1. .NET Reflector 8 documentation . . . 2
- 1.1 Using the .NET Reflector installer . . . 3
  - 1.1.1 Downloading .NET Reflector as a .zip file . . . 8
- 1.2 Working with .NET Reflector . . . 9
  - 1.2.1 .NET Reflector tips - Keyboard shortcuts . . . 10
  - 1.2.2 Using the .NET Reflector Power Commands . . . 12
- 1.3 Worked examples . . . 14
  - 1.3.1 Debugging a SharePoint customization . . . 15
  - 1.3.2 Debugging into SharePoint and seeing the locals . . . 17
  - 1.3.3 Introduction to building .NET Reflector add-ins . . . 24
  - 1.3.4 Exporting source code . . . 26
- 1.4 Troubleshooting . . . 27
  - 1.4.1 Error messages . . . 28
    - 1.4.1.1 The certificate for a digital signature in this extension is not valid . . . 29
- 1.5 Release notes and other versions . . . 30
  - 1.5.1 .NET Reflector 8.5 release notes . . . 31
  - 1.5.2 .NET Reflector 8.4 release notes . . . 32
  - 1.5.3 .NET Reflector 8.3 release notes . . . 33
  - 1.5.4 .NET Reflector 8.2 release notes . . . 34
  - 1.5.5 .NET Reflector 8.1 release notes . . . 35
  - 1.5.6 .NET Reflector 8.0 release notes . . . 36
  - 1.5.7 .NET Reflector 7.7 release notes . . . 38
  - 1.5.8 .NET Reflector 7.6 release notes . . . 39
  - 1.5.9 .NET Reflector 7.5 release notes . . . 40
  - 1.5.10 .NET Reflector 7.4 release notes . . . 41
  - 1.5.11 .NET Reflector 7.3 release notes . . . 42
  - 1.5.12 .NET Reflector 7.2 release notes . . . 43
  - 1.5.13 .NET Reflector 7.1 release notes . . . 44
  - 1.5.14 .NET Reflector 7.0 release notes . . . 45
  - 1.5.15 .NET Reflector 6.5 release notes . . . 46
  - 1.5.16 .NET Reflector Pro 6.5 release notes . . . 47
  - 1.5.17 .NET Reflector and .NET Reflector Pro 6.1 release notes . . . 48
  - 1.5.18 .NET Reflector 6.0 release notes . . . 49
  - 1.5.19 .NET Reflector Pro 6.0 release notes . . . 50
- 1.6 Support for Visual Studio 2005 and 2008 deprecated . . . 51

.NET Reflector 8 documentation

About .NET Reflector

With .NET Reflector, you can decompile and debug .NET assemblies and executables and disassemble the source code into your chosen .NET language, so you can see the contents of a .NET assembly.

For more information, see the .NET Reflector product page.

The .NET Reflector 8 documentation is incomplete. For full documentation, see the .NET Reflector 6 documentation.

Quick links

Walkthrough: Debugging a SharePoint customization

.NET Reflector forums

Using the .NET Reflector installer

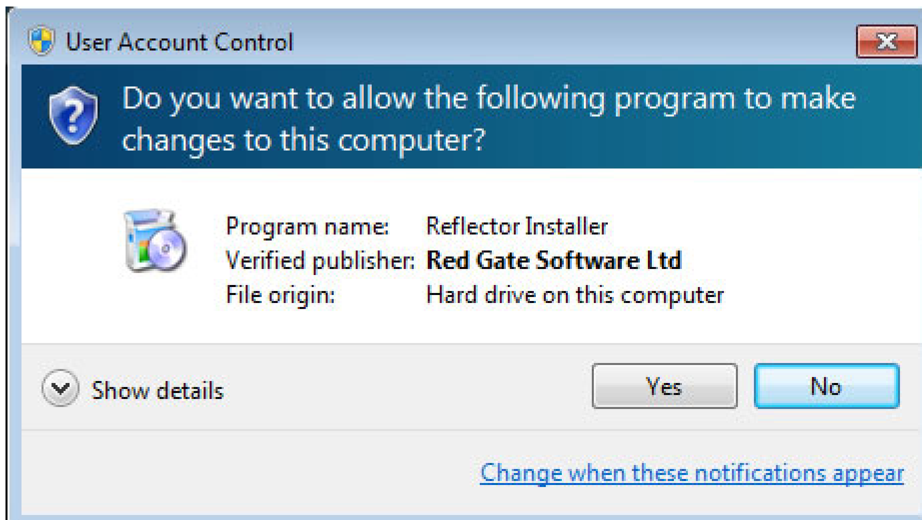
The screenshots on this page are for .NET Reflector 7.6, but the information applies to all later versions of .NET Reflector.

Before version 7.6, .NET Reflector did not have an installer and was downloaded as a .zip file. In version 7.6 we implemented an installer.

You are recommended to uninstall all previous versions of .NET Reflector before you install v7.6. This will ensure a clean installation process as you move from the old zip system to the new installer.

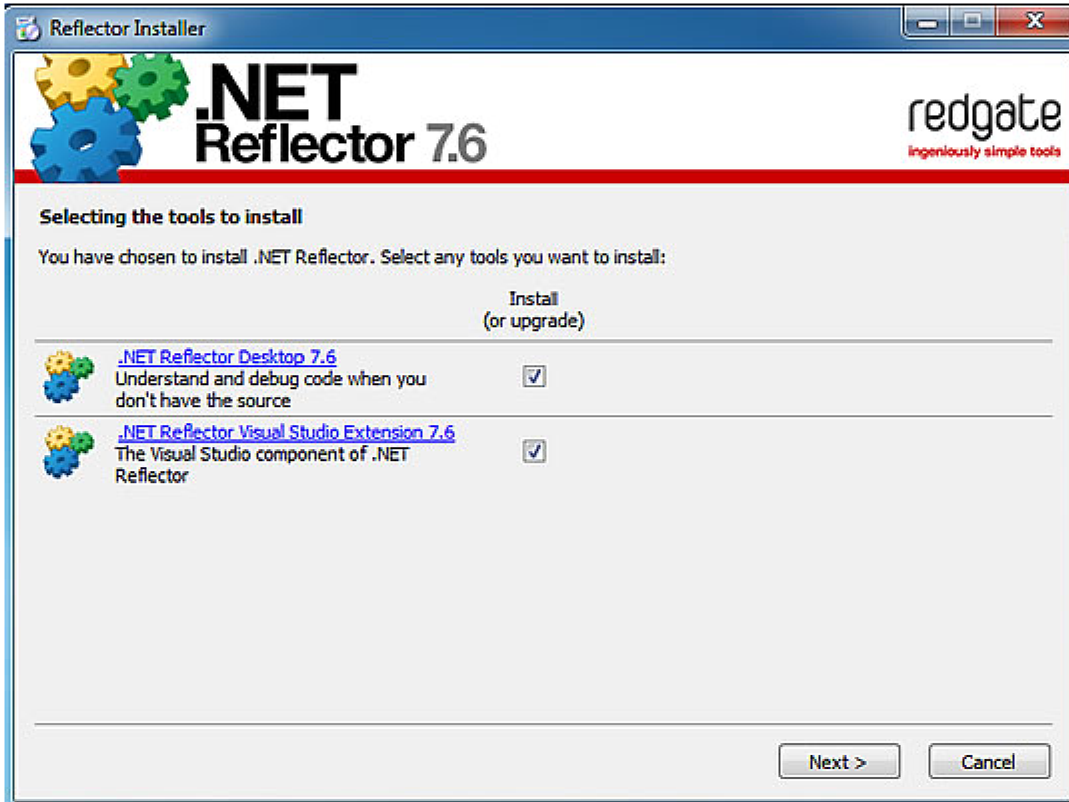
Walkthrough

The installation process is elevated, so you'll see an elevation prompt after you run the installer.

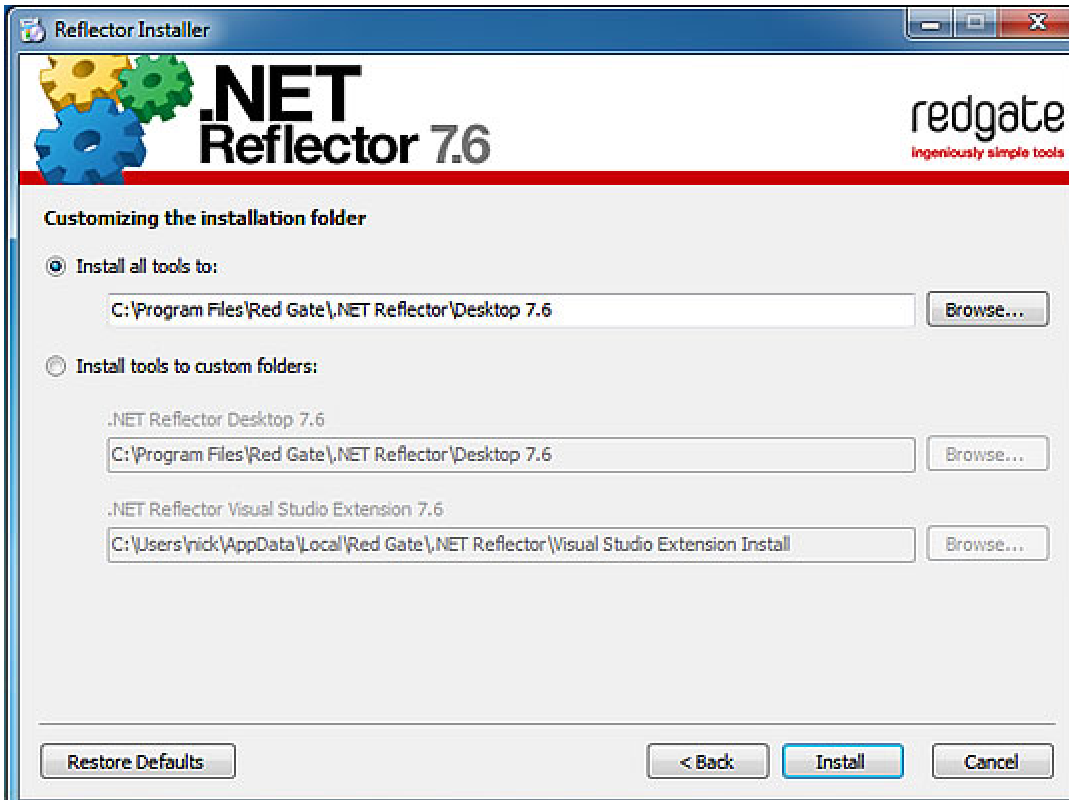


If you're not a local administrator on the machine you're trying to install onto, you'll be prompted for an admin password. If this is a problem, you should still be able to get hold of a zipped version:

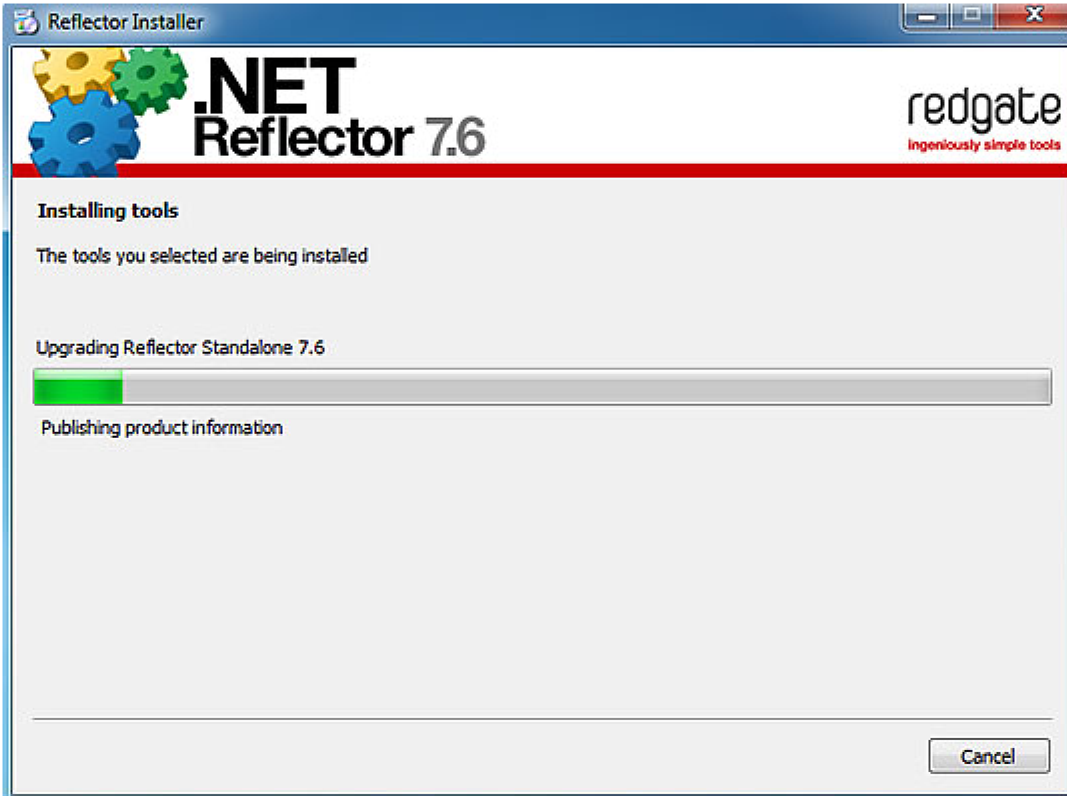
We've now separated the Visual Studio package from the stand-alone tool. Some people we've spoken to only want one or the other, while some want both. If you choose to only install one, you can re-run the installer later to install the other part:



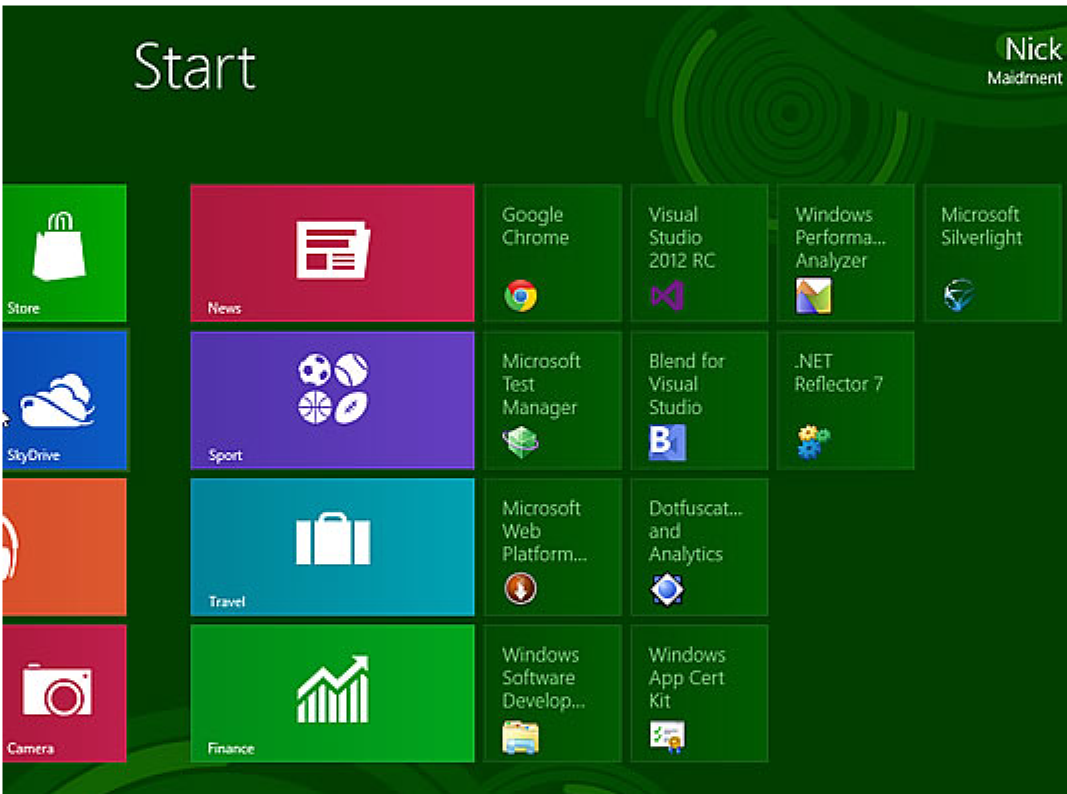
Following this, there is the license agreement and, after that, choosing where to install:



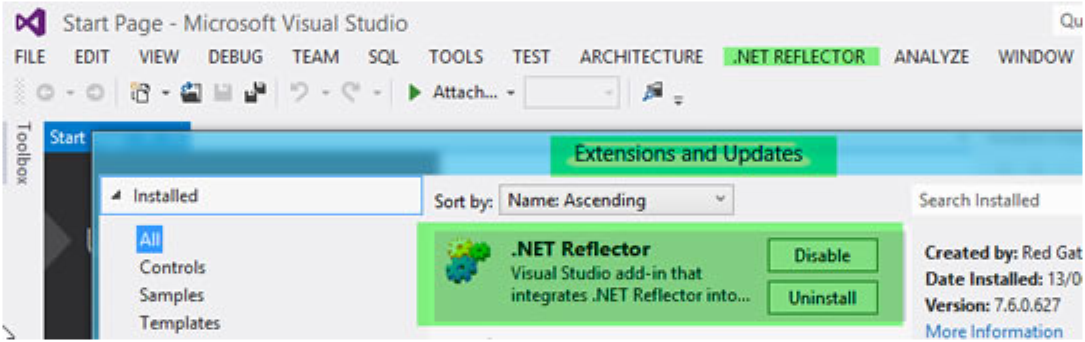
The default location is set to Program Files. We put the Visual Studio extension installer (.vsix) into the current user's app data folder during installation, but we clear it up if you choose to uninstall later. The next screen should show you the installation progress:



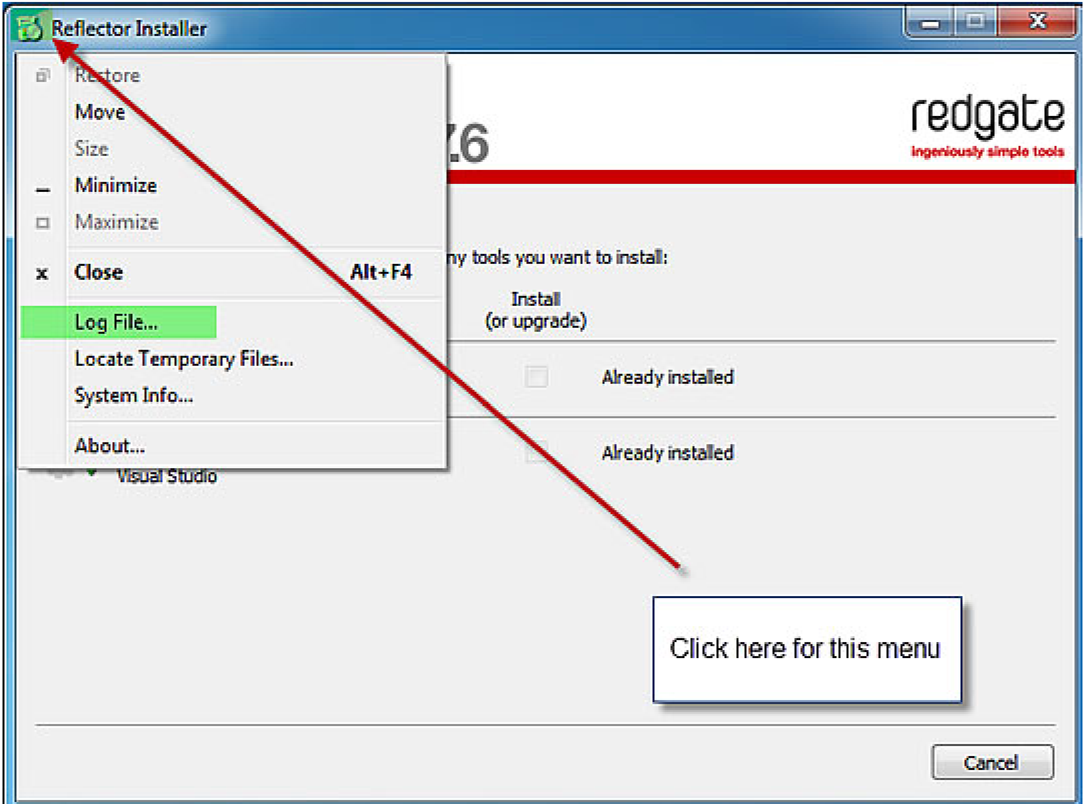
Once that's done, you should get a confirmation screen. If you've installed Reflector Desktop, you should see it in your start menu (or on the metro screen if you're running windows 8):



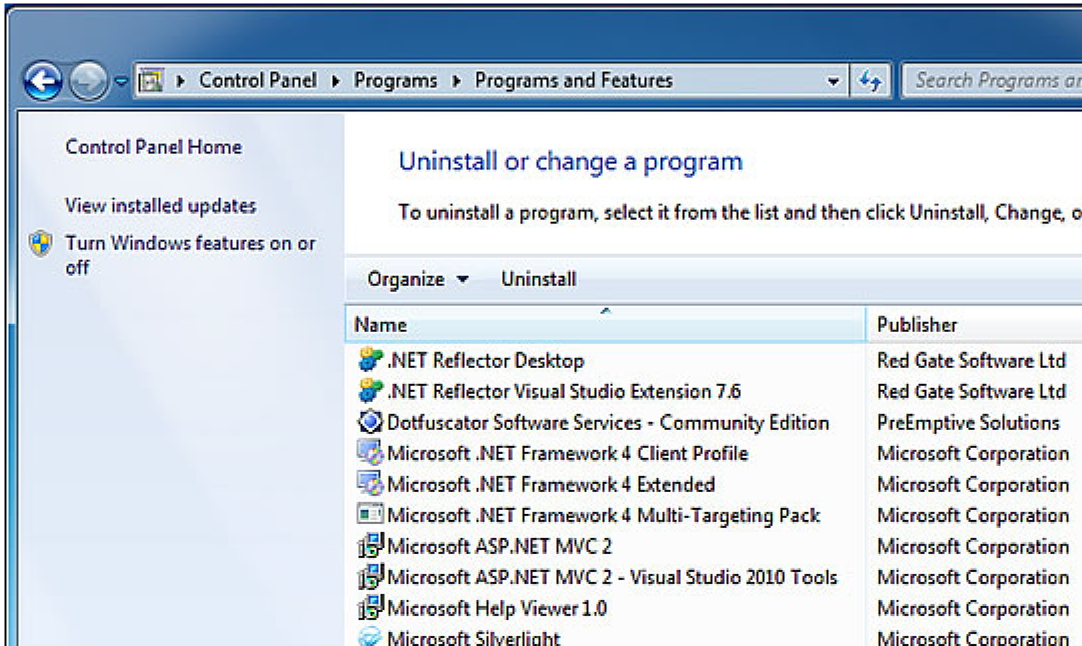
If you've installed the Visual Studio extension, it should appear in Visual Studio 2010 / 2012 when you next fire it up:



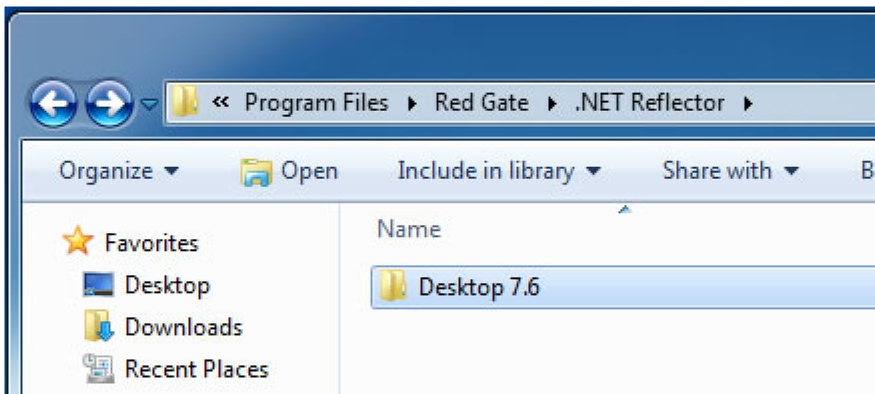
If anything goes wrong, there is a log file, found by left-clicking in the top left hand corner of the installer dialog, which you can inspect and send to us if you need a hand getting anything working:

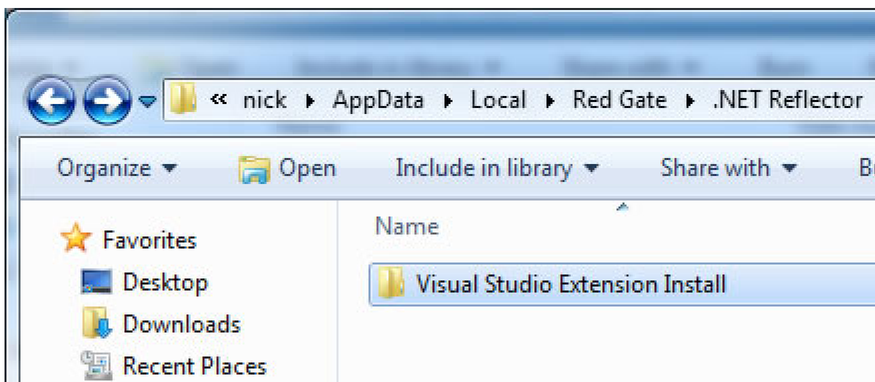


And if you don't feel you need Reflector any more you can now find us in the control panel:



Don't worry, .NET Reflector is still portable! If you don't want to use the installer to create a "portable" version of the tool, install the tool to your desktop as normal, and then go to your installation folder:





If you want the desktop tool, you can copy the files in "Desktop 7.6? to wherever you want them and run reflector.exe from there. For the Visual Studio package, you can copy the .vsix file from the "Visual Studio Extension Install" folder to another machine, and then run it to install into that machine's Visual Studio instances.

Installing from a .zip file

If you are having problems using the installer, you can still download a zipped copy of Reflector

Downloading .NET Reflector as a .zip file

Why are we moving to an installer?

Based on our research it's a better experience for most of our users, both in terms of usability and control. It also gives us a way to uninstall much more cleanly, which is something which gets queried a lot on our forums.

Does the installer still give us a portable tool?

Absolutely! You can either just run the compact installer on your new machine and get a fresh install, or you can copy the tool to wherever you want to run it.

How long will we continue to support the .zip file?

For the foreseeable future. Given that our installer elevates up front, and we know that not everyone has the luxury of installing with elevated permissions, we'll be keeping the .zip file in sync so that everyone get's the latest features and fixes.

Please tell us why you prefer to use a .zip file

Based on our earlier research and the feedback we're still getting, we're working to ensure that the installer gives all the benefits of the .zip file, and also brings you a better experience. Of course, we'd still love to know what you think, so if you still need to use a .zip file instead than the installer, please tell us why.

Working with .NET Reflector

- .NET Reflector tips - Keyboard shortcuts
- Using the .NET Reflector Power Commands

.NET Reflector tips - Keyboard shortcuts

.NET Reflector has a number of keyboard shortcuts. This article details most of them.

Open assembly Ctrl+O

Opens a dialog which allows you to browse to an assembly and open it.

Open assembly list Ctrl+L

This brings up the Assembly list management dialog. You can have multiple assembly lists so that you can easily switch back and forth between different versions of the framework or any other set of assemblies you choose to define:

Export Assembly Source Code Ctrl+S

This is only available at the top level of an assembly. Takes you to the export dialog which allows you to generate a visual studio project for an assembly. While not necessarily fully compilable it does go some way to recovering the source code you may have lost.

We should probably assign this to a key rather than "S" in future, and an issue has been raised.

Open new tab Ctrl+left mouse button click (or mouse 3)

Opens a new tab containing the code for the thing you were clicking on.

Close Current Tab Ctrl+F4

Closes the tab which currently has focus.

Open Bookmarks Pane F2

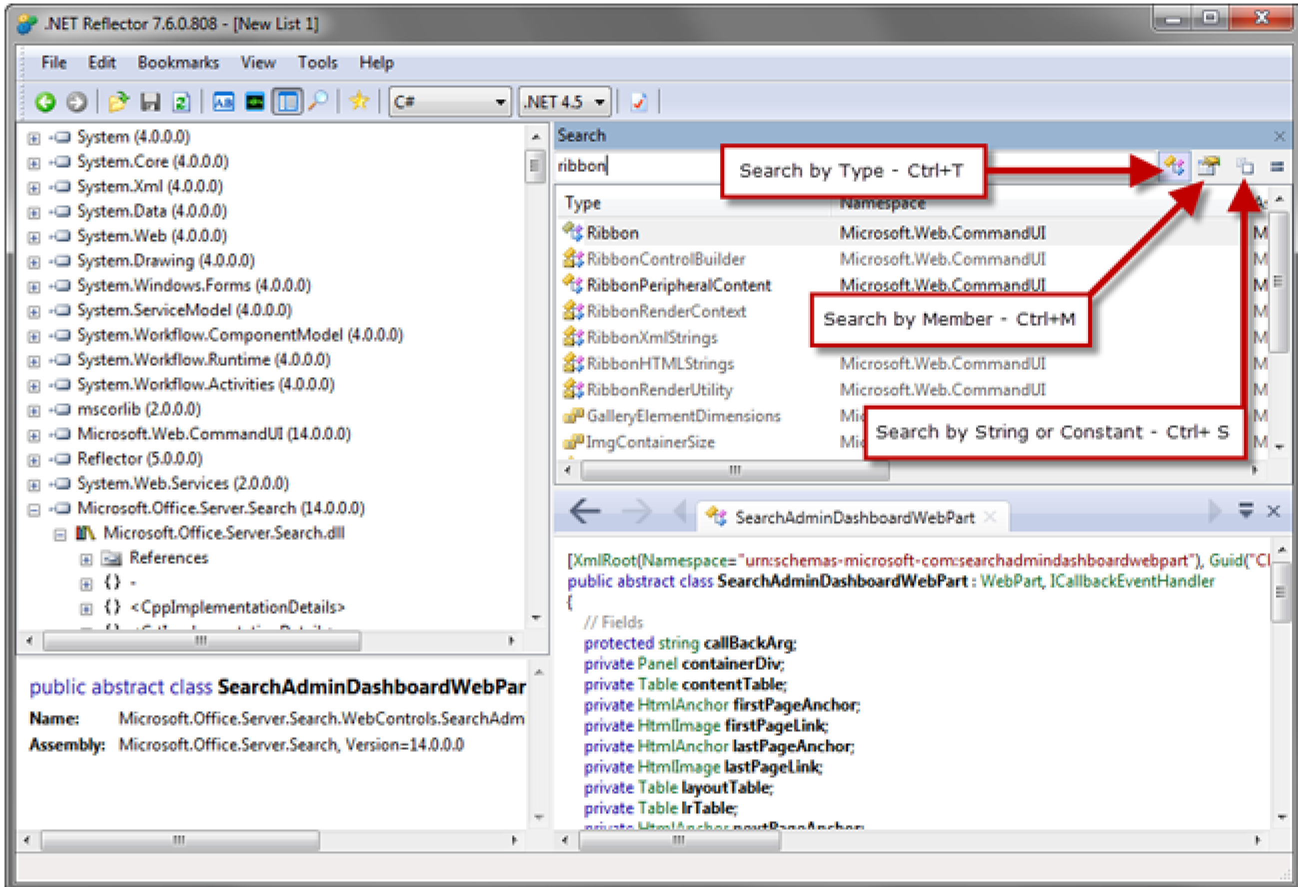
Opens the bookmark pane and displays links to any code that you've previously bookmarked:

Toggle bookmark Ctrl+K

Allows you to set up a bookmark for a particular method, type or assembly.

Open Search F3

Opens the search panel. It's also worth noting that you can change what you search by when this pane has focus:



Decompile Assembly Space or mouse click

This displays code in the currently active tab. The click part is reasonably intuitive, but you might not know about Space.

Open Analyze Pane Ctrl+R

Opens the Analyze pane so that you can do some analytical investigation on whatever was highlighted in the assembly tree:

Close assembly Delete

Closes the currently highlighted assembly in the assembly tree.

Search MSDN Ctrl+M

Available at the namespace level for framework classes Searches MSDN for documentation on this namespace.

Using the .NET Reflector Power Commands

If you've tried the .NET Reflector 7.7 release, you'll notice some new commands have appeared across the program. These are the remains of the Power Commands we integrated into the tool in an earlier release. These commands could previously be enabled from the options menu, but they hadn't been worked on in a long time and had run into disrepair. Some were confusing and others were broken.

Here's a rundown of the updated commands and how to access them from within Reflector.

Collapse All Assemblies

There is now a command to Collapse All Assemblies in the File menu. Since we don't have a root node in the assembly browser (like a solution in Visual Studio has) this doesn't make sense in a context menu, so it's in the File menu for now. As Jason Haley put it when he originally wrote the Power Commands add-in: "This exposes the underlying tree view's CollapseAll method functionality which means it will collapse all expanded nodes that are currently open." It's great for when you have a ton of assemblies and you need to go back to the default state before you expanded them all.

Copy As

When viewing code for a class, type, property or field, you can now use the right-click context menu in the code window to copy all the code to the clipboard in a format of your choice. Note that for a class this will expand all the methods automatically, so in the picture above selecting "Text" will copy the entire ConcurrentBag <T> class to the clipboard.

Hopefully this will fix some of the issues where you need to quickly reproduce the code generated by Reflector, but were having problems with those pesky code hyperlinks. The original copy function is still there if you still want to copy a selection as well, and becomes available when you highlight a section of code.

Import/Export Assembly List

We're still looking at ways to improve our assembly list management functionality, but for now there's a way to save your assembly list setup by using the Export Assembly List command, and then load it back using Import. Perhaps you might want to do this as part of backing up your Reflector configuration, or to move between different computers with the same setup. Exporting opens another dialog where you can choose which assemblies from the lists to export, and save this as an XML file for later importing.

It's not immediately clear that importing won't make changes to the assemblies which are currently loaded into Reflector, only the assembly lists. So once you've imported your assembly lists you will still need to Open Assembly List and select the list you want to load into Reflector.

Open File Location

This opens an explorer window at the location on disk where the assembly can be found. Useful for when you quickly need to get to the DLL or EXE so you can manipulate the file outside of Reflector.

Open With

At several levels in the assembly browser there's an option to open the selected item with an application of your choice. Separate commands for common applications formed a large proportion of the original Power Commands add-in, so we've distilled these into default applications for one larger Open With command.

This means slightly different functionality for different levels in the tree. Notably, code snippets are written to a temporary file (using the Copy As function described earlier, but writing to a file instead of the clipboard) so they can be opened, and resources are extracted before they open normally. Resource tables are opened in Visual Studio as .RESX files so they can be viewed more easily. And assemblies open as expected.

Selecting Choose Application will bring up a dialog where you can add your own applications. Adding an application to the list will save it so that next time you select Choose Application you will be able to select the desired application faster.

Open Zip

Did you know you can drag a .ZIP archive into Reflector and it will decompile the compressed assemblies without you having to extract them yourself? This was originally a Power Command, but while looking for ways to merge the code I discovered native functionality in Reflector for opening Silverlight applications (XAP files) which could be easily extended to handle .ZIP archives as well. You can also select .ZIP archives now from the drop down menu in the Open Assembly dialog, and open them through the menu.

Referenced By

The analyzer now contains the functionality to show which of the assemblies loaded into Reflector depend on the selected assembly or module.

This means you can now check dependencies both ways: from the assembly browser for references of the current assembly, and from the analyzer for assemblies referencing the current assembly.

At the moment the analyzer will only show the list of assemblies. But if it's useful, it might be an exciting possible improvement for the future to attempt to show the code which calls into the current assembly by making the analyzer tree expandable to show the classes and methods which do this! That way you will be able to see not only where your assembly is being used, but also how it is being used by other assemblies.

Other Power Commands

If you were previously familiar with the Power Commands in the options menu and there was one you frequently used which I haven't detailed here (for example the Query Editor) then don't panic. We've packaged up a few of the other useful commands and plan on releasing a new Power Commands add-in with the final release of 7.7. This will put the option to enable the extra commands back into the options menu so you can select them from there.

The reason we opted for this approach was to streamline the application to remove those commands which would only be used in niche cases. We really wanted to get rid of the option to enable/disable the Power Commands, as discoverability was really low while they were turned off by default. These commands didn't seem valuable enough to be always enabled in Reflector, but still worked to the point where people might use them, so a separate add-in seemed best.

As always, we're constantly looking for feedback and feature requests! If there are any new commands you'd really like to see in the next version of Reflector then there are the usual channels: the forums or info@reflector.net

Worked examples

- Debugging a SharePoint customization
- Debugging into SharePoint and seeing the locals
- Introduction to building .NET Reflector add-ins
- Exporting source code

Debugging a SharePoint customization

SharePoint has a large and complex code base, and some of the details aren't as transparent or well-documented as developers need. This is a short overview showing how bugs can arise when working with incomplete documentation, and how we can use .NET Reflector to fix those bugs.

In this scenario we're working on a web portal, and we want to add a set of controls to the ribbon. This is simple enough, but we run into an unexpected error. We use .NET Reflector to explore the SharePoint API that is throwing the exception, so we can understand the problem and troubleshoot the control.

Defining custom controls

We're looking at an issue in SharePoint 2010, where XML comments in definition files cause an exception. The issue can arise when creating custom fields, content types, and other custom features that require XML definition files. The XML schema for SharePoint Features is documented, but the documentation doesn't always give rich implementation information.

The custom action behind the control is defined in the XML file, and it's part of a group of controls being used in the ribbon. Because these files quickly become long and complex, we want to comment the code.

This is a simplified "hello world" example of a custom action definition file:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <CustomAction Id="Ribbon.WikiPageTab.CustomGroup" Location="CommandUI.Ribbon">
  <CommandUIExtension>
   <CommandUIDefinitions>
    <CommandUIDefinition Location="Ribbon.WikiPageTab.Groups._children">
     <groups>
      <!-- Group containing the new "hello world" controls -->
      <Group Id="Ribbon.WikiPageTab.CustomGroup" Sequence="55" Description="Custom
Group" Title="Custom" Command="EnableCustomGroup"
Template="Ribbon.Templates.Flexible2">
       <Controls Id="Ribbon.WikiPageTab.CustomGroup.Controls">
        <Button Id="Ribbon.WikiPageTab.CustomGroup.CustomGroupHello"
Command="CustomGroupHelloWorld" LabelText="Hello, World" TemplateAlias="o2"
Sequence="15" />
        <Button Id="Ribbon.WikiPageTab.CustomGroup.CustomGroupGoodbye"
Command="CustomGroupGoodbyeWorld" LabelText="Good-bye, World" TemplateAlias="o2"
Sequence="18" />
       </Controls>
      </Group>
     </groups>
    </CommandUIDefinition>
   </CommandUIDefinitions>
   <CommandUIHandlers>
    <CommandUIHandler Command="EnableCustomGroup" CommandAction="javascript:return
true;" />
    <CommandUIHandler Command="CustomGroupHelloWorld"
CommandAction="javascript:alert('Hello, world!');" />
    <CommandUIHandler Command="CustomGroupGoodbyeWorld"
CommandAction="javascript:alert('Good-bye, world!');" />
   </CommandUIHandlers>
  </CommandUIExtension>
 </CustomAction>
</Elements>
```

(In practice, we could be dealing with hundreds of lines of XML)

When the portal page renders, the controls appear. But when the control is actually used, SharePoint displays an error.

Debugging the error

SharePoint error messages typically have a Correlation ID , a GUID that lets us track the error through the logs. From these logs, we see that the error is a null reference exception.

This is surprising, because in building our control, we've followed the SharePoint documentation pretty thoroughly. At first glance, there appears to be nothing wrong with any of our own code, and the custom action definitions match the documented schema.

To solve the problem, we look into the code with .NET Reflector.

We see that the exception is thrown by Microsoft.Web.CommandUI.Ribbon.CreateRenderContext, so we search for that library using .NET Reflector desktop, and decompile it to view the underlying source code.

The code .NET Reflector shows us is:

```
DataNode node4 =
uiproc.GetResultDocument().SelectSingleNode("/spui:CommandUI/spui:Ribbon/spui:Tabs/spu
i:Tab[@Id='" + this.InitialTabId + "']/spui:Groups");
if (node4 == null)
{
 node4 =
uiproc.GetResultDocument().SelectSingleNode("/spui:CommandUI/spui:Ribbon/spui:Contextu
alTabs/spui:ContextualGroup/spui:Tab[@Id='" + this.InitialTabId + "']/spui:Groups");
}
Hashtable hashtable = new Hashtable();
if (node4 != null)
{
 foreach (DataNode node5 in node4.ChildNodes)
{
XmlAttribute attribute2 = node5.Attributes["Id"];
if (attribute2 != null)
{
 hashtable[attribute2.Value] = node5;
}
```

If a node is a comment, rather than a conventional element, the value returned is null. This is because XmlNode.Attributes returns as null for all nodes that are not of the XmlNodeType.Element type. This null result is unexpected, so an exception is thrown.

Luckily, the solution is simple, and removing the comments prevents the exception. This may not be an ideal workaround, but because the limitation lies in a third-party assembly, it can't be fixed directly.

Improving on documentation

The XML comments issue is a small example of an undocumented limitation. It's simple to fix with decompilation tools, but would be a substantial debugging task without the ability to look inside the underlying code.

Unfortunately, incomplete (or entirely absent) documentation is quite common for 3rd party tools libraries, and frameworks. SharePoint itself is elaborate, complex, and difficult to debug and understand using only the published documentation. In practice, you can encounter unusual errors, or require customizations that depend on entirely undocumented behavior.

In these cases .NET Reflector saves troubleshooting time, and lets you explore 3rd party libraries in context, to better understand how to integrate with them and build upon them.

Debugging into SharePoint and seeing the locals

The debugging functionality in .NET Reflector is based in Visual Studio. It lets you use the Visual Studio debugger with decompiled code, so you can step through it, set breakpoints, and so on. However, for debugging scenarios like working with SharePoint, the assemblies are hosted and loaded outside Visual Studio. Reflector generates PDB files and allows you to step through the code, but the optimizations made by the CLR mean that you cannot see the values of the local variables.

This limits debugging because you cannot watch these values as they change, and properly follow the data flow. In this article, our technical lead Clive discusses a method for enabling locals for debugging sessions attached to the SharePoint `w3wp`processes.

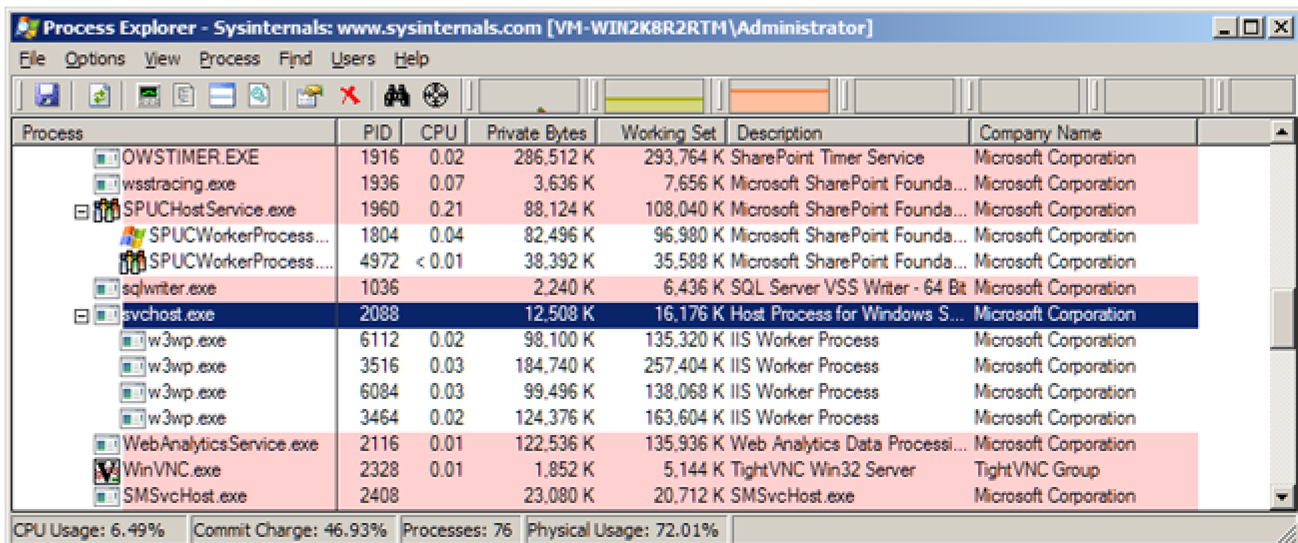
Clive's investigation

The CLR is fairly keen on generating efficient code, and so, if you don't have a debugger attached at the time the code is JIT-ed, it's going to do its very best to generate fast (and therefore undebuggable) code. If you later attach a debugger, the CLR as it currently stands generally won't re-JIT methods, so the debugger is not going to give you a very good debugging experience.

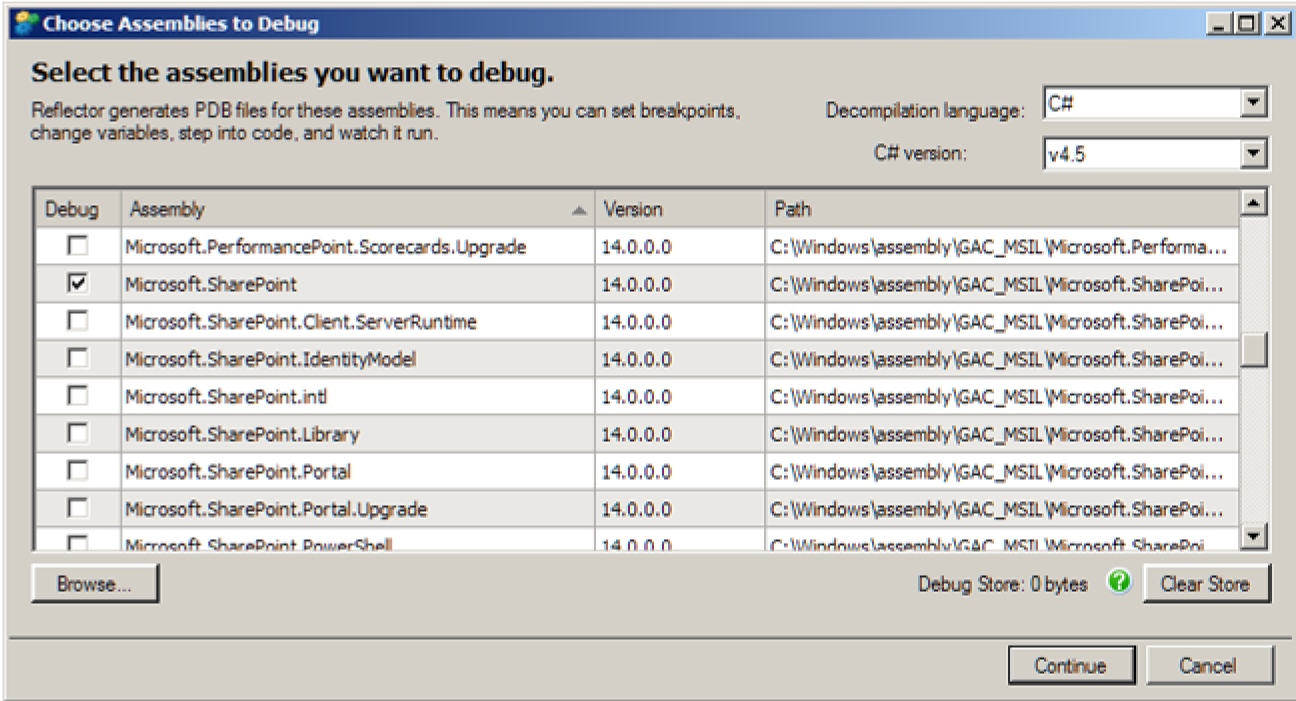
This became clear recently when I did some experimenting on debugging SharePoint using Reflector VSPro. I'm no expert on SharePoint, and it took quite a while to get a virtual machine together. Moreover, running SharePoint inside a VM on my work machine brings the machine to a crawl. But we learned a lot from the debugging experience, and I'm going to walk you through it.

Setting up, and having problems

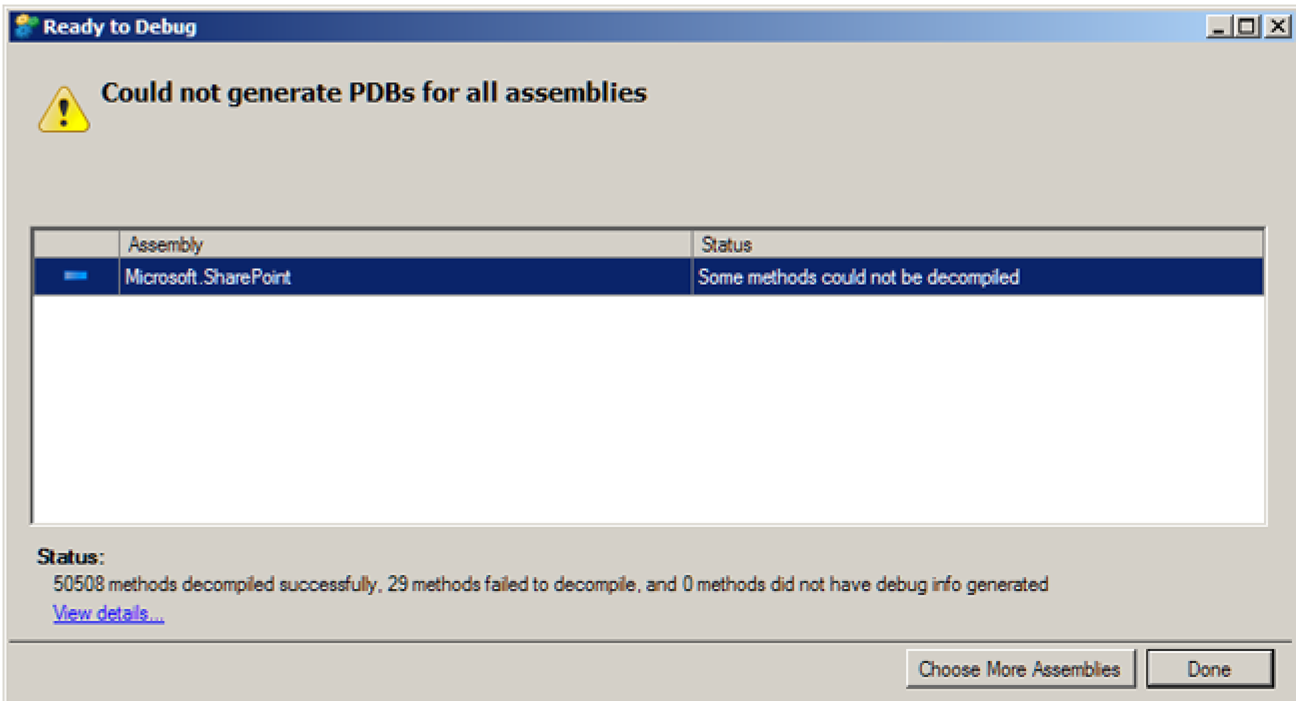
Initially, I kicked off a web site and watched the w3wp.exe instances being created using Process Explorer:



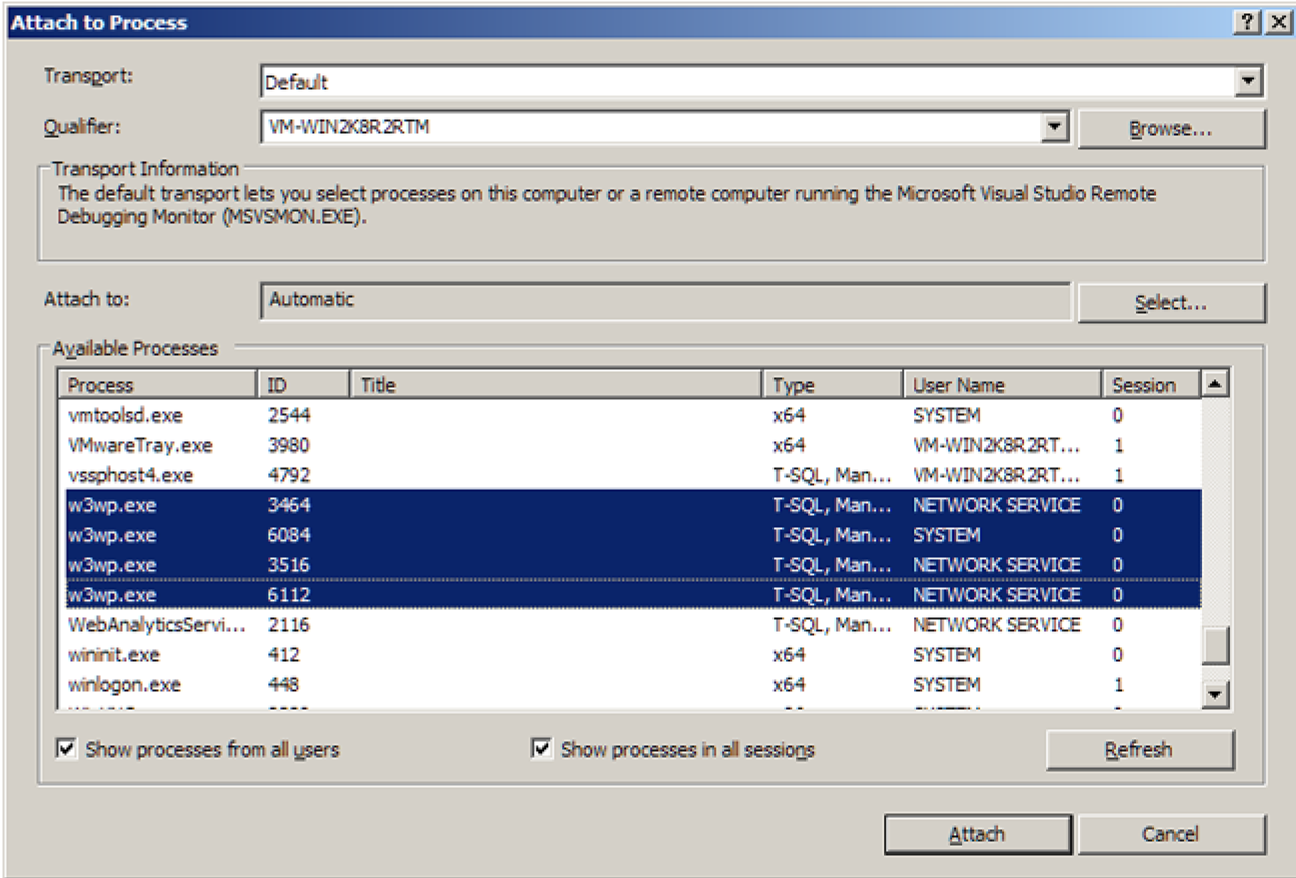
To enable debugging, I then decompiled the SharePoint assembly using Reflector VSPro:



There were a couple of methods that couldn't be decompiled:

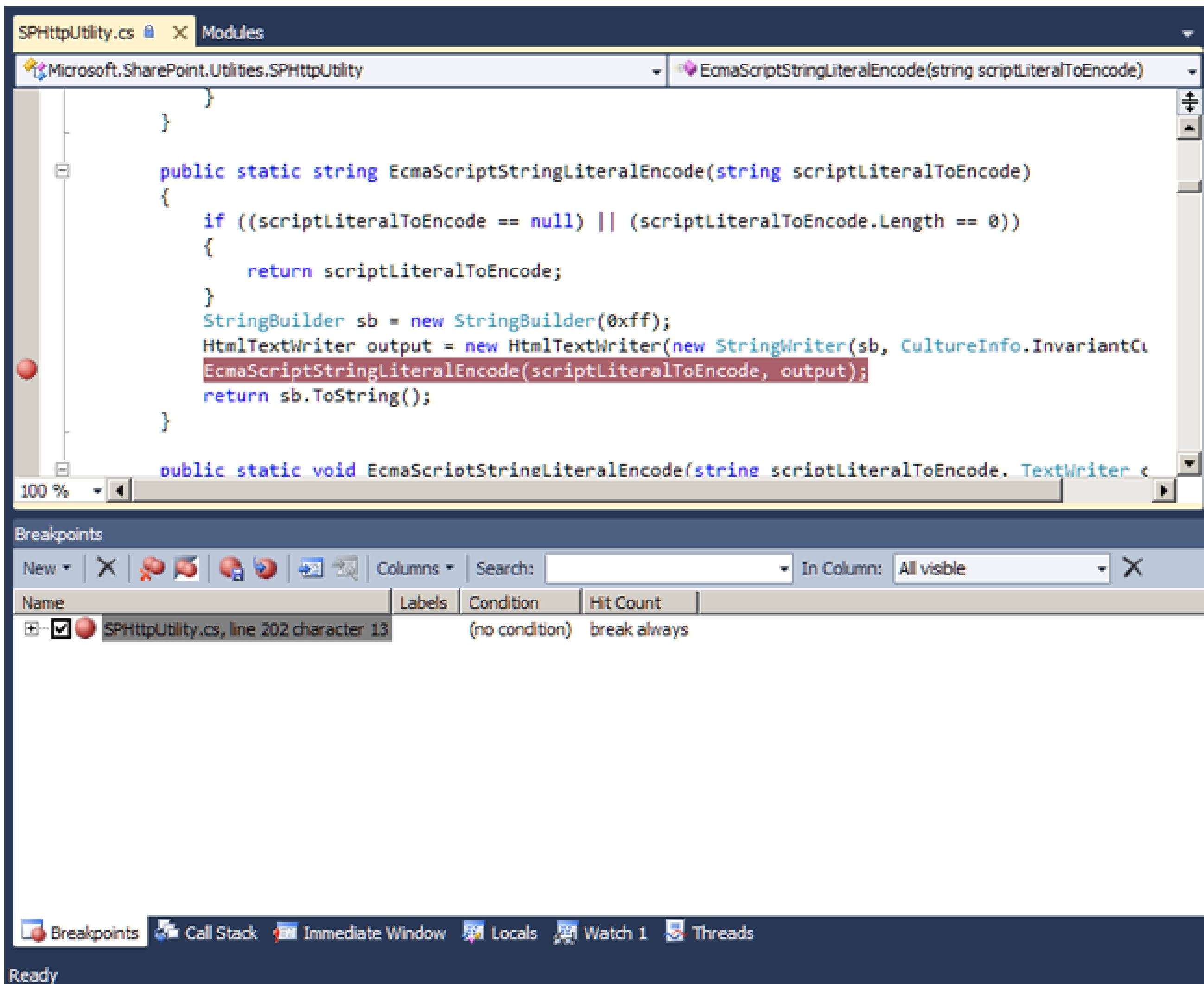


But it's still possible to attach the debugger to the four worker processes, and start to look at what's going on:

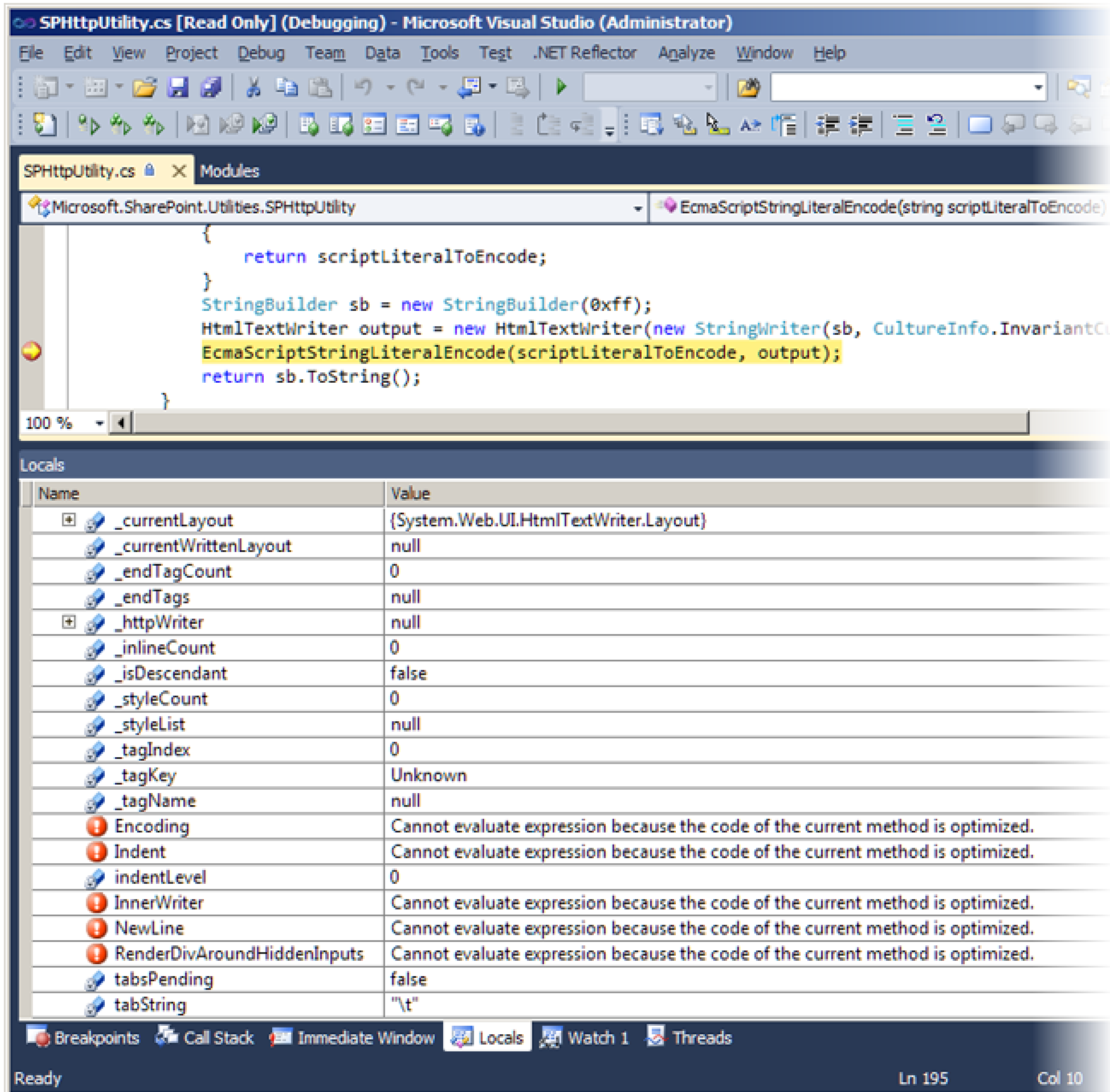


In order to start debugging, I needed to locate a source file corresponding to a class that I knew was going to be called. So I navigated into the Reflector cache directory and found the file `SPHttpUtility`, which I knew would contain the code for the class of the same name.

Finding this in Visual Studio, I set a breakpoint on one of its methods:



I then used a web browser to get the SharePoint to execute the code. Imagine my dissatisfaction when the debugger couldn't display the local variable values:



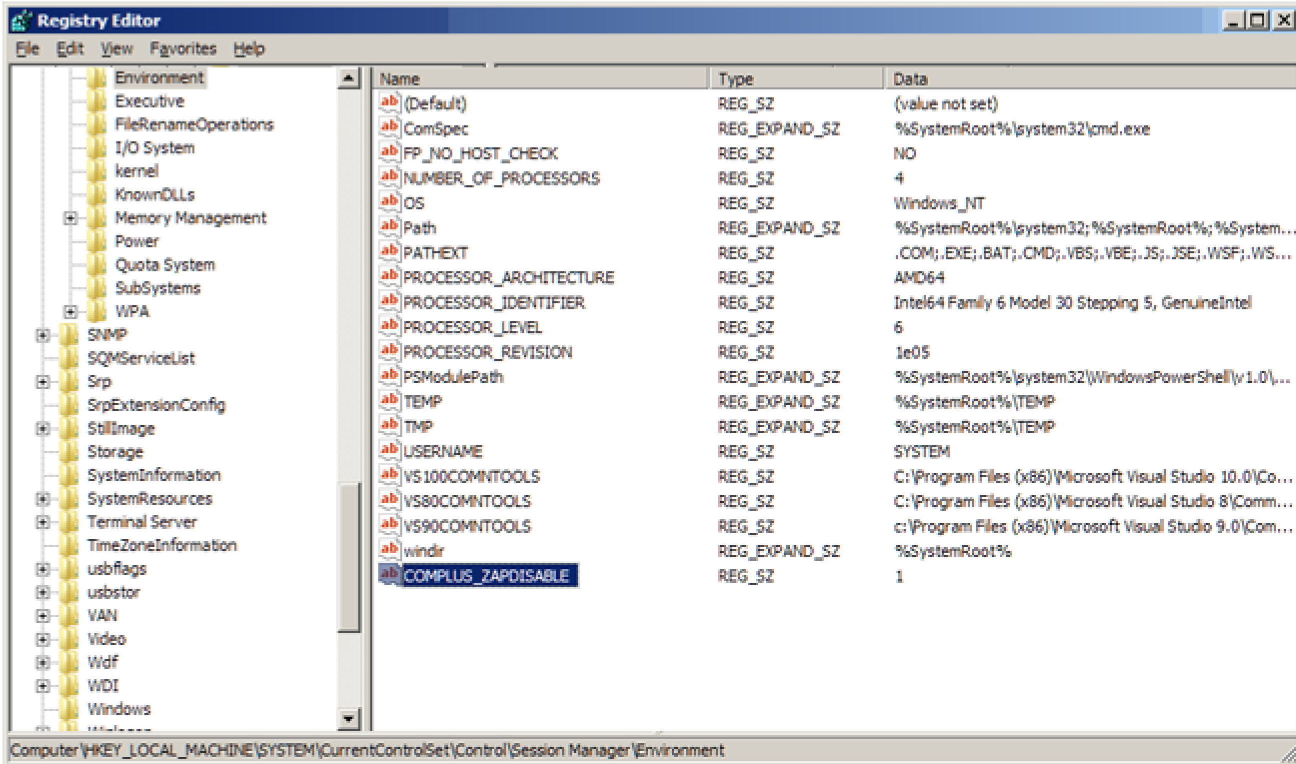
Without the local values, we can't follow the flow of data through the code, and we can't debug as accurately as we would like.

Enabling debugging for SharePoint locals

One problem is that SharePoint is made up of ngen'ed assemblies, and you can't see what's going on in that code. The assemblies are loaded automatically by the CLR, and so prevent us debugging.

Disabling optimizations with `COMPLUS_ZAPDISABLE`

To see the locals, we need to prevent the CLR loading the ngen'd assemblies. Fortunately, this can be done by setting the`COMPLUS_ZAPDISABLE` environment variable in the process that loads the CLR itself.



This issue is also documented in the MSDN blog post: How to disable optimizations when debugging Reference Source

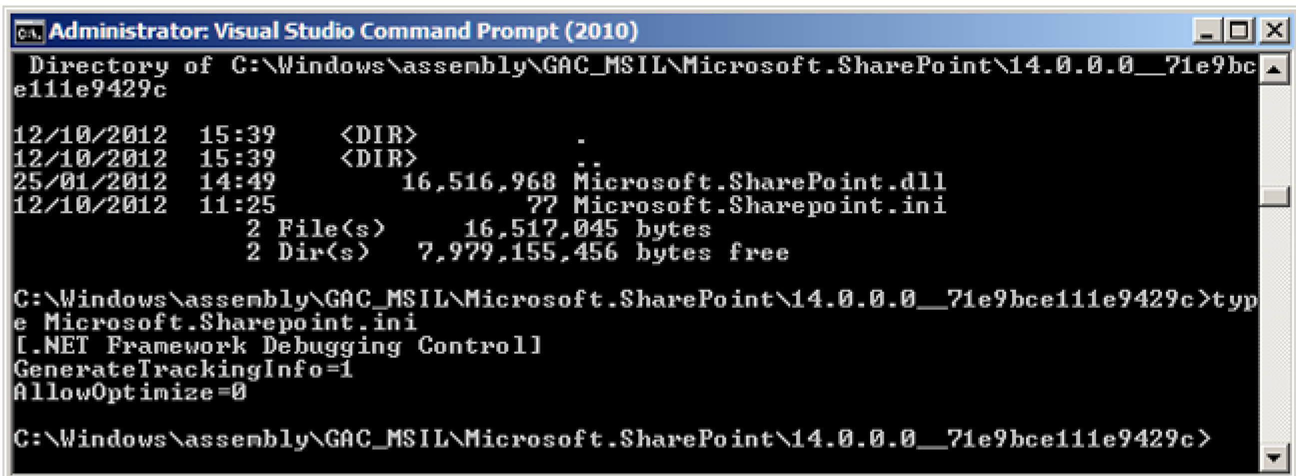
With IIS, I find this easiest to do using the registry entry described in this article on improving the debugging experience.

The environment variable prevents the CLR loading the precompiled version of an assembly. If you set this entry, you'll need to restart the various worker processes, and a useful trick is to use process explorer to check that the environment variable is set in the process, by using the properties tab in the context menu when the process is selected

Preventing optimization with .ini files

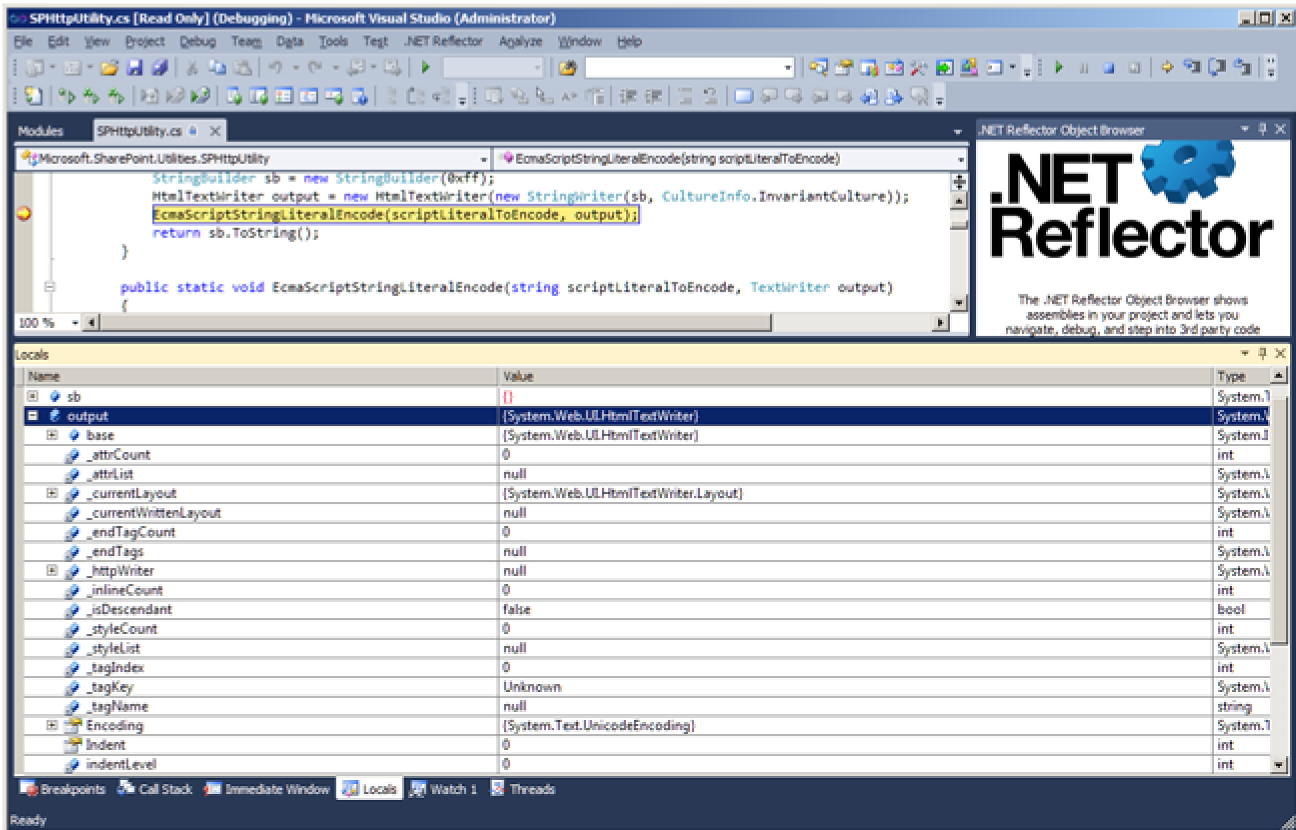
The second problem is that the methods were JIT-ed before the debugger was attached, and hence the code is optimized to some extent. The trick now is to use a .ini file, which the JIT will detect and which can be used to override the optimization level specified in the assembly itself.

I went into the GAC, using the Modules window inside Visual Studio to determine where the assembly was actually loaded from. I then made a .ini file, named just as the assembly but with the extension ini instead of dll, and containing the following three lines:



In order to try this again, I recycled the `w3wp.exe` processes in my case by using Process Explorer to kill them, although recycling IIS might have been a slightly tidier way to do it. I then hit the web page, let them start up and attached again.

This time, at the same breakpoint, we can see all of the variable values because the code is now unoptimised:



Conclusions

There are two things to take away, if you want to debug into the SharePoint assemblies, with all the locals visible:

1. Prevent the loading of precompiled assemblies
   Set the `COMPLUS_ZAPDISABLE` environment variable in the registry
2. Prevent the optimization of loaded assemblies using the JIT
   Create .ini files in the same locations as the .dll files you're looking at.

This will give you a much better debugging experience, even if you attach the debugger to the process after it has started.

In my tests there was no impact on system stability, and it's easy to remove the files and re-set the variables when you're done.

Introduction to building .NET Reflector add-ins

.NET Reflector has an extensive add-in framework, and there are plenty of add-ins already available to use as examples of what can be done.

A .NET Reflector add-in is fundamentally a dll/exe assembly file that contains packages. A package is a class that implements the `IPackage` inte rface, which defines a Load and Unload method. An `IServiceProvider` interface is passed during loading, and gives access to a set of services which are part of the .NET Reflector object model (the most common of which we'll see below).

Available services

The following table lists the most commonly-used services that can be accessed through the `GetService` method on`IServiceProvider`

Service	Description
IAssemblyBrowser	Maintains the currently selected Code Model object in the `ActiveItem` property. You can assign a Code Model object like IMethodDeclaration to the `ActiveItem` to programatically change the currently selected item in the browser window. ActiveItemChanged notifies that the selected item has changed.
IWindowManager	Manages the application window and pane windows. You can add your own pane windows to the Windows collection which will create an IWindow hosting frame. ShowMessage can be used to show notification messages to the user.
ICommandBarManager	Manages the Reflector menu bar, tool bar and context menus. You can lookup a context menu by its identifier and add items to it.
IConfigurationManager	Manages the sections from the Reflector configuration file as a set of IConfiguration objects. Lists of items are represented as properties named "0?, "1?, "2?, and so on.
IAssemblyManager	Maintains the list of currently loaded assemblies. LoadFile can be used to load an assembly file from disk. Unload allows you to unload an assembly from memory. The Assemblies collection holds all the currently loaded assemblies.
ILanguageManager	Manages formatting modules for different programming languages. The ActiveLanguage property exposes the ILanguage object currently used for rendering. You can add your own language rendering code by implementing the ILanguage interface. Use RegisterLanguage to add your add-in to ILanguageManager.

Although the .NET Reflector API exposes more interfaces than this, these are the most commonly used ones.

Building a HelloWorld add-in

A simple "HelloWorld" add-in can be created by implementing the `IPackage` interface.

The Load method is implemented to ask the `IServiceProvider` for the `IWindowManager` service, which allows you to communicate with .NET Reflector's windowing system. Finally, the `ShowMessage` method is used to show a message to the user:

```
using System;
using Reflector;
internal class HelloWorldPackage : IPackage
{
 private IWindowManager windowManager;
 public void Load(IServiceProvider serviceProvider)
 {
  this.windowManager = (IWindowManager)
serviceProvider.GetService(typeof(IWindowManager));
  this.windowManager.ShowMessage("Loading HelloWorld!");
 }
 public void Unload()
 {
  this.windowManager.ShowMessage("Unloading HelloWorld!");
 }
}
```

The code can be compiled into an add-in dll, which is referencing Reflector.exe as a library:

```
csc.exe /target:library /out:HelloWorld.dll *.cs /r:Reflector.exe
```

The add-in can then be copied to your Reflector directory and loaded using the View Add-Ins menu. While this is a very basic add-in, the fundamentals of the construction and implementation don't change.

Adding items to command bars and context menus

The `ICommandBarManager` service allows you to add menu items to the .NET Reflector main menu and context menus. Each sub-menu and context menu is registered in the `CommandBars` collection with an identifier name, and the following table lists the most commonly used identifiers:

Identifier	Description
Tools	The tools menu shown as part of the main menu.
Browser.Assembly	The context menu for the currently selected assembly.
Browser.Namespace	The context menu for the currently selected namespace.
Browser.TypeDeclaration	The context menu for the currently selected type declaration.
Browser.MethodDeclaration	The context menu for the currently selected method declaration.

Learning more

Thoroughly documenting the .NET Reflector API is something we hope to improve in future.

We currently recommend this series of articles by Jason Haley:

- Getting Started with .NET Reflector add-ins
- Create your own add-in : The basics
- Create your own add-in : More details
- Wrapping .NET Reflector

For more examples, see:

- .NET Reflector Add-in Tutorial (Peli de Halleux)
- Building the .NET Reflector Add-in (Jamie Cansdale)

Exporting source code

If you‘ve lost the source code for one of your assemblies and you want to recreate the files, you can do this by using .NET Reflector's **Export source code** option.

1. Open the .NET Reflector desktop application.
2. On the **Tools** menu, under **Options**, then under **Disassembler**, make sure that language and optimization are set as needed.
3. Load your assembly into the *Reflector object browser.*
4. Right-click the assembly and choose **Export source code**.

   > Make sure you right-click the *assembly* node. If you're on a *namespace* or *class* node, the option isn't available.

5. Click **Start.**
   The C# or VB source files and *.csproj* / *.vbproj* file are created for the project in the specified output directory. You can open the project in Visual Studio, fix any errors and changes the code as needed, and rebuild the project

> Exported source code isn't perfect. There may be errors in the exported code that you need to fix manually.

Troubleshooting

Error messages

- The certificate for a digital signature in this extension is not valid

Error messages

- The certificate for a digital signature in this extension is not valid

The certificate for a digital signature in this extension is not valid

When attempting to update .NET Reflector's Visual Studio extension, the installation can fail with a message:

> *The certificate for a digital signature in this extension is not valid.*

This will affect people trying to use check for updates on XP / Windows Server 2003 to upgrade the Visual Studio extension in Visual Studio 2010

This is a Microsoft bug and is detailed here: http://support.microsoft.com/kb/2581019

The issue occurs because Windows XP and Windows Server 2003 handle the Certificate Revocation Lists (CRL) differently than other operating systems.

When Extension Manager handles the Certificate Revocation Lists (CRL), Extension Manager uses the same method for all operating systems. However, Windows XP and Windows Server 2003 handle the Certificate Revocation Lists (CRL) differently than other operating systems. Therefore, issue 2 occurs.

The Workaround:

To work around this bug:

1. Go to Visual Studio > Tools > Extension Manager and uninstall the old .NET reflector package.
2. Download the new installer from the Reflector website
3. Run the installer.

Let us know if you still have any trouble with this bug, and we'll investigate further.

Release notes and other versions

Version 9.0 (current)	November 16th, 2015 (latest)	Release notes	Documentation
Version 8.5	March 4th, 2015	Release notes	Documentation
Version 8.4	October 27th, 2014	Release notes	
Version 8.3	January 20th, 2013	Release notes	
Version 8.2	June 16th, 2013	Release notes	
Version 8.1	May 7th, 2013	Release notes	
Version 8.0	February 20th, 2013	Release notes	
Version 7.7	October 12th, 2012	Release notes	*Documentation not available*
Version 7.6	July 16th, 2012	Release notes	
Version 7.5	February 13th, 2012	Release notes	
Version 7.4	November 12th, 2011	Release notes	
Version 7.3	July 15th, 2011	Release notes	
Version 7.2	July 7th, 2011	Release notes	*Documentation not available*
Version 7.1	May 4th, 2011	Release notes	
Version 7.0	March 7th, 2011	Release notes	
Version 6.5 - Pro	July 14th, 2010	Release notes	Documentation
Version 6.5	July 14th, 2010	Release notes	
Version 6.1	February 24th, 2010	Release notes	
Version 6.0 - Pro	February 9th, 2010	Release notes	
Version 6.0	February 9th, 2010	Release notes	

.NET Reflector 8.5 release notes

March 4th, 2015

This is a minor release of .NET Reflector which includes the following enhancements and bug fixes:

Features

- Supports Microsoft Visual Studio 2015 CTP 6

Fixes

- Fixed a problem where .NET Reflector could crash Visual Studio if it didn't find its SQLite instance
- Improved .NET Reflector's shutdown code to stop it causing Visual Studio to crash while closing

Known issues

- In Visual Studio 2015, pressing F12 in your own code takes you to the metadata view, and not to the decompiled code

.NET Reflector 8.4 release notes

October 27th, 2014

This is a minor release of .NET Reflector, which includes the following bug fixes:

Enhancements

- Supports Visual Studio 2014 CTP.

Fixes

- Fix an issue with the .NET Reflector Visual Studio add-in, which could cause Visual Studio to crash.

.NET Reflector 8.3 release notes

.NET Reflector 8.3.3 - 20th January, 2014

Fixes

- RP-3550 – Fixed several issues to do with debugging, including performance issue when stepping
- Reflector no longer offers to enable debugging for reference assemblies

.NET Reflector 8.3.0 - December 10th, 2013

Features

- Analyzer pane available in the Visual Studio add-in

Fixes

- RP-2619 - Issue with decompiling *mscorlib* on Windows 8.1 no longer occurs
- RP-3514 - Go To Member now only appears on appropriate items

Known issue

- On Vista x86, the installer may crash on the final step of installation. Reflector will still be fully installed, despite the crash.

.NET Reflector 8.2 release notes

July 16th, 2013

This minor release includes:

- Support for Visual Studio 2013
- Support for .NET 4.5.1
- On-hover hex/decimal value conversion in Reflector Desktop
- Local variable highlighting in Reflector Desktop
- Code Map view in Reflector Desktop
- Fix for the Enable just my code bug in the Visual Studio extension

The full .NET Reflector 8.2 release notes are available at the .NET Reflector Blog.

.NET Reflector 8.1 release notes

May 7th, 2013

This minor release includes fixes for issues with Visual Studio crashes, and a licensing bug.

The full .NET Reflector 8.1 release notes are available at the .NET Reflector Blog.

.NET Reflector 8.0 release notes

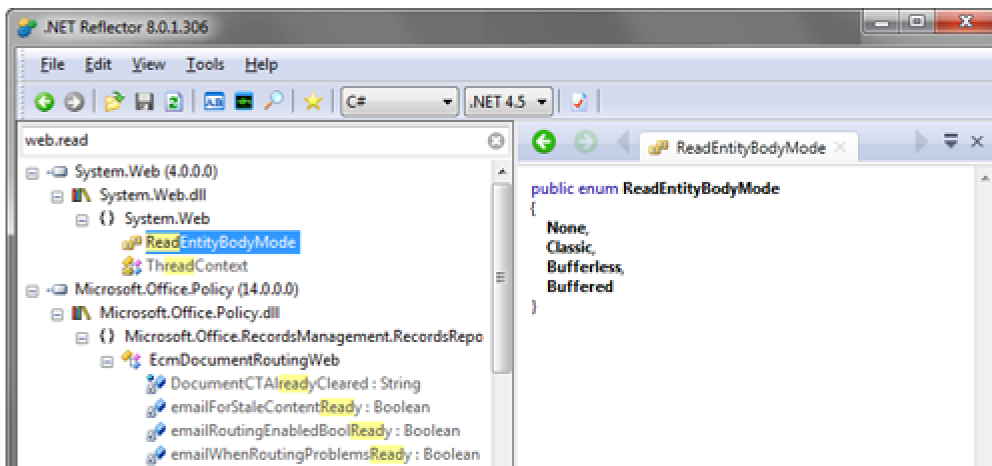
February 20th, 2013 - Version 8.0.2.313 patch release

On 20/02/2013 we released a small patch fix for .NET Reflector. This release includes a fix for an issue with repeated Visual Studio crashes that had been affecting some users.

January 14th, 2013

Search and filtering

Version 8 of .NET Reflector introduces a new search filter to the object browser in both Reflector Desktop and the Visual Studio extension:



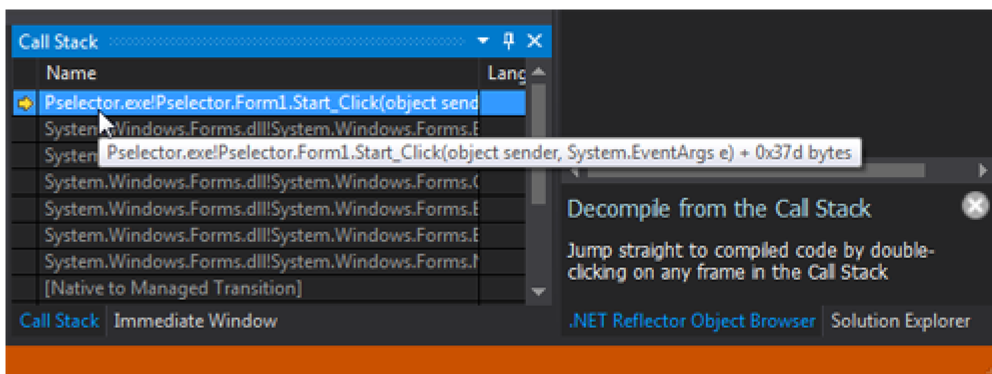
When you type in the search box, the contents of the object explorer are filtered based on your search terms.

The search includes basic scoping. So, for example, typing *web.read* would filter the object explorer to show only items with names containing "*re ad*" whose parent items contain "*web*".

Debugging in Visual Studio

The ability to generate PDB files for decompiled code was introduced in version 7.5

Version 8 improves on the debugging workflow in Visual Studio by allowing you to navigate to decompiled code from frames in the call stack, and by decompiling automatically if a debugging session encounters an exception in code without source. Additionally, setting a breakpoint in decompiled code now triggers automatic PDB generation, making it easier to step through 3rd party code:



Known issue - upgrading from version 7.x

If you are upgrading from .NET Reflector version 7 you may see incorrect text in the Check For Updates notification screen.

The check For Updates mechanism may show version 8 as a free trial.

This is not true, and updating will install the full version of .NET Reflector 8.0

Version 8 was made available as a free upgrade to all customers with a version 7 license, and should activate normally without requiring you to enter your serial number.

.NET Reflector 7.7 release notes

October 12th, 2012

- Collapse All Assemblies powercommand has now moved to file menu
- Collapse All Assemblies powercommand now has updated icon
- Collapse Assembly option added to assembly root node context menu
- Copy As powercommand moved to context menu
- Import/Export Assembly list powercommand moved to file menu
- Open file location moved to to assembly root node context menu
- Open with Powercommand added to context menu
- Open Zip by dragging zip into Reflector
- Referenced by Powercommand added to analyzer
- Removed older non-functional powercommands
- Archived less commonly used powercommands into powercommands addin
- Re-ordered Reflector context menu
- Added new icons for bookmark/remove bookmark
- Refresh icons in Visual Studio Extension
- Reflector intercept of f12 functionality in visual studio is now togglable.

.NET Reflector 7.6 release notes

July 16th, 2012

- Installer now "per-user" again, making management via VS smoother
- Coupled with work in Build 7, gives Reflector a true installer
  Installation process now attempts to elevate by default
    - Known issue: Installing alongside Reflector 7.6.0.604 causes duplicate menus in visual Studio
- Updating Core tool splash screen
- Adding an installer
- Better support for SharePoint DLLs
- Further improvements and bug fixes for Async support
- Adding a dynamic welcome screen
- Extending our C#5 async functionality support
- Continued re-plumbing Reflector UI to better support Dev11 theming
  ROB & context menus now accept theming correctly
    - Known bug: some text in the ROB is rendered too light
- Continued improvements of C#5 async support.
  Support for most new async methods in System.IO
    - Known issue: nested Try blocks aren't currently handled very well
    - If we're not confident about async decompilation, Reflector emits code for .NET 4.0
- Updated "about" dialog in standalone tool
- Control over whether Delphi & Oxygene are available decompilation options
- WinMD files are now discoverable on both 32-bit and 64-bit machines.
- Continued work on theming support for Dev 11
- Further development on C#5 async functionality support
- Added early theming support for Dev 11
- Added early support C#5 async functionality
- Added functional support for Dev 11
- Added support for .NET Framework 4.5 and it's assemblies
- Added support for WinRT winmd libraries.
- Removed VS Addin support for VS 2005 / 2008

.NET Reflector 7.5 release notes

Version 7.5.2.1

- Reflector automatically detects the winmd directory, and can load the contained files into the assembly list

Version 7.5.2

- The ROB now remembers its state each time you close & re-open Visual Studio
- Check For Updates will automatically update the Visual Studio Integration from older add-ins and packages

Version 7.5.1.13

- Fixed several com exceptions, mostly relating to race conditions
- Updated the trial dialog string

February 13th, 2012 - version 7.5

- Many string changes to make things easier to understand
- Assorted design changes to dialogs and screens
- Improvements to the installation experience of the package
- Improved the speed of type by type decompilation
- Improved handling of the case where the add-in is superseded by the package (we remove the old menu items)
- Error reporting experience is more fine-tuned. Users have options to be notified of work-arounds and fixes
- Hide menu items such as "enable debugging", instead of just disabling them
- Make "enable debugging" work on all items of the tree, rather than just the top level assembly item
- "Go to decompiled definition" more commonly enabled.
- Setting a break point causes the necessary PDB file to be automatically generated
- Decompiled code can now be stepped-into like any other code
- Attempting to step into inaccessible code triggers instructions on how to debug it correctly
- Multiple PDBs can now be generated in parallel
    - PDB generation can currently take a little time, especially if you're generating several or the assemblies are particularly large
- The trial dialog screen has been updated to be more informative and clear
- Re-introduced "Go to Decompiled Definition" right-click menu item
- Improved support for 'Go to Definition'(F12) in code without source
- A number of simple usability enhancements
- A number of licensing and installation bug fixes
- Added support for 'Go to Definition'(F12) in code without source
- Improved path to decompiled code
- Improved path to pdb generation for code you want to debug
- Completed move to a VS Package
- Started transition from a VS Add-in to a VS Package
- Added support for VS11
- Turned on SmartAssembly feature usage reporting on by default for all EA builds
- Added loaded project references to the Reflector Object Browser (ROB)
- Double click on any type in the 'ROB' to decompile in a new VS editor pane
- Added a Reflector 'Go to Definition' context menu item to navigate through code without source
- Added a new 'Reflector Object Browser'(ROB) into Visual Studio (will eventually offer decompilation by type inside VS)
- Changed the way Reflector shows the version number to display the correct build number
- Bug fixes:
    - "Open in Reflector" context menu not working.
    - XAML Source Code is incorrectly delimited.
    - Missing end tag in XAML Translation.
    - Code generation for different versions of the same assembly
    - 'Flatten Namespaces' dialog
    - Various decompilation and assembly-loading problems have been fixed drop us a note if you'd like to know more.

Note: This is not an exhaustive list. This has been compiled from blog and forum posts, but will much more representative from V7.5 onwards.

.NET Reflector 7.4 release notes

November 12th, 2011

- Improved translation of hitherto less common decompilation cases (e.g. Lambda expressions).
- Improved handling of exceptions.
- Better handling of decompilation cases involving obfuscation.
- A sprinkling of UX improvements that you won't overtly notice, but which will make life more pain-free.

.NET Reflector 7.3 release notes

July 15th, 2011

- BAML to XAML decompilation in the core product
- Fixed a bug that caused Reflector to hang on some systems when opened from Windows Explorer shell integration.

.NET Reflector 7.2 release notes

July 7th, 2011

- Fixed the start-up performance problem that was plaguing V7.2.
- Many decompilation improvements, particularly when dealing with expressions and LINQ queries.
- Backwards compatibility for add-ins built for previous versions of .NET Reflector.
- Many additional bug fixes to cure stability problems, along with a number of UI glitches.

.NET Reflector 7.1 release notes

May 4th, 2011

- Improved VB support
- Better handling of resource types
- Lots of improvements to C# decompilation
- Addition of command line activation/deactivation
- Fixes for 105 other bugs, including a number of stability improvements:
    - VB ByRef argument handled incorrectly.
    - Incorrect do-while-continue generated in C#.
    - Export Assembly Source Code creates an invalid VB project file.
    - "A break point could not be inserted at this location" error.

.NET Reflector 7.0 release notes

March 7th, 2011

Features and Enhancements

.NET Reflector is a class browser, analyzer, and decompiler for .NET assemblies.

Highlights in this new release include:

- Better decompilation, including the addition of iterator block (yield) support, as well as improvements to the .NET 4.0 decompilation,
- Tabbed decompilation,
- Ability to decompile and explore source code for referenced assemblies in Visual Studio,
- Support for Silverlight XAP files
- PowerCommands integration

How to Install and Use the Visual Studio Add-in

To install the Visual Studio add-in:

1. Unzip everything into a suitable location.
2. Run **Reflector.exe**. Then click **Tools > Visual Studio and Windows Explorer** Integration on the main menu, and select the versions of Visual Studio in which you want to install the add-in.

You can then open Visual Studio, and use the **.NET Reflector** menu to select referenced assemblies you would like to debug into. These assemblies are then decompiled to C# or VB. You can then step into them whilst debugging, set breakpoints, see the values of variables and parameters, etc. In fact, the only debugger functionality that will not work is edit and continue.

You can also decompile and explore source if you right-click on a project reference, and then click **Decompile and Explore** on the context menu.

To remove the Visual Studio add-in:

1. On .NET Reflector's **Tools** menu, click **Visual Studio and Windows Explorer** and clear the appropriate **Visual Studio Integration** chec kbox. The next time you open Visual Studio the add-in will be removed.

Supported .NET Framework Versions

.NET Reflector requires Microsoft .NET Framework 3.5 or later to run, but can open assemblies compiled against any version of the .NET framework, or Mono.

Supported OS Versions

- ®Windows XP SP2 or later, ®Windows Server 2003, ®Windows Vista, ®Windows Server 2008 & 2008 R2, ®Windows 7

Supports 32-bit and 64-bit versions of all listed operating systems, where applicable.

Supported Visual Studio Versions

The following versions of Microsoft Visual Studio are supported by the .NET Reflector add-in:

- Visual Studio 2005, 2008, and 2010

Improvements

- Introduction of a tabbed browsing model.
- UI improvements to make features more discoverable.
- Inclusion of Jason Haley's PowerCommands add-in.
- Improves to decompilation, such as handling iterator blocks.

.NET Reflector 6.5 release notes

July 14th, 2010

This release fixes a number of bugs in version 6.1. It also adds support for some of the new language features in .NET 4:

- The dynamic type in C# 4
- Co- and Contra- variance markers in interfaces and delegates
- The new expansion of the lock() { } statement
- Optional parameters

.NET Reflector Pro 6.5 release notes

July 14th, 2010

In response to customer feedback, this release changes the installation method for the .NET Reflector Visual Studio add-in so that the add-in is no longer installed automatically. This version also includes the bug fixes and support for new features in .NET Reflector version 6.5.

.NET Reflector and .NET Reflector Pro 6.1 release notes

February 24th, 2010

These releases fix several problems that were present in the 6.0 release:

- Support for using a copy of Reflector.cfg stored alongside Reflector.exe has been re-enabled so users upgrading from 5.x releases will not lose their settings.
- Fixed unhandled exception on exit of Visual Studio when .NET Reflector add-in used in conjunction with TestDriven.NET add-in.
- Added better support for dealing with framework assemblies, which only contain meta-data, in the “Referenced Assemblies” folder.
- Fixed problem where attempted decompilation with CppCliLanguage add-in would lead to display of a page on the Redgate website.
- Added option to activate .NET Reflector Pro to .NET Reflector menu in Visual Studio after receiving feedback from a number of users that it was hard to figure out how to activate the product.

.NET Reflector 6.0 release notes

February 9th, 2010

.NET Reflector is a class browser, analyzer, and decompiler for .NET assemblies.

.NET Reflector 6 includes a new Visual Studio add-in that allows developers to jump into Reflector from your source code.

This version of .NET Reflector supports:

- .NET 4.0 assemblies
- Opening assemblies from the Global Assembly Cache
- Improved graphics and usability

.NET Reflector Pro 6.0 release notes

February 9th, 2010

The new .NET Reflector Pro is an add-in to Visual Studio that lets you debug third-party code and assemblies. It decompiles the assembly to either C# or VB code, and lets you step through it in Visual Studio 2008, just as you would do with your own code. This is all controlled from a Visual Studio add-in, so you don't need to leave your coding environment.

- Full support for .NET 1.0, 1.1, 2.0, 3.0, 3.5 and 4.0
- Decompile an entire assembly to either C# or VB to view and debug in Visual Studio
- Step-through debugging of any assembly in Visual Studio (as long as it's not obfuscated):
    - Step into and set breakpoints anywhere in any assembly
    - Watch variables in the decompiled code
    - Use Visual Studio's advanced debugging features in decompiled code: Set Next Statement, modify variable values, and dynamic expression evaluation in the immediate window

Support for Visual Studio 2005 and 2008 deprecated

To make sure we can stay focused on the building the best possible decompiler and Visual Studio debugging enhancer, we've made the tough decision to deprecate add-in support for Visual Studio 2005 and 2008.As more and more developers are upgrading their IDE, the vast majority of Reflector users are now working in Visual Studio 2010.

What exactly does this mean for you?

The Visual Studio extensions in future versions of .NET Reflector Pro & VSPro will only support Visual Studio 2010 (or higher). If you need to work in Visual Studio 2005 or 2008, you can download a "sunset" build of .NET Reflector 7.5 below, which will support older versions of Visual Studio.

Please bear in mind that this sunset build will not contain the full set of features available in the more current versions of Reflector.

How can I use .NET Reflector with Visual Studio 2005 or 2008?

Download the sunset build of .NET Reflector 7.5 below, open it up on the machine running the older version of Microsoft's IDE, and then run Reflector as usual. Running .NET Reflector will allow you to install the Visual Studio extension, and applying your current license key will enable you to use the tool as normal.

Download 7.5 build