



"I'll show you how to view incoming WMI queries as close to real time as possible, then correlate where the query is coming from."

An Easier Way to View Incoming WMI Queries

A helpful script outputs timestamped WMI queries in an easy-to-read format

Windows Management Instrumentation (WMI) is a widely used technology on both Windows server and client, employed for tasks such as inventory of hardware and software, determining whether services or processes are running, and many others. Usually, WMI just works, but when it doesn't and you need to investigate it, doing so can be very difficult. For instance, one of the most common problems with WMI is high CPU spikes for the one of the provider processes (wmiprvse.exe). The provider process is where the WMI queries are executed, and the WMI service process (svchost.exe) is where the query results are returned to the process that executed the WMI query. As an administrator, you've probably either run into a high-CPU-usage issue with WMI or needed to view incoming WMI queries. In the course of investigating a high-CPU issue, it would be very beneficial to see the queries being executed and determine where the query came from.

Unfortunately, viewing incoming WMI queries is at best a cumbersome process that can frustrate administrators trying to answer the simple question "What WMI queries are being executed on my system?" In this article, I'll show you how to view the incoming queries as close to real time as possible, then correlate where the query is coming from. This will enable you to immediately troubleshoot common WMI high-CPU issues without having to perform complicated debugging steps or scan through hundreds of log file events—or make that dreaded support call.

Retrieving WMI Trace Logs

As described in the blog post "WMI Debug Logging," tinyurl.com/3a6b7ls, you can turn on the tracing mechanism inside the Event Viewer to view the tracing logs for WMI. However, because of the large number of events collected, it's difficult to determine the latest incoming queries, when they executed, and where they are coming from. Even the filtering mechanism built into the Event Viewer menu does not provide the ability to filter only on "WMI queries."

Enter wevtutil.exe, the Windows events command-line utility. Wevtutil, included in Windows Vista and later, lets you retrieve information about event logs as well as export, run queries, and set properties on the log files. For instance, Wevtutil can set the tracing flag on the WMI-Activity event log either locally or remotely, so that you can start receiving WMI trace logs in the Event Viewer interface.

Here is the exact syntax to set the tracing flag on the WMI-Activity log file (start an elevated command prompt), along with a message displayed after you enter the command:

```
C:\Windows\system32>wevtutil sl Microsoft-Windows-WMI-Activity/Trace /e:true
```

```
**** Warning: Enabling this type of log clears it. Do you
want to enable and
clear this log? [y/n]:
y
```

Notice that you are asked to clear the log, which is required to enable the tracing option. (Note: It is not required to clear the log to disable the tracing flag.) There is also an /r option that lets you set the tracing flag on a remote system.

After the tracing flag is set, you will start to receive "Informational" events in the WMI-Activity event log. However, the verbosity is high, and even if you wanted to use the filtering mechanism, you still would not be able to filter for just WMI queries. Again we will use wevtutil.exe to help us manually view the incoming WMI queries. Here is the syntax to view the WMI queries using Wevtutil (after enabling the tracing flag and from an elevated command prompt).

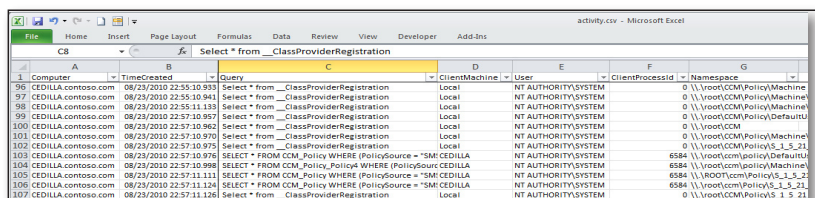
```
<Event xmlns='http://schemas.microsoft.com/win/2004/08/
events/event'><System><Provider Name='Microsoft-
Windows-WMI-Activity' Guid='{1418ef04-b0b4-4623-bf7e-
d74ab47bbdaa}'/><EventID>1</EventID><Version>0
</Version><Level>4</Level><Task>0</Task><Opcode>0</Opco
de><Keywords>0x8000000000000000</Keywords><TimeCreated
SystemTime='2010-08-20T17:26:52.911870500Z'><Event
RecordID>17</EventRecordID><Correlation><Executio
n ProcessID='1156' ThreadID='8136' ProcessorID='1'
KernelTime='25' UserTime='28'><Channel>Microsoft-Windows-
WMI-Activity/Trace</Channel><Computer>Computer1
.na.corp.contoso.com</Computer><Security UserID='S-
1-5-18'></System><UserData><Operation_
xmlns:auto-
ns2='http://schemas.microsoft.com/win/2004/08/
events' xmlns='http://manifests.contoso.com/win/2006/
windows/WMI'><GroupOperationId>467724</GroupOperation
Id><OperationId>467725</OperationId><Operations>Start
IwbemServices::ExecQuery - select * from Win32_
OperatingSystem</Operation><ClientMachine>Computer1
</ClientMachine><User>na/user1</User><ClientProcessId>5900
</ClientProcessId><NamespaceName>\\.root\\cimv2</
NamespaceName></Operation_></UserData></Event>
```

Figure 1: Wevtutil output

■ WHAT WOULD MICROSOFT SUPPORT DO?

```
GroupOperationId = 467724; OperationId = 467725; Operation = Start
IwbemServices::ExecQuery - select * from Win32_OperatingSystem; ClientMachine =
User1; User = NAuser1; ClientProcessId = 5900; NamespaceName = \\.\root\cimv2
```

Figure 2: Weventutil output in text format



Computer	TimeCreated	Query	ClientMachine	User	ClientProcessId	Namespace
CECILIA.contoso.com	08/23/2010 22:55:10.943	Select * from __ClassProviderRegistration	Local	NT AUTHORITY\SYSTEM	0	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:55:10.943	Select * from __ClassProviderRegistration	Local	NT AUTHORITY\SYSTEM	0	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:55:11.139	Select * from __ClassProviderRegistration	Local	NT AUTHORITY\SYSTEM	0	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:57:10.967	Select * from __ClassProviderRegistration	Local	NT AUTHORITY\SYSTEM	0	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:57:10.962	Select * from __ClassProviderRegistration	Local	NT AUTHORITY\SYSTEM	0	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:57:10.970	Select * from __ClassProviderRegistration	Local	NT AUTHORITY\SYSTEM	0	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:57:10.975	Select * from __ClassProviderRegistration	Local	NT AUTHORITY\SYSTEM	0	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:57:10.976	SELECT * FROM CCM_Policy WHERE (PolicySource = "SM:CECILIA	Local	NT AUTHORITY\SYSTEM	6584	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:57:10.986	SELECT * FROM CCM_Policy WHERE (PolicySource = "SM:CECILIA	Local	NT AUTHORITY\SYSTEM	6584	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:57:11.111	SELECT * FROM CCM_Policy WHERE (PolicySource = "SM:CECILIA	Local	NT AUTHORITY\SYSTEM	6584	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:57:11.124	SELECT * FROM CCM_Policy WHERE (PolicySource = "SM:CECILIA	Local	NT AUTHORITY\SYSTEM	6584	\\.\root\cimv2\PolicyMachine
CECILIA.contoso.com	08/23/2010 22:57:11.135	Select * from __ClassProviderRegistration	Local	NT AUTHORITY\SYSTEM	0	\\.\root\cimv2\PolicyMachine

Figure 3: Output from WMIActivity.vbs script

```
c:\>wevtutil qe Microsoft-Windows-WMI-
Activity/Trace /q:"*[System[(Level=4
or Level=0) and (EventID=1)]]"
|findstr /i execquery
```

The first option `qe` means “query events from a log file”; then the name of the log file is entered (Microsoft-Windows-WMI-Activity/Trace). The `/q:` option says to filter the event log for just those specific events; in this case we are asking for Level 4 or Level 0 AND Event ID=1, which is the event ID that all WMI queries are logged as. Then we pipe the results to `findstr` (find string) and filter for just the `/execquery` commands.

The output from the previous command looks similar to that shown in Figure 1. This output is for just one query—imagine if you received hundreds of queries! That would be a lot of data to sort through. However, we can change the format of this output from XML, to text by making a small change to the `Weventutil` command:

```
C:\>wevtutil qe Microsoft-Windows-WMI-
Activity/Trace /q:"*[System[(Level=4
or Level=0) and (EventID=1)]]"
/f:text |findstr /i execquery
```

Notice the change we made was simply adding the `/f:` option, which means “format,” and supplying the format as text. Now our output, shown in Figure 2, is more manageable. Although this output is less verbose, it still too verbose for the simple task of viewing incoming WMI queries, plus it doesn’t provide any timestamp to indicate when the query occurred.

Automating WMI Activity Tracking

It would be helpful to have a script that you can schedule to run and would provide manageable output, show timestamps to indicate when a query occurred, and let you see the latest queries since the last time the script was run. And that is exactly what the Microsoft support team has done for you. We’ve created a script that shows you the incoming WMI queries and when they occurred.

Depending on the version of Windows used, you’ll be able to tell which machine and process ID the query came from, either locally or remotely.

The script, `WMIActivity.vbs`, queries for WMI events and outputs only the WMI query

activity. (You can download the script at www.windowsitpro.com, InstantDoc ID 125876; simply rename the text file containing the script with a `.vbs` extension to make it executable.)

Figure 3 shows the output in Excel spreadsheet format. Notice that the output includes several columns of data that give you as the administrator insight into when the query executed, the query itself, the machine where the query came from (ClientMachine), and the Client Process ID.

I should mention that the ClientMachine and ClientProcessID columns will be populated only for systems running Windows 7 or Windows Server 2008 R2 or later. Vista and Windows Server 2008 don’t include the code to capture the client that sent the query or the client process ID.

The first time the script is run, a timestamp is put into the registry under the key `HKLM\Software\WinITPro`. Each time the script is run, the timestamp is checked and only the log entries newer than the timestamp are pulled. That way you see only the newest queries rather than having to wade through much of the same information or navigate through a cumbersome Event Viewer-style menu.

You will need to format the `TimeCreated` column with a custom format. You can do so by selecting the entire column, right-clicking, and selecting `Format Cell`, as Figure 4 shows. You could also create a pivot table, so that you can view the queries from a “Count” perspective.

A Little Extra Troubleshooting Help

With help from the `WMIActivity.vbs` script file, you’ll be able to view the WMI queries being sent to your system as close to real time as possible. This script will give you some extra help in troubleshooting WMI high-CPU issues as well as better insight into what and who is interrogating your system.

InstantDoc ID 125876

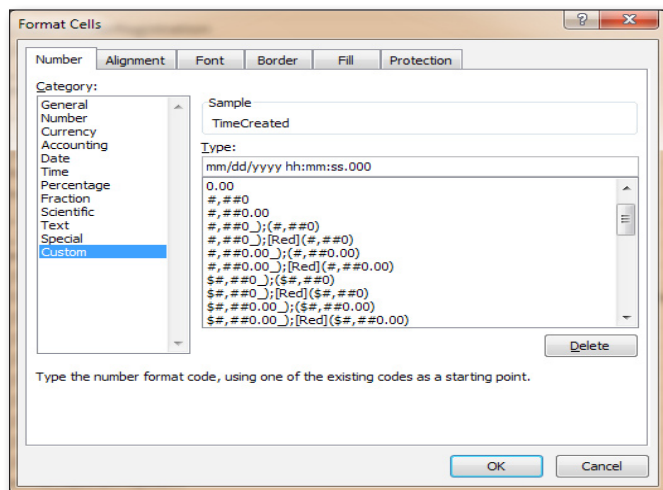


Figure 4: Formatting the TimeCreated column

