# 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 w3wpprocesses.

## 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:

🂐 Process Explorer - Sysinternals:	www.sy	sinternals.	com [VM-W	/IN2K8R2RTM	\Administrator]		
Elle Options View Process Find I	Jsers H	elp					
J 🛃 🛃 🚍 🖻 🚍 🚳 🚰	×   #	) 😔 🛛					
Process	PID	CPU Pr	vate Bytes	Working Set	Description	Company Name	▲
OWSTIMER.EXE	1916	0.02	286,512 K	293,764 K	SharePoint Timer Service	Microsoft Corporation	
wsstracing.exe	1936	0.07	3,636 K	7,656 K	Microsoft SharePoint Founda	Microsoft Corporation	
SPUCHost Service.exe	1960	0.21	88,124 K	108,040 K	Microsoft SharePoint Founda	Microsoft Corporation	
Arr 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	
w 3wp.exe	6084	0.03	99,496 K	138,068 K	IIS Worker Process	Microsoft Corporation	
w 3wp.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	*
CPU Usage: 6.49% Commit Charge: 4	6.93%	Processes: 7	6 Physical U	lsage: 72.01%			1.

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

	enerates PDB files for these assemblies. This means you ca ables, step into code, and watch it run.	C# version: v4.5	2	
Debug	Assembly	Version	Path	4
	Microsoft.PerformancePoint.Scorecards.Upgrade	14.0.0.0	C:\Windows\assembly\GAC_MSIL\Microsoft.Perform	ma
	Microsoft.SharePoint	14.0.0.0	C:\Windows\assembly\GAC_MSIL\Microsoft.ShareF	Poi
	Microsoft.SharePoint.Client.ServerRuntime	14.0.0.0	C:\Windows\assembly\GAC_MSIL\Microsoft.ShareF	°oi
	Microsoft.SharePoint.IdentityModel	14.0.0.0	C:\Windows\assembly\GAC_MSIL\Microsoft.ShareF	°oi
	Microsoft.SharePoint.intl	14.0.0.0	C:\Windows\assembly\GAC_MSIL\Microsoft.ShareF	Poi
	Microsoft.SharePoint.Library	14.0.0.0	C:\Windows\assembly\GAC_MSIL\Microsoft.ShareF	Poi
	Microsoft.SharePoint.Portal	14.0.0.0	C:\Windows\assembly\GAC_MSIL\Microsoft.ShareF	°oi
	Microsoft.SharePoint.Portal.Upgrade	14.0.0.0	C:\Windows\assembly\GAC_MSIL\Microsoft.ShareF	oi
	Microsoft SharePoint PowerShell	14000	C+\Windows\assembly\GAC_MSTI_Wirrosoft_ShareF	ni 🔳

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

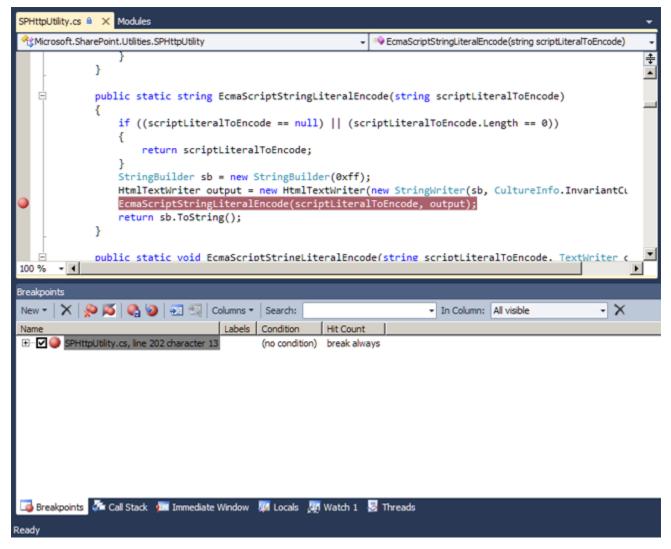
2	Ready to Debug Could not generate PDBs for all as		
ſ	Assembly	Status	
	Microsoft.SharePoint	Some methods could not be decompiled	
	Status: 50508 methods decompiled successfully, 29 methods failed <u>View details</u>	to decompile, and 0 methods did not have debug info generated	
		Choose More Assemblies Don	e

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

-	Default					
ualifier:	VM-WIN:	K8R2RTM		•	Browse	
ransport Information - The default transport   Debugging Monitor (MS		ect processes on this computer or a remote comp E).	outer running the I	Microsoft Visual Studio	Remote	
tach to:	Automat	c			<u>S</u> elect.	
vailable Processes	ID	Title	Type	User Name	Session	
vmtoolsd.exe	2544		x64	SYSTEM	0	_
VMwareTray.exe	3980		x64	VM-WIN2K8R2RT	1	
vssphost4.exe	4792		T-SQL, Man	VM-WIN2K8R2RT	1	
w3wp.exe	3464		T-SQL, Man	NETWORK SERVICE	0	
w3wp.exe	6084		T-SQL, Man	SYSTEM	0	
w3wp.exe	3516		T-SQL, Man	NETWORK SERVICE	0	
w3wp.exe	6112		T-SQL, Man	NETWORK SERVICE	0	
WebAnalyticsServi	2116		T-SQL, Man	NETWORK SERVICE	0	
wininit.exe	412		x64	SYSTEM	0	
winlogon.exe	448		x64	SYSTEM	1	•
Show processes fro	m all <u>u</u> sers	Show processes in all s	essions		Refresh	

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:

SPHttpUtility.cs [Read Only] (Debugging	) - Microsoft Visual Studio (Administrator)						
Elle Edit View Project Debug Team Data Tools Tegt .NET Reflector Analyze Window Help							
V							
SPHttpUtility.cs A X Modules							
Microsoft.SharePoint.Utilities.SPHttpUtility	<ul> <li>EcmaScriptStringLiteralEncode(string scriptLiteralToEncode)</li> </ul>						
} StringBuilder sb HtmlTextWriter ou	<pre>{     return scriptLiteralToEncode; } StringBuilder sb = new StringBuilder(0xff); HtmlTextWriter output = new HtmlTextWriter(new StringWriter(sb, CultureInfo.InvariantC EcmaScriptStringLiteralEncode(scriptLiteralToEncode, output);</pre>						
}	ig();						
100 % 🔹 📢							
to a to							
Locals	Webs.						
Name	Value						
_currentLayout	{System.Web.UI.HtmlTextWriter.Layout}						
currentWrittenLayout							
_endTagCount	0						
endTags	null						
🕀 🕜 _httpWriter	null						
	0						
.jsDescendant	false						
	0						
.styleList	null						
tagIndex	0						
🕜 _tagKey	Unknown						
🕜 _tagName	null						
Encoding	Cannot evaluate expression because the code of the current method is optimized.						
Indent	Cannot evaluate expression because the code of the current method is optimized.						
ge indentLevel	0						
InnerWriter	Cannot evaluate expression because the code of the current method is optimized.						
NewLine	Cannot evaluate expression because the code of the current method is optimized.						
RenderDivAroundHiddenInputs	Cannot evaluate expression because the code of the current method is optimized.						
🕜 tabsPending	false						
🕜 tabString	"\t"						
👼 Breakpoints 🛛 🖓 Call Stack 🚈 Immediate	Window 🧱 Locals 🎘 Watch 1 😞 Threads						
Ready	Ln 195 Col 10						

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 environ ment variable in the process that loads the CLR itself.

	-	Name	Type	Data
— Executive		ab (Default)	REG_SZ	(value not set)
— FileRenameOperations		ab ComSpec	REG_EXPAND_SZ	%SystemRoot%\system32\cmd.exe
I/O System		ab FP_NO_HOST_CHECK	REG_SZ	NO
		ab NUMBER_OF_PROCESSORS	REG SZ	4
- KnownDLLs		ab OS	REG SZ	Windows_NT
Memory Management		abPath	REG EXPAND SZ	%SystemRoot%\system32;%SystemRoot%;%System
- Power		ab PATHEXT	REG SZ	.COM; EXE: .BAT; .CMD; .VBS; .VBE; .JS; .JSE; .WSF; .WS.
Quota System		ab PROCESSOR ARCHITECTURE	REG SZ	AMD64
SubSystems		ab PROCESSOR IDENTIFIER	REG SZ	Intel64 Family 6 Model 30 Stepping 5, GenuineIntel
E- WPA		ab PROCESSOR LEVEL	REG SZ	6
B- SNMP		ab PROCESSOR REVISION	REG SZ	1e05
SQMServiceList		ab PSModulePath	REG EXPAND SZ	%SystemRoot%\system32\WindowsPowerShell\v1.0\.
SrpExtensionConfig		ab TEMP	REG EXPAND SZ	%SystemRoot%\TEMP
E Stillmage		ab TMP	REG_EXPAND_SZ	%SystemRoot%\TEMP
Storage		abUSERNAME	REG_SZ	SYSTEM
SystemInformation		ab VS100COMNTOOLS	REG_SZ	C:\Program Files (x86)\Microsoft Visual Studio 10.0\Co.
SystemResources		ab VS80COMNTOOLS	REG_SZ	C:\Program Files (x86)\Picrosoft Visual Studio 10.0(co.
Terminal Server		ab VS90COMNTOOLS	REG_SZ	c: Program Files (x86) Microsoft Visual Studio 8 (Comm. c: Program Files (x86) Microsoft Visual Studio 9.0 (Com.
TimeZoneInformation		ab windir		%SystemRoot%
🗉 🌆 usbflags		COMPLUS ZAPDISABLE	REG_EXPAND_SZ	765ystemRoot76
usbstor		COMPLUS_ZAPOISABLE	REG_SZ	1
NAN VAN				
- Video	_			
e 🚺 wdf				
E WDI				
	-1			

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_71e9bc					
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 77 Microsoft.Sharepoint.ini</dir></dir>					
2 File(s) 16,517,045 bytes 2 Dir(s) 7,979,155,456 bytes free					
C:\Windows\assembly\GAC_MSIL\Microsoft.SharePoint\14.0.0.071e9bce111e9429c>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:

ः SPHttpUtility.cs [Read Only] (Debugging) - Microsoft Visual Studio (Admi	nistrator)	a Di X					
Ele Edit View Project Debug Team Data Tools Test .NET Reflector							
□-□-≌」2  2  2  2  2  0+♡-□+□+□  >   □  2  -□+□  2  2  2  2  2  2  2  2  2  2  2  2  2							
Modules SPHttpUtility.cs + X	NET Refector Object transer	* 8 X					
🌱 (Microsoft, SharePoint, Utilities, SPHittpUtility • 🗣 EcmaScriptStringUteralEncode(string scriptLiteralToEncode) • 💽 🚺							
StringBuilder sb = new StringBuilder(0xff); HtmlTextWriter output = new HtmlTextWriter/	new StringWriter(sb, CultureInfo.InvariantCulture));						
EcmaScriptStringLiteralEncode(scriptLiteral	ToEncode, output);						
return sb.ToString();		tor					
}							
public static void EcmaScriptStringLiteralEncod	<pre>le(string scriptLiteralToEncode, TextWriter output) The .NET Reflector Object Bro </pre>	wser shows					
100 % - 4	assembles in your project an	d lets you					
	navigate, debug, and step into 3	rd party code					
Locals		* # ×					
Name	Value	Type 🔺					
🗷 🔮 sb	0	System.1					
E 🕏 output	(System.Web.ULHtmlTextWriter)	System.V					
E 🖗 base	(System.Web.UI.HtmlTextWriter)	System.1					
⊘ _attrCount	0	int					
g _attrlist	null	System.\					
E ge_currentLayout	(System.Web.UI.HtmlTextWriter.Layout) null	System.\					
	0	System.\ int					
♂ _endTags	null	System.\					
E A httpWriter	null	System.v					
2 inlineCount	0	int					
isDescendant	false	bool					
	0	int					
.styleList	null	System.\					
	0	int					
	Unknown	System.\					
	null	string					
E 😁 Encoding	{System.Text.UnicodeEncoding}	System.1					
🚰 Indent	0	int					
ge indentLevel	0	int 💌					
📑 Breakpoints 🖓 Call Stack 🚛 Immediate Window 🏭 Locais 🛒 Watch 1	Threads						
Dawfy							

### 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.