Microsoft

Troubleshooting with the Windows Sysinternals Tools

Guidance from the tools' creator

MARK RUSSINOVICH | AARON MARGOSIS

Microsoft

Troubleshooting with the Windows Sysinternals Tools

Mark Russinovich
Aaron Margosis

If you selected the Include Stack Traces option when you saved the XML, each event also includes a <*stack*> element. For events that include a stack trace, the <*stack*> element contains one or more <*frame*> child elements containing the following elements:

- **depth** The zero-based position in the stack, with 0 at the top of the stack.
- **address** The return address of the stack frame in the process' virtual memory.
- **path** The path of the module loaded at the stack frame's return address.
- **location** If the Resolve Stack Symbols option was selected and the symbol could be resolved for this stack frame, the <*location*> element shows the symbol name + offset, and if possible the source file and line number. Otherwise, <*location*> shows the module name and the offset from the module's base address.

Opening saved Procmon traces

Procmon can open traces saved in its native PML file format. Procmon running on an x86 system can open only traces captured on an x86 system. Procmon running on an x64 system can open x86 or x64 traces, but it must be in the correct mode for the architecture. To open an x86 trace on x64, Procmon must be started with the **/Run32** command-line option to run the 32-bit version of Procmon. Note that when running in 32-bit mode on x64, Procmon cannot capture events.

If Procmon is already running, open the File Open dialog box by clicking the Open toolbar icon. You can open a Procmon log file from the command line with the **/OpenLog** command-line option as follows:

- For x86 traces on x64:

  ```
  procmon.exe /run32 /openlog logfile.pml
  ```

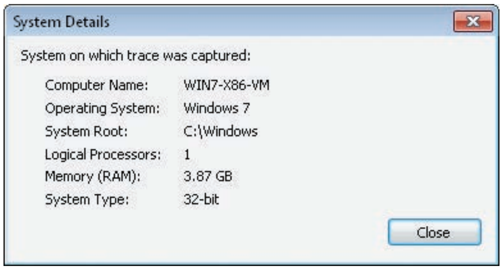
- For everyplace else:

  ```
  procmon.exe /openlog logfile.pml
  ```

Each time you run Procmon, it registers a per-user file association for .PML to the current Procmon path with the **/OpenLog** option. So after you have run Procmon one time, you can open a Procmon log file simply by double-clicking it in Explorer. If you run Procmon with the **/Run32** option, that option will also be added to the file association. So if you're analyzing a set of 32-bit logs, you can do so from Explorer. The **/Run32** option will be removed from the association if you later run Procmon without that option.

Procmon does not require administrative rights to open an existing log file, and it won't prompt for elevation on Windows Vista and newer versions when started with the **/OpenLog** option. However, if you later want to capture events, you'll need to restart Procmon with administrative rights.

The log file includes information about the system on which the data was collected, including the computer name, operating system version and whether it is 32-bit or 64-bit, system root path, number of CPUs, and amount of RAM. You can see this in the System Details dialog box (shown in Figure 5-19) on the Tools menu.



System on which trace was captured:

Computer Name:	WIN7-X86-VM
Operating System:	Windows 7
System Root:	C:\Windows
Logical Processors:	1
Memory (RAM):	3.87 GB
System Type:	32-bit

FIGURE 5-19 System Details dialog box.

To view symbols in stack traces, the system on which the trace was captured does not need to have debugging tools installed nor symbols configured, but the system on which the trace is viewed must have both. In addition, it must have access to symbol files and binaries for the trace system. For Windows files, the Microsoft public symbol server will usually provide these. Note that the 32-bit version of Procmon needs to load a 32-bit Dbghelp.dll. Because the 32-bit version of Procmon stores all its configuration settings in a different registry key from the 64-bit version, configure symbols for x86 traces after starting Procmon with the **/Run32** option.

Logging boot, post-logoff, and shutdown activity

Up to this point in the chapter, everything that has been described about Procmon assumes you're logged on at an interactive desktop. Procmon also provides ways to monitor system activity when no one has logged on and after users have logged off.

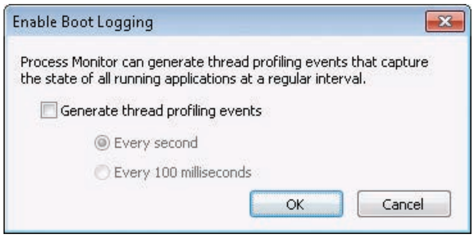
Boot logging

You can configure Procmon to begin logging system activity from a point very early in the boot process. This is the feature you need if you're diagnosing issues that occur before, during, or in the absence of user logon, such as those involving boot-start device drivers, autostart services, the logon sequence itself, or shell initialization. Boot logging also enables you to diagnose issues that occur during user logoff and system shutdown.

Boot logging is the only Procmon mode that is tolerant of hard resets. Because of this, it can help diagnose system hangs and crashes, including those occurring during startup or shutdown.

In addition to file, registry and process events, boot logs include Procmon-generated process profiling events.[1] When you choose Enable Boot Logging from the Options menu, Procmon also gives you the option to generate thread-profiling events with the dialog box shown in Figure 5-20, either once per second or ten times per second. You can click Cancel at this point if you decide not to enable boot logging. The Enable Boot Logging menu option shows a check mark when it is enabled; you can cancel boot logging by toggling that menu option. You can also enable boot logging by running Procmon with the **/EnableBootLogging** command-line option.

[1] The tracing of network events depends on Event Tracing for Windows (ETW) and is not available in boot logs.


Enable Boot Logging

Process Monitor can generate thread profiling events that capture the state of all running applications at a regular interval.

☐ Generate thread profiling events

● Every second

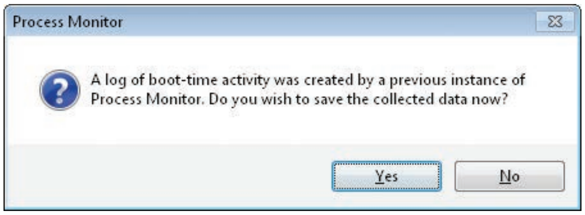
○ Every 100 milliseconds

FIGURE 5-20 Boot-logging options

When you enable boot logging, Procmon configures its driver to run as a boot start driver that loads very early in the boot sequence at the next system startup, before most other drivers. Procmon's driver will log activity into %windir%\Procmon.PMB, and it will continue logging through shutdown or until you run Procmon again. Thus, if you don't run Procmon during a boot session, you'll capture a trace of the entire boot-to-shutdown cycle. As a boot start driver, it remains loaded very late into the shutdown sequence.

After the boot-start driver loads, it changes its startup configuration to be a demand-start driver for subsequent boots. Consequently, when you enable boot logging, it is only for the next boot. To enable boot logging for subsequent boots, you must explicitly enable it again each time.

When you run Procmon, it looks to see whether an unsaved boot log has been generated, either from the current session or from a previous boot session. If Procmon finds one, it asks you whether you want to save the processed boot log output file and where you want to place it. (See Figure 5-21.) Procmon then opens and displays the saved log. If you do not save the boot log to another location, it will be overwritten the next time you capture a boot-time log. You can automate the converting of the unsaved boot log and skip the dialog box by running Procmon with the ***/ConvertBootLog pml-file*** option, which looks for an unsaved boot log, saves the captured data to the location that you specify, and then exits.


Process Monitor

A log of boot-time activity was created by a previous instance of Process Monitor. Do you wish to save the collected data now?

FIGURE 5-21 Procmon asks whether you want to save a boot log.

When looking at boot-time activity, remember that the System process is the only process early in a boot and that activity originating from the System process is filtered by default. Choose Advanced Output on the Filter menu to see System process activity.

If you configure boot logging and the system crashes early in the boot, you can deactivate the boot logging by choosing the Last Known Good option from the Windows boot menu. Press F8 during Windows startup to access this option.

Keeping Procmon running after logoff

Boot logging is the only option Procmon offers to capture events very late in the shutdown sequence. If you need to capture events that occur during or after user logoff but don't need a complete trace of the shutdown, boot logging always remains an option. However, in addition to the post-logoff data you want to capture, you'll end up with a log of the entire boot session from system startup on, which might be far more data than you want. Another option, then, is to start Procmon in a way that survives user logoff.

One way to monitor a user's logoff is to leverage terminal services, using either Fast User Switching or Remote Desktop. With the target user already logged on, start a new session as a different user and start Procmon. Switch back to the original user's session and log off. Return to the second session, and stop capturing events. Set a filter on the Session attribute to see only the events that occurred within the original user's terminal services session.

Another effective way to capture post-logoff activity is to use PsExec with the **–s** option to run Procmon as System in the same environment in which noninteractive System services run. There are some tricks to this, though, because you won't be able to interact with this instance of Procmon:

- You need to specify a backing file on the command line with **/BackingFile**. Remember that this setting sticks. So if you run Procmon and capture data again as System without specifying a different backing file, you'll overwrite your previous trace.
- You must specify **/AcceptEula** and **/Quiet** on the command line to ensure that Procmon doesn't try to display dialog boxes that cannot be dismissed.
- Procmon must be shut down cleanly. To do this without shutting the system down, you must run **Procmon /Terminate** in the exact same manner as the original command.

See the "Backing files" and "Automating Procmon: command-line options" sections in this chapter for more information about these options. See "Sessions, window stations, desktops, and window messages" in Chapter 2 to better understand the underlying concepts covered here. And see Chapter 7, "PsTools," for more information about PsExec.

Here is an example command line to start a Procmon trace that survives logoff:

```
PsExec -s -d Procmon.exe /AcceptEula /Quiet /BackingFile C:\Procmon.pml
```

And the following command line will stop that trace:

```
PsExec -s -d Procmon.exe /AcceptEula /Terminate
```

The PsExec **–d** option allows PsExec to exit without waiting for the target process to exit.

If a PsExec-launched instance of Procmon is running as System during a clean system shutdown, Procmon will stop logging when CSRSS tears down user-mode processes. To capture events beyond this point, boot logging is the only option.

Long-running traces and controlling log sizes

Procmon trace files can become very large, particularly with boot logging or other long-running traces. Therefore, Procmon provides several ways to control log file size.

Drop filtered events

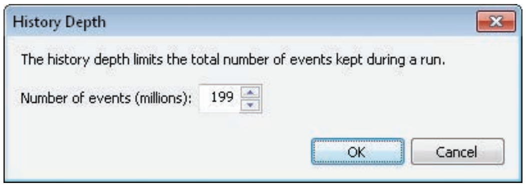
Ordinarily, Procmon will log all system activity, including events that are normally never displayed because of the active filters. That way, you always have the option to set a filter, explore the resulting output, and then change the filter to see a different set of output. However, if you know in advance of a long-running trace that you'll never need to see events for, you can keep them from taking space in the log by choosing the Drop Filtered Events option in the Filter menu.

When Drop Filtered Events is chosen, events that don't meet the filter criteria are never added to the log, reducing the impact on log size. Obviously, that event data cannot be recovered later. This option affects only newly collected events. Any events that were already in the log are not removed.

Note that filtering is not applied while a boot log is being collected, so Drop Filtered Events will not reduce disk usage impact during a boot log trace. But also note that the filters—and the Drop Filtered Events setting—are applied when the boot log is processed. So if you elect to drop events and need to see System process activity or other low-level events, make sure to choose Enable Advanced Output (Filter menu) before rebooting.

History depth

Process Monitor watches committed memory usage and stops capturing events when system virtual memory runs low. By opening the History Depth dialog box (shown in Figure 5-22) from the Options menu, you can limit the number of entries kept so that you can leave Process Monitor running for long periods and ensure that it always keeps the most recent events. The range goes from a minimum of 1 million to 199 million events. The default is 199 million.


History Depth

The history depth limits the total number of events kept during a run.

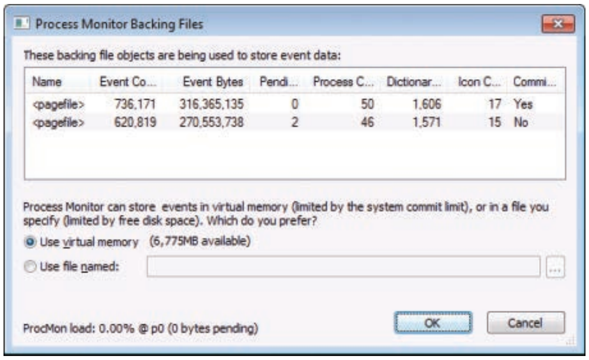
Number of events (millions): 199

FIGURE 5-22 History Depth dialog box.

Backing files

By default, Procmon uses virtual memory to store captured data. If virtual memory runs low, Procmon automatically stops logging and displays an error message. If your logging needs exceed the capacity of virtual memory, you can configure Procmon to store captured data to a named file on disk. The capacity limit when using a named file is the amount of free space on the hard drive.

You can configure and see information about backing files by choosing Backing Files from the File menu. The Process Monitor Backing Files dialog box, shown in Figure 5-23, opens. Backing file configuration changes take effect the next time you begin capturing a new log or clear the current log.



These backing file objects are being used to store event data:

Name	Event Co...	Event Bytes	Pendi...	Process C...	Dictionar...	Icon C...	Commi...
<pagefile>	736,171	316,365,135	0	50	1,606	17	Yes
<pagefile>	620,819	270,553,738	2	46	1,571	15	No

Process Monitor can store events in virtual memory (limited by the system commit limit), or in a file you specify (limited by free disk space). Which do you prefer?

[✓ Use virtual memory] (6,775MB available)

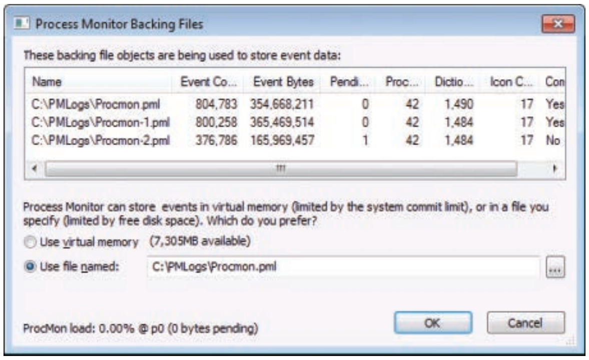
Use file named:

ProcMon load: 0.00% @ p0 (0 bytes pending)

FIGURE 5-23 Process Monitor Backing Files dialog box.

Note that if you choose a named file, Procmon might create additional files to keep individual file sizes manageable. Files will have the same base name, with an incrementing number appended, as shown in Figure 5-24. As long as the files are kept in the same directory and with the same base name, Procmon will treat the file set as a single log.

The Backing Files dialog box also displays diagnostic information, including the number of events captured and the number of processes observed.



These backing file objects are being used to store event data:

Name	Event Co...	Event Bytes	Pendi...	Proc...	Dictio...	Icon C...	Con
C:\PMLogs\Procmon.pml	804,783	354,668,211	0	42	1,490	17	Yes
C:\PMLogs\Procmon-1.pml	800,258	365,469,514	0	42	1,484	17	Yes
C:\PMLogs\Procmon-2.pml	376,786	165,969,457	1	42	1,484	17	No

Process Monitor can store events in virtual memory (limited by the system commit limit), or in a file you specify (limited by free disk space). Which do you prefer?

Use virtual memory (7,305MB available)

[✓ Use file named:] C:\PMLogs\Procmon.pml

ProcMon load: 0.00% @ p0 (0 bytes pending)

FIGURE 5-24 Process Monitor Backing Files dialog box with named files.