

On/Off Transition Performance Analysis of Windows Vista

July 31, 2009

Abstract

This document discusses the importance of on/off transition performance, methodologies for measuring this performance, and how to analyze the results. The information in this paper is intended to help OEMs and system analysts improve system response times.

This information applies to the Windows Vista® operating system.

This documentation accompanies the Windows® 2008 Software Development Kit release of the xperf tools.

For the latest information, see:

http://www.microsoft.com/whdc/system/sysperf/on-off_transition.msp

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, MSDN, PreFetch, Windows, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Document History

Date	Change
September 8, 2008	First publication

Contents

Introduction	6
Windows Vista On/Off Transitions.....	6
Power-On Transitions	7
Power-Off Transitions.....	7
Measuring On/Off Transition Performance	7
Creating a Baseline Measurement.....	7
Software Tools and Performance Data.....	8
Capturing a Trace.....	8
Dealing with Raw Dumps.....	9
The Effect of a Network Connection on Performance Measurements	10
Performance Analysis Overview	11
Boot Transition.....	11
Performance Analysis	11
Overview	12
Boot Traces	12
Capturing Boot Traces	12
Analyzing Boot Traces.....	13
Phases	13
The OS Loader Phase.....	13
MainPathBoot Phase.....	14
The PostBoot Phase.....	19
Key Issues.....	20
Suspend/Resume and Standby/Resume Transitions	20
Performance Analysis	20
Suspend Phase Overview.....	20
SuspendApps Subphase.....	Error! Bookmark not defined.
SuspendServices Subphase.....	Error! Bookmark not defined.
Query Devices Subphase	Error! Bookmark not defined.
SuspendDevices Subphase	Error! Bookmark not defined.
HiberfileWrite Subphase	Error! Bookmark not defined.
Resume Phase Overview.....	22
BIOSInit Subphase	Error! Bookmark not defined.
HiberfileRead Subphase	Error! Bookmark not defined.
ResumeDevices Subphase	Error! Bookmark not defined.
PostResume Subphase	Error! Bookmark not defined.
Standby/Resume Transition: Capturing Standby Traces	23
Hibernate/Resume Transition: Capturing Hibernate Traces	23
Suspend/Resume Transition: Capturing Resume Traces.....	23
Suspend/Resume Transition: Analyzing Captured Traces	23
Suspend Phase: SuspendApps Subphase	24
Suspend Phase: SuspendServices Subphase	24
Suspend Phase: QueryDevices and SuspendDevices Subphases	25
Resume Phase: ResumeDevices Subphase.....	25
Resume Phase: ResumeServices and ResumeApps Subphases	25
Resume Phase: PostResume Subphase	25
Summary.....	26
Shutdown Transition.....	26

Performance Analysis	26
Overview	26
Capturing Shutdown Traces.....	27
Analyzing Shutdown Phase Traces.....	27
User Session Shutdown.....	27
System Session Shutdown	28
Kernel Shutdown.....	28
Summary	28
Conclusions	29
Resources	29
Appendix: XML Reference	30
The Boot Process	31
boot	31
processSummary	32
expectedProcesses	32
unexpectedLonglived	33
unexpectedShortlived	33
unexpectedVeryShortlived	33
process	33
timing	34
interval	35
perProcess	35
perProcessCPUUsage	35
perPriority	36
perPriorityCPUUsage	36
cpuUsage	36
diskIO	37
byExtension	Error! Bookmark not defined.
byFile	37
file	38
byExtension	38
extension	38
services	39
serviceTransition	39
pnP	40
phase	40
pnPObject	40
groupPolicy	41
process	41
The Suspend Process	42
suspend	43
scenario	43
queryapps	44
queryservices	44
suspendapps	44
app	45
suspendservices	45
service	45
suspendshowui	46

superfetchpagein	46
suspendwinlogon	46
suspendlockpageablesections	46
presleepcallbacks	47
suspendswapiinworkerthreads	47
flushvolumes	47
querydevices	47
suspenddevices	48
resumedevices	48
device	48
driver	48
The Shutdown Process	49
shutdown	49
timing	50
perSessionInfo	50
sessionShutdown	50
preShutdownNotification	50
shutdownProcess	51
intervals	51
interval	52
unresponsiveServices	52
unresponsiveService	52
services	52
serviceTransition	52
groupPolicy	52

Introduction

Good performance during a Windows Vista® boot, shutdown, and other state transitions not only improves the perceived quality of a computer, but also greatly affects daily computer usage patterns and system reliability. You must therefore spend time improving performance and creating a fast and consistent on/off transition.

The importance of transition performance is highlighted in the following customer scenarios:

Frequent system updates protect Windows Vista users and introduce new features, but users quickly become frustrated by a lengthy update and reboot cycle.

Users often consider the time that is required to boot a computer to be a benchmark of operating system performance.

A lengthy shutdown time can irritate users and can increase service times for system administrators. Longer shutdown times can also adversely affect the reliability of mobile systems. For example, they create more risk of power being removed unexpectedly.

Mobile users rely on their computer's ability to do suspend/resume and hibernate/resume transitions. Suspend/resume performance is critical to maintaining data integrity. If the resume transition takes too long, users ignore suspend and select the less convenient option to shut down their computer instead.

Hibernate provides a way to store the system state without using any power. But as memory sizes increase, so do performance challenges such as responsiveness after the system has entered resume.

You can use the tools in the Windows Performance Tools Kit (xperf, xperfview, and xbootmgr) to measure and analyze the performance of these transitions. This paper describes techniques for performing a targeted analysis of Windows Vista power on/off transitions by focusing on CPU, disk, services, and Plug and Play issues. You can automate the generation of traces and view the results in an XML file. This consistent testing methodology allows for easier communication and more effective collaboration. The XML schema is described in the appendix to this paper.

Windows Vista On/Off Transitions

A computer that is running Windows Vista can transition between on/off states in the following ways:

Booting

Shutting down

Entering or resuming from standby (S3)

Entering or resuming from hibernate (S4)

Each transition is vulnerable to performance issues that are both general to all transitions and specific to each transition.

Powering-on Transitions

The powering-on transitions are as follows:

Boot (S5)

Resume from standby (S3)

Resume from hibernate (S4)

Powering-on transitions can be considered to consist of two main phases:

The first phase corresponds to the completion of the critical path such as the time that is required to create the desktop process.

The second phase corresponds to the time that is required for the system to reach an "idle" state, when the background service initialization is complete.

Each phase has unique challenges and vulnerabilities.

Powering-off Transitions

The powering-off transitions are as follows:

Shutdown

Suspend to standby (S3)

Suspend to hibernate (S4)

Powering-off begins when the transition is initiated and ends when the system is "off." To help analysis, you can divide the powering-on transition into phases.

Measuring On/Off Transition Performance

You can simplify measuring the performance of on/off transitions by using an accurate and repeatable testing methodology. A consistent approach makes it easier to compare the performance of applications, services, and drivers against a baseline system, without requiring a detailed knowledge of the transition.

The Windows Vista self-optimizing features make it difficult to create repeatable performance results. For example, for the boot prefetcher Windows Vista uses the disk I/O pattern from the previous boots to predict the I/O pattern of the next boot.

To help you create consistent measurements, the Windows Performance Tools Kit automates transition testing and ensures that all Windows Vista optimizations are performed.

Creating a Baseline Measurement

The recommended way to measure performance is to perform side-by-side testing on two systems that have identical hardware. The first system is used as a benchmark. It should be running a retail installation of Windows Vista that has all drivers installed and updated but has no additional features or applications. This setup lets you use the second system to perform an incremental evaluation of third-party extensions and other features. By performing tests against a benchmark system, you can evaluate features without a detailed knowledge of any internal processes.

Software Tools and Performance Data

The command-line tools in Table 1 measure performance and are in the Windows Performance Tools Kit as part of Windows Performance Analyzer tool suite. They rely on Event Tracing for Windows (ETW) to log events for disk I/O, CPU usage, Plug and Play, and other key indicators. The tools kit includes introductory documentation.

Table 1. Windows Performance Tools Kit Performance Analyzer Command-line Tools

Tool	Purpose
xbootmgr.exe	Automates collecting repeatable and reliable on/off performance traces. It integrates with the Windows Vista self-optimizing features to ensure that the test system was trained exactly like a real-world system.
xperf.exe	In context of on/off transition analysis, generates summary reports for all on/off transitions in XML and provides raw event dumps for detailed low-level investigations.
xperfview.exe	Provides a detailed visualization of performance data.

Capturing a Trace

Capturing and analyzing data for on/off transitions resembles capturing performance traces. You use the xperfview and xperf command-line tools to filter and analyze the data. However, some features are unique to on/off transitions. For example, Plug and Play device enumeration and initialization are unique to the boot process and do not occur during resume from standby or hibernate.

The trace capture process is as follows:

1. Use xbootmgr to capture the trace during the required on/off transition.
 2. Use xperf to filter the trace and create an XML summary or a raw data file.
- or-
- Use xperfview to create a graphical representation of the trace to aid analysis.

Table 2 shows the common command-line arguments that xbootmgr uses.

Table 2. Xbootmgr command-line arguments

Option	Purpose
-numRuns <i>n</i>	<i>N</i> specifies the number of traces to be collected.
-resultPath <i>path</i>	<i>Path</i> is the fully qualified path where the traces should be stored. Network paths are not supported.
-runTag <i>tag</i>	<i>Tag</i> is a string that is prepended to each trace file to make testing easier.
-noTraceFlagsInFileName	A deprecated option that generates simpler trace file names. This is a legacy command from when the trace flags were not logged to the trace file directly, and should no longer be used.
-help	Displays detailed command-line help and defaults.

For a full list of xbootmgr command, see the Help file that accompanies xbootmgr.exe.

The xperf tool generates specific summary reports for each transition. A typical xperf command to process a boot trace resembles the following command:

```
xperf -i trace.etl -o summary.xml -a boot
```

Xperf creates an XML output file that is generated in a logical hierarchy. To examine the file, use a viewer that supports expanding and collapsing nodes (such as Microsoft® Internet Explorer).

The xperf and xperfview tools use the service graph to help you examine on/off-specific transitions. This graph displays service startup and shutdown times for boot and shutdown traces. The CPU and disk resources data graph can also be useful.

The simplest way to identify performance issues is to compare the results for each phase against a baseline system.

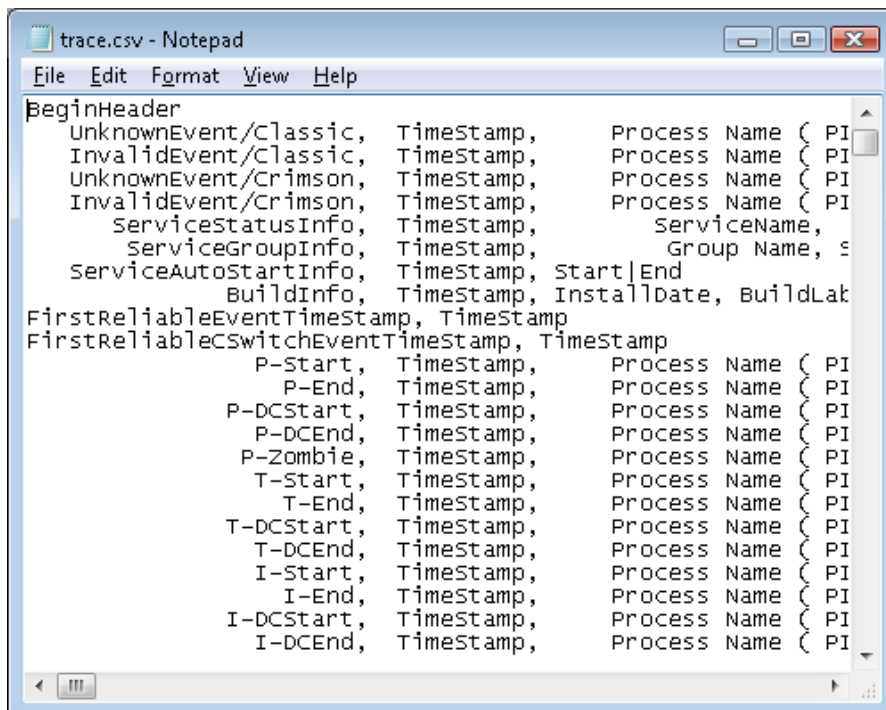
Note: Not all performance problems can be identified or root cause determined from the summary report. Deep analysis by using xperfview might be necessary.

Dealing with Raw Dumps

The xperf tool can generate a raw per-event dump of the trace and summary reports. You can look for some specific information in a raw trace. First, create the trace by typing the following at the command prompt:

```
xperf -i trace.etl -o trace.csv -a dumper
```

As you can see in Figure 1, the resulting dump file is very detailed. You might find it difficult to understand and use the file without a good comprehension of the system and events. However, the dump file contains some information that is currently not available in the summaries.



```

BeginHeader
UnknownEvent/Classic, Timestamp, Process Name ( PI
InvalidEvent/Classic, Timestamp, Process Name ( PI
UnknownEvent/Crimson, Timestamp, Process Name ( PI
InvalidEvent/Crimson, Timestamp, Process Name ( PI
ServiceStatusInfo, Timestamp, ServiceName,
ServiceGroupInfo, Timestamp, Group Name, s
ServiceAutoStartInfo, Timestamp, start|End
BuildInfo, Timestamp, InstallDate, BuildLak
FirstReliableEventTimestamp, Timestamp
FirstReliableCswitchEventTimestamp, Timestamp
P-Start, Timestamp, Process Name ( PI
P-End, Timestamp, Process Name ( PI
P-DCStart, Timestamp, Process Name ( PI
P-DCEnd, Timestamp, Process Name ( PI
P-Zombie, Timestamp, Process Name ( PI
T-Start, Timestamp, Process Name ( PI
T-End, Timestamp, Process Name ( PI
T-DCStart, Timestamp, Process Name ( PI
T-DCEnd, Timestamp, Process Name ( PI
I-Start, Timestamp, Process Name ( PI
I-End, Timestamp, Process Name ( PI
I-DCStart, Timestamp, Process Name ( PI
I-DCEnd, Timestamp, Process Name ( PI

```

Figure 1. A sample dump file created by xperf

The general format of the dump is as follows:

The first column represents the event type.

The second column represents the time stamp of the event in microseconds since the start of the trace.

Any remaining columns contain event-specific information.

The raw dump files are often tens or hundreds of megabytes and require filtering for effective analysis. The easiest way to filter the file is to use a utility such as `findstr.exe`, which is included with Windows Vista.

The `findstr.exe` arguments follow regular expression syntax. For command-line help, type the following:

```
findstr.exe /?
```

For example, to obtain a list of all events that refer to `winusb.dll` in the "trace.csv" dump file, type the following:

```
findstr winusb.dll trace.csv
```

Notes

Do not use the system during testing because activity will be included in the trace and might distort the analysis.

The test process includes many delays, during which countdown dialog boxes appear.

You must be a member of the Administrators group to run the `xbootmgr` tool, or you will be required to provide confirmation on each boot/resume. If User Account Control (UAC) is enabled, you must also provide confirmation.

Boot/shutdown testing requires that the user account be configured for auto-logon by using the Windows Vista `autologon.exe` power toy.

S3/S4 transitions are supported only when the **On resume, display logon screen** check box under **Screen Saver Settings** is cleared.

All results are stored in the directory that is specified by `-resultPath` argument or in the `xbootmgr.exe` directory by default.

Two files are created for each iteration: `*.CAB` and `*.ETL`. The `*.ETL` files are required for this analysis.

The `*.ETL` files should be copied to a separate Windows system for processing. The trace files can be large, and the processing operation can interfere with the state of the testing machine.

The Effect of a Network Connection on Performance Measurements

The presence of a network connection can influence the performance of on/off transitions. The following suggestions reduce the variability that networking can introduce:

Leave wireless adapters enabled, but not connected to any network.

Delete all persistent network connections before testing, by using the following command from an elevated command line:

```
net * /d
```

When the test system is joined to a domain, the group policies that the domain administrator deploys can affect performance. Therefore, consider group policies when you design your test scenarios.

Performance Analysis Overview

The following sections discuss the analysis of these on/off transitions:

Boot

Suspend/Resume and Standby/Resume

Shutdown

For each transition type, we will:

Look at the commands that are required to generate the trace.

Take an overview of the transition itself and point out potential vulnerabilities.

List the XML nodes to which you should pay particular attention in the trace summary.

Each on/off transition consists of multiple phases and subphases, as illustrated in Figure 2. This paper examines each phase in detail.

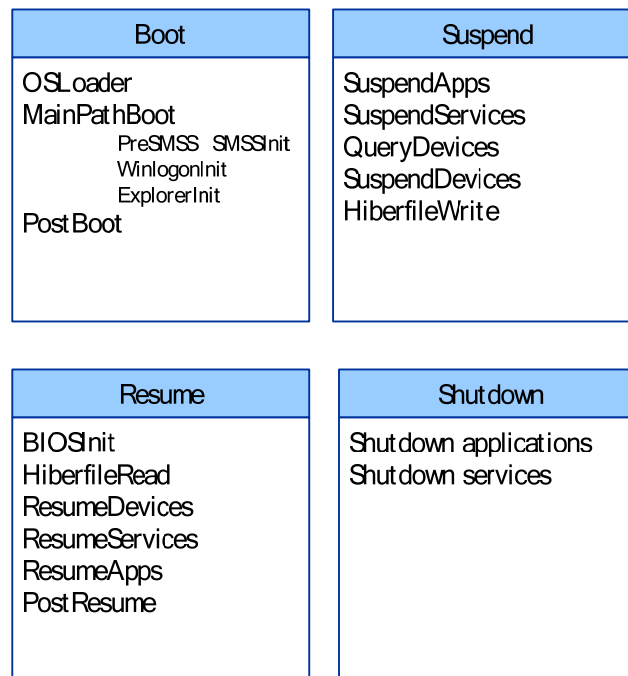


Figure 2. An overview of the phases and subphases associated with on/off transitions

Boot Transition

This section examines the boot process and discusses ways to measure and improve performance bottlenecks.

Performance Analysis

Improving the time that a computer requires to start not only increases user satisfaction, but also is frequently used as an operating system performance metric.

Users such as system administrators who restart computer systems frequently quickly become dissatisfied with long boot times.

Overview

The boot process is divided into three primary phases that are shown in Figure 3.

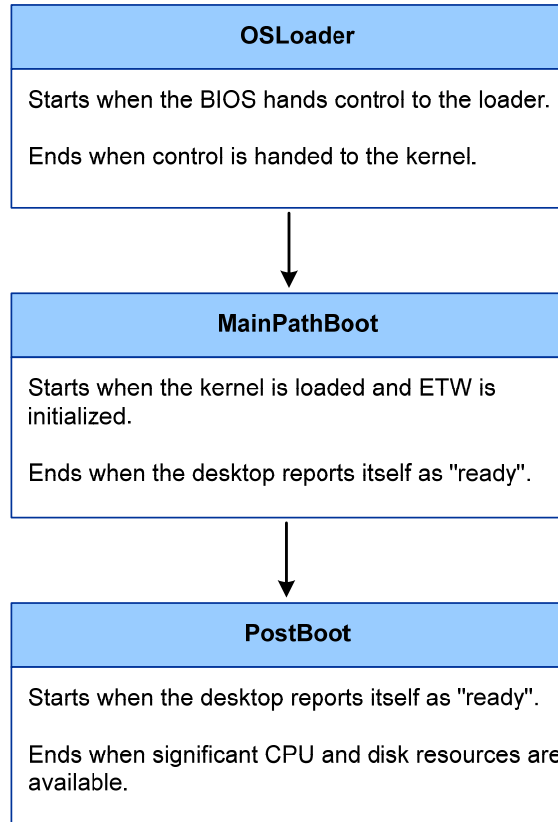


Figure 3. The main phases that comprise the boot process

The PostBoot phase is considered complete when background activity has dropped to a level at which users can perform their usual work. This phase can be complete while the disk is still busy with low-priority activity, so the disk activity LED is not a good indicator that the PostBoot phase is complete.

Boot Traces

This section covers capturing and analyzing traces during the boot transition.

Capturing Boot Traces

You can use the `xbootmgr` tool to capture a trace of the boot process. For example, the following example command generates a boot trace by restarting the system three times:

```
xbootmgr -trace boot -numRuns 3 -resultPath %systemdrive%\traces -
prepSystem -traceflags base
```

For full information about `xbootmgr` options, see the `xbootmgr` Help file.

The following are some things to remember:

The test system reboots within 5 seconds after you enter this command.

Using **-prepSystem** causes Windows Vista self-optimizations to occur.

If **prepSystem** is specified, two preparatory boots occur to enable Windows Vista self-optimizations to run until they are complete.

The **-traceflags base+cswitch** option provides a lightweight and consistent trace from a timing perspective.

One boot per iteration is performed, and a status dialog box appears after each boot.

No final status dialog box appears when testing is complete.

Analyzing Boot Traces

To generate an XML summary of the boot phase, use the **-a boot** action with xperf. For example, the following command takes as input the trace.etl trace file and generates the following summary.xml output file:

```
xperf -i trace.etl -o summary.xml -a boot
```

When you examine the boot process traces, remember the following:

The time unit that is used (for example, milliseconds) is specified at the top of the XML summary report.

Most intervals and operations are described by as a trio of start time, end time, and duration.

Many intervals and operations can overlap. This overlap can make it difficult to identify a specific problem in the boot phase.

Comparing the time in an interval or operation against a baseline can help highlight issues and reduce ambiguity that is related to overlap.

Phase timing information primarily appears under the timing node in the XML report.

Phases

The OS Loader Phase

During the OS loader phase, only basic initialization occurs. This initialization includes the initialization of the loader and the loading of all drivers that are marked **BOOT_START**. The drivers are not initialized in this phase.

Note: When examining the trace:

The **osLoaderDuration** attribute of the **timing** node contains the time that was spent in this phase.

OS Loader Phase Vulnerabilities

BOOT_START drivers that are not embedded-signed cause significant performance problems in this phase. The operating system must search the CAT files for a valid signature, which leads to reduced performance.

To obtain a list of the BOOT_START drivers, type the following command:

```
findstr I-DCStart trace.csv
```

Ensure that all the drivers in this list are embedded-signed. Use signtool.exe from the Windows Driver Kit to verify the embedded signatures, substituting the appropriate driver name. For example:

```
signtool verify /kp /v driver.sys
```

MainPathBoot Phase

Most boot activities occur during the MainPathBoot phase, including Plug and Play enumeration and initialization, session initialization, and some Service and Group Policy activity. This phase includes the time that is spent at the logon screen including when the user enters credentials, but the tool subtracts any idle time during these interactions.

Services are started during this phase, but they can continue to start after the phase is complete.

If the test machine is part of a network domain, remember that Group Policy can potentially block specific activities that occur as part of the boot process.

Note: When examining the trace:

The **bootDoneViaExplorer** attribute of the **timing** node is the time that was spent in the MainPathBoot phase.

Timings for the minor phases that are divisions of this phase are reported in the **interval** node, which is a child of the **timing** node.

MainPathBoot Subphases

MainPathBoot consists of several minor subphases, as shown in Figure 4. Each subphase appears by name in the trace summary.

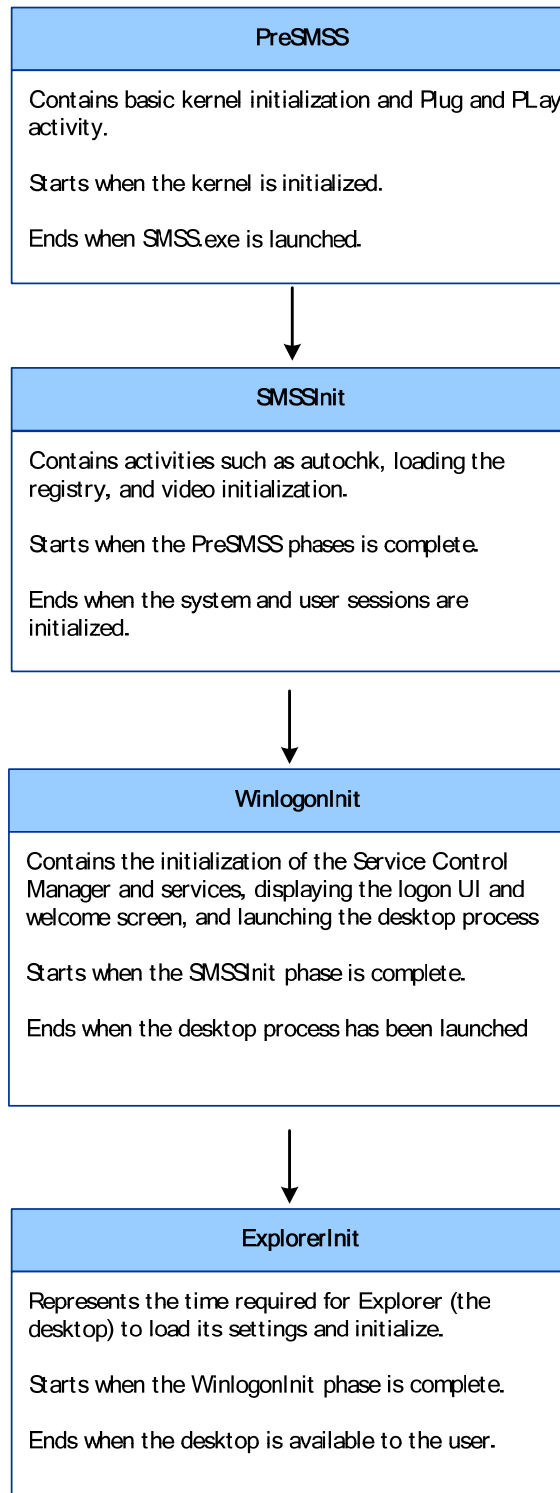


Figure 4. The MainPathBoot subphases

Note: Background processing that is related to boot can continue after the ExplorerInit subphase is complete. For example, not all services might be started when ExplorerInit is complete.

Each minor subphase that is part of MainPathBoot has unique performance vulnerabilities.

PreSMSS Subphase Vulnerabilities

Inbox, devices, and third-party drivers can affect the time that is spent to perform Plug and Play enumeration and initialization during the PreSMSS subphase. By using the existing trace.etl XML trace file that was created by using xperf (see "Analyzing Boot Traces" earlier in this paper), you can generate a CSV text file report to look for Plug and Play activity by typing the following command:

```
xperfnfo.exe -i trace.etl -o tracepnp.csv -a pnp
```

The header of the CSV file contains a summary of the time that was spent in boot and system Plug and Play phases.

The CSV file contains a header, followed by a dump of the Plug and Play data, and then followed by per-device and per-driver summaries. For every device and driver operation, the raw Plug and Play data contains a different EventType.

When examining the trace, note the following:

bootStart corresponds to the initialization of the BOOT_START drivers that were loaded by the OS Loader.

systemStart corresponds to the loading and initialization of all other devices and drivers.

DeviceEnum represents a Plug and Play IRP_MN_QUERY_DEVICE_RELATIONS request to the driver that is associated with the device. This request enumerates the device's children.

BootInit calls the **DriverEntry** function of the associated driver.

DeviceStart marks the time of the IRP_MN_START_DEVICE IRP.

DriverLoad marks the loading and call to **DriverEntry** for the associated driver.

Each Plug and Play operation has three time stamps, all of which are measured from the beginning of the trace:

StartTS is the time at which the operation was started.

PendTS is the time at which the target pended the operation. For operations that are not pended, PendTS is set to EndTS.

EndTS is the time at which the operation was complete.

Each operation has an associated **InstancePath** attribute that corresponds to a symbolic name for a device or driver.

The Plug and Play summary data is generated from this raw data.

The Summary Plug and Play data contains separate sections for per-device and per-driver results. Both sections follow the same format. The best way to analyze the data is to load it into Microsoft Excel® and sort the **PrePendInitTime** and

PostPendInitTime columns from largest to smallest. Comparing the results against a baseline system highlights the additional time that a driver or device consumed.

When examining the trace, note the following:

InstancePath contains the summary, which includes the total time that was required to initialize a device or driver.

The time measurements are divided into **PrePendInitTime** and **PostPendInitTime**.

Multiple devices and drivers can be loaded and initialized in parallel. Therefore, reducing a single driver's initialization time by a given number of seconds can reduce the overall boot time by a smaller amount because of overlap. To determine the overlap, examine the raw data that contains the start and end time stamps.

The **DeviceEnum** and **DeviceStart** events correspond to the processing of the **IRP_MN_QUERY_DEVICE_RELATIONS** and **IRP_MN_START_DEVICE IRPs** respectively. The time taken to process these IRPs can have a significant impact on boot time if the IRP is processed directly in the driver's dispatch routine. By returning **STATUS_PENDING** the PnP manager is able to process other device requests while the driver is processing the IRP.

For additional details please see:

IRP_MN_START_DEVICE

<http://msdn.microsoft.com/en-us/library/ms806454.aspx>

IRP_MN_QUERY_DEVICE_RELATIONS

<http://msdn.microsoft.com/en-us/library/ms806482.aspx>

SMSSInit Subphase Vulnerabilities

A common source of performance problems in the SMSSInit subphase is the video driver. The video driver must be initialized first in the system session and then in the user session.

Initialization in the user session is typically much faster because of common initialization costs that were already encountered in the system session.

You can infer the start times and end times of these initialization operations from the events in the raw dump of the trace. Measuring these values is critical to evaluating an updated video driver for boot performance.

Calculating the System Session Video Initialization Process. No specific events are associated with the system session video initialization process, but you can infer the time that is required by determining the time difference between the completion of the first CSRSS.EXE launch by SMSS.EXE, and the launch of WININIT.EXE by SMSS.EXE.

To calculate the length of time that is required to initialize the video driver in the system session

1. Filter out all SMSS events of interest:

```
findstr.exe /C:"Microsoft-Windows-Subsys-SMSS" trace.csv > trace.csv.filtered
```

```
findstr.exe /i "csrss wininit" trace.csv.filtered
```

2. Locate the end of the first launch of CSRSS.EXE
 Event type is in the first column:
 Microsoft-Windows-Subsys-SMSS/smss:Executelmage/Stop
 Time stamp is in the second column.
 Process name is in the sixth column.
 Note that two launches of CSRSS.EXE are expected, one per session, with the first corresponding to the system session.
3. Locate the start of WININIT.EXE launch
 Microsoft-Windows-Subsys-SMSS/smss:Executelmage/Start
4. To obtain the time that is spent during system session video initialization, calculate the difference between these two time stamps.

Calculating the User Session Video Initialization Process. You can infer the time that the user session video initialization process requires by noting when the second CSRSS.EXE launch by SMSS.EXE is complete and when the launch of WINLOGON.EXE by SMSS.EXE begins.

To calculate the length of time that is required to initialize the video driver in the user session

1. Filter out all SMSS events of interest:
 findstr.exe /C:"Microsoft-Windows-Subsys-SMSS" trace.csv > trace.csv.filtered
 findstr.exe /i "csrss winlogon" trace.csv.filtered
2. Locate the end of the second launch of CSRSS.EXE
 Note that two launches of CSRSS.EXE are expected, one per session, with the second corresponding to the system session
3. Locate the start of WINLOGON.EXE launch
 Microsoft-Windows-Subsys-SMSS/smss:Executelmage/Start
4. To obtain the time that is spent during user session video initialization, calculate the difference between these two time stamps.

WinlogonInit Phase Vulnerabilities

The presence of the network and preexisting persistent network connections can cause variable delays in the WinlogonInit subphase. Therefore, to ensure that each run is consistent, we recommend that you delete all persistent network connections before you test.

If synchronous Group Policy was deployed on that particular domain, testing that is performed while the machine is connected to a domain can show a significant time that is spent in this subphase. The actual time varies depending on the specifics of the policy or policies.

During this subphase, services begin to start and continue to start throughout subsequent subphases. Services are arranged in "load order groups." Groups are started serially, but the services within each group are started in parallel (unless a service has explicitly declared a dependency on a second service, in which case they

will start serially). Long delays during any service initialization can have a negative impact on the time required for the boot phase to complete.

Analyzing Service Start. Service start information is in the **services** node in the XML summary. Every service has its own subnode, and the time that is required for the service to begin is in the **totalTransitionTimeDelta** attribute.

The Service Control Manager (SCM) is blocked from starting the next load order group until the current service reports itself as running. Therefore, services should report themselves as running immediately to unblock the SCM. Background initialization and processing can occur in parallel on a worker thread.

Note: When examining the trace:

services contains the service start information.

The time that is required for the service to start is specified by the **totalTransitionTimeDelta** attribute.

ExplorerInit Phase Vulnerabilities

Analysis of this phase is secondary to the goals of this paper and will not be discussed further.

The PostBoot Phase

To improve performance, the goal for this phase should be to limit and quantify the background processing that continues after the desktop is visible such as launching startup applications, creating tray icons, and so on. When the system is reasonably idle and responsive to user input, the PostBoot phase can be considered complete.

The duration of the PostBoot phase is defined as the time that is required for the system to reach a performance level that does not interfere with typical usage. Low-priority CPU and disk activity is ignored.

Note: When examining the trace:

The PostBoot metric, **bootDoneViaPostBoot**, is only meaningful if the trace was captured with a **-traceflags** argument that includes the **cswitch** parameter. That is, **-traceflags base+cswitch**.

The timing for the PostBootPhase is in the **timing** node of the XML summary.

The **bootDoneViaPostBoot** attribute represents the end of the PostBoot phase.

The difference between **bootDoneViaPostBoot** and **bootDoneViaExplorer** is the length of the PostBoot phase.

The PostBoot Phase Vulnerabilities

Any service initialization that began during the WinlogonInit subphase can continue into the PostBoot phase, so short service startup times are important.

To study this phase, use xperfview to examine the CPU usage per-process and the disk I/O on a per-file and per-process basis. Compare the length of the PostBoot phase on the test system to a baseline because this helps determine whether a problem exists.

Identifying PostBoot in xperfview

The start of the PostBoot phase corresponds to the end of the MainPathBoot phase. You can most easily find the end of this phase by looking for the creation of the explorer.exe desktop process.

To find the beginning of the phase, use the XML trace file and locate the **bootdoneviaexplorer** node. When examining the trace in xperfview, the start time attribute is the beginning of the PostBoot phase.

Key Issues

The following is a checklist of items that can help improve boot phase performance:

Ensure that all BOOT_START drivers are embedded-signed.

Ensure that all devices and drivers initialize quickly.

Verify that services have a short startup time to avoid blocking progress by the SCM.

Ensure that services and startup applications on the system do not adversely affect and lengthen the PostBoot phase.

The most important thing to remember is that you should perform testing in a controlled way and make comparisons against a valid baseline. The baseline system must consist of identical hardware that is running the correct operating system and software configuration. When you add drivers, services, and extensions to an image, measure the effect incrementally to avoid regressions.

Suspend/Resume Transition

Resume transitions, whether from suspend or from hibernate are key Windows Vista customer scenarios. Improving performance on these transitions increases convenience and improves battery life to users. However, these transitions offer many potential vulnerabilities and challenges.

Performance Analysis

Suspend/resume transitions are divided into many phases, each of which is vulnerable to specific performance problems because of software inefficiencies, bugs, and configuration issues.

An important phase for performance is the PostResume subphase (which is described later in this paper). Having minimal activity during this subphase is critical to giving users a high-quality experience when they resume their systems.

This section discusses resume transitions and examines how to generate and analyze traces.

Overview of the Suspend Transition

The phases of the suspend transition are shown in Figure 5.

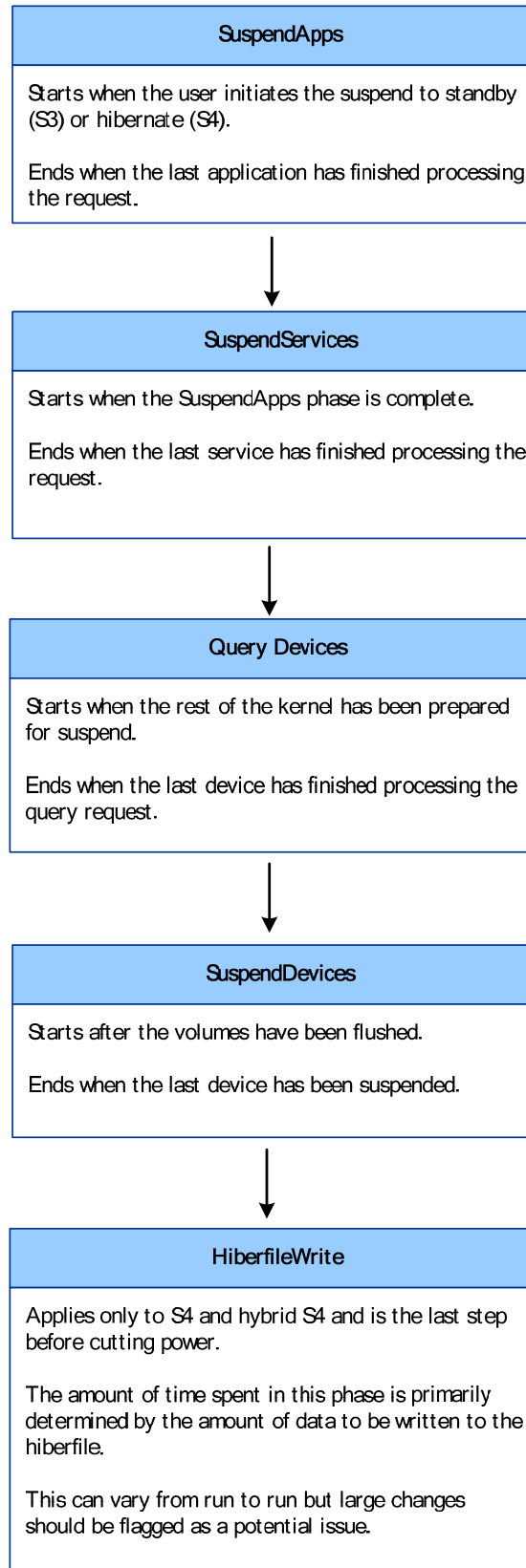


Figure 5. The phases of the suspend transition

Overview of the Resume Transition

The phases of the resume transition are shown in Figure 6.

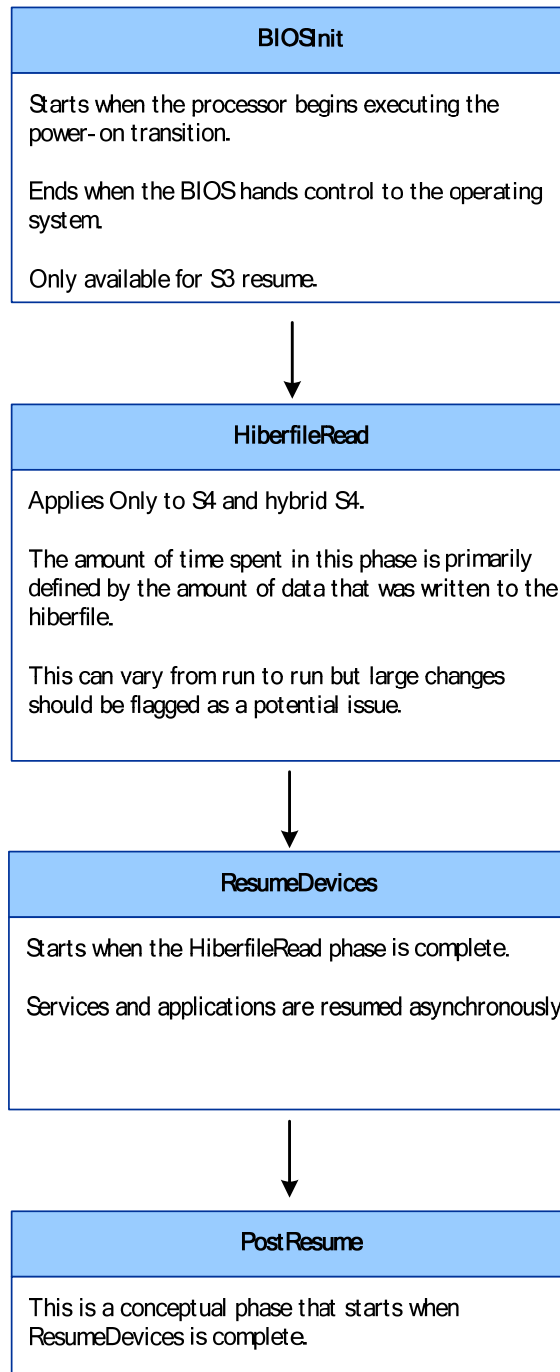


Figure 6. The phases of the resume transition

Standby/Resume Transition: Capturing Standby Traces

The following example command uses `xbootmgr` to capture a trace of the standby/resume transition. The system restarts three times, and the trace is saved to a directory on the root drive that is called `traces`:

```
xbootmgr -trace standby -numRuns 3 -resultPath %systemdrive%\traces -  
postBootDelay 120 -traceFlags base
```

The following are some key points to remember:

The test system is not rebooted for standby/resume testing.

The **-postBootDelay** value specifies the time-out in seconds that the system waits after resuming before stopping the trace. The default is 120, which is usually acceptable for standby/resume testing.

The **-traceFlags base** option provides the more lightweight and consistent trace from a timing perspective.

A status dialog box appears throughout each iteration.

After testing is complete, the final trace is created, but no dialog box appears.

Hibernate/Resume Transition: Capturing Hibernate Traces

The following example command uses `xbootmgr` to capture a trace of the hibernate/resume transition. The system restarts three times, and the trace is saved to a directory on the root drive that is called `traces`:

```
xbootmgr -trace hibernate -numRuns 3 -resultPath %systemdrive%\traces -  
postBootDelay 120 -traceFlags base
```

The following are some key points to remember:

The test system is not rebooted for hibernate/resume testing.

The **-postBootDelay** value specifies the time-out in seconds that the system waits after resuming before stopping the trace. The default is 120, which is usually acceptable for standby/resume testing.

The **-traceFlags base** option provides the more lightweight and consistent trace from a timing perspective.

A status dialog box appears throughout each iteration.

After testing is complete, the final trace is created but no dialog box appears.

Suspend/Resume Transition: Capturing Resume Traces

Suspend/resume transition information is captured as part of the standby or hibernate trace capture procedure.

Suspend/Resume Transition: Analyzing Captured Traces

To generate an XML summary of the suspend/resume transition, use the **-a suspend** action with `xperf`, as shown in the following example command:

```
xperf -i trace.etl -o summary.xml -a suspend
```

When you read through the report, note that most intervals and operations are described by start time, end time, and duration.

Many phases involve substantial processing by applications, services, and devices. Detailed timing information is given in the subnode that is associated with each **scenario** node.

Note: When examining the trace:

The **scenario node** contains high-level summary information.

duration is the total time for the entire suspend/resume transition.

suspend is the time that is required to suspend the system.

biosinit is the time that is spent in the BIOS to resume the system. This is available only for standby, not for hibernate.

resumecritical is the time that is spent performing actions that are critical to resuming the system (such as resuming devices).

resume is the time that the thread that controls the resume transition requires to finish executing the resume work (sending notifications and so on).

Hibernate/resume transitions have **hiberwrite** and **hiberread** attributes.

Note: Currently no attribute corresponds to the PostResume subphase.

Suspend Phase: SuspendApps Subphase

The SuspendApps subphase corresponds to the processing of the suspend message by all windowed applications.

This phase starts when the WM_POWERBROADCAST message is sent to all applications serially. Applications are permitted to use up to 2 seconds to process the message.

Most applications are not required to do significant work in this phase. Any such work directly correlates to a delay in suspending the system. The application does not always have sufficient time to complete its work before the 2-second time-out elapses.

Note: When examining the trace:

Detailed per-process timing for this phase is reported in the **suspendapps** node.

Suspend Phase: SuspendServices Subphase

The SuspendServices subphase corresponds to the processing of the suspend notification by services. Only services that are registered to receive power messages receive the notification, which is sent to services serially. No time-out is enforced, but the service does not always have sufficient time to complete its work before the system enters suspend.

Most services are not required to complete significant work during this subphase. However, any work that occurs uses hardware resources, which can delay the system from entering suspend.

Note: When examining the trace:

Detailed per-process timing for this subphase is given in the **suspendservices** node.

Suspend Phase: QueryDevices and SuspendDevices Subphases

Devices and drivers can significantly affect the suspend time of the system. In the QueryDevices subphase, the system queries each device, starting at the leaves of the device tree and moving up the hierarchy. After the QueryDevices subphase is complete, the system moves to the SuspendDevices subphase.

Drivers can delay the suspend transition in both the QueryDevices and SuspendDevices subphases.

To identify performance problems in these subphases, compare the results against a baseline measurement.

Note: When examining the trace:

queryDevices and **suspendDevices** provide per-driver and device summaries.

Resume Phase: ResumeDevices Subphase

The performance of the devices and drivers can have a large effect on the resume time of the system because all devices must resume before the system can restore the desktop.

Device initialization begins almost immediately after the BIOS gives control to the operating system.

As with the QueryDevices and SuspendDevices **subphases**, it is important to validate driver and device changes against the correct baseline.

Note: When examining the trace:

resumedevices provides a per-driver and device summary.

Resume Phase: ResumeServices and ResumeApps Subphases

These two subphases are a common source of performance problems. For example, consider how much time passed while the system was in suspend. After the system resumes, many timers and scheduled tasks can start and many applications and services can receive concurrent notifications.

The ResumeServices and ResumeApps subphases are asynchronous, and the system considers them complete after the infrastructure that manages the resume process is notified to do so.

The activity in this phase effectively falls into the PostBoot phase.

Resume Phase: PostResume Subphase

The PostResume subphase is a conceptual subphase that covers the time that follows the resume transition. As with the boot transition, the goal is to get the system to a fairly idle state as quickly as possible.

The CPU Utilization and Disk Utilization graphs can provide insight into how busy the system is following a resume, but a valid baseline measurement is important when you analyze PostResume activity. Currently, no tools support the calculation of a PostResume metric. Instead, we strongly recommend that you compare traces from

the baseline and the test system to identify anomalous CPU and disk activity that occurs during the PostResume window.

Summary

The phases of the suspend/resume transition are complex, but also very important phases to optimize. The following is a list of important things to consider:

Remember that applications, services, and drivers can all affect suspend/resume transitions.

Ensure that applications and services are not taking a long time to process their messages and notifications.

Ensure that drivers and their devices are not taking a long time to suspend or resume.

Evaluate updated software against the correct baseline. You can take a baseline measurement from a clean installation that does not have a device driver.

Performance in the PostResume subphase is absolutely critical to customer satisfaction.

Use xperf to examine CPU and disk usage during these phases and identify large consumers of hardware resources.

Shutdown Transition

Similar to the boot process, a lengthy shutdown transition can cause a poor user experience.

Performance Analysis

Services and applications can have a significant impact on shutdown performance.. CPU and disk activity during the shutdown process can also potentially delay the shutdown transition.

Overview

Figure 7 shows the three main phases of the shutdown transition.

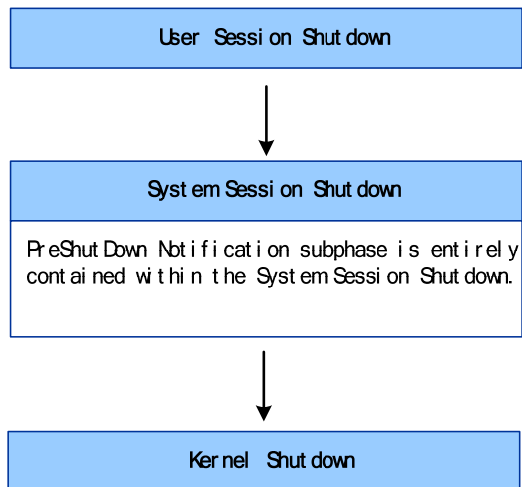


Figure 7. The phases of the shutdown transition

Capturing Shutdown Traces

The following example uses `xbootmgr` to capture a trace of the shutdown transition. The system restarts three times, and the trace is saved to a directory on the root drive called `traces`:

```
xbootmgr -trace shutdown -numRuns 3 -resultPath %systemdrive%\traces -
postBootDelay 180 -traceFlags base
```

The following are some key points to remember:

The `-postBootDelay` value specifies the time-out in seconds that the system waits after booting before the next shutdown trace is initiated. The default is 120, but 180 is recommended for shutdown traces.

The `-traceFlags base` option provides a lightweight and consistent trace.

An initial reboot is performed before you collect traces.

A reboot is performed for each iteration, and a status dialog box appears throughout each iteration.

After testing is complete a final reboot is performed, but no status dialog box appears.

Analyzing Shutdown Phase Traces

Use the `-a shutdown` action with `xperf` to generate an XML summary of shutdown, as shown in the following example command:

```
xperf -i trace.etl -o summary.xml -a shutdown
```

Note: Detailed per-phase timing information is available in the XML report. The `services` node provides per-device timing information, and the `perSessionInfo` node provides per-application timings.

User Session Shutdown

Application shutdown is controlled for each session, for each session's `CSRSS.EXE`. During shutdown, applications must perform a handshake with `CSRSS`, and any applications that fail to do this correctly are timed out and terminated.

Applications can have a significant impact on shutdown performance due to the fact the timeouts are long in order to give applications plenty of time to shutdown and persist user data..

The time that is required to shut down an application varies among applications. Application shutdown timings are reported in the `sessionShutdown` subnode of the `perSessionInfo` node, specifically a node of the following form:

```
<sessionShutdown sessionID="1" startTime="600" endTime="7160"
duration="6560">
```

Note: The `sessionID` attribute in this phase is 1.

System Session Shutdown

During the System Session Shutdown phase, the SCM shuts down many services in parallel and all processes that are running in session 0. During shutdown, each service must perform a handshake with the SCM. If a service fails to perform the handshake correctly, the SCM waits 20 seconds and then terminates the service.

Note: Services that do not perform the SCM handshake correctly and therefore delay shutdown are identified in the **unresponsiveServices** node.

The time for this phase is reported specifically in a node of the following form:

```
<sessionShutdown sessionID="0" startTime="600" endTime="7160"
duration="6560">
```

Note: The sessionID attribute in this phase is 0.

This phase includes the preShutdownNotification subphase that applies only to session 0, which contains Windows Vista system services. Because preshutdown notifications let services take longer than 20 seconds before timing out, you should use them very sparingly.

Kernel Shutdown

In the kernel shutdown phase, the rest of the system is shut down.

You can calculate the time that was spent in this phase as follows:

1. Within the XML trace file, locate the overall shutdown time, for example:


```
<results timeFormat="msec">
<shutdown>
<timing shutdownTime="29917" servicesShutdownDuration="20137">
```
2. Subtract the time that is required to shut down the system and user sessions from **shutdownTime**. In this example, the kernel shutdown phase lasted 9,780 milliseconds.

Summary

The following is a list of important things to consider as you improve shutdown performance:

Remember that services and applications can significantly affect shutdown performance.

Ensure that no services appear in the **unresponsiveServices** node.

Ensure that the applications that are running on the system respond quickly to shutdown notifications.

Use the **perSessionInfo** and **sessionShutdown** nodes to see timing for each application.

Perform testing in a controlled way and make comparisons against a valid baseline (that is, the same hardware running the correct operating system and software configuration).

Conclusions

You can identify and solve most on/off transition performance problems by using the tools and techniques that are discussed in this paper. For example, system integrators can incorporate on/off testing by using bootmgr.exe, xperf.exe, and xperfview into the preinstallation image validation procedure can provide significant value to users.

Furthermore, by using these tools during system development, you can discover software and hardware issues that might otherwise go unnoticed.

Resources

Benchmarking Vista white paper

<http://www.microsoft.com/whdc/system/sysperf/default.aspx>

Windows Performance Tools Kit

<http://www.microsoft.com/whdc/system/sysperf/perftools.aspx>

QUERY_SERVICE_CONFIG Structure on MSDN

[http://msdn2.microsoft.com/en-us/library/ms684950\(VS.85\).aspx](http://msdn2.microsoft.com/en-us/library/ms684950(VS.85).aspx)

SetProcessShutdownParameters Function on MSDN

[http://msdn.microsoft.com/en-us/library/ms686227\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms686227(VS.85).aspx)

autologon.exe

<http://shellrevealed.com/files/folders/code/entry4411.aspx>

Appendix: XML Reference

All traces that the Windows Performance Tools Kit generates follow a specific XML schema. The following sections list the information that is reported in each XML node and define any XML attributes in the node.

This XML reference contains the following three sections:

Boot Process

Resume/Suspend Process

Shutdown Process

Each section begins with a diagram that explains the node hierarchy and continues with a description of each node.

The Boot Process

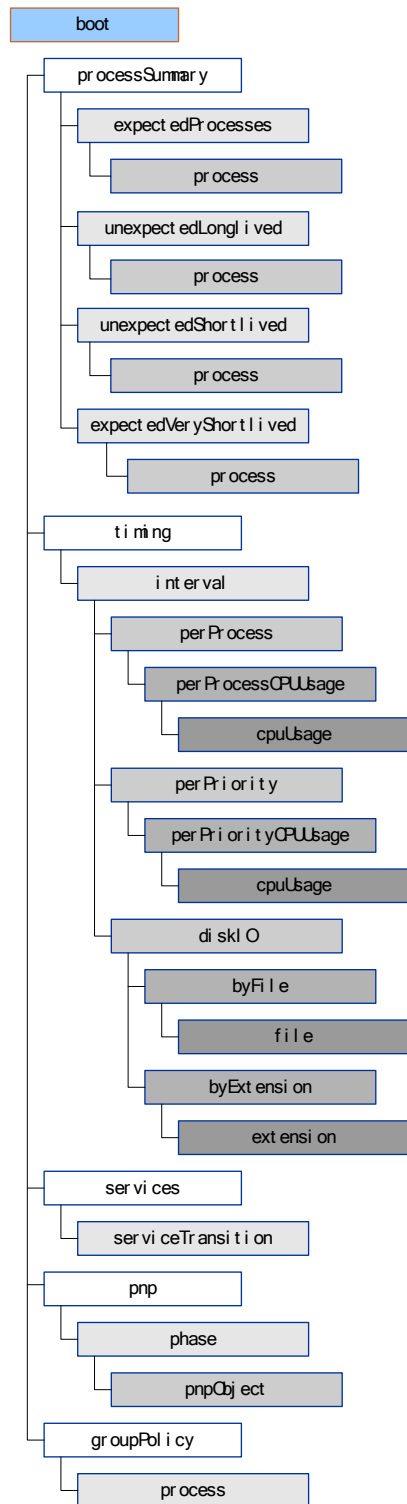


Figure A-1. The XML nodes in the boot transition report

boot

The **boot** node is the root node for the boot phase and contains all the trace information that has been captured as part of the boot process.

The boot node contains the following subnodes:

processSummary

timing

services

pnp

groupPolicy

Each of these nodes contains subnodes and attributes that hold specific trace information.

Notes:

All times—other than **osLoaderDuration**—are relative to the start of the trace. This start is approximately the time at which the kernel starts, excluding BIOS and WinLoad times.

DiskIO times refer to service time, instead of I/O time.

processSummary

This node is part of the boot phase and includes all trace data that relates to processes. The details of each captured process appear under one of the following subnodes:

expectedProcesses

unexpectedLonglived

unexpectedShortlived

unexpectedVeryShortlived

Note: The division of processes among these subnodes is essentially for internal Microsoft use and is irrelevant for analysis. You should focus on the existence of the processes, not the subnode under which they are stored.

Attributes

The **processSummary** node has the following attributes:

numProcesses	The number of processes that are captured as part of the trace.
numUnexpectedLonglived	The number of processes that are classified as unexpectedly long-lived.
numUnexpectedShortlived	The number of processes that are classified as unexpectedly short-lived.
numUnexpectedVeryShortlived	The number of processes that are classified as unexpectedly very short-lived.

expectedProcesses

This node is a subnode of **processSummary** and contains the details of each process that is classified as "expected." Note that the classification is only for internal Microsoft use and is irrelevant for analysis.

The **expectedProcesses** node contains one or more instances of the following subnode:

process

The trace information is within this subnode.

unexpectedLonglived

This node is a subnode of **processSummary** and contains the details of each process that is classified as "unexpectedly long-lived." Note that the classification is only for internal Microsoft use and is irrelevant for analysis.

The **unexpectedLonglived** node contains one or more instances of the following subnode:

process

The trace information is within this subnode.

unexpectedShortlived

This node is a subnode of **processSummary** and contains the details of each process that is classified as "unexpectedly short-lived," that is, is still active by the time that the trace ends. Note that the classification is only for internal Microsoft use and is irrelevant for analysis.

The **expectedShortlived** node contains one or more instances of the following subnode:

process

The trace information is within this subnode.

unexpectedVeryShortlived

This node is a subnode of **processSummary** and contains the details of each process that is classified as "unexpectedly very short-lived," that is, has a lifetime that is less than 100 ms by default. Note that the classification is only for internal Microsoft use and is irrelevant for analysis.

The **unexpectedVeryShortlived** node contains one or more instances of the following subnode:

process

The trace information is within this subnode.

Attributes

The **unexpectedVeryShortlived** node has the following attribute:

threshold	If a process lives for less than the threshold time (which is expressed in global report time units that are defined in the timing node), it is classified as very short-lived.
------------------	---

process

This node is a subnode of the **processSummary** subnode and contains the specific details of each captured process.

Attributes

The **process** node has the following attributes:

name	The name of the process.
startTime	The time when the process began, as measured from the start of the trace.
endTime	The time when the process ended, as measured from the start of the trace. A value of -1 signifies that the process was still alive after the trace ended.
lifeTime	The lifetime of the process. A value of -1 signifies that the process was still alive after the trace ended.

timing

This node is part of the boot phase and includes all timing data for the subphases that are part of the boot process. The details for each phase are in an **interval** node. These **interval** nodes include the following:

PreSMSS

SMSSInit

WinlogonInit

ExplorerInit

PostExplorerPeriod

TraceTail

Attributes

The **timing** node has the following attributes. All times are in global report time units. The default is milliseconds:

bootDoneViaExplorer	The time that was spent in the MainPathBoot phase.
bootDoneViaPostBoot	The time at which the PostBoot phase ended.
osLoaderDuration	The time that was spent in the OS Loader phase (which is not included in the main path boot time of bootDoneViaExplorer).
postBootRequiredIdleTime	See the following notes.
postBootDisturbance	See the following notes.
pnpBootStartStartTime	The time that the system started loading BootStart drivers.
pnpBootStartEndTime	The time that the system finished loading BootStart drivers.
pnpBootStartDuration	The difference between the previous two values.
pnpSystemStartStartTime	The time that the system started loading SystemStart drivers.
pnpSystemStartEndTime	The time that the system finished loading SystemStart drivers.
pnpSystemStartDuration	The difference between the previous two values.

Notes

The difference between **bootDoneViaPostBoot** and **bootDoneViaExplorer** is the length of the PostBoot phase.

The duration of the PostBoot phase, which is reported by the boot action, is an extrapolated value. This value is based on system behavior after the last critical activity (Explorer initialization, which is reported as **bootDoneViaExplorer**) is complete.

System activity (specifically, CPU and disk usage at typical and greater priorities) is analyzed to determine if a user could run applications and interact with the system.

A scanning algorithm does the analysis and tries to locate a total of 10 seconds (by default) of available resources, starting with the **bootDoneViaExplorer** time.

The time at which 10 seconds of theoretically available resources have been observed is reported as **bootDoneViaPostBoot**.

The amount of available time that the algorithm looks for (the default 10 seconds) is reported as **postBootRequiredIdleTime**. The amount of time that resources were being used while the algorithm scanned for idle time is reported as **postBootDisturbance**.

Therefore, **bootDoneViaPostBoot** is the sum of **bootDoneViaExplorer**, **postBootRequiredIdleTime**, and **postBootDisturbance**.

interval

This node is part of the **timing** node of the boot phase and includes all data for the various activities that occur during each subphase. Additional information is in the following subnodes:

perProcess

perPriority

diskIO

Attributes

The **interval** node has the following attributes. All times are in global report time units. The default is milliseconds:

name	The name of the subphase.
startTime	The time when the subphase began, as measured from the start of the trace.
endTime	The time when the subphase ended, as measured from the start of the trace.
duration	The total length of the subphase.

perProcess

This node is part of the **interval** subnode of the **timing** node and lists the amount of CPU usage for each process. Each process has its own **perProcessCPUUsage** subnode.

perProcessCPUUsage

This node is part of the **perProcess** subnode of the **timing** node and contains the exact amount of CPU usage for each process. More details are stored in the **cpuUsage** subnode.

Attributes

The **perProcessCPUUsage** node has the following attributes. All times are in the global report time units, which by default is milliseconds:

name	The name of the process.
time	The length of time that the process was active.
percentOfInterval	The percentage of the subphase's total duration that was spent on this process.

perPriority

This node is part of the **interval** subnode of the **timing** node and lists the amount of CPU usage for each process. This number is organized by the priority of the process's threads in one or more **perPriorityCPUUsage** subnodes.

perPriorityCPUUsage

This node is part of the **perPriority** subnode of the **timing** node and lists the CPU usage for each process, organized by priority, that is, the CPU usage at a given priority across all processes and all threads. The details are stored in the **cpuUsage** subnode.

Attributes

The **perPriorityCPUUsage** node has the following attributes. All times are in global report time units. The default is milliseconds:

priority	The priority of the process.
time	The length of time that the process was active.
ofInterval	The percentage of the subphase's total duration that was spent on this process.

cpuUsage

This node appears as a subnode to both **perPriorityCPUUsage** and **perProcessCPUUsage**.

Subnode to perPriorityCPUUsage

In this form, the **cpuUsage** nodes list the amount of CPU time that all the processes of a specific priority used.

Attributes

The **cpuUsage** node has the following attributes. All times are in global report time units. The default is milliseconds:

process	The name of the process that executed at the given priority.
time	The amount of CPU time that this process executed at the given priority.
ofPriority	The percentage of total CPU execution time at the given priority that this process accounted for.

Subnode to perProcessCPUUsage

In this form, the **cpuUsage** nodes list the amount of CPU time that each process used during the interval, sorted by priority.

Attributes

The **cpuUsage** node has the following attributes. All times are in global report time units. The default is milliseconds:

priority	The priority of the process.
time	The amount of time that was spent executing at the specified priority within this process.
cumTime	The cumulative amount of time that was spent executing this process at the specific priority.
ofProcess	The percentage of the process's total CPU time that was accounted for by the time that was spent at this priority.
ofInterval	The percentage of the entire interval's time that was accounted for by the time that was spent (by this process) at this priority.
cumOfProcess	The percentage of the process' total CPU time accounted for by the time that was spent at this and lower priorities.

diskIO

This node is part of the **interval** subnode of the **timing** node and lists the amount of disk I/O for each process. For a more precise analysis, the trace data is also available in the following two subnodes:

byFile

byExtension

Attributes

The **diskIO** node has the following attributes. All times are in global report time units. The default is milliseconds:

readBytes	The number of bytes that were read during the trace.
readOps	The number of read operations.
readTime	The time that was spent performing read operations.
avgBytesPerRead	The average number of bytes that were read for each read operation.
medBytesPerRead	The median number of bytes that were read for each read operation.
writeBytes	The number of bytes that were written during the trace.
writeOps	The number of write operations.
writeTime	The time that was spent performing write operations.
avgBytesPerWrite	The average number of bytes that were written for each write operation.
medBytesPerWrite	The median number of bytes that were written for each write operation.
totalBytes	The total number of bytes that were read and written during the trace.
totalOps	The total number of read and write operations.
totalTime	The total time that was spent reading and writing data during the trace.

byFile

This node is part of the **diskIO** subnode of the **timing** node and lists disk usage, organized by each file.

The **byFile** node contains one or more instances of the following subnode:

file

The **file** subnode contains information on each disk operation.

Attributes

The **byFile** node has the following attributes. All times are in global report time units. The default is milliseconds:

name	The name of the file that was read from or written to.
readBytes	The number of bytes that were read from the file.
readOps	The number of read operations.
readTime	The time that was spent performing read operations.
writeBytes	The number of bytes that were written to the file.
writeOps	The number of write operations.
writeTime	The time that was spent performing write operations.

file

This node is part of the **byFile** subnode of the **diskIO** node and lists disk usage, organized by each file.

Attributes

The **file** node has the following attributes. All times are in global report time units. The default is milliseconds:

name	The name of the file that was read from or written to.
readBytes	The number of bytes that were read from the file.
readOps	The number of read operations.
readTime	The time that was spent performing read operations.
writeBytes	The number of bytes that were written to the file.
writeOps	The number of write operations.
writeTime	The time that was spent performing write operations.
totalBytes	The total number of bytes that were read or written during this file activity.
totalOps	The total number of read or write disk operations.
totalTime	The total time that was spent on disk activity either reading from or writing to this file.

byExtension

This node is part of the **diskIO** subnode of the **timing** node and lists disk usage, organized by each file name extension.

The **byExtension** node contains one or more instances of the following subnode:

extension

The **extension** subnode contains information on each disk operation.

extension

This node is part of the **byExtension** subnode of the **diskIO** node and lists the disk usage, organized by each file name extension.

Attributes

The **file** node has the following attributes. All times are in global report time units. The default is milliseconds:

name	The file name extension.
readBytes	The number of bytes that were read from the file(s).
readOps	The number of read operations.
readTime	The time that was spent performing read operations.
writeBytes	The number of bytes that were written to the file(s).
writeOps	The number of write operations.
writeTime	The time that was spent performing write operations.
totalBytes	The total number of bytes that were read or written in this operation.
totalOps	The total number of read or write disk operations.
totalTime	The total time that was spent on disk activity either reading from or writing to the file(s).

services

This node contains the details of all the services and drivers that were started or shut down during the trace.

The **services** node contains one or more instances of the following subnode:

serviceTransition

Attributes

The **services** node has the following attributes:

autoStartStartTime	The time that the autoStart phase began, measured from the start of the trace.
autoStartEndTime	The time that the autoStart phase ended.
autoStartDuration	The total length of time that was spent in the autoStart phase.

The **autoStart** phase includes the time that the SCM initializes services that are marked for **autoStart**, excluding any that were marked for delayed **autoStart**. Services that are marked for demand-start but are required as explicit dependencies by **autoStart** services are also initialized and included.

serviceTransition

The **serviceTransition** node contains timing and other information for each service that was started or shut down.

Attributes

The **serviceTransition** node has the following attributes:

name	The name of the service.
group	A definition of the “load order group” of the service. The “transition” attribute indicates whether it is a start/stop and so on.
transition	The total time that was required to complete the transition.
totalTransitionTimeDelta	The difference between the end time of the transition and the start time of the transition.
firstCheckpointTimeDelta	The difference between the time of the first checkpoint (that is, a ping from service) and the start time of the transition.
processingTimeDelta	The difference between the end time of transition and the time of first checkpoint. The totalTransitionTimeDelta is equal to firstCheckpointTimeDelta plus processingTimeDelta . If there is no checkpoint, processingTimeDelta is zero.
container	The process name and process ID of the process that contains the service. If the service is hosted in an svchost (and so not publicly available), the svchost name is also given.
startedAt	The time when the transition began.
firstCheckpointedAt	The time when the first checkpoint occurred.
endedAt	The time when the transition ended (that is, finished starting, failed, or shut down).

For more information on the transition attributes, see “QUERY_SERVICE_CONFIG Structure” on MSDN®.

In particular, see the **IpLoadOrderGroup** member of the QUERY_SERVICE_CONFIG data structure.

pnP

This node is part of the boot phase and includes the timing data for the various Plug and Play activities that are part of the boot process. Each Plug and Play activity is in one of the following **phase** nodes:

bootStart

systemStart

remainder

Notes

The **bootStart** phase corresponds to the initialization of the BOOT_START drivers that the OS Loader loaded.

The **systemStart** phase encapsulates the loading, initializing, and configuration of system start drivers and their own services.

phase

This node is part of the **pnP** node and includes all timing data for the various Plug and Play activities that occur during the boot process.

More information is stored in the **pnPObject** subnode.

Attributes

The **phase** node has the following attributes. All times are in global report time units. The default is milliseconds:

name	The Plug and Play activity name.
startTime	The time when the Plug and Play activity began, measured from the start of the trace.
endTime	The time when the Plug and Play activity ended.
duration	The total time that was taken by the Plug and Play activity.

pnPObject

This node is part of the Plug and Play **phase** node and includes detailed timing data for the various Plug and Play activities that occur during the boot process.

Attributes

The **pnPObject** node has the following attributes. All times are in global report time units. The default is milliseconds:

name	The Plug and Play object name, often known as the "instance path" for the device or driver.
type	The object type.
activity	The type of Plug and Play activity, such as Enum or Start.
startTime	The time when the Plug and Play activity began, measured from the start of the trace.
endTime	The time when the Plug and Play activity ended.
duration	The total time that was taken by the Plug and Play activity.
prePendTime	If the driver "pended" the IRPs it received to potentially unblock other parts of the Plug and Play operation, the difference between when the driver "pended" the IRPs and the startTime.
description	The Plug and Play description text.
friendlyName	An optional friendly name for the Plug and Play object.

groupPolicy

This node is a subnode of the **boot** node and includes the Group Policy information that covers the execution of scripted Group Policy applications, that is, gpscript.exe. Additional Group Policy settings can be applied in another way.

More information is stored in the **process** subnode.

process

This node is part of the **groupPolicy** node and shows the process tree of execution of Group Policy scripts. Child processes are reflected in child **process** subnodes.

Attributes

The **process** node has the following attributes. All times are in the global report time units, which by default is milliseconds:

Name	The process name.
Pid	The process ID.
startTime	The time when the process was started.
endTime	The time when the process exited.
Duration	The total lifetime of the process.
[local,cum]LowPriCPUtime	The CPU time that was spent executing at lower-than-normal priority. local: by this process alone. cum: by this process and all its child processes.
[local,cum]RegPriCPUtime	The CPU time that was spent executing at typical and higher priority. local: by this process alone. cum: by this process and all its child processes.
[local,cum]LowPriDiskBytes	The amount of disk I/O that was initiated at lower-than-normal priority. local: by this process alone. cum: by this process and all its child processes.
[local,cum]RegPriDiskBytes	The amount of disk I/O that was initiated at typical and higher priority. local: by this process alone. cum: by this process and all its child processes.

The Suspend Process

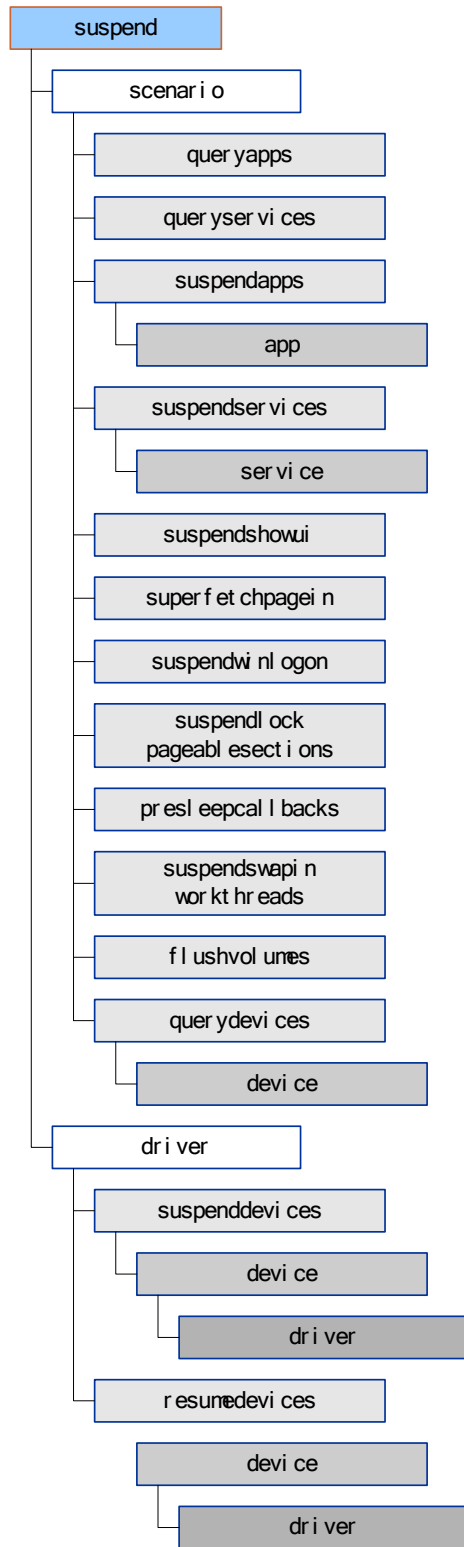


Figure A-2. Nodes in the suspend transition report

suspend

The **suspend** node is the root node for the suspend trace. It contains the following subnode:

scenario

Attributes

The **suspend** node has the following attributes:

time_unit	The time unit in which subsequent times are reported. The default is "us", which means microseconds.
time_precision	The number of significant figures that were used to display times.
min_reported	A filter to remove events below a certain threshold. For example, if min_reported is set to 10 and time_unit is set to us, events shorter than 10 microseconds are not reported. This filter can be set by specifying a value to the "-min" argument of the "suspend" action of xperf.

scenario

Each scenario node has information about one suspend/resume transition that occurred in this trace.

The **scenario** node contains the following subnodes, which represent the following phases of the transition:

queryapps

queryservices

suspendapps

suspendservices

suspendshowui

superfetchpagein

suspendwinlogon

suspendlockpageablesections

presleepcallbacks

suspendswapiworkerthreads

flushvolumes

querydevices

suspenddevices

resumedevices

Attributes

The **scenario** node has the following attributes:

start	The time when the suspend or hibernate transition began, measured from the start of the trace.
duration	The total duration of the suspend or hibernate transition.
suspend	The total time that was spent to suspend the system.
resumecritical	The total time that was spent performing actions that are critical to resuming the system (such as resuming devices).

hiberwrite	The total time that was spent writing the hiberfile to disk. Writing the hiberfile is performed by using a special driver stack because other drivers are currently in sleep.
hiberread	The total time that was spent reading the hiberfile from disk. Reading the hiberfile is performed by using a special driver stack because other drivers are currently still in sleep.
resume	The total time that was taken to resume the machine. It does not include the hiberread time or the BIOSInit time, but does include the resumecritical time.

queryapps

This node contains information on the phase during which running applications are sent a preliminary notification of the suspend operation. Earlier than Windows Vista, applications could veto suspend during this phase and therefore prevent the system from entering sleep. In Windows Vista this action has been disabled by default and applications are no longer able to veto it.

Attributes

The **queryapps** node has the following attributes:

start	The time when the queryapps phase began, measured from the start of the trace.
duration	The total duration of the queryapps phase.

queryservices

This node contains information on the phase during which services are sent a preliminary notification of suspend. Earlier than Windows Vista, services could veto suspend in this phase and prevent the system from entering sleep. In Windows. this action has been disabled by default and services are no longer able to veto it. For application compatibility, it is possible to change this default setting.

Attributes

The **queryservices** node has the following attributes:

start	The time when the queryservices phase began, measured from the start of the trace.
duration	The total duration of the queryservices phase.

suspendapps

The **suspendapps** node describes the phase in which the applications are sent a message that states that the machine is entering suspend.

During this phase, a WM_POWERBROADCAST message that has the PBT_APMSUSPEND event type is sent to all applications synchronously. Each application is given a maximum of 2 seconds to process the notification. Any activity that is performed after this time is suspended and the system continues to the next phase. This phase is synchronous and blocks the suspend path, so applications that do not respond quickly to this message directly add to the suspend time.

This node contains one or more instances of the following subnode:

app

Attributes

The **suspendapps** node has the following attributes:

start	The time when the suspendapps phase began, measured from the start of the trace.
duration	The total duration of the suspendapps phase.

app

Each instance of this node contains information on the time that each application took to process the suspend message as part of the suspend/hibernate transition.

Attributes

The **app** node has the following attributes:

start	The time at which the application began processing the suspend message, measured from the start of the trace.
duration	The length of time that the application spent processing the suspend message.
exe	The name of the application's .exe file.
pid	The application's process ID.

suspendservices

The **suspendservices** node details the phase that begins after the **suspendapps** phase is complete. It ends when the last service has finished processing the suspend request.

Services can register a handler to receive the suspend notification from the kernel power manager. All registered services receive this notification. Services that take a long time to return from this synchronous notification directly block the suspend path for that amount of time.

The **suspendservices** node contains one or more instances of the following subnode:

service

Attributes

The **suspendservices** node has the following attributes:

start	The time when the suspendservices phase began, measured from the start of the trace.
duration	The total duration of the suspendservices phase.

service

Each instance of this node contains information on the service that is being asked to enter suspend.

Attributes

The **service** node has the following attributes:

start	The time at which the service began processing the suspend notification, measured from the start of the trace.
duration	The length of time that the service spent processing the suspend notification.
name	The service name.

suspendshowui

A notification is sent to winlogon to show the logon UI.

Attributes

The **suspendshowui** node has the following attributes:

start	The time when the suspendshowui phase began, measured from the start of the trace.
duration	The total duration of the suspendshowui phase.

superfetchpagein

To prefetch pages that are needed for the resume from sleep, the **superfetchpagein** phase receives a maximum of 4 seconds for S3 and fastS4 (also known as “hybrid sleep”) transitions and 10 seconds for a hibernate transition.

This performance optimization improves the resume experience. The prefetch starts at the very beginning of the suspend transition, and this phase provides SuperFetch™ with additional time if it is needed.

Attributes

The **superfetchpagein** node has the following attributes:

start	The time when the superfetchpagein phase began, measured from the start of the trace.
duration	The total duration of the superfetchpagein phase.

suspendwinlogon

Winlogon uses this notification to lock the workstation during the sleep transition. The power policy can be changed to leave the workstation unlocked.

Attributes

The **suspendwinlogon** node has the following attributes:

start	The time when the suspendwinlogon phase began, measured from the start of the trace.
duration	The total duration of the suspendwinlogon phase.

suspendlockpageablesections

During this phase, the memory manager is requested to lock all pages that are stored in the PAGE_LK section of the kernel into memory. The power management routines that cannot be paged during the power transition are all placed in this section to ensure that they can be accessed without page faulting. The duration of this phase is calculated as the time between the end of the **superfetchpagein** phase and the start of the **PreSleepCallbacks** phase.

Attributes

The **suspendlockpageablesections** node has the following attributes:

start	The time when the suspendlockpageablesections phase began, measured from the start of the trace.
duration	The total duration of the suspendlockpageablesections phase.

presleepcallbacks

Drivers can register to receive this notification if they want to perform a specific action during suspend.

Attributes

The **presleepcallbacks** node has the following attributes:

start	The time when the presleepcallbacks phase began, measured from start of the trace.
duration	The total duration of the presleepcallbacks phase.

suspendswapinworkerthreads

During this phase, worker threads are swapped in so that the system is not required to page-fault on them during resume.

The duration of this phase is calculated as the time between the end of **PreSleepCallbacks** and the start of **QueryDevices**.

Attributes

The **suspendswapinworkerthreads** node has the following attributes:

start	The time when the suspendswapinworkerthreads phase began, measured from the start of the trace.
duration	The total duration of the suspendswapinworkerthreads phase.

flushvolumes

During this phase, the list of volumes is iterated and the removable drives are flushed and removed. This ensures that the system still behaves reliably if the user removes any drives while the system is in hibernate.

Attributes

The **flushvolumes** node has the following attributes:

start	The time when the flushvolumes phase began, measured from the start of the trace.
duration	The total duration of the flushvolumes phase.

querydevices

During this phase, devices are notified to prepare for the suspend or hibernate transition. The **querydevices** phase begins when the rest of the kernel has been prepared for the suspend action. It ends when the last device has finished processing the query request.

This node contains one or more instances of the following subnode:

device

Attributes

The **querydevices** node has the following attributes:

start	The time when the querydevices phase began, measured from the start of the trace.
duration	The total duration of the querydevices phase.

suspenddevices

During this phase, notifications are sent to devices to tell them to suspend. The **suspenddevices** phase starts after the volumes have been flushed and ends when the last device has been suspended.

This node contains one or more instances of the following subnode:

device

Attributes

The **suspenddevices** node has the following attributes:

start	The time when the suspenddevices phase began, measured from the start of the trace.
duration	The total duration of the suspenddevices phase.

resumedevices

The **resumedevices** phase starts almost immediately after the operating system receives control from the BIOS and ends when the last device that has an installed driver has resumed.

During this phase, the kernel power manager notifies all drivers that the system is about to resume.

This node contains one or more instances of the following subnode:

device

Attributes

The **resumedevices** node has the following attributes:

start	The time when the resumedevices phase began, measured from the start of the trace.
duration	The total duration of the resumedevices phase.

device

The **device** node contains trace information that relates to a specific device.

This node contains one or more instances of the following subnode:

driver

Attributes

The **device** node has the following attributes:

start	The time when the device began to process the suspend or resume notification, measured from the start of the trace.
duration	The time that was spent processing the suspend or resume notification.
name	The name of the device.

driver

The **driver** node contains trace information that relates to a specific driver. Each **device** node has one or more **driver** nodes associated with it.

Attributes

The **driver** node has the following attributes:

start	The time when the driver began to process the suspend or resume notification, measured from the start of the trace.
duration	The time that was spent processing the suspend or resume notification.
name	The name of the driver.

The Shutdown Process

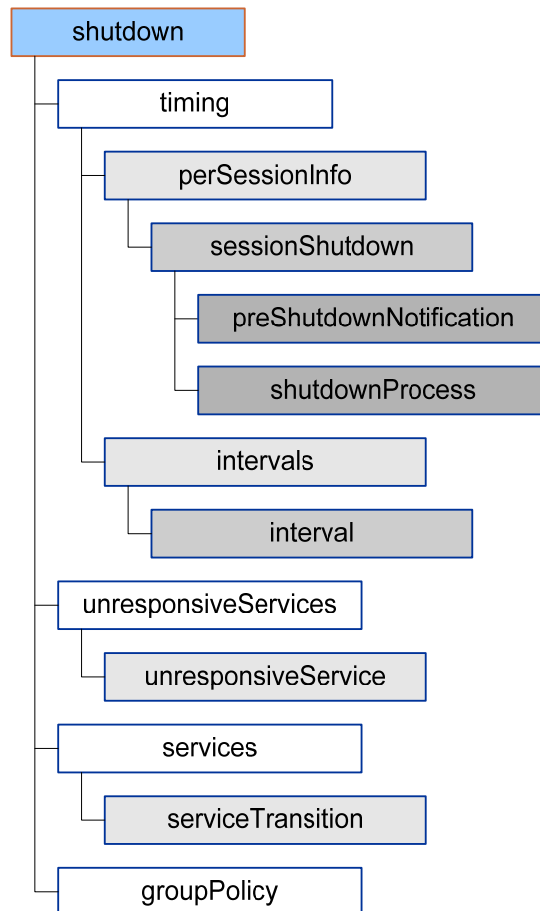


Figure A-3. Nodes in the shutdown transition report

shutdown

The **shutdown** node contains all the captured trace information.

The **shutdown** node contains the following subnodes:

timing

unresponsiveServices

services

groupPolicy

Application shutdown timings are reported in the **sessionShutdown** subnode, whereas service shutdown timings are reported in **unresponsiveServices** and **services**.

Each of these nodes contains subnodes and attributes that hold specific information.

timing

This node contains all the trace timing information that was captured for application shutdown and other phases.

The **timing** node contains the following subnodes:

perSessionInfo

intervals

Attributes

The **timing** node has the following attributes:

shutdownTime	The end-to-end shutdown time.
servicesShutdownDuration	The total duration of the service shutdown part of the shutdown transition.

perSessionInfo

The **perSessionInfo** node contains timing information for all the processes that are running within the session as indicated by the **sessionID** attribute of each **sessionShutdown** node.

The **perSessionInfo** node contains one or more instances of the following subnode:

sessionShutdown

sessionShutdown

The details of every running process that was shut down are included in this node.

The **sessionShutdown** node contains one or more instances of the following subnodes:

preShutdownNotification

shutdownProcess

Note: The **preShutdownNotification** node applies only to session 0, the session that contains Windows Vista system services. Preshutdown notifications let services take longer than 20 seconds before timing out, and so should be used extremely sparingly.

Attributes

The **sessionShutdown** node has the following attributes:

sessionID	The session ID.
startTime	The time when the sessionShutdown phase began, measured from the start of the trace.
endTime	The time when the sessionShutdown phase ended.
duration	The total duration of the sessionShutdown phase.

preShutdownNotification

During this phase, the system notifies the SSCM that the computer is shutting down. The SCM then has a default time of approximately 20 seconds to shut down all services that are running before the system terminates the SCM process

(services.exe). This procedure does not always let services gracefully shut themselves down.

Attributes

The **preShutdownNotification** node has the following attributes:

startTime	The time when the preShutdownNotification phase began, measured from the start of the trace.
endTime	The time when the preShutdownNotification phase ended.
duration	The total time that was spent by the preShutdownNotification phase.

shutdownProcess

This node contains the details of a process that has been shut down.

Attributes

The **shutdownProcess** node has the following attributes:

name	The name of the process.
shutdownStartTime	The time when the process began to shut down, measured from the start of the trace.
shutdownEndTime	The time when the process finished shutting down.
processEndTime	The time that the process shut down. A value of -1 means that the process did not complete its shutdown before timing out.
shutdownDuration	The total length of time that the process spent shutting down.
shutdownLevel	The shutdown priority for a process that is relative to other processes in the system. For more information, refer to "SetProcessShutdownParameters Function" on MSDN.
shutdownFlags	For Microsoft internal use only.

intervals

The **intervals** node contains details of the phases that are part of the shutdown transition.

These phases include the following:

WaitForWinstationShutdown

PreShutdownNotification

NtShutdownSystem

ZeroHiberFile

FlushVolumes

ZeroPageFile

IoShutdownSystem

WaitForProcesses

CmShutdownSystem

The **intervals** node contains one or more instances of the following subnode:

interval

interval

Each **interval** subnode marks a phase in the shut-down transition.

Attributes

The **interval** node has the following attributes:

name	The name of the interval.
startTime	The time at which the interval begins, measured from the start of the trace.
endTime	The time at which the interval ends. A value of -1 signifies that the phase did not end cleanly and that some component might have timed out.
duration	The duration of the interval.

unresponsiveServices

Services that do not perform the SCM handshake correctly, and therefore delay the shutdown transition, are identified in this node.

The **unresponsiveServices** node contains one or more instances of the following subnode:

unresponsiveService

Attributes

The **unresponsiveServices** node has the following attribute:

numUnresponsiveServices	The number of unresponsive services, and therefore the number of unresponsive Service subnodes.
--------------------------------	---

unresponsiveService

The name of a service that did not perform the SCM handshake correctly appears in this node.

Attributes

The **unresponsiveService** node has the following attribute:

name	The name of the services that did not respond.
-------------	--

services

For information about the **services** node, see “services” in the “Boot Phase” section of this appendix.

serviceTransition

For information about the **servicetransition** node, see “serviceTransition” in the “Boot Phase” section of this appendix.

groupPolicy

For information about the **groupPolicy** node, see “groupPolicy” in the “Boot phase” section of this appendix.