure 3**.

Figure 3

When you build the project, you get a simple DLL that returns a text value. You can use this DLL transparently from .NET because Type I marshalling supports both data types.

Before you can use any DLL in VB.NET, you need to add a reference to it in your project. This holds true for both .NET DLLs and COM DLLs, and VS.NET allows you to add references to both types of DLLs from the same dialog.

In VS.NET, use the Project | Add Reference menu option to bring up the Add Reference dialog. Click on the COM tab, which gives you a list of all the COM components registered currently on your system. Scroll down and select the COMObject entry (see **Figure 4**).

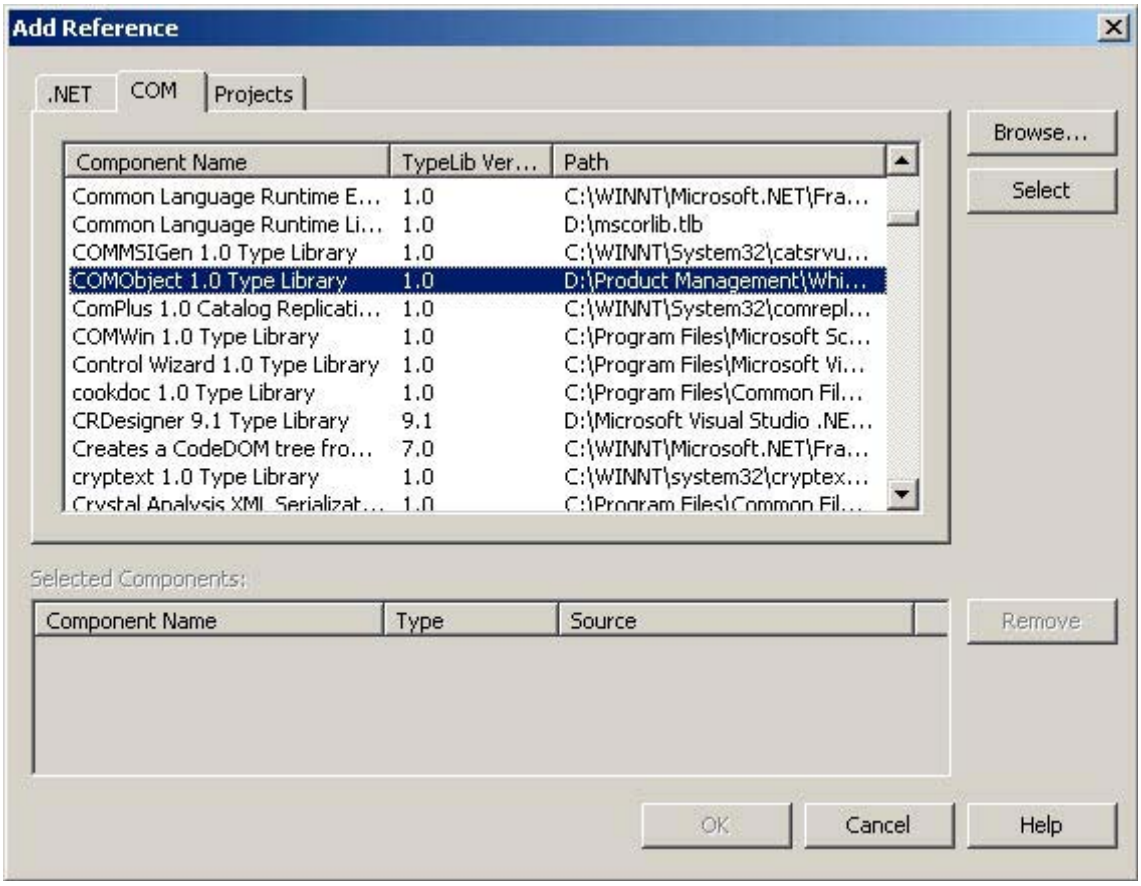

Figure 4

VS.NET creates a *runtime callable wrapper* (RCW) assembly automatically when you click on OK. This important step occurs behind the scenes. The RCW is a .NET assembly that your .NET code sees much like a COM DLL. When you write code that calls the COMObject component, your code is really calling the RCW, which then calls the underlying COM component on your behalf. The RCW takes care of the marshalling process, making the actual call from the .NET platform over to the COM platform. It can also let you share COM components among many .NET apps. Add a TextBox control and a Button control to the form, then double-click on the Button control to bring up the code window. Add this code within the Click event:

```
Dim obj As New COMObject.COMObjectClass()
TextBox1.Text = obj.GetString
Marshal.ReleaseComObject(obj)
```

This code creates an instance of the COMObject component, then calls the method on that object to populate the display. The code then calls Marshal.ReleaseComObject to release the reference to the underlying COM object. You do the same thing when you call Object.Release() in Team Developer because

COM objects are destroyed when they're released, and they expect to be released as soon as you're done using them.

On the other hand, objects in VB.NET aren't destroyed until the .NET garbage collection algorithm gets around to it. By calling ReleaseComObject, you force .NET to more closely simulate the behaviour of COM—always a good idea when working with COM objects.

Before this code will work, you need to add an Imports statement to the top of the file:

```
Imports System.Runtime.InteropServices
```

This provides the namespace where the Marshal class resides.
Now you can run the application and click on the button. The form should then display the data retrieved from the Team Developer COM DLL (see **Figure 5**).

Figure 5

PART II – Consuming a VB .NET Assembly from Team Developer

You'll find it nearly as easy to reverse the process. Of course, without your intervention, a COM-based client—such as a Team Developer app—won't be able to use a VB.NET DLL. This takes only a few steps to accomplish, though.

In VS.NET, create a new Class Library project named ManagedClass. Remove the template code from the Class1.vb file and write this code, and then rename Class1.vb to ManagedClass.vb:

```
<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)> _
Public Interface INETInterface
    Function GetString() As String
End Interface

Public Class NETClass
    Implements INETInterface
    Function GetString() As String _
    Implements INETInterface.GetString
        Return "Congratulations! You just accessed .NET Code."
    End Function
End Class
```

This code provides the same functionality as the COM DLL you created earlier to make it available to Team Developer. Use the <InterfaceTypeAttribute()> attribute to mark the managed INETInterface as **IDispatch** only when exposed to COM.

Of course, COM relies on globally unique identifiers (GUIDs) to identify your class. If you don't specify them, .NET will create some for you automatically.

You must mark the project itself so VS.NET registers the DLL with COM. Right-click on the ManagedClass project in the Solution Explorer and select Properties. Click on Configuration Properties and Build in the left-hand pane, and check the option named Register for COM Interop (see **Figure 6**).

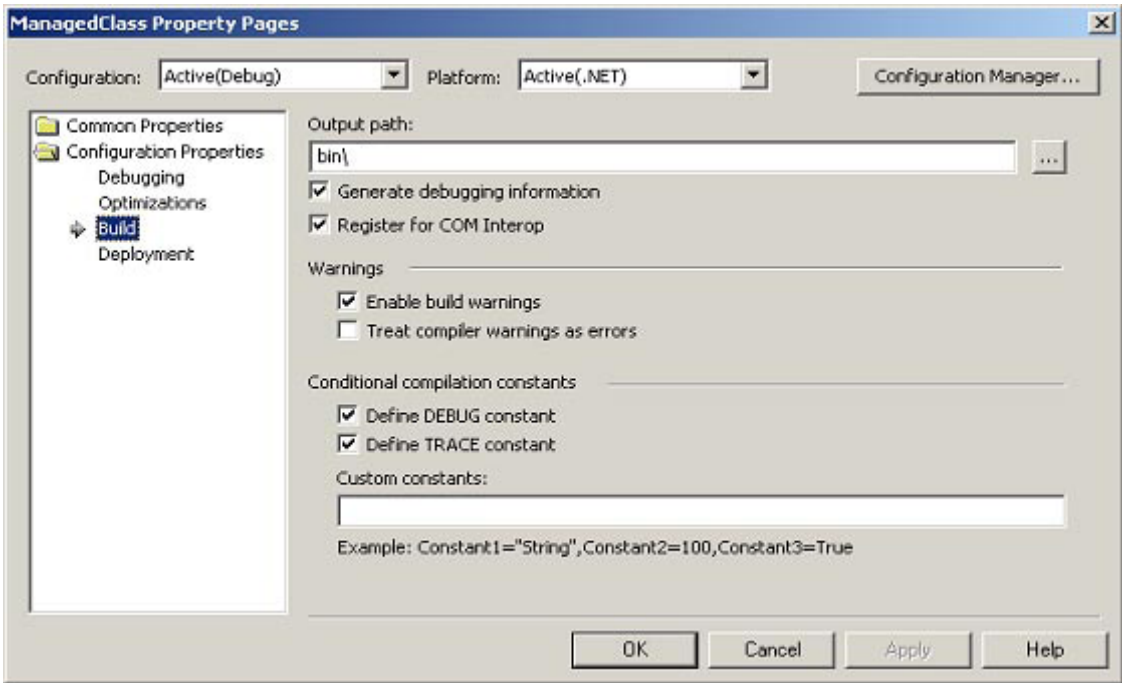

Figure 6

With this option checked, VS.NET creates a COM-*callable wrapper* (CCW) and a COM type library for your class the next time you build the solution. VS.NET also

registers the DLL in the Windows Registry, making it available to Team Developer. All this goes on under the hood—the only visible indication is a line in the Output window.

The CCW resembles the RCW that VS.NET created for you earlier. This assembly handles the marshalling and all the details that let a COM client app invoke your VB.NET object. Build the project to create the DLL, then open the Team Developer IDE.

Save the project as .NETClient.apt. Open the ActiveX Explorer and locate the ManagedClass component. You can generate the SAL proxy just as you would for any COM component (see Figure 7).

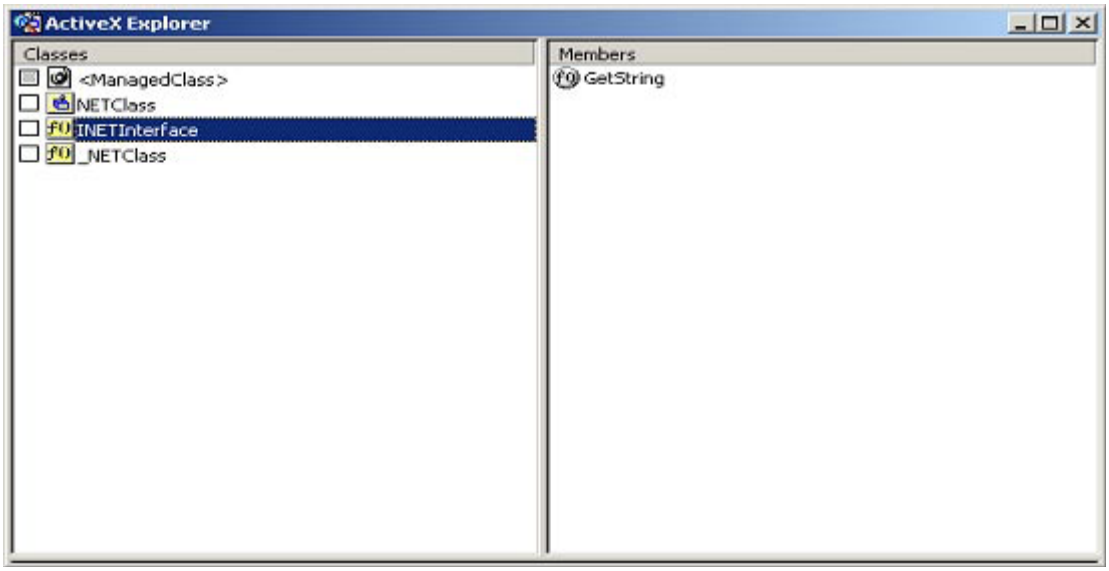

Figure 7

Now add the Common Language Runtime Library.apl to the Libraries section. Create a new Form Window and add a Data Field and Pushbutton to the Form Window. Create an instance of INETInterface and add the following code to the Pushbutton:

```
On SAM_Click
  If myInterface.CreateObject( "ManagedClass.NETClass" )
    Call myInterface.GetString( sNETString )
    Set dfNET = sNETString
```

Finally, run the app and click on the button. The display should show the value retrieved from the VB.NET assembly (see **Figure 8**).

Figure 8

Conclusion

This is what we need to do to access .NET components from unmanaged code. If you want to develop some of your application components in .NET, you should consider that many of the features of .NET components are not exposed to COM clients. This will require some design-time decisions and may affect the way you design your object hierarchies (because inheritance hierarchy of .NET components is flattened for COM clients).

.NET's Interop services deliver a useful and usually simple way to reuse existing Team Developer COM Objects from VB.NET, and to use VB.NET DLLs from Team Developer as you migrate your components over time. In most cases, Interop services do all the hard work for you, and VS.NET often automates the process. If you're creating multithreaded .NET apps or passing complex data types through Variants, things can get complex and you'll need to do more work by hand.

Requirements

- SQLWindows 2.1 or above
- The included Common Language Runtime Library.apl
- Visual Studio .NET

About Unify

Unify is a global provider of software development technology and solutions that helps IT customers participate in Service-Oriented Architecture (SOA). Unify's productive and re-liable development tools, migration solutions and databases enable organizations to build and modernize business essential applications for SOA. Composer for Lotus Notes offers a complete, like for like, production to production migration solution for Lotus Notes applications. Unify's award-winning NXJ Developer enables IT teams to be extremely productive, learn new technologies fast and deliver Web services-based applications on time and on budget. The Team Developer, SQLBase, DataServer, ACCELL and VISION product families enable cross-platform rapid development on Java/J2EE, Linux or Windows. Unify has a rich heritage in delivering rich, cost-effective technologies to its thousands of IT customers and ISV, VAR and distributor partners. Unify is headquartered in Sacramento, Calif., and can be reached at (916) 928-6400 or by visiting www.unify.com.

Unify Corporation
2101 Arena Blvd., Suite 100
Sacramento, CA 95834
USA
Phone: 1.916.928.6400
Toll Free: 1.800.468.6439
Fax: 1.916.928.6404
Munich: +49 8 115 55430
United Kingdom: +44 (0)1753 245 510
France: +33 (0)1 34 58 28 30