Issue	August 2002
Section	Columns
Main file name	lowyt1.rtf
Listing file name	
Sidebar file name	
Table file name	
Screen capture file names	lowyf1.bmp
Infographic/illustration file names	
Photos or book scans	
Special instructions for Art dept.	
Editor	SP
Spellchecked (set Language to English U.S.)	
PM review	
Character count	
Package length	
ToC blurb	

Overline:

Q & A

Byline:

Technology Toolbox:
VB.NET
C#
SQL Server 2000
ASP.NET
XML
VB6

Deck:

Q.

Sign Interop Assembly

I have a legacy COM DLL I need to use from my .NET assembly. I added a reference to it using the Visual Studio .NET Add Reference dialog wizard and the COM tab. VS.NET creates an interoperation assembly and adds it to the project. It all works fine until I assign a strong name to my assembly. When I sign my assembly, VS.NET refuses to compile it, giving the error message: "Assembly generation failed—Referenced assembly does not have a strong name." What can I do?

A.

The problem you encounter isn't specific to interop assemblies. An assembly with only a friendly name can add a reference to any strongly-named assembly. For example, all the .NET Framework classes reside in strongly-named assemblies, and every user assembly can use them freely. However, the reverse isn't true: A strongly-named assembly can only reference and use other strongly-named assemblies. The compiler refuses to build and assign a strong name to any assembly that uses types defined in assemblies equipped with only a friendly name.

The reason is clear: A strong name implies the client can trust the assembly, regarding authenticity and integrity of the service provider.

The client also assumes when referencing an assembly with a strong name that the client will always get this exact version or a compatible one. This cycle of trust is breached if the strongly-named assembly can use other assemblies with potentially dubious origins and unverifiable versions. So, strongly-named assemblies can only reference other strongly-named assemblies.

Using the VS.NET wizard to generate an interop assembly generates an interop assembly with a friendly name only. In fact, the wizard is a mere UI wrapper around a command-line utility called TlbImp. TlbImp reads the COM type library and generates an interop assembly containing metadata about the COM types, and .NET is responsible at runtime for bridging the two component technologies transparently from the .NET client perspective. If the .NET client is a strongly-named assembly, then it can't use that interop assembly. To overcome this, you have to call TlbImp directly, using the /keyfile: command line switch to specify a key filename to assign a strong name to the interop assembly. For example, say the COM definitions are in MyServer.tlb, your strong name key file name is MyKey.snk, and you want to call the strongly-named interop assembly MyServerInterop.dll. You'd use TlbImp this way:

```
tlbimp MyServer.tlb
  /out: MyServerInterop.dll

  /keyfile: MyKey.snk
```

Next, add a reference manually to MyServerInterop.dll in the VS.NET project, this time using the .NET tab, and you're good to go.—*J.L.*

C# projects have built-in support for preparing an interop assembly and signing it at the same time. Bring up the C# Project properties dialog. Under Common Properties | General is the "Wrapper Assembly Key File" item (see Figure 2) . If you specify there the path and name of the file containing your keys, Visual Studio.NET will use those keys to automatically sign the interop assembly using the keys. Unfortunately, Visual Basic.NET have no such support in Visual Studio.NET. If your strongly named client assembly is developed in Visual Basic.NET, you must use the *TLBImp* command line utility, and the /keyfile as just shown.

Q.

Treating Warnings as Errors

In Visual C++ 6.0, I could instruct the compiler to treat compilation warnings as errors, and refuse to build in case of warnings. This was an essential part of coding standard. How do you achieve the same functionality in VS.NET?

A.

Treating warnings as errors is indeed intrinsic to productive development. Your code should compile flawlessly, and if the compiler complains about something, you'd better attend to it. By default, VS.NET lets you build and ship with warnings. Fortunately, you can instruct VS.NET to treat warnings as errors, and by doing do, the compiler won't build your assembly, even if it encounters only warnings. Both C# and Visual Basic.NET projects can take advantage of this feature, although the configuration dialog box is different. In a C# project, right-click on the project in the Solution Explorer, and select Properties. This brings up the Project Properties page, which is as close as you get to your old Visual C++ project settings. Expand Configuration Properties, and select Build (see Figure 1).

Setting the Treat Warnings As Errors drop-down box to true causes the compiler to treat warnings as errors. Note that you can set different project configuration settings for debug or release builds, by selecting a build target from the Configuration box. While you have the dialog open, verify that the warning level is set to Warning Level 4, to get as much warning (implicit help from the compiler) as possible. Always compile your code at level 4. You can treat warnings as errors in a VB.NET project by bringing up the Project Properties page, expanding Configuration Properties, selecting Build, and checking the "Treat compiler warnings as errors" checkbox.—*J.L.*

About the Author:

Juval Löwy is the principal of IDesign, a consulting and training company focused on .NET design and migration. Juval is the author of *COM and .NET Component Services* (O'Reilly), and he speaks frequently at major conferences. Juval chairs the program committee of the .NET California Bay Area User Group. Contact him at www.idesign.net

Captions:

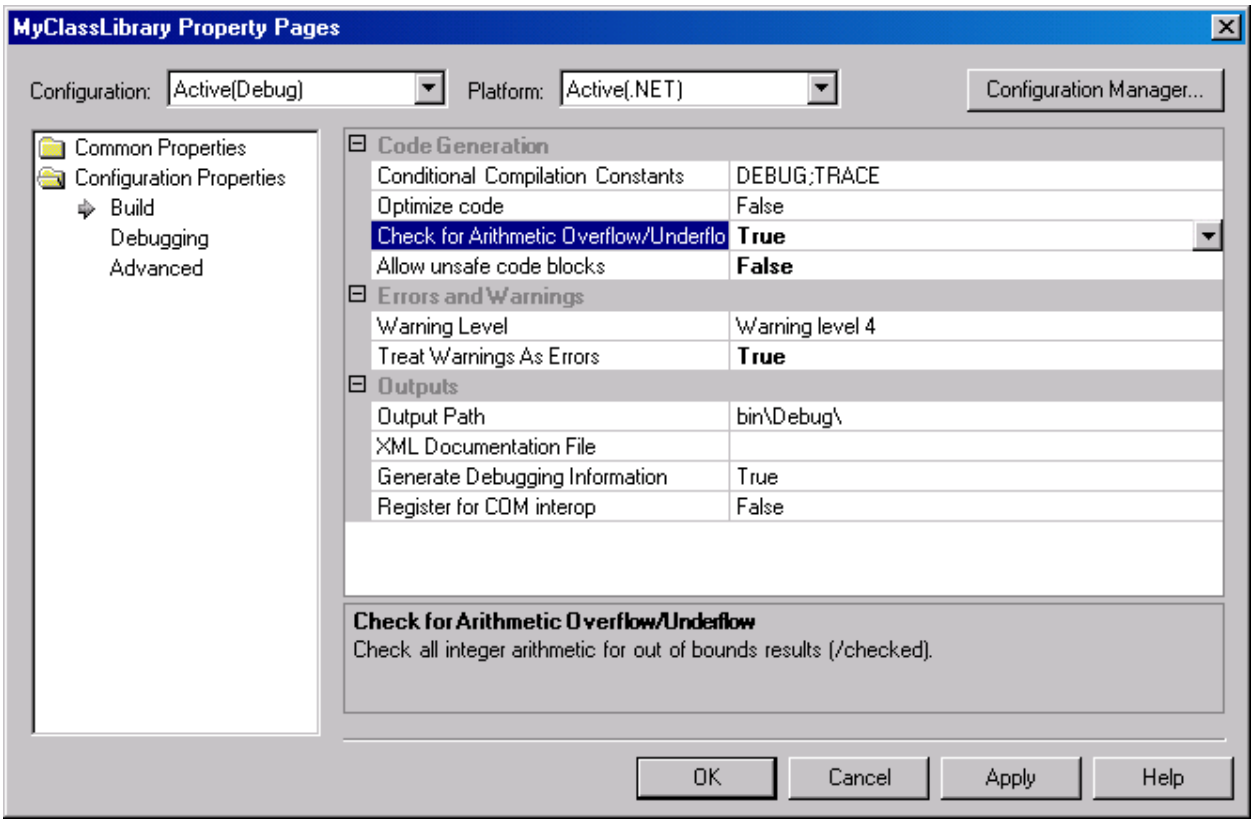

Figure 1

Configure Project Settings. The Properties page lets you configure project settings, including warning level, unsafe code support, unchecked code support, and treating warnings as errors. You can configure different settings for Debug and Release builds.

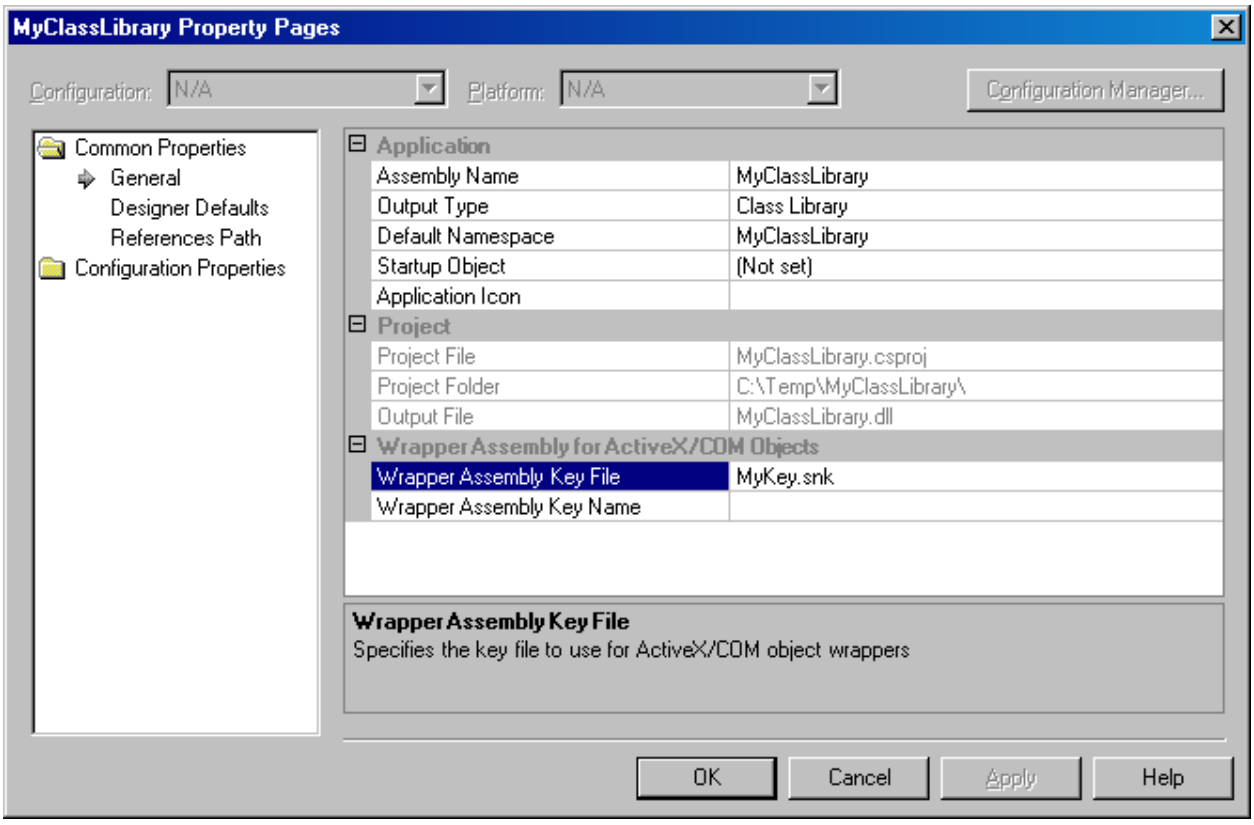

Figure 2

Signing an Interop Assembly. In C# projects, you can provide the name of a file containing your strong name keys (typically the file used to sign your assembly). VS.NET will automatically sign with these keys any COM interop assembly you import.