Windows On/Off Transitions Solutions Guide

May 8, 2009

Abstract

This document provides an overview of Windows® on/off state transitions and recommendations for PC system manufacturers and software developers to achieve superior on/off transition performance. The information in this paper is intended to help system manufacturers and developers optimize their respective systems, drivers, applications, and services to improve system response times during on/off transitions.

Except as noted, this information applies to the following operating systems:
 Windows 7
 Windows Vista®

References and resources discussed here are listed at the end of this paper.

For the latest information, see:
 <http://www.microsoft.com/whdc/>system/pnppwr/powermgmt/OnOffTrans.mspx

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.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, 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 |  |  |  |
| May 8, 2009 | Final Draft |

Contents

[Introduction 4](#_Toc229381301)

[About This Guide 5](#_Toc229381302)

[Testing Methodology 5](#_Toc229381303)

[Establishing Baseline Measurements 5](#_Toc229381304)

[Reducing Test Result Variance 6](#_Toc229381305)

[Part 1: Overview of On/Off Transitions 7](#_Toc229381306)

[Boot Transition 7](#_Toc229381307)

[BIOS Initialization 8](#_Toc229381308)

[OS Loader 8](#_Toc229381309)

[Main Boot Path 8](#_Toc229381310)

[Post-Boot 9](#_Toc229381311)

[Sleep and Hibernate Transitions 9](#_Toc229381312)

[Suspend Phase 9](#_Toc229381313)

[Resume Phase 10](#_Toc229381314)

[Shutdown Transition 12](#_Toc229381315)

[Part 2: Best Practices for PC System Manufacturers 13](#_Toc229381316)

[Recommendations on All Transitions for PC System Manufacturers 13](#_Toc229381317)

[Boot Transition Recommendations for PC System Manufacturers 14](#_Toc229381318)

[Sleep and Hibernate Transition Recommendations for PC System Manufacturers 15](#_Toc229381319)

[Part 3: Best Practices for Driver Developers 15](#_Toc229381320)

[Recommendations for Driver Developers on All Transitions 15](#_Toc229381321)

[Boot Transition Recommendations for Driver Developers 16](#_Toc229381322)

[Sleep/Hibernate Transition Recommendations for Driver Developers 17](#_Toc229381323)

[Part 4: Best Practices for Service Developers 18](#_Toc229381324)

[Boot Transition Recommendations for Service Developers 19](#_Toc229381325)

[Sleep and Hibernate Transition Recommendations for Service Developers 19](#_Toc229381326)

[Shutdown Transition Recommendations for Service Developers 20](#_Toc229381327)

[Part 5: Best Practices for Application Developers 21](#_Toc229381328)

[Boot Transition Recommendations for Application Developers 21](#_Toc229381329)

[Sleep and Hibernate Transition Recommendations for Application Developers 21](#_Toc229381330)

[Shutdown Transition Recommendations for Application Developers 22](#_Toc229381331)

[Part 6: Summary of Recommendations 23](#_Toc229381332)

[Checklist for PC System Manufacturers 23](#_Toc229381333)

[Checklist for Driver Developers 24](#_Toc229381334)

[Checklist for Service Developers 24](#_Toc229381335)

[Checklist for Application Developers 24](#_Toc229381336)

[Feedback 25](#_Toc229381337)

[Resources 25](#_Toc229381338)

[Performance Analysis Tools 25](#_Toc229381339)

[Power Management Best Practices 26](#_Toc229381340)

[Task and Service Best Practices 26](#_Toc229381341)

[Resources for Driver Developers 26](#_Toc229381342)

[Reference 27](#_Toc229381343)

# Introduction

Good performance during Windows® on/off transitions—boot, sleep, and shutdown—is critical for a good user experience. Customers can become frustrated by systems that take too long to boot or to shut down. Mobile systems that have lengthy sleep and hibernate transitions can unnecessarily deplete battery life.

This document provides recommendations for system manufacturers and developers of device drivers, applications, and services to achieve superior performance on all on/off state transitions. The paper describes each such transition and explains any implementation differences between Windows Vista® and Windows 7.

System extensions (applications, drivers, services, and devices) can negatively affect on/off transition times if they are not properly optimized. Poorly optimized extensions commonly result in the following:

* Delays
* Lack of parallelism
* Excessive resource consumption

By following the best practices that are outlined in this paper, you can avoid these problems and greatly reduce on/off transition times. We applied these performance optimizations to many systems in our laboratories and reduced boot-to-desktop time on some systems nearly 50 percent. On some systems, boot time decreased by a total of 40 to 50 seconds. However, the effect of each driver, service, or application on transition times is unique, and your results might differ.

System builders affect transition times when they select devices. With hard work and informed decision-making, you can achieve exceptionally fast on/off transition times. Systems with properly matched CPU, disk, and memory capabilities that have been thoroughly tested and optimized can post impressive numbers, and even systems that are designed to achieve low prices can perform well.

Table 1 shows Windows 7 boot times that were measured on four different real-world machine configurations. These boot times are measured from the end of BIOS until the end of the post-boot phase, when the system becomes relatively idle following a boot transition. Boot phases are further described in “Boot Transition” later in this document.

Table 1. Optimized Boot Times on Machines Running Windows 7

|  |  |  |
| --- | --- | --- |
| System type | General specifications | Optimized boot time |
| High-performance desktop | * Dual-core 2.8-GHz CPU
* 10,000-RPM disk
* 3-GB memory
 | 10.0 seconds |
| Thin laptop, solid-state disk (SSD) | * Dual-core 2-GHz CPU
* SSD
* 3-GB memory
 | 12.2 seconds |
| Thin laptop | * Dual-core 2.4-GHz CPU
* 5,400-RPM disk
* 3-GB memory
 | 19.3 seconds |
| Small notebook PC | * 1.6-GHz CPU
* 5,400-RPM disk
* 1-GB memory
 | 33.5 seconds |

Although these systems have good boot times, each one can still benefit from proprietary device and driver improvements. For example, booting the Windows operating system requires that a large amount of data be read from the machine’s hard disk. This data is not always sequential. Because frequent disk accesses can delay boot, good boot performance requires hard disk I/O optimizations.

By following the best practices in this document, system manufacturers can more effectively meet business goals and deliver better on/off transition performance with less engineering time and lower costs.

# About This Guide

The best practices in this guide are the result of several years of performance analysis that identified key performance issues and evaluated their impact on on/off transitions. Each section of the guide contains best practices for optimizing a specific component, so that you can easily find the information that most directly applies to your job role.

The first section of this document provides a detailed overview of all on/off state transitions. You can use this information to evaluate the performance impact of your devices, drivers, applications, or services. The next four sections describe best practices for system manufacturers, driver developers, service developers, and application developers, respectively. Finally, the guide summarizes all the recommendations.

“Resources” at the end of this guide provides a comprehensive reference to additional documents and tools that can help you improve on/off transitions.

# Testing Methodology

When you test a system for on/off transition performance, you should always establish a baseline measurement to use for comparison. You can then add the service, application, or device and driver that you want to assess and again measure the relevant transitions. If the transition time increases, the added system extensions might adversely affect user experience.

The Windows Performance Toolkit (WPT) can help you ensure that drivers, services, and applications do not negatively affect on/off transitions. The toolkit includes the Xbootmgr and Xperf tools, which can simplify on/off transition analysis. You can find extensive information about how to analyze on/off transitions in “On/Off Transition Performance Analysis of Windows Vista,” which is listed in “Resources.”

For more details on the WPT, see “Resources.”

## Establishing Baseline Measurements

Before adding any applications or services, establish a baseline measurement for proper comparison by measuring transition times on a clean installation of Windows (on the same hardware). To determine the net impact of each component on transition times, system manufacturers should first choose a minimal reference platform and then add devices, drivers, services, and applications one by one.

Follow these steps during evaluation:

1. Use Xbootmgr and Xperf in the WPT to measure on/off transition performance on a baseline system.

2. Install an application, service, or device and driver on the test system.

3. Use Xperf and Xperfview to identify issues and possible improvements.

4. Update and reinstall the driver, application, or service.

5. Re-measure on/off transition times and compare improvement.

6. Repeat from step 2 until all system extensions are added.

System manufacturers should also test for variability in on/off performance over a long period of time. In internal testing at Microsoft, we booted a system over 2,000 times to look for variability in boot time. This system showed standard deviations of up to 3.1 seconds. Such testing can also help detect issues that occur infrequently and ensure that the addition of more applications and services does not cause large variations in on/off transition times.

## Reducing Test Result Variance

To ensure the reliability of performance test data, you must minimize variance. The Xperf and Xbootmgr tools capture many system events, so the key to minimizing variance is controlling system activity. During testing, you must carefully control the boot prefetcher, network connections, log‑on process, and any user interactions.

Windows Vista and Windows 7 have a self-optimizing feature—called the boot prefetcher—that uses the disk I/O pattern from the previous boots to predict the I/O pattern of the next boot. To improve boot performance, the prefetcher predicts what data is required and reads this data from the hard disk before the data is actually required. This optimization can make it difficult to create repeatable performance results.

The boot prefetcher is a heuristic algorithm, so results do not always improve with each run. However, the algorithm is designed to “settle,” which generally reduces variance as more training boots are performed. Table 1 demonstrates the effects of the prefetcher on a system that is undergoing five training boots. The “Untrained” entry in the table is the system without optimization.

Table 2. Boot Prefetcher Effects on Windows 7 Boot Times

|  |  |  |
| --- | --- | --- |
|  | Time to Explorer.exe start(in milliseconds—ms) | Time to boot completion(in ms) |
| Untrained | 23,696 | 40,696 |
| Boot 1 | 23,098 | 38,698 |
| Boot 2 | 22,005 | 37,805 |
| Boot 3 | 21,553 | 38,353 |
| Boot 4 | 21,855 | 38,055 |
| Boot 5 | 21,827 | 37,527 |

To reduce variance, we recommend that you run three to five “training” boots before you take test measurements to allow the prefetcher to fully self-optimize. Wait at least 2 minutes between training runs to allow prefetcher training to complete.

The WPT can perform automated boot training and ensure that all optimizations are performed. To automate the training process, enable training in WPT and specify the number of training boots to perform. In response, WPT reboots the system twice, writes the prefetcher data, and then performs the specified number of boot transitions.

Code Integrity checks occur after new driver installation. If you want to run a set of training boots, allow systems that have newly installed drivers to idle for 15 minutes before you start the training run to ensure that code integrity checks do not interfere with results.

The presence of a network connection can affect 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

Any unnecessary user interaction will appear in performance traces. You should not interact with the systems under test during test runs because such system activity is included in the trace and can distort the results. To minimize variance from user interactions, make the user account that runs Xbootmgr a member of the Administrators group, disable User Account Control, and configure user profiles for autologon.

# Part 1: Overview of On/Off Transitions

Windows systems perform three types of on/off transitions:

* Boot
* Sleep and hibernate
* Shutdown

Each transition has several phases. Before you can evaluate the performance impact of a driver, service, or application, you should understand how each type of on/off transition works.

For more detail about on/off transitions, see “On/Off Transition Performance Analysis of Windows Vista,” which is listed in “Resources.”

## Boot Transition

*PC Performance Focus Groups*, a commissioned study that Forrester Consulting conducted for Microsoft in 2009, shows that customers perceive boot time as a primary indicator of system performance. Customers whose machines boot quickly are more likely to feel positive about their overall computing experience.

The boot transition involves the following four phases:

* BIOS initialization
* OS loader
* Main path boot
* Post-boot

Figure 1 shows the phases of the boot transition.

Figure 1. Phases of the boot transition

### BIOS Initialization

During the BIOS initialization phase, the platform firmware identifies and initializes hardware devices and then runs a power-on self-test (POST). The POST process ends when the BIOS detects a valid system disk, reads the master boot record (MBR), and starts the Windows Bootmgr executable. Bootmgr.exe locates and calls Winload.exe on the Windows boot partition, which begins the OS loader phase.

### OS Loader

During the OS loader phase, the Windows loader binary (Winload.exe) loads essential system drivers and initializes the system until the Windows kernel can begin to execute. During this phase, device drivers that are categorized as BOOT\_START and the system registry hive are loaded into memory. OS loader is an I/O-bound phase, so fast storage can significantly reduce the duration of this phase.

### Main Boot Path

Most work that is required to boot the system is performed in the main path boot phase. To better explain the best practices, we divide the main path boot phase into four subphases, as Figure 2 shows.



Figure 2. Main path boot subphases in boot transition

The following are the four subphases of the main path boot phase:

* **PreSMSS.** The PreSMSS subphase begins when winload.exe invokes the kernel. During this subphase, the kernel initializes data structures and system components. As a part of this initialization process, the kernel starts the PnP manager, which initializes the BOOT\_START drivers that were loaded during the OS loader phase.
* **SMSSInit.** The SMSSInit subphase begins when the kernel passes control to the session manager process (Smss.exe). During this phase, video drivers are initialized and the subsystem start processes—including Smss.exe, Csrss.exe, Wininit.exe, and Winlogon.exe—are started. SMSSInit ends when control passes to Winlogon.exe.
* **WinLogonInit.** Although Service Control Manager (SCM) launches auto-start services at the end of SMSSInit, most of its work occurs during the WinLogonInit subphase. The logon screen appears, and the desktop starts during WinLogonInit. The subphase ends when Explorer.exe starts.
* **ExplorerInit.** Explorer.exe is initialized during the Explorerinit subphase. Services and applications continue to start during this subphase.

### Post-Boot

After main path boot is complete, the desktop is visible and the user can interact with it. The post-boot phase then begins. During this phase, additional services and applications start and can limit the user’s ability to perform tasks even though the desktop is visible. The post-boot phase continues until Windows achieves a reasonably idle level of activity after the desktop appears.

## Sleep and Hibernate Transitions

Mobile users rely on their computer’s ability to quickly perform sleep and hibernate transitions. On mobile systems, improved performance on these key Windows scenarios increases user convenience and extends battery life. If a sleep or hibernate transition takes too long, users might choose to shut down their computer instead and lose key advantages. For instance, wake-on-LAN (WOL), an extremely useful feature for conserving energy and allowing remote access to a computer, is possible only if the system is in the sleep or hibernate state.

During sleep transition, the system enters a low-power state and system context is retained in main memory.

During the hibernate transition, system context is saved on the hard drive and the system completely shuts down. However, systems that have large amounts of RAM might face performance challenges because more information must be stored on the hard drive during suspend and restored into memory during resume.

The sleep and hibernate transitions involve two phases: suspend and resume. The following sections provide more detail about these two phases.

### Suspend Phase

During the suspend phase, Windows saves system context and shuts off the devices that are required to enter the target state—either sleep or hibernate. Figure 3 shows the subphases of the suspend phase during sleep and hibernate transitions.



Figure 3. Subphases of the suspend phase for sleep and hibernate transitions

During a sleep transition, suspend involves the following subphases:

* **SuspendApps.** Suspend notifications (WM\_POWERBROADCAST messages that have an event type of PBT\_APMSUSPEND) are serially sent to all applications. For each application, the system sends the WM\_POWERBROADCAST message, waits up to 2 seconds for the application to return from processing the message, and then sends the message to the next application. After Windows notifies all applications, it continues to the SuspendServices subphase. Any applications that fail to complete all their work before the 2-second time-out can continue any remaining processing after the system resumes from sleep.
* **SuspendServices.** All services that registered to receive power management events receive suspend notifications.
* **QueryDevices.** All device drivers receive a power I/O request packet (IRP) that has the IRP\_MN\_QUERY\_POWER minor IRP code and requests a system power state. This request notifies the driver that the system power state is about to change and gives the driver the opportunity to prepare for the change. Most drivers do very minimal work to process this IRP because the typical duration of this subphase is less than 20 milliseconds.
* **SuspendDevices.** All device drivers receive a system power IRP with the IRP\_MN\_SET\_POWER minor IRP code and requests a lower power state. Device drivers must then save appropriate device context and put the device in the appropriate state for sleep or hibernate.

A hibernate transition involves the following additional subphase:

* **HiberfileWrite.** This subphase applies only to hibernate and hybrid sleep transitions. Hybrid sleep, which was introduced in Windows Vista, combines sleep and hibernate. The system enters sleep after it saves a minimal system context to disk. This allows the system to be restored even if an unexpected power failure occurs. The Hiberfile is written to disk before the system enters sleep.

### Resume Phase

During the resume phase, the system returns to power state S0, the working state. System context is restored to memory, and devices are restarted. The subphases of the resume phase differ, depending on whether the system is resuming from sleep or from hibernate. Figure 4 shows the subphases of the resume phase for the sleep transition.



Figure 4. Subphases of the resume phase for sleep transition

Resume from sleep involves the following subphases:

* **BIOS Initialization.** When a system resumes from sleep (S3 state), the first subphase is BIOS initialization. In this subphase, the platform firmware restores the processor and chipset state.
* **ResumeDevices.** The Windows power manager notifies drivers that the system is resuming by sending system set-power IRPs to all device stacks. This subphase is considered complete when all drivers complete the system set-power IRPs.
* **ResumeApps.** The system sends to each application a WM\_POWERBROADCAST message with the *wParam* event value of PBT\_APMRESUMEAUTOMATIC. User-mode applications might also receive this message with the PBT\_APMRESUMESUSPEND value if a user-initiated action, such as pressing a power button, triggered wake.
* **ResumeServices.** The power managercalls an interface in the user-mode PnP manager, which then calls each service’s power management event handler.
* **PostResume.** The PostResumesubphase is a conceptual subphase that encompasses the time that follows the resume transition until the system returns to a fairly idle state.

As Figure 5 shows, resume from the hibernate transition includes the POST subphase instead of BIOS Initialization and adds the HiberfileRead subphase. All other phases are the same as in resume from sleep.

Figure 5. Subphases of the resume phase for hibernate transition

The POST and HiberfileRead subphases are as follows:

* **POST.** When a system resumes from hibernate (ACPI S4 state), the first subphase is POST.
* **HiberfileRead.** During HiberfileRead, drivers in a special driver stack read the Hiberfile into memory because the normal disk device stack has not yet been loaded. This subphase applies only to hibernate and hybrid sleep transitions.

## Shutdown Transition

Reducing shutdown time is vital for a successful overall user experience. A lengthy shutdown time can frustrate users, increase service times for system administrators, and adversely affect system reliability. For instance, if shutdown takes too long, the user might press the power button instead and potentially cause data loss.

Services and applications can significantly affect shutdown performance. Excessive CPU and disk activity can also potentially delay system shutdown, which causes a poor user experience.

Figure 6 shows the three main phases of the shutdown transition.

Figure 6. Phases of the shutdown transition

The shutdown transition includes the following phases:

* **User session shutdown.** During this phase, all applications that are running in the user session—with session ID 1—are shut down.

When the user chooses **Shut Down** from the **Start** menu, Windows sends one or two shutdown messages (WM\_QUERYENDSESSION and WM\_ENDSESSION messages) to each user interface (UI) thread in each graphical user interface (GUI) application. If after 5 seconds any application blocks shutdown, Windows displays the dialog box in Figure 7 so that users can choose to force or cancel shutdown.



Figure 7. Shutdown dialog box in Windows Vista and Windows 7, waiting for user input

To shut down a console application, Windows invokes the console control handler by sending CTRL\_LOGOFF\_EVENT and waits up to 5 seconds for a response.

If the user selects **Force shut down** from the dialog box, the system gives all applications 0.25 second to respond to WM\_QUERYENDSESSION messages and 0.5 second to respond to WM\_ENDSESSION messages, and then terminates any remaining applications.

For system-forced shutdowns, Windows does not display a dialog box. Instead, it immediately terminates any hung applications and terminates all GUI and console applications after the 5‑second time-out. Shutdowns that are triggered by the installation of updates are part of this category.

* **System session shutdown.** This phase includes the pre-shutdown notification and shutdown notification subphases:
* **Pre-shutdown notification.** Windows serially shuts down all services that registered to receive pre-shutdown notifications. Ordered services—services that have set up the shutdown order of dependent services—are shut down before non-ordered services.
* **Shutdown notification.** All services that registered to receive shutdown notifications are shut down in parallel.

If all services have not exited after 20 seconds (in Windows Vista) or 12 seconds (in Windows 7 client operating systems), the system continues the shutdown. Processes and services that do not shut down in a timely manner are left running as the system shuts down.

* **Kernel shutdown.** The remainder of the system, such as all devices and drivers, are shut down during the kernel shutdown phase.

# Part 2: Best Practices for PC System Manufacturers

System manufacturers can significantly improve on/off transition performance by optimizing their hardware, software, and firmware configurations. This section discusses specific best practices that can help system manufacturers to achieve performance goals.

## Recommendations on All Transitions for PC System Manufacturers

The following recommendations apply to all on/off transitions.

* Select devices carefully because configuration affects transition performance.

Differently configured systems have different transition performance characteristics. As the mix of installed devices changes, so do delays, parallelism, and resource usage during transitions. A system manufacturer can improve performance by making smart configuration decisions.

As Table 1 in “Introduction” shows, choosing a fast solid-state drive (SSD) instead of a slow rotational disk can significantly reduce boot time. Incorporating integrated device electronics (IDE) instead of Serial ATA (SATA) is a less dramatic example, but still results in a 1‑second delay during Plug and Play enumeration of the IDE channels, as described later in this section.

In this sense, transition performance is a useful metric for a system manufacturer. You can vary the installed devices and then test transition times to find configurations that meet desired points on the price/performance curve.

* Ensure that all drivers, applications, and services that you install on the system are optimized for performance.

For more information, see:

* [PART 3: Best Practices for Driver Developers](#_PART_3:_Best)
* [PART 4: Best Practices for Service Developers](#_PART_4:_Best)
* [PART 5: Best Practices for Application Developers](#_PART_5:_Best)

## Boot Transition Recommendations for PC System Manufacturers

* Minimize BIOS initialization time.

Boot transition time depends on both the BIOS and the operating system. BIOS initialization is becoming an increasingly larger part of the boot transition process. To minimize BIOS effects on boot transition, select a BIOS with minimal initialization time or configure the system BIOS for fast POST.

If the fast POST flag is set, the fastest possible BIOS POST time is important. A POST of 1.5 seconds or less lets system configurations that include SSDs begin booting the operating system more quickly than most mechanical drives can even spin up. A good target for optimization is any BIOS that has a fast POST that takes longer than 3 seconds.

* Apply the latest firmware updates to all system components.

On some recently tested systems, common system devices such as optical disk drives had significant (up to a 10‑second) initialization delays during boot. Often these problems were resolved by updating the device firmware. To reduce problems during boot transition, obtain and apply the latest firmware versions for system components.

* Balance CPU, disk, and memory for best boot time.

Boot phases are sometimes CPU bound and at other times I/O bound. The best way to ensure a quick boot time is to balance memory, disk, and processor speeds and capacities. For example, a system that has top-of-the-line processors, 1 GB of memory, and a 4,200‑RPM disk usually results in poor boot performance. For best results, match speeds and capacities.

* Choose SATA rather than IDE/PATA for the storage bus.

Each PATA channel requires a minimum 1‑second delay during enumeration, which represents a significant delay during a critical section of the boot process. SATA storage buses that are configured in native mode do not experience these enumeration delays. For the best Plug and Play initialization time, choose native-mode SATA.

Consider disabling unused PATA channels on mobile computers if the channels cannot be connected to a device. In such nonextensible system scenarios, you can avoid the enumeration delay and minimize the risk of support issues.

* Additional RAM can improve boot performance.

To save memory space on systems that have less than 2 GB of RAM, prefetched data is compressed. Compression requires CPU cycles and thus can slow boot time. This effect is especially pronounced on PCs that have slower processors. By providing 2 GB or more of memory, you can improve boot times, but you should base this decision on the target market, cost, and usage model of the PC.

* Optimize the disk layout.

For better sequential read performance during boot, optimize rotational disk layout. On the 5,400‑RPM rotational disks that we tested, sequential read from the outer tracks delivered as much as 1.75 times more data per second than sequential read from the inner tracks. Boot involves some sequential read activity, so you can potentially increase performance if you locate the system partition that contains the Windows installation on the outer tracks. To lay out system partitions, use the Diskpart.exe tool that is included with Windows. For best performance, create the system partition on the outer edge of the disk (the lowest numbered disk offsets).

For more information on Diskpart.exe, see “Diskpart Command-Line Options,” which is listed in “Resources.”

## Sleep and Hibernate Transition Recommendations for PC System Manufacturers

* Optimize S4 resume by enhancing BIOS INT13 performance.

You can optimize performance in the HiberfileRead subphase by carefully analyzing and enhancing BIOS INT13 performance. Windows uses these routines for disk I/O to read the hibernate image file from the disk during S4 resume. Consider the following optimizations:

* Tune INT13 performance for maximum read performance.
* Enable your disk controller for the highest available direct memory access (DMA) mode in the INT13 calls.
* Optimize the INT13 calls for large read requests (64 KB).
* Optimize for sequential sector reads, that is, multiple 64‑KB reads that are issued in a row, if requests are physically adjacent.

# Part 3: Best Practices for Driver Developers

Drivers can significantly impact the performance of all on/off transitions. For example, a driver can significantly increase boot time by not returning STATUS\_PENDING for the appropriate IRPs during initialization. Drivers that do not implement fast resume can also adversely affect resume from sleep/hibernate performance.

## Recommendations for Driver Developers on All Transitions

* Use the Windows Driver Foundation (WDF) to write drivers.

Driver developers should use WDF for writing drivers. WDF includes frameworks for developing both kernel-mode and user-mode drivers and includes several driver verification tools. By using WDF, driver developers can focus on the functional requirements of their hardware instead of the low-level requirements of the operating system.

WDF greatly simplifies the implementation of Plug and Play and power management support for drivers. It provides intelligent default handling for Plug and Play and power requests, so that drivers are not required to handle requests that do not apply to their devices. Instead, a driver “opts in” to handle the events that are relevant to its device—and those events map to a well-defined task.

WDF implements Plug and Play and power management by using several internal state machines that define such tasks at a much finer level than the WDM IRP\_MJ\_PNP and IRP\_MH\_POWER requests. Therefore, WDF drivers are more efficient and require significantly less Plug and Play and power management code than Windows Driver Model (WDM) drivers.

For example, WDF drivers are not required to handle system power IRPs or track the system power state to determine which state their devices should be in. Instead, they inform the framework of the device capabilities and implement callback functions that are invoked to perform device-specific tasks at specific points during on/off transitions.

For more information on WDF, see the WHDC Web site, which is listed in “Resources.”

## Boot Transition Recommendations for Driver Developers

* Embedded-sign BOOT\_START drivers.

During the OS loader subphase of boot, Windows verifies that BOOT\_START drivers are signed. By embedded-signing the binary driver file, you can eliminate the performance cost of retrieving the signature from a catalog file. Otherwise, the loader must find several catalog files on disk and load them into memory, which reduces performance.

* Configure a driver as BOOT\_START only if required.

Only the minimal set of drivers that are required to boot the system should be configured as BOOT\_START. Adding unnecessary drivers to the BOOT\_START set prolongs the OS loader subphase and potentially increases boot time.

For more details on selecting a startup type for your drivers, see “Specifying Driver Load Order,” which is listed in “Resources.”

* Return STATUS\_PENDING from initialization IRPs.

During Plug and Play enumeration, the PnP manager detects Plug and Play devices, loads their drivers into memory, and initializes the drivers.

During initialization, the PnP manager sends the drivers an IRP\_MN\_START\_DEVICE request to start their devices, followed by IRP\_MN\_QUERY\_DEVICE\_RELATIONS to enumerate child devices. This is a serial process and the PnP manager waits until a driver returns STATUS\_PENDING or STATUS\_SUCCESS. With the exception of some miniport and bus drivers, drivers can return STATUS\_PENDING before their device startup and enumeration routines have completed, so that the PnP manager can continue to start other devices at the same level of the device tree. The PnP manager cannot, however, continue to a new level of the tree until all drivers at the current level return STATUS\_SUCCESS.

By returning STATUS\_PENDING, a driver can increase the amount of parallelization during system PnP initialization and thereby reduce boot time. To ensure the best boot transition times, all drivers should return STATUS\_PENDING from these two IRPs.

**Note:** Drivers that use the kernel-mode driver framework (KMDF) are not required to handle these requests. KMDF invokes any driver-specific callback functions and automatically returns STATUS\_PENDING for the IRP\_MN\_START\_DEVICE and IRP\_MN\_QUERY\_DEVICE\_RELATIONS requests.

For more information, see “Resources for Driver Developers” that are listed in “Resources.”

* Use the SYSTEM registry hive in driver code.

The SYSTEM hive is loaded during the early phases of boot. Conversely, the SOFTWARE hive is loaded on demand during SMSSInit. A driver that tries to access the SOFTWARE registry hive in its **DriverEntry** routine might delay all other execution block while the system loads the SOFTWARE hive. For best boot performance, use the SYSTEM registry hive to store any configuration data.

* Reduce driver file size.

Large driver files add to disk access and I/O transfer times and therefore can delay initialization during boot. If most of the code in a kernel-mode driver is marked nonpaged, a large driver file might also have a large resident memory footprint. Reduce driver file size as much as possible. If a single driver binary file supports multiple devices in a device class, consider using one driver file per device to improve boot time and possibly reduce the memory footprint.

An easy way to reduce the size of kernel-mode driver files is to use KMDF. KMDF drivers have smaller binary files than WDM drivers do because they require less code. KMDF drivers dynamically link with the framework at load time, and one instance of the framework supports all KMDF drivers on the system.

* Reduce display adapter initialization time.

Most of the SMSSInit phase of boot is spent initializing video in session 0 and session 1. Initialization can take several seconds. By reducing the display adapter initialization time, you might be able to reduce overall boot time by a similar amount.

## Sleep/Hibernate Transition Recommendations for Driver Developers

* Implement fast resume.

WDF automatically handles fast resume for KMDF and user-mode driver framework (UMDF) drivers.

To achieve fast resume performance in a WDM driver, follow these guidelines. The guidelines apply only to drivers for leaf node devices—that is, devices that have no child devices in the device hierarchy:

1. Set an I/O completion routine for the IRP\_MN\_SET\_POWER request for the S0 system power state.

2. Send the IRP\_MN\_SET\_POWER IRP for the S0 system power state down the stack.

3. In the I/O completion routine for the S0 system power state IRP\_MN\_SET\_POWER IRP:

1. Send an IRP\_MN\_SET\_POWER request for the device D0 power state.
2. Return STATUS\_SUCCESS from the I/O completion routine for the S0 system power state IRP\_MN\_SET\_POWER request.

4. Complete any device initialization that is required to bring the device online in the I/O completion routine for the D0 set-power IRP.

5. Begin handling any queued I/O requests after the device has returned to the D0 state.

In addition, remember the following guidelines when you optimize a driver for fast resume:

* Windows records “resume complete” after all devices have completed the S0 set-power IRPs. Devices are not required to complete their D0 set-power IRPs or to be fully functioning at this point.
* For better resume performance, drivers can complete the S0 set-power IRP before the D0 set-power IRP is completed. The power manager can then send the S0 set-power IRP to other devices sooner, which reduces serialization delays and enhances overall system resume performance.
* To ensure the best user experience, drivers should queue any I/O requests received while they process the D0 set-power IRP. Failing I/O requests during this time might cause applications to hang or produce time-out error messages.
* Optimize CPU resource usage on resume.

To enhance the post-resume user experience, drivers can optimize use of CPU resources when the system resumes from sleep or hibernate. The performance of deferred procedure call (DPC) routines and other code that runs at IRQL >= DISPATCH\_LEVEL is an important consideration.

Excessively long DPC routines can block other threads from executing and can block other DPC routines that are queued and ready to execute. When threads or DPCs are blocked, the duration of the device initialization phase increases and extends the overall system resume time. The best resume times are achieved when individual DPC routines do not execute for more than 100 µs. If a task must run at DISPATCH\_LEVEL for longer than 100 µs, the DPC routine should terminate after 100 µs and schedule one or more DPC timer routines to complete the task later.

# Part 4: Best Practices for Service Developers

Services can have a significant impact on on/off transitions. For example, in Windows 7 a service that is unresponsive during shutdown can delay system shutdown by 12 seconds. This section discusses important best practices that service developers can follow to achieve superior performance on all on/off transitions.

## Boot Transition Recommendations for Service Developers

* Convert nonessential services to demand start or trigger start.

A service that uses the automatic startup type starts as part of the boot transition and therefore affects boot performance. Use the automatic startup type only if the service provides critical functionality that cannot wait until after boot is complete.

If a service is not required during boot, change the startup type to demand start or trigger start. This change makes critical system resources available for kernel and system service initialization. It also results in less activity during the later stages of the boot transition and provides an effective alternative to designs that would otherwise wake at regular intervals or at system idle.

**Note:** Demand start and trigger start for services are available only in Windows 7.

For more information on demand start and trigger start, see “Developing Efficient Background Processes for Windows,” which is listed in “Resources.”

* Do not use load order groups to express service dependencies.

Services can declare dependencies or use load order groups to ensure that they start in a specific order. Windows processes load order groups in serial order. If a service is part of an early load order group, delays in that service can block subsequent load order groups and possibly block the boot process in general. To achieve the best boot performance, remove services from load order groups altogether.

* Ensure that services that are running report this status as soon as possible.

The Service Control Manager (SCM) cannot start subsequent load order groups until all services in the current group report themselves as running. Dependencies within a load order group similarly block the SCM while services are serially started. To avoid delaying the SCM and the entire boot process, services should report their status as running as soon as possible.

* Avoid using managed code on the boot path.

To write services that have a minimal impact on the startup path in managed code is challenging. You must be very selective in the common language runtime (CLR) services that you use. Although managed code has definite advantages, it is not suitable for the startup path.

## Sleep and Hibernate Transition Recommendations for Service Developers

* Avoid registering to receive suspend notifications.

If a service is not required to complete any significant work during the suspend phase, it should not opt in to receive suspend notifications. Windows serially sends these notifications to services, and each such notification adds to the overall suspend time of the system.

A driver that defers any required work until the resume notification and then performs it in the background without interrupting the user helps to minimize suspend times.

* Minimize post-resume activity.

During the ResumeServices subphase, the kernel power manager calls an interface in the user-mode PnP manager, which then calls each service’s power management event handler to notify the service that the system has resumed to the working state. These notifications are asynchronous, and the kernel power manager considers this subphase complete as soon as it finishes its call to the user-mode PnP manager. As a result, services should follow these guidelines to ensure a satisfying post-resume user experience:

* Optimize CPU and disk usage immediately after the system resumes. During the post-resume phase, the user can interact with the system and might try to quickly access the calendar or other data at the same time that several services and applications respond to resume events. By analyzing the CPU and Disk Utilization graphs, as well as comparing the graphs against valid baseline measurements, you can identify anomalous CPU and disk activity during the ResumeServices subphase.
* Consider using the task scheduler or trigger start mechanism for periodic service work such as software updates so that these tasks do not start immediately after resume.
* If your service depends on a network, maintain awareness of connectivity status and respond reasonably and transparently to transitions in network connectivity.

For more details on power management events and measuring system resume performance, see “Task and Service Startup Best Practices” and “Power Management Best Practices” that are listed in “Resources.”

## Shutdown Transition Recommendations for Service Developers

* Avoid opting in to receive pre-shutdown and shutdown notifications.

Services should opt to receive pre-shutdown and shutdown notifications only if the services must handle these notifications. In the pre-shutdown notification subphase, the SCM serializes the waits. Therefore, these services block system shutdown until they exit or until the *wait hint* time-out expires.

Services are not guaranteed to have enough time to finish all their work in the shutdown notification subphase before the system shuts down.

* Respond quickly to shutdown notifications from the SCM.

A service that registers to receive shutdown notifications through its service control handler must respond quickly to such notifications from the SCM. Services are shut down in parallel. Therefore, the total duration of the shutdown notification subphase is approximately the longest time that any one service takes to shut down. To properly shut down without adversely affection shutdown transition time, a service should:

* Mark itself as STOP\_PENDING immediately after it receives the shutdown notification, which returns control to the SCM.
* Enter the stopped state (SERVICE\_STOP) as soon as possible.

For specific guidelines on writing service control handler functions, see “Service Control Handler Function” on MSDN, as listed in “Resources.”

# Part 5: Best Practices for Application Developers

Applications with poor performance can significantly affect the overall user experience on a system. Application developers should follow the guidelines in this section to avoid delays during boot, standby, and shutdown, and to ensure a responsive post-boot and post-resume user experience.

## Boot Transition Recommendations for Application Developers

* Reduce application resource consumption during post-boot.

Post-boot is a busy time on the system. Resources are limited and generally have considerable contention in the seconds after the desktop is ready. To improve system responsiveness and user perceptions of system performance, applications that are scheduled to start as a part of the boot process should minimize their use of resources such as disk and CPU.

* Avoid using managed code on the boot path.

Properly writing applications for the startup path in managed code is challenging. You must be very selective in the use of CLR services. Although managed code has definite advantages, it is not suitable for the startup path.

* Avoid starting applications from the **RunOnce** key on every boot.

As its name implies, the **HKCU\...\RunOnce** registry key was designed for the *one-time* application start. It is the only “startup” key on the system that blocks Explorer.exe. If an application must start on every boot, you can shorten the time to desktop and increase parallelism during the boot transition by starting it from the **Run** key or the **Startup** folder instead of **RunOnce**. To ensure the best transition time, do not start nonessential applications during boot.

## Sleep and Hibernate Transition Recommendations for Application Developers

* Avoid WM\_POWERBROADCAST processing delays.

During the SuspendApps subphase, Windows serially sends to all applications a WM\_POWERBROADCAST message with an event type of PBT\_APMSUSPEND. After the system sends the WM\_POWERBROADCAST message, it waits up to 2 seconds for the application to return from processing the message before it sends the message to the next application. The system then continues the suspend process. Any applications that do not complete all their work before the time-out must continue any remaining processing after the system resumes from sleep.

Because of the serial notification, every application that does not respond quickly to the WM\_POWERBROADCAST message blocks the suspend path and directly adds to the overall suspend time on the system. Applications do not always have sufficient CPU time to complete their work before the 2-second time-out elapses. Because of this, several other processes might run concurrently on the system and consume CPU resources.

* Minimize post-resume activity.

During the ResumeApps subphase, Windows sends to all applications a WM\_POWERBROADCAST message with the *wParam* event value of PBT\_APMRESUMEAUTOMATIC. User-mode applications might receive this message with the PBT\_APMRESUMESUSPEND value if a user-initiated action, such as pressing a power button, triggered wake. These notifications are asynchronous, and the kernel power manager considers this subphase complete after it notifies the USER subsystem.

Follow these general guidelines when handling these events in an application:

* Maintain awareness of connectivity status and respond reasonably and transparently to transitions in network connectivity. Automatically reconnect to network resources in the background without interrupting the user.
* Manage resource usage immediately after a resume-from-sleep transition. Several applications can respond to resume events at the same time, and the user might attempt to quickly access the calendar or other data. Applications that perform resource-intensive system scans should wait until the system is idle before they start such scans.
* If tracking user presence is critical to the application, update this user information immediately upon resume. Use the PBT\_APMRESUMESUSPEND event to determine whether the user is present at the system.

To improve system responsiveness, applications should minimize CPU and disk utilization during the post-resume subphase. By analyzing the CPU and Disk Utilization graphs and comparing them against valid baseline measurements, you can identify anomalous CPU and disk activity during this subphase.

For more details on power management events and measuring system resume performance, see “Application Power Management Best Practices for Windows Vista,” which is listed in “Resources.”

## Shutdown Transition Recommendations for Application Developers

* Respond quickly to shutdown notifications.

Windows serially shuts down applications. Consequently, each delay in a process’s shutdown path contributes to the total duration of shutdown. To ensure a speedy shutdown, each application must respond quickly to shutdown notification messages (WM\_QUERYENDSESSION and WM\_ENDSESSION).

Although time-outs are lengthy so that applications have enough time to shut down and preserve user data, failure to respond quickly to shutdown notifications can significantly affect system shutdown performance, which leads to a poor user experience.

* Reduce shutdown path delays.

Ensure that the shutdown path contains no significant delays. For example, a word processing application that does not implement “Auto-save*”* might store considerable information in memory and be required to write all modified pages back to the respective files. Such unnecessary processing delays the application’s response to shutdown notifications and therefore delays shutdown itself.

To reduce application shutdown delays, consider the following guidelines:

* Avoid excessive disk flushes at the time of shutdown, such as the following two examples:
* Disk flushes that the application initiates (such as calling **FlushFileBuffers**). Minimizing the amount of unsaved data at shutdown avoids excessive disk flushes. You can do this by performing regular disk flushes (that is, by regularly and automatically saving data during processing), keeping track of data that is saved to disk, and saving only unsaved data at shutdown.
* Registry flush on shutdown. Avoid calling explicit flushes during shutdown because this increases overall shutdown time.
* Avoid network dependencies during shutdown. Network problems can delay the shutdown of applications.

# Part 6: Summary of Recommendations

This section summarizes the best practices that were discussed in this guide. PC system manufacturers, driver developers, service developers, and application developers who follow these recommendations can achieve superior on/off transition performance:

* Use the WPT to measure performance during all on/off transitions.
* Perform testing in a controlled way, and make comparisons against a valid baseline. Obtain a baseline measurement on a system with as few system extensions as possible. Add devices, applications, and services one at a time, and test for unacceptable regressions in on/off transition times.

## Checklist for PC System Manufacturers

* Use the WPT to measure system performance during all on/off transitions.
* Manage transition performance by carefully selecting devices.
* Ensure that all drivers, applications, and services that are installed on the system are optimized for performance.
* Minimize BIOS initialization time.
* Apply the latest firmware updates to all system components.
* Balance CPU, disk, and memory to achieve the shortest boot time.
* Choose SATA rather than IDE/PATA for the storage bus.
* Consider installing more RAM to minimize compression of prefetched data during boot.
* Optimize the disk layout to improve sequential read performance during boot.
* Optimize resume from hibernate performance by enhancing BIOS INT13 performance.

## Checklist for Driver Developers

* Use the WPT to measure driver performance during all on/off transitions.
* Use Windows Driver Foundation (WDF) to write drivers.
* Ensure that all WDM drivers return STATUS\_PENDING from initialization IRPs to increase parallelization during Plug and Play initialization.
* Ensure that all BOOT\_START drivers are embedded-signed to avoid the performance cost of checking catalog files.
* Mark drivers as BOOT\_START only when required.
* Ensure that all drivers implement fast resume and optimize CPU resource usage on resume to achieve better resume performance.
* Use the SYSTEM registry hive instead of the SOFTWARE hive in driver code.
* Reduce driver file sizes.
* Reduce display adapter initialization time.

## Checklist for Service Developers

* Use the WPT to measure the performance of services during all on/off transitions.
* Ensure that all nonessential services are converted to demand start or trigger start to make system resources available during boot.
* Avoid using load order groups to express service dependencies.
* Ensure that all running services report this status as soon as possible during boot to avoid blocking the Service Control Manager (SCM).
* Avoid using managed code for services on the startup path.
* Ensure that all services do not opt in to receive pre-shutdown and shutdown notifications (SERVICE\_CONTROL\_PRESHUTDOWN and SERVICE\_CONTROL\_SHUTDOWN control codes) unless absolutely required.
* Ensure that all services that have opted to receive shutdown notifications respond quickly to the SCM.
* Verify that services do not opt in to receive suspend notifications unless absolutely required.
* Ensure that all services respond quickly to resume events and minimize post-resume CPU, disk, and network usage.

## Checklist for Application Developers

* Avoid using managed code for applications on the critical boot path.
* Ensure that all applications respond quickly to shutdown notifications (WM\_QUERYENDSESSION and WM\_ENDSESSION messages).
* Reduce delays in the shutdown path of services and applications by minimizing CPU, disk, and network activity in response to shutdown notifications.
* Avoid delays in processing the suspend notification (WM\_POWERBROADCAST message).
* Respond quickly to resume events and minimize post-resume CPU, disk, and network usage.
* Reduce application resource consumption post-boot.
* Do not start applications from the **RunOnce** key on every boot.

# Feedback

Please provide feedback on any technical rationale, implementation details, case studies, additional tools, and other information that you want from Microsoft to help you address on/off transition performance issues.

To provide any comments or feedback, send e‑mail to:
 onofffb@microsoft.com

# Resources

This section has links to important documents and tools that you can use to verify and measure the impact of the recommendations in this paper. These tools enable system manufacturers, and application, service and driver developers to achieve superior on/off transition performance across all systems.

## Performance Analysis Tools

#### Windows Performance Toolkit (WPT)

WPT, the official toolkit for Windows performance analysis, is built and used by the Windows development group within Microsoft and is publicly available. You can use it to analyze all on/off transitions in great detail and determine the cause of many other performance issues such as long service, application, or driver delays, high CPU and disk usage, and shortened battery life.

To download the latest version of WPT, see the following:

Windows Performance Analysis

http://msdn.microsoft.com/en-us/performance/default.aspx

More WPT resources are available on the following sites:

Windows Performance Toolkit documentation on MSDN

http://msdn.microsoft.com/en-us/library/cc305187.aspx

Windows Performance Analysis forum

<http://social.msdn.microsoft.com/Forums/en-US/wptk_v4/threads/>

Event Tracing for Windows on MSDN

http://msdn.microsoft.com/en-us/library/aa363787.aspx

#### Additional Performance Tools and Information

On/Off Transition Performance Analysis of Windows Vista

<http://www.microsoft.com/whdc/system/sysperf/On-Off_Transition.mspx>

Diskpart Command-Line Options

<http://technet.microsoft.com/en-us/library/cc766465.aspx>

Measuring Performance in Windows Vista

<http://www.microsoft.com/whdc/system/sysperf/Vista_perf.mspx>

Measuring System Resume Performance on Windows Vista

<http://www.microsoft.com/whdc/system/sysperf/resumeperf.mspx>

On/Off Transition Trace Capture Tool

[http://msdn.microsoft.com/en-us/library/cc526988.aspx](http://msdn.microsoft.com/en-us/library/cc526988.aspx%20)

## Power Management Best Practices

Mobile Battery Life Solutions: A Guide for Mobile Platform Professionals

<http://www.microsoft.com/whdc/system/pnppwr/mobile_bat.mspx>

Application Power Management Best Practices for Windows Vista

<http://www.microsoft.com/whdc/system/pnppwr/powermgmt/PM_apps.mspx>

Optimizing Windows Vista Platforms for Energy Efficiency

<http://www.microsoft.com/whdc/system/pnppwr/powermgmt/Optimize_Power.mspx>

## Task and Service Best Practices

Developing Efficient Background Processes for Windows

<http://www.microsoft.com/whdc/system/pnppwr/powermgmt/BackgroundProcs.mspx>

Service Control Handler Function

http://msdn.microsoft.com/en-us/library/ms685149(VS.85).aspx

## Resources for Driver Developers

#### Driver Information on MSDN

Specifying Driver Load Order

http://msdn.microsoft.com/en-us/library/ms790185.aspx

Handling IRPs: What Every Driver Writer Needs to Know

http://www.microsoft.com/whdc/driver/kernel/IRPs.mspx

IRP\_MN\_START\_DEVICE

<http://www.microsoft.com/whdc/driver/kernel/IRPs.mspx>

IRP\_MN\_QUERY\_DEVICE\_RELATIONS

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

#### Windows Driver Foundation

Windows Driver Foundation on the WHDC Web site: <http://www.microsoft.com/whdc/driver/wdf/default.mspx>

#### Windows Driver Kit (WDK)

The WDK contains several tests and tools that are designed to improve the stability, performance, and functionality of drivers that are built to run on Windows. It includes many of the same tests and tools that the Windows development team uses.

The WDK is available to MSDN subscribers and as a download from the WHDC Web site.

WDK on the WHDC Web site

<http://www.microsoft.com/whdc/devtools/WDK/default.mspx>

WDK Documentation blog

<http://blogs.msdn.com/wdkdocs/default.aspx>

Monthly updates of the WDK documentation

<http://www.microsoft.com/whdc/DevTools/WDK/WDKdocs.mspx>

Driver Development Newsgroup

<http://msdn.microsoft.com/newsgroups/default.aspx?dg=microsoft.public.development.device.drivers&lang=en&cr=US>

Contact the WDK Team

WDKPMFT@microsoft.com (WDK Product Management team)

WDKQA@microsoft.com (WDK Test team)

DDKSurv1@microsoft.com (WDK Docs team)

# Reference

“PC Performance Focus Groups,” a commissioned study conducted by Forrester Consulting on behalf of Microsoft, January 2009.