

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.

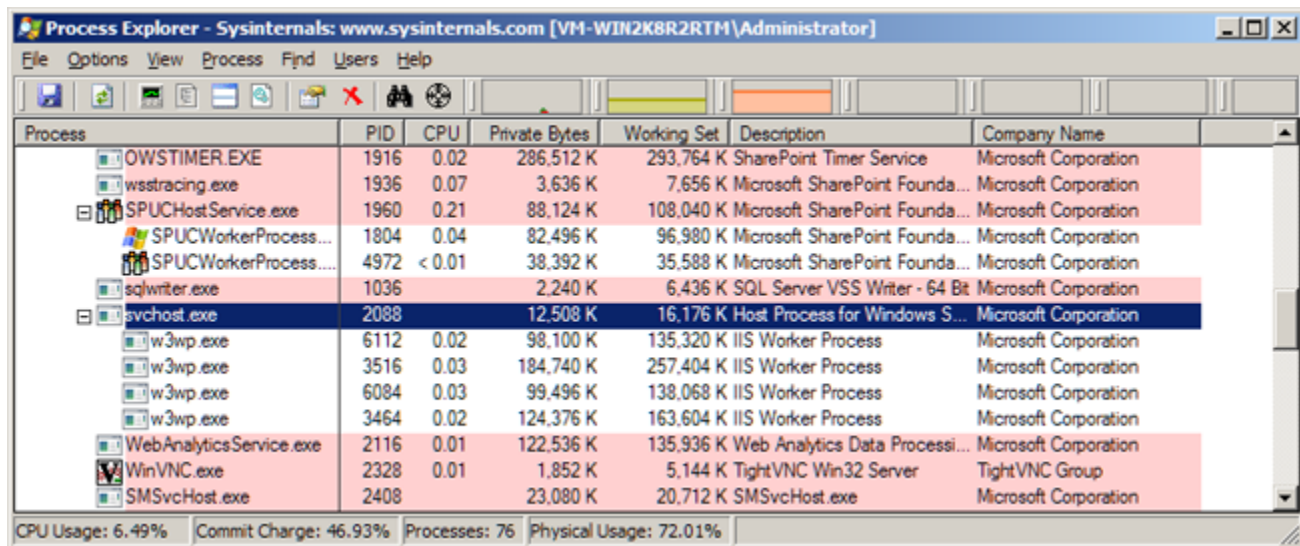
Clive's investigation

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

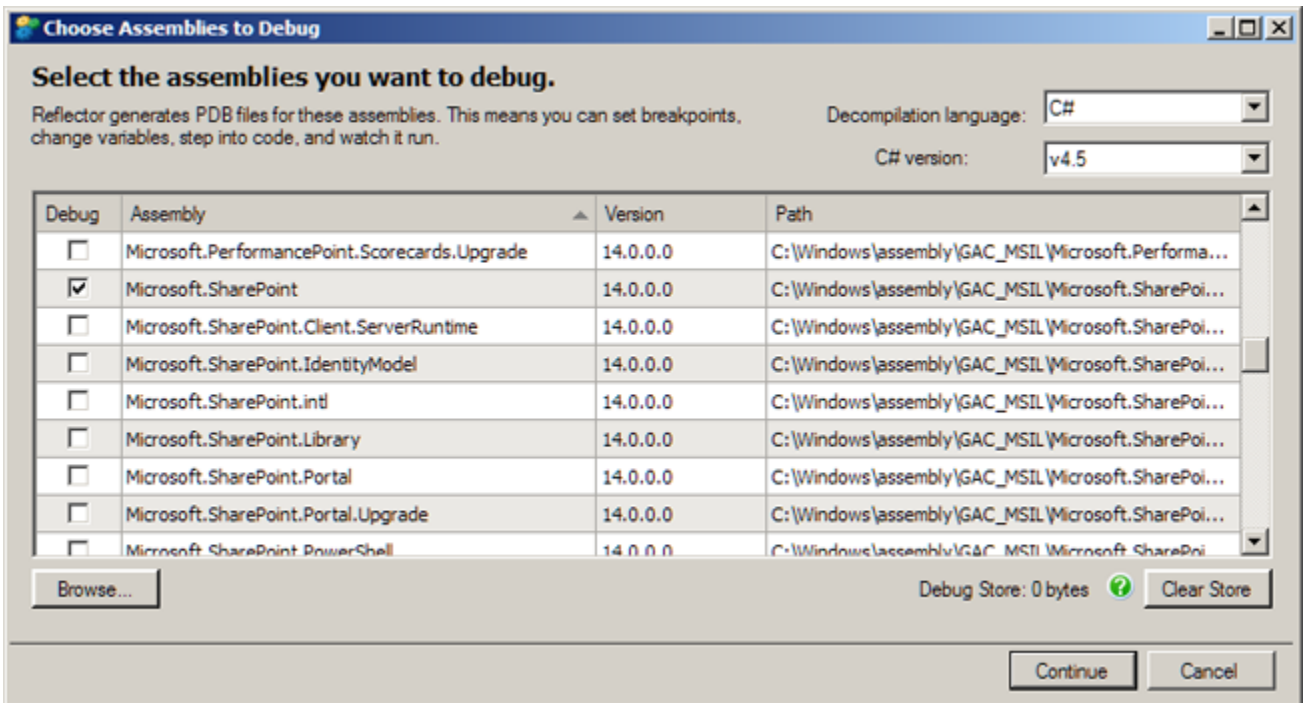
Initially, I kicked off a web site and watched the `w3wp.exe` instances being created using Process Explorer:



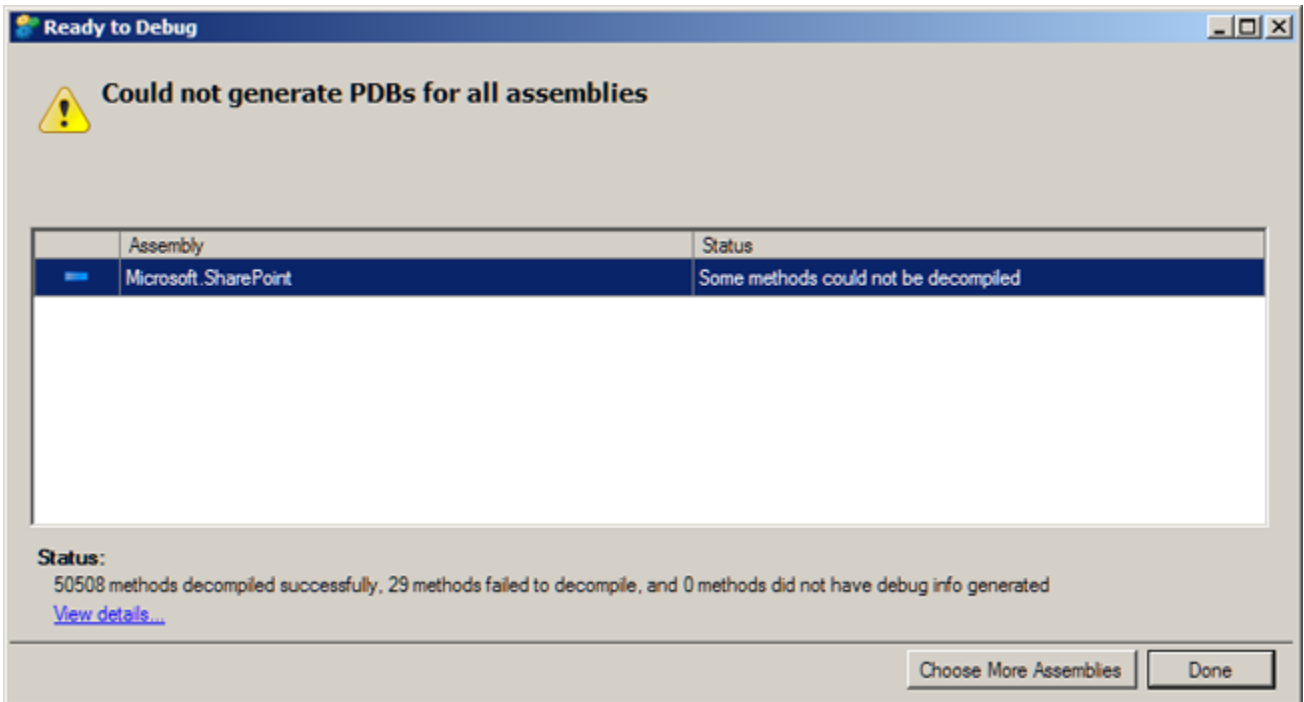
The screenshot shows the Process Explorer window from Sysinternals. The title bar reads 'Process Explorer - Sysinternals: www.sysinternals.com [VM-WIN2K8R2RTM\Administrator]'. The menu bar includes File, Options, View, Process, Find, Users, and Help. The toolbar contains icons for file operations and process management. The main pane displays a list of processes with columns for Process, PID, CPU, Private Bytes, Working Set, Description, and Company Name. The processes listed include OWSTIMER.EXE, wstracing.exe, SPUCHostService.exe, SPUCWorkerProcess..., sqlwriter.exe, svchost.exe, w3wp.exe (multiple instances), WebAnalyticsService.exe, WinVNC.exe, and SMSvcHost.exe. The status bar at the bottom shows CPU Usage: 6.49%, Commit Charge: 46.93%, Processes: 76, and Physical Usage: 72.01%.

Process	PID	CPU	Private Bytes	Working Set	Description	Company Name
OWSTIMER.EXE	1916	0.02	286,512 K	293,764 K	SharePoint Timer Service	Microsoft Corporation
wstracing.exe	1936	0.07	3,636 K	7,656 K	Microsoft SharePoint Founda...	Microsoft Corporation
SPUCHostService.exe	1960	0.21	88,124 K	108,040 K	Microsoft SharePoint Founda...	Microsoft Corporation
SPUCWorkerProcess...	1804	0.04	82,496 K	96,980 K	Microsoft SharePoint Founda...	Microsoft Corporation
SPUCWorkerProcess....	4972	< 0.01	38,392 K	35,588 K	Microsoft SharePoint Founda...	Microsoft Corporation
sqlwriter.exe	1036		2,240 K	6,436 K	SQL Server VSS Writer - 64 Bit	Microsoft Corporation
svchost.exe	2088		12,508 K	16,176 K	Host Process for Windows S...	Microsoft Corporation
w3wp.exe	6112	0.02	98,100 K	135,320 K	IIS Worker Process	Microsoft Corporation
w3wp.exe	3516	0.03	184,740 K	257,404 K	IIS Worker Process	Microsoft Corporation
w3wp.exe	6084	0.03	99,496 K	138,068 K	IIS Worker Process	Microsoft Corporation
w3wp.exe	3464	0.02	124,376 K	163,604 K	IIS Worker Process	Microsoft Corporation
WebAnalyticsService.exe	2116	0.01	122,536 K	135,936 K	Web Analytics Data Processi...	Microsoft Corporation
WinVNC.exe	2328	0.01	1,852 K	5,144 K	TightVNC Win32 Server	TightVNC Group
SMSvcHost.exe	2408		23,080 K	20,712 K	SMSvcHost.exe	Microsoft Corporation

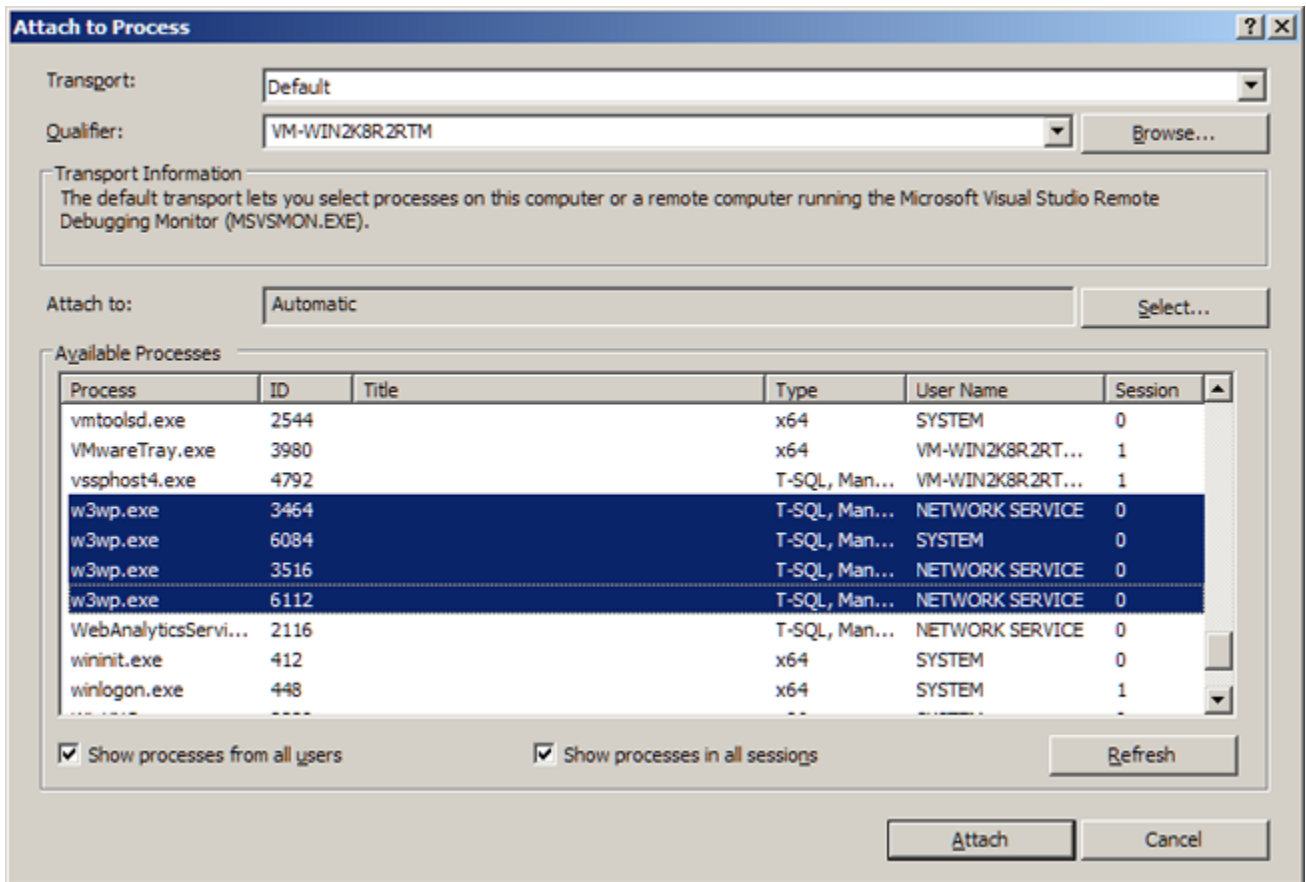
To enable debugging, I then decompiled the SharePoint assembly using Reflector VSPRO:



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

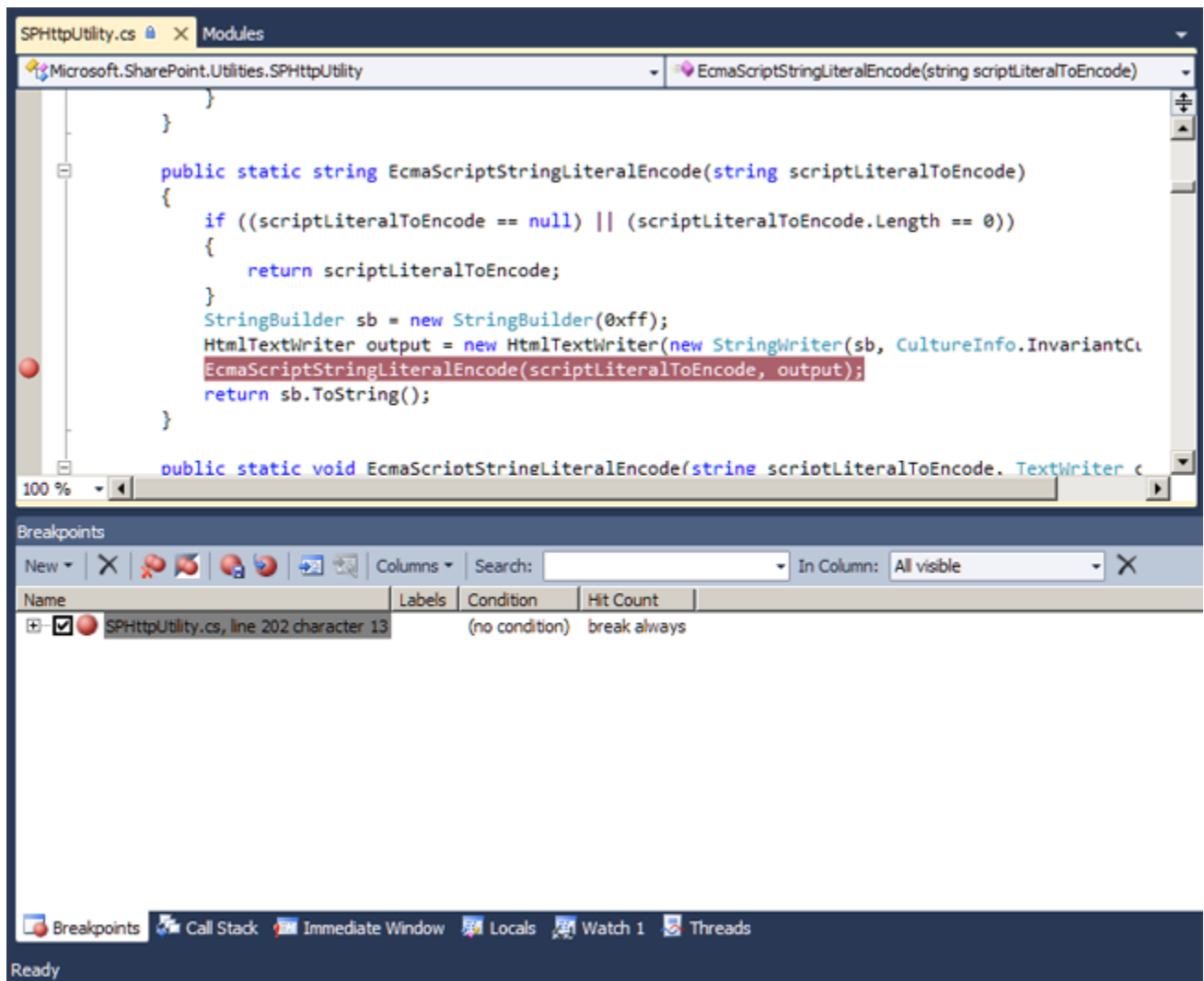


But it's still possible to attach the debugger to the four worker processes, and start to look at what's going on:

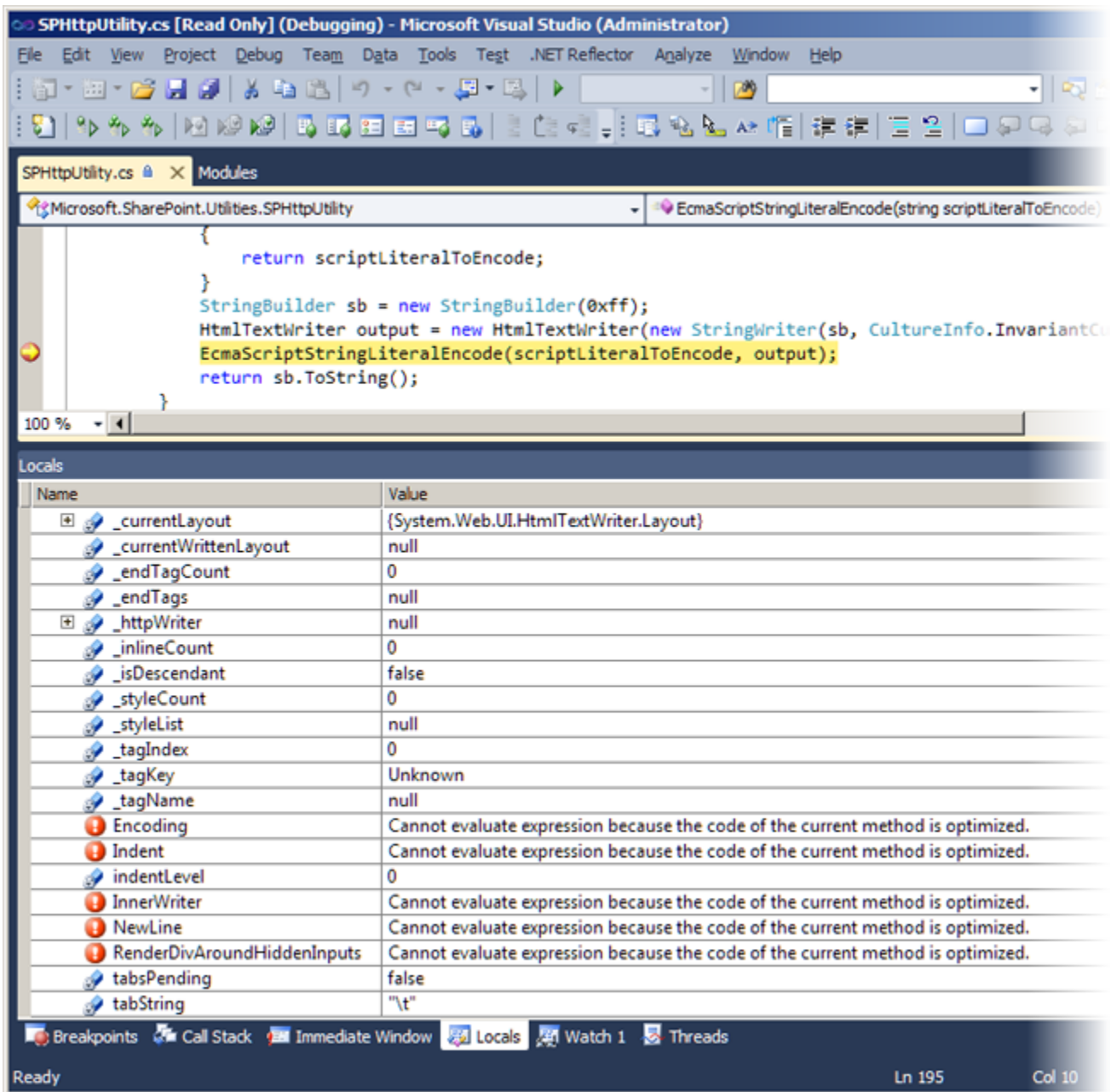


In order to start debugging, I needed to locate a source file corresponding to a class that I knew was going to be called. So I navigated into the Reflector cache directory and found the file `SPHttpUtility`, which I knew would contain the code for the class of the same name.

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:



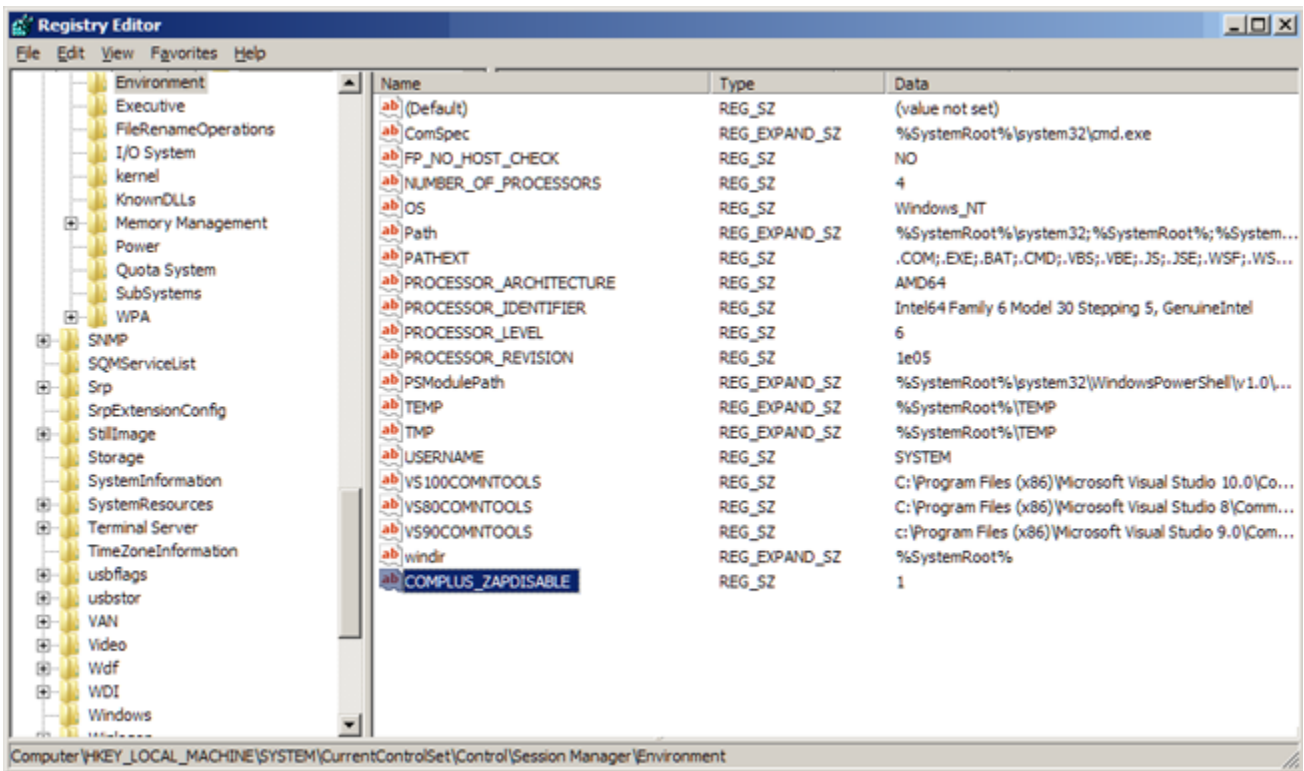
Without the local values, we can't follow the flow of data through the code, and we can't debug as accurately as we would like.

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:

```

Administrator: Visual Studio Command Prompt (2010)

Directory of C:\Windows\assembly\GAC_MSIL\Microsoft.SharePoint\14.0.0.0__71e9bce111e9429c
e111e9429c
12/10/2012  15:39    <DIR>          .
12/10/2012  15:39    <DIR>          ..
25/01/2012  14:49        16,516,968  Microsoft.SharePoint.dll
12/10/2012  11:25             ??  Microsoft.Sharepoint.ini
                2 File(s)          16,517,045 bytes
                2 Dir(s)          7,979,155,456 bytes free

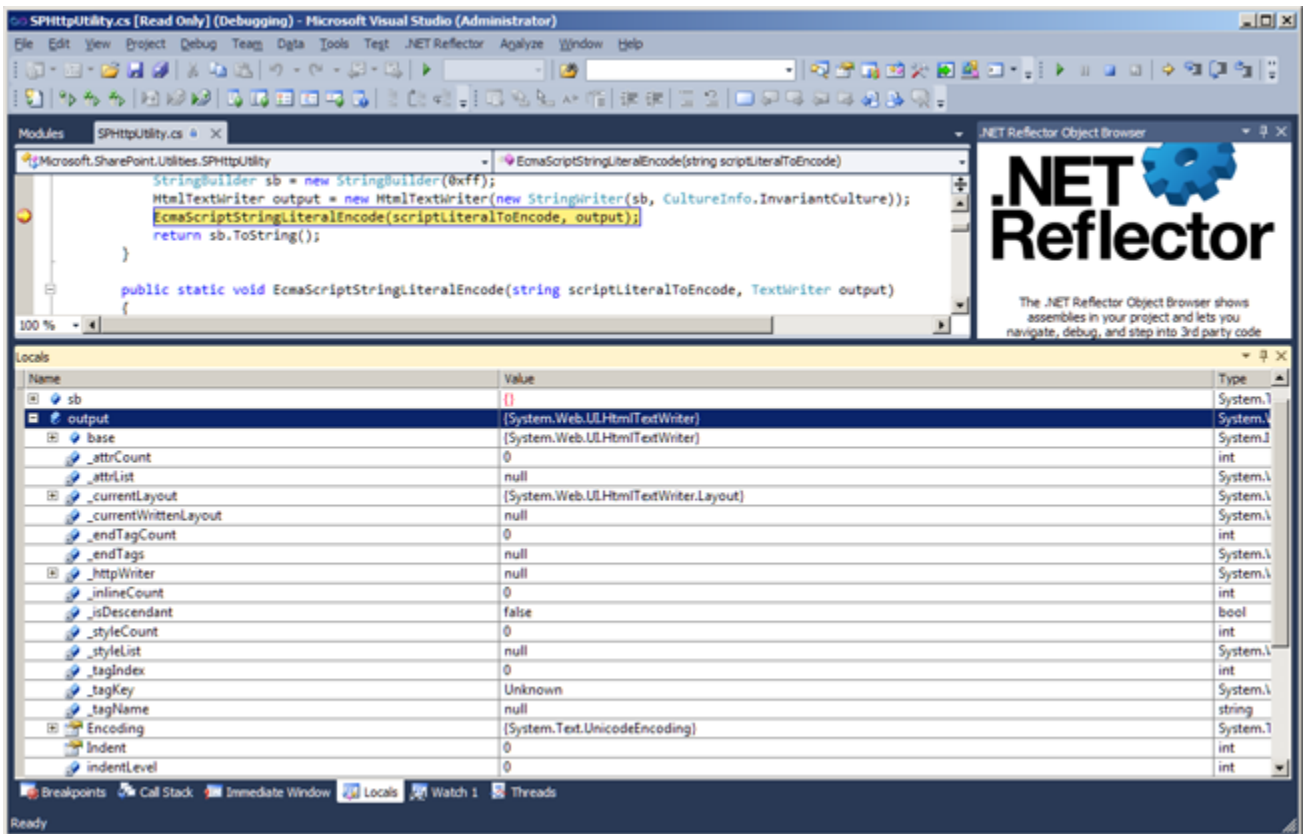
C:\Windows\assembly\GAC_MSIL\Microsoft.SharePoint\14.0.0.0__71e9bce111e9429c>typ
e Microsoft.Sharepoint.ini
[.NET Framework Debugging Control]
GenerateTrackingInfo=1
AllowOptimize=0

C:\Windows\assembly\GAC_MSIL\Microsoft.SharePoint\14.0.0.0__71e9bce111e9429c>

```

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.