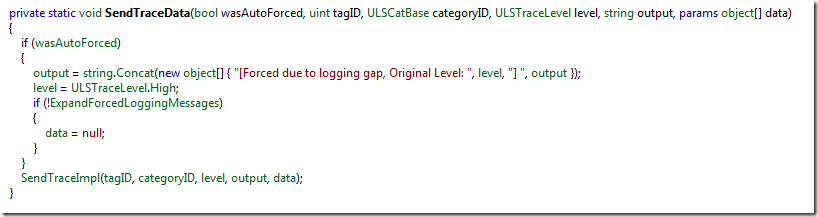
**[ULS\_EXPAND\_FORCED\_LOGGING\_MESSAGES\_” Environment Variable Explained](http://blog.falchionconsulting.com/index.php/2014/07/_uls_expand_forced_logging_messages_-environment-variable-explained/" \o "\“_ULS_EXPAND_FORCED_LOGGING_MESSAGES_\” Environment Variable Explained)**

Have you ever been looking through the SharePoint ULS logs trying to troubleshoot one issue or another and come across entries such as this:

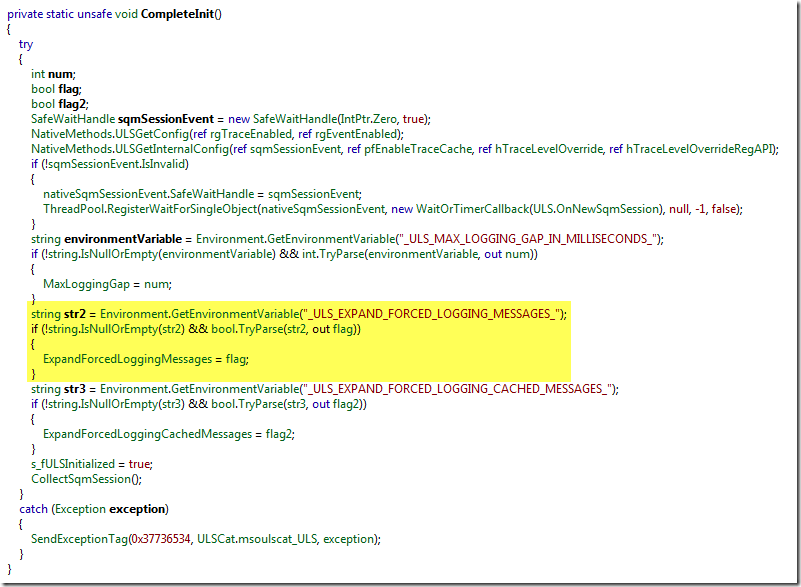
[Forced due to logging gap, Original Level: <TraceLevel>] <Some message> {0}

I was recently working with a client who noticed an message like this in the ULS logs and was baffled by the fact that the {0} was not being replaced with the relevant data referred to in the message text. I ‘d noticed this in the past but never bothered looking into it to figure out what was up with it so I decided to crack open Reflector and poke around a bit.

What I found was that there was a method, **SendTraceData()**, which is part of the **Microsoft.SharePoint.Diagnostics.ULS** class and it is within this method that the *[Forced due to logging gap…]*message is appended to the string that is to be written to the ULS logs (this happens when the time between log writes exceeds the default 50ms). Within the routine that is adding this string there is a test to see if the **ExpandForcedLoggingMessages** property is **false**, and if it is then set whatever data was provided to **null**, thereby clearing the relevant information that would be used in any subsequent string.Format() call to add the data to the message. Here’s a screenshot showing that code so you can see what I mean:



So the trick to figuring out why the {0} is not being replaced with the appropriate value is to understand how that **ExpandForcedLoggingMessages** property is set. I did some more digging with the Reflector analyze tool and found that in the **CompleteInit()** method of the same **Microsoft.SharePoint.Diagnostics.ULS** class there is code which sets the property based on the value of the **\_ULS\_EXPAND\_FORCED\_LOGGING\_MESSAGES\_** environment variable:



This variable, by default, is not set (and I can’t find one lick of documentation about it) but if we create the variable and set it to a value of **true** and reboot the server we will find that suddenly the values are being populated (or expanded) on that server (for this to take affect farm wide you need to do this on each server in the farm).

To set the variable you can either do it manually via the computer properties or you can use this simple PowerShell command:

[System.Environment]::SetEnvironmentVariable("\_ULS\_EXPAND\_FORCED\_LOGGING\_MESSAGES\_", "true", "Machine")

Don’t forget to reboot the machine after you set the variable.

Because I didn’t want to wait for a particular event to occur to validate that this actually worked I decided to create a simple test script which called the SendTraceData() method. Because the containing class is internal I had to use some reflection to make this work but you shouldn’t need to run this code – I only include it here for completeness to show how I validated that the change worked as expected:

$bindings = @("InvokeMethod", "NonPublic", "Instance", "Static", "Public", "GetField", "GetProperty")

$asm = [System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SharePoint")

$type = $asm.GetType("Microsoft.SharePoint.Diagnostics.ULS")

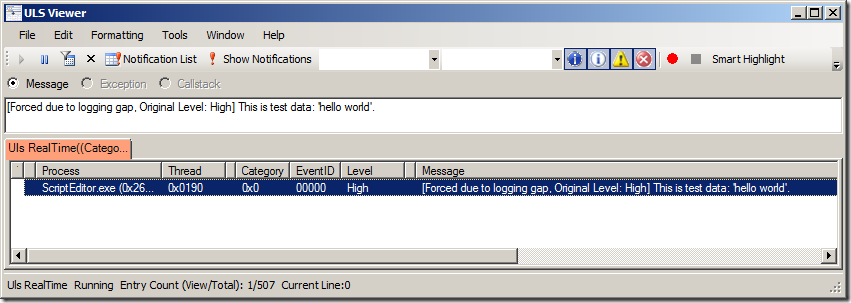
$ulscatType = $asm.GetType("Microsoft.SharePoint.Diagnostics.ULSCat")

$ulscat = $ulscatType.GetProperty("msoulscatUnknown").GetValue($null, $null)

$method = $type.GetMethod("SendTraceData", $bindings)

$method.Invoke($null, @($true, [System.UInt32]0, $ulscat, 20, "This is test data: '{0}'.", @("hello world")))

If we look at the ULS logs using ULSViewer we can see our test event got created and the data was preserved:



And now you know how to get that {0} populated with data which may or may not be useful but I personally would rather have more data available when troubleshooting an issue and hate the idea of having possibly key information nullified.

Oh, and while you’re looking at the CompleteInit() method you may have noticed that there’s also a **\_ULS\_MAX\_LOGGING\_GAP\_IN\_MILLISECONDS\_** environment variable and a **\_ULS\_EXPAND\_FORCED\_LOGGING\_CACHED\_MESSAGES\_** environment variable. The later is similar to the one I’ve been talking about and is related to messages you see that look like this:

[Forced due to logging gap, cached @ <Date>, Original Level: <TraceLevel>] <Some message> {0}

So if you’re going to enable the one you should probably enable the other as well.

The **\_ULS\_MAX\_LOGGING\_GAP\_IN\_MILLISECONDS\_** environment variable controls how big, in milliseconds, the maximum logging gap should be. By default the value is 50 milliseconds. I would recommend *not* changing this value but if you want events written more frequently to the ULS logs then you can lower the value or likewise if you want the events written less frequently you can bump up the value (note that you won’t lose events, it just affects how often SharePoint is writing to the file).

I hope this information helps you with your troubleshooting adventures!

-Gary