

# VISUAL STUDIO HACKS™

*Tips & Tools for Turbocharging the IDE*



O'REILLY®

*James Avery*

HACK  
#64

## Examine the Innards of Assemblies

ILDASM is great for digging into assemblies, but Reflector is a powerful, free tool that gives you even more.

When you compile your source code in Visual Studio, the compiler translates the high-level source code, not into machine-specific instructions, but into an intermediate language known as Microsoft Intermediate Language (MSIL). This Intermediate Language (IL), along with additional security, versioning, sharing, and other related metadata, is packaged into one or more DLLs or executable files. The complete package is referred to as an assembly. As you saw in “Examine the IL Generated by Your Code” [Hack #63], there are free tools that can examine the IL of an assembly.

While examining the IL of an assembly can be useful at times, it requires familiarity with MSIL. More often than not, the average developer is much more comfortable with a high-level programming language like C# or Visual Basic rather than IL. Fortunately, a free tool called Reflector can translate the intermediate language of a .NET assembly into either C# or Visual Basic code. In addition to converting IL to C# or Visual Basic code, Reflector provides an outline of the assembly’s classes and its members, the ability to view the IL for an assembly, and support for third-party add-ins.

### Download and Run Reflector

Reflector is a free program created by Lutz Roeder, a Microsoft employee. It is one of those essentials that every serious .NET developer should have in her toolbox. Reflector is updated frequently; the latest version is available at <http://www.aisto.com/roeder/dotnet>. At the time of this writing, when you download Reflector, you download a zip file containing just two files: *Reflector.exe* and *ReadMe.htm*. After unzipping these two files to some directory, you can run Reflector by simply double-clicking the *Reflector.exe* file.

By default, Reflector opens a handful of common assemblies: *mscorlib*, *System*, *System.Data*, *System.Drawing*, and so on. Each opened assembly is listed in Reflector’s main window (see Figure 7-23). Clicking on the + icon next to an assembly will expand the tree, showing the assembly’s namespaces. Each namespace has a corresponding + icon next to it as well that, when clicked, will show the namespace’s classes. Additionally, each class can be expanded to show the class’s members—its events, fields, methods, and properties.

To view the details of other assemblies, such as assemblies you’ve created, go to the File menu and choose Open. Next, browse to the assembly you

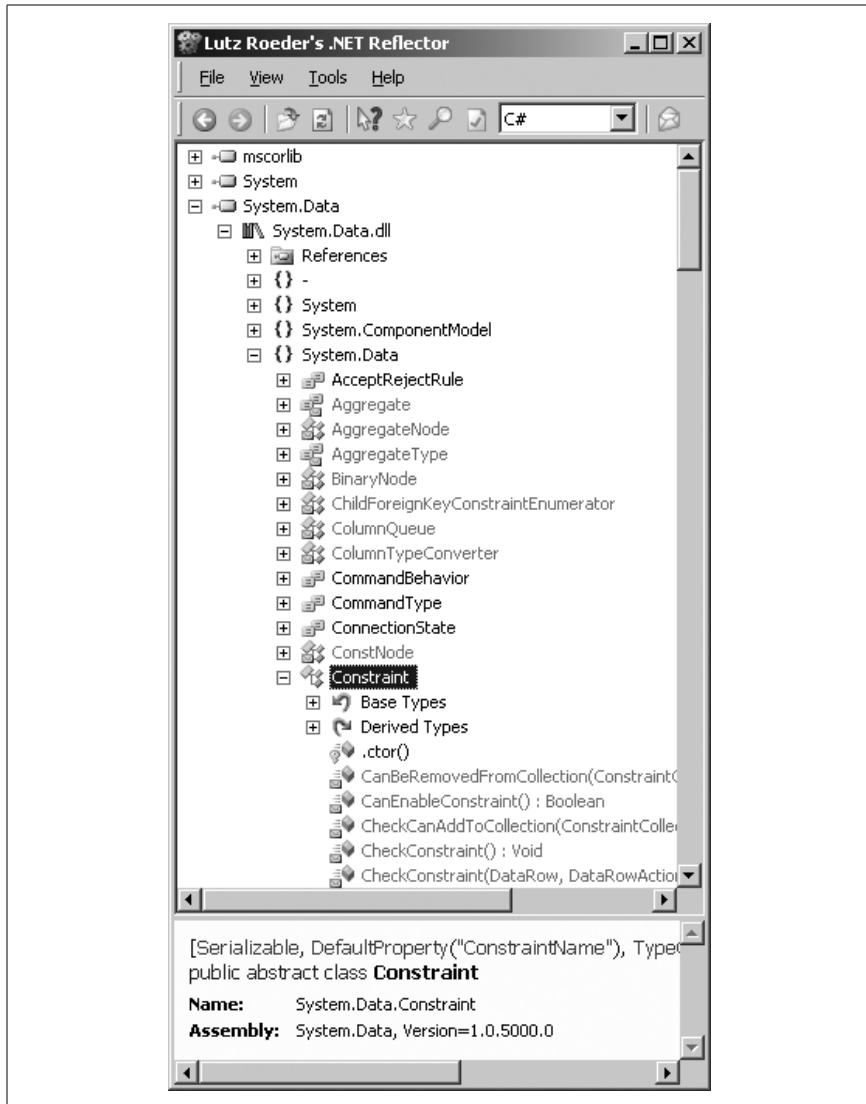


Figure 7-23. Reflector main window

want to view. Once you have selected a valid .NET assembly, the assembly will be displayed in Reflector's main window along with the default assemblies. To remove an assembly from Reflector's main window, right-click on the assembly and choose Close.

## Disassemble Assemblies with Reflector

While being able to browse through assemblies, namespaces, and classes is handy, Reflector's true usefulness shines through in its disassembling capabilities. Once you have drilled down to a class-level member, you can disassemble the class-level member by going to the Tools menu and choosing Disassembler. This will open up a second pane, showing the disassembled content in either C#, Visual Basic, Delphi, or IL. (You can specify what language the disassembled output should be shown in through the View → Options dialog or via the drop-down list in the toolbar.) Figure 7-24 shows the disassembled contents of the DataSet class's GetXml() method in C#.

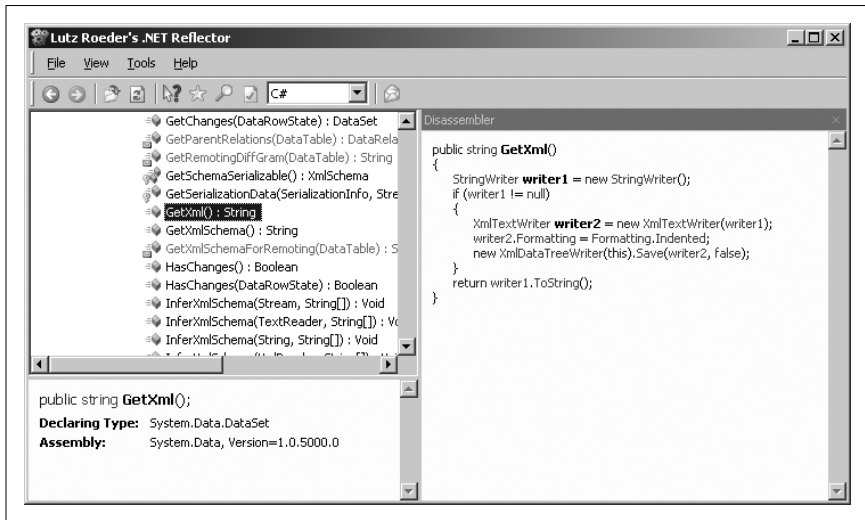


Figure 7-24. Disassemble methods in your language of choice with Reflector

With its disassembling capabilities, Reflector makes it easy to investigate the guts of the .NET Framework Base Class Library. You can also examine the source code of assemblies that you have created or are using but don't have the original source code for.



Seeing that Reflector can provide source code view of an assembly might give you pause if your company generates its revenue from selling a .NET application. The last thing you'd likely be interested in is having your customers—or worse, your competitors—examining the source code of the application. There's no way to prevent anyone from viewing the IL of a .NET assembly (and thereby disassembling it to C# or Visual Basic), but you can *obfuscate* the IL. Obfuscation is the process of changing the IL so that it still behaves as intended, but is illegible and unreadable for a human. A number of .NET obfuscation products—such as: PreEmptive Solution's Dotfuscator [Hack #80], WiseOwl's Demeanor, and Remotesoft's .NET Obfuscator—are available.

## Additional Reflector Features

In addition to serving as an object browser and disassembler, Reflector can display call and callee graphs for class and class members, offer one-click access to search Google or MSDN, and provide a framework that allows third-party developers to create add-ins for Reflector.

To view the call or callee graphs, simply select a member in the tree view, go to the Tools menu, and select the Call Graph or Callee Graph option. The Call Graph lists the members called by the selected item, whereas the Callee Graph lists those members that call the selected item. For example, as Figure 7-25 shows, the `ArrayList` class's `Clone()` method calls the `System.Array.Copy()` method and the `ArrayList`'s constructor (as it created a new `ArrayList` instance), and works with the `ArrayList`'s `_items`, `_size`, and `_version` private member variables.

The callee graph is the inverse of the call graph. It shows those members that call the selected item. For example, the `ArrayList`'s `Clone()` method's callee graph shows that the `System.NET.SocketPermission` class's `Copy()` method and the `System.Xml.XPath.XsltFunction` class's `Clone()` members, among others, call the `ArrayList`'s `Clone()` method.

Reflector's functionality can be further extended through the use of add-ins. There are add-ins for displaying assembly dependency graphs, for automatically loading the currently running assembly, for outputting the disassembled contents of an entire assembly, and for hosting Reflector within Visual Studio. These add-ins, and more, are listed at <http://www.freewebs.com/csharp/Reflector/AddIns> and are all worth checking out.

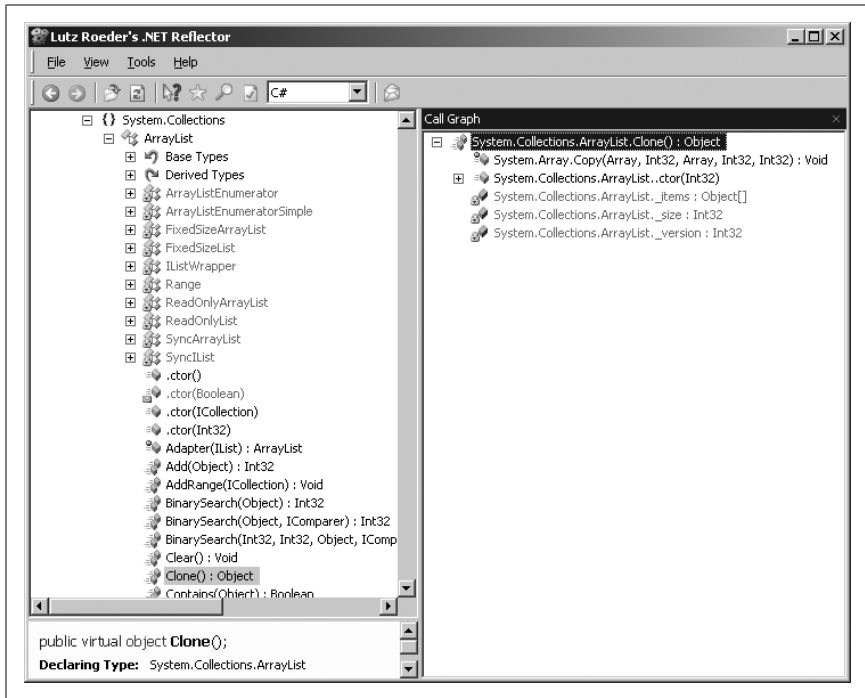


Figure 7-25. Call Graph shows members called by selected item

## Using Reflector Within Visual Studio

Of particular interest is the Reflector.VisualStudio Add-In. This add-in, created by Jaime Cansdale, allows for Reflector to be hosted within Visual Studio. With this add-in, you can have Reflector integrated within the Visual Studio environment. To get started, you will need to have the latest version of Reflector on your machine. Once you have downloaded Reflector, download the latest version of the Reflector.VisualStudio Add-In from <http://www.testdriven.NET/reflector>. The download contains a number of files that need to be placed in the same directory as `Reflector.exe`. To install the add-in, drop to the command line and run:

```
Reflector.VisualStudio.exe /install
```

After the add-in has been installed, you can start using Reflector from Visual Studio. You'll notice a new menu item, `Addins`, which has a menu option titled `Reflector`. This option, when selected, displays the Reflector window, which can be docked in the IDE (see Figure 7-26). Additionally, the add-in provides context menu support. When you right-click in an open code file in Visual Studio, you'll see a `Reflector` menu item that expands into a submenu with options to disassemble the code into `C#` or `Visual Basic`, display the call

## Examine the Innards of Assemblies

graph or callee graph, and other related choices. The context menu also includes a Synchronize with Reflector menu item that, when clicked, syncs the object browser tree in the Reflector window with the current code file.

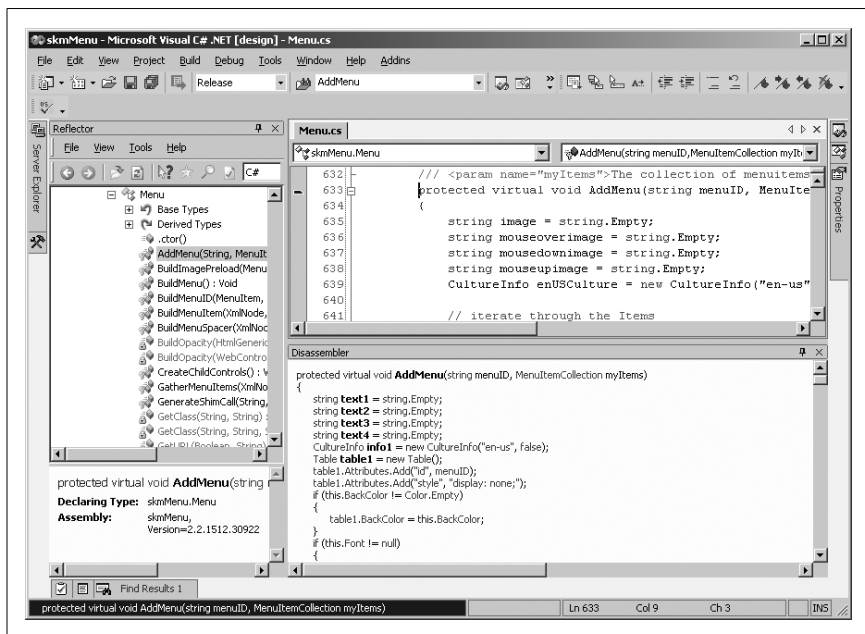


Figure 7-26. With the Reflector.VisualStudio Add-In, Reflector can be hosted in Visual Studio

Reflector is an object browser, disassembler, and so much more, all wrapped up into one program that can be hosted through Visual Studio. Reflector is useful for inspecting the source code of the .NET Framework's Base Class Library, as well as a helpful tool for inspecting your own assemblies. With its bevy of features and add-ins, Reflector is an indispensable tool that every .NET developer should know of and use.

—Scott Mitchell