Application Power Management Best Practices for Windows Vista

November 20, 2007

Abstract

The Windows Vista® operating system features significant changes to power management infrastructure, functionality, and user experiences. These changes impact all Microsoft® Windows® components, including third-party applications and services. Application developers must be aware of the changes to power management in Windows Vista, and design and test their applications accordingly.

This paper details the power management best practices for Windows Vista applications, including correctly handling sleep and resume transitions, responding to common system power events and designing for entertainment and media PC scenarios.

This information applies for the following operating systems:
 Windows Server® 2008
 Windows Vista

The current version of this paper is maintained on the Web at:
 <http://www.microsoft.com/whdc/system/pnppwr/powermgmt/PM_apps.mspx>

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

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.

© 2006-2007 Microsoft Corporation. All rights reserved.

Microsoft, MSDN, PowerPoint, Visual Studio, Windows, Windows Server, 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** |  |  |  |
| --- | --- | --- | --- | --- |
| December 1, 2006 | First publication |
| November 20, 2007 | Correct location for obtaining Power Event Monitoring tool. |

Contents

[Introduction 4](#_Toc183182615)

[Handling Sleep and Resume Transitions 5](#_Toc183182616)

[Sleep, Stand By, Hibernate, and Hybrid Sleep 5](#_Toc183182617)

[Sleep and Resume Events 5](#_Toc183182618)

[Changes to Sleep and Resume Events 6](#_Toc183182619)

[Responding to Sleep Events 6](#_Toc183182620)

[Responding to Resume Events 7](#_Toc183182621)

[Handling Network Connections and Files 8](#_Toc183182622)

[Preventing System Idle Timeouts 8](#_Toc183182623)

[Display Idle Timeout 9](#_Toc183182624)

[Sleep Idle Timeout 9](#_Toc183182625)

[Using SetThreadExecutionState 9](#_Toc183182626)

[SetThreadExecutionState Notes 11](#_Toc183182627)

[Designing for Extended Battery Life 11](#_Toc183182628)

[Best Practices for Extending Battery Life 12](#_Toc183182629)

[Battery Life Impact of High-Resolution Periodic Timers 12](#_Toc183182630)

[Responding to Common Power Events 14](#_Toc183182631)

[Registering for Power Events 14](#_Toc183182632)

[Responding to Power Events 15](#_Toc183182633)

[Unregistering for Power Events 16](#_Toc183182634)

[System Power Source (AC/DC) Notifications 17](#_Toc183182635)

[Power Plan Personality Notifications 18](#_Toc183182636)

[Monitor Power State Notifications 19](#_Toc183182637)

[Battery Capacity Remaining Notifications 19](#_Toc183182638)

[Background Task Notifications 20](#_Toc183182639)

[Designing for Entertainment and Media PC Scenarios 21](#_Toc183182640)

[Responding to Away Mode Notifications 21](#_Toc183182641)

[Requesting Away Mode 22](#_Toc183182642)

[Using Power Management APIs from Managed Code 23](#_Toc183182643)

[Responding to Sleep and Resume Events 23](#_Toc183182644)

[Responding to Common Power Events 24](#_Toc183182645)

[Additional Managed Code Samples and Information 26](#_Toc183182646)

[Designing for Earlier Versions of Windows 26](#_Toc183182647)

[Responding to Power Events within a Windows Service 27](#_Toc183182648)

[Testing Applications for Power Management 28](#_Toc183182649)

[Introduction to the PwrTest Utility 28](#_Toc183182650)

[Using PwrTest to Automate Sleep and Resume Transitions 28](#_Toc183182651)

[Using PwrTest to Observe SetThreadExecutionState Calls 29](#_Toc183182652)

[Introduction to the Power Event Monitoring Tool 29](#_Toc183182653)

[Call to Action 30](#_Toc183182654)

[References 31](#_Toc183182655)

# Introduction

The Windows Vista operating system features significant changes to power management infrastructure, functionality, and user experiences. These changes are designed to enable the following key goals for Windows Vista power management:

* **Energy-saving features that are enabled by default.** Windows Vista helps extend mobile PC battery life and reduce desktop and workstation power consumption by enabling energy-saving features by default.
* **Deterministic and reliable sleep transitions.** Sleep and resume transitions in Windows Vista are responsive and deterministic. Applications, services, and device drivers cannot block in-progress sleep transitions, which gives users additional confidence when they close the lid of a mobile PC or power off a desktop system.
* **System-wide responsiveness to user power policy.** Windows Vista users can easily change the overall energy-savings or performance behavior of their system.
* **Media and entertainment PC scenarios.** Windows Vista introduces Away Mode, a power management feature that enables the PC to look and feel off while it remains on to record television content or service network requests for media sharing.

Realizing each of these power management goals requires changes to the Windows® operating system as well as hosted applications and services.

This paper describes the power management changes for Windows Vista and demonstrates power management best practices for applications, including:

* **Handling sleep and resume transitions.** Applications must be able to correctly handle sleep and resume transitions. In Windows Vista, applications can no longer block in-progress sleep transitions.
* **Temporarily preventing system idle timeouts**. Applications can temporarily prevent the system sleep and display idle timeouts to ensure that background tasks such as device synchronization, hard disk defragmentation, and media recording complete successfully.
* **Responding to common power events.** Applications must be able to run on mobile platforms and respond to common system power events, such as power plan personality and system power source (AC/DC) changes, to help extend system battery life.
* **Enabling media and entertainment PC scenarios.** Applications might be required to respond to Away Mode transitions and to pause local media playback to help ensure that the PC looks and sounds off while it is in Away Mode.

The Windows ecosystem supports many application types and designs. The power management best practices that this paper demonstrates might not address all applications or scenarios. Application developers should use this paper and additional online resources from Microsoft to ensure that their applications correctly handle power management scenarios.

# Handling Sleep and Resume Transitions

All PCs running Windows Vista will experience sleep and resume transitions. Mobile PCs will continue to use the sleep state as a way to extend working battery life. However, desktop PCs running Windows Vista will have a sleep idle timer that is enabled by default so that, after a period of inactivity, the system automatically enters the sleep state.

Sleep transitions in Windows Vista are improved to help ensure responsiveness and deterministic behavior. These improvements prevent applications from blocking an in-progress sleep transition. Developers must design and test applications so that they can handle these changes to sleep and resume transitions in Windows Vista.

## Sleep, Stand By, Hibernate, and Hybrid Sleep

Windows uses several terms to identify low-power sleep states. Windows Vista supports the following low-power sleep states:

* **Sleep** (suspend-to-RAM). System context is saved to main memory, and the memory is kept in a low-power (self-refresh) mode while the system is asleep.
* **Hibernate** (suspend-to-disk). System context is saved on the system hard disk drive.
* **Hybrid sleep** (a combination of sleep and hibernate). System context is saved both in memory and on the hard disk drive.

Applications receive the same set of sleep and resume events for each low-power sleep state. They are not required to handle sleep or resume differently for each low-power sleep state and should not attempt to do so.

## Sleep and Resume Events

The WM\_POWERBROADCAST message notifies Windows applications of a system sleep or resume transition. Applications receive this message when a power management event occurs, including a sleep or resume transition, a power setting change, or a battery status change.

The WM\_POWERBROADCAST message includes one of the following event types that indicate the exact power management event that has occurred on the system:

* **PBT\_APMSUSPEND**
This event signals that the system is suspending. It is delivered each time the system suspends.
* **PBT\_APMRESUMEAUTOMATIC**
This event signals that the system is resuming. It is delivered on every resume and does not indicate whether a user is present at the system.
* **PBT\_APMRESUMESUSPEND**This event signals that the system has resumed and a user is present at the system. It is delivered only if a user presses a button (such as the power or sleep button) or generates input through a mouse or keyboard to indicate his or her presence at the system.

## Changes to Sleep and Resume Events

Windows Vista includes two primary changes to sleep and resume events:

* The PBT\_APMQUERYSUSPEND event is no longer delivered to applications. In earlier versions of Windows, the PBT\_APMQUERYSUSPEND event allowed applications to block the sleep transition from completing by returning BROADCAST\_QUERY\_DENY.

Some developers incorrectly use this message to avoid designing and testing for sleep and resume transitions. For example, an application might return BROADCAST\_QUERY\_DENY and then display user interface (UI) that asks for input. However, if the system is a mobile PC with the lid closed, prompting for user input is ineffective.

Some applications return BROADCAST\_QUERY\_DENY to help protect access to a remote network resource such as an open file. However, these same applications must correctly handle transient network conditions, which are prevalent in wireless networks.

* Applications must process the PBT\_APMSUSPEND event within 2 seconds. This change is designed to help increase the performance and responsiveness of suspend transitions. If an application has not returned from the PBT\_APMSUSPEND event after 2 seconds, any remaining processing completes after the system resumes from the sleep state.

To ensure that applications correctly handle sleep and resume messages, developers should follow the best practices in the following sections of this paper.

## Responding to Sleep Events

Windows delivers the PBT\_APMSUSPEND event to applications when the system enters the sleep state. Applications have 2 seconds to finish processing the PBT\_APMSUSPEND event before Windows continues with the sleep transition.

Application developers should understand the process by which Windows transitions the system to a sleep state:

1. A user or the system sleep idle timer initiates the sleep transition.

2. The Windows kernel power manager notifies applications, services, and device drivers with the appropriate suspend events, including PBT\_APMSUSPEND.

3. Windows places the system hardware in a sleep state and saves system context in memory, on disk, or both in memory and on disk.

4. The system resumes from the sleep state later due to the user powering on the system, a timer expiring, or a wake-on-LAN event.

5. Device drivers, services, and applications are notified that the system has resumed with the appropriate events.

User-mode threads, including those threads of applications and services, continue to execute during the entire sleep and resume transition.

The implication of this design is that many applications do not need to do anything specific to correctly handle sleep transitions. The following are examples of applications that are unaffected by sleep and resume events and that might not be required to process these events:

* A calculator application that relies solely on user input for activity
* Entertainment applications such as a simple game
* A photo application that displays pictures in a sideshow format
* A Web browser that displays a Web page

However, some applications might be required to respond to the PBT\_APMSUSPEND event within 2 seconds when the system is entering the sleep state. Typically, applications do very little work in response to the PBT\_APMSUSPEND message, such as updating the application state to indicate that the system has entered sleep so that a corresponding action can be taken when the system resumes.

The following types of applications might respond to the PBT\_APMSUSPEND event:

* Peer-to-peer or network applications that track user presence might transmit a message that indicates the user is absent.
* Optical media creation applications might attempt to stop the in-progress media creation activity. The application should attempt to automatically restart the media creation operation when the system resumes or prompt the user that the creation was interrupted.
* A print or scan application might tell the printer or scanner device to stop the current operation and update its state to automatically retry the operation when the system resumes.
* An application that requires the system to wake automatically in the future from a timer might program a waitable timer by using the SetWaitableTimer function.

## Responding to Resume Events

The PBT\_APMRESUMEAUTOMATIC and PBT\_APMRESUMESUSPEND events notify applications that the system has resumed. Applications have up to 20 seconds to respond to these resume events.

The PBT\_APMRESUMEAUTOMATIC event is delivered to applications on every resume, but the PBT\_APMRESUMESUSPEND event is delivered on resume only when a user is present at the system. Windows uses notifications from the platform hardware and user input at the keyboard or mouse to determine that a user is present at the system. Applications receive both the PBT\_APMRESUMEAUTOMATIC and PBT\_APMRESUMESUSPEND event when the system resumes from sleep and a user is present.

Applications typically use the resume events to automatically start or restart tasks, reconnect to network resources, and update the information that they are presenting to the user.

When handling resume events in applications, developers should ensure that:

* Applications automatically reconnect to network resources in the background without interrupting the user.
* A task that was interrupted due to the system sleeping (such as an optical media creation) restarts automatically, if possible.
* Applications prompt the user for action if a task that was interrupted due to a sleep state cannot be automatically restarted.
* Applications appropriately manage their resource usage immediately after a resume-from-sleep transition. Multiple applications can respond to resume events at the same time, and the user might be attempting to quickly access calendar or other data. Applications that perform system scans should wait approximately 1 minute or longer after receiving the PBT\_APMRESUMESUSPEND message before starting a resource-intensive scan.
* Applications that track user presence on the network update the user presence information when the resume event is received. These applications should use the PBT\_APMRESUMESUSPEND event to determine whether the user is present at the system.

## Handling Network Connections and Files

A common issue with sleep and resume transitions is the availability of network connections and remote files when the system resumes from sleep. Due to the prevalence of mobile PCs and wireless networks, it is likely that remote network resources will not be available when the system resumes from sleep.

Generally, application developers should design their applications to handle transient network conditions such as those that are possible with a wireless network. Sleep and resume transitions are no different in this respect—a network resource might be unavailable when the system resumes from the sleep state. Developers should design and test the application to handle this condition in the same way that the application handles an unavailable network connection.

Application developers should follow the following guidance when designing for transient network conditions:

* Maintain awareness of connectivity status and respond reasonably and transparently to transitions in network connectivity.
* If a network resource becomes disconnected, defer data transfers until later and provide appropriate user feedback.
* When the user establishes a new network connection, resume pending transfers, if sufficient bandwidth exists, and transfer information that might be needed later.
* Cache data locally, so that working files are protected from network interruptions, and synchronize the cache when connectivity is present.

For more information on designing for network connections and files, see the Mobile PC Development Guide on MSDN® listed at the end of this paper.

# Preventing System Idle Timeouts

Windows automatically turns off the display and places the system in the sleep state after a period of inactivity. Although users can configure the duration of the period of inactivity, Windows Vista enables aggressive timeouts by default for both the display idle timeout and the system sleep idle timeout. These timeouts are enabled on desktop systems and mobile PCs to help reduce power consumption and to extend battery life.

To ensure correct operation, applications might have to temporarily prevent the system or display idle timeout. For example, presentation applications disable both the display and sleep idle timeouts while the user is making a presentation in full-screen mode.

## Display Idle Timeout

To reduce power consumption, Windows automatically turns off the display device after a period of user inactivity. To determine user activity, Windows tracks user input through attached keyboards and mouse devices. If the elapsed time since the last user input is larger than the display idle timeout that is specified in power policy, the display is turned off.

Generally, applications must prevent the display from being turned off when they are displaying information to the user for long periods of time but the user is not providing input, such as during a full-screen presentation or full-screen media visualization.

When preventing the display idle timeout, however, developers should remember that the display consumes a large amount of power. This is particularly important on mobile PCs where the LCD backlight consumes a large percentage of the overall system power budget.

## Sleep Idle Timeout

If a system has been idle longer than the sleep idle timeout in the power policy, Windows automatically places the system in the sleep state. Windows uses both user input and processor usage to determine if a system is idle and tracks this information over time.

Generally, applications must prevent the system from entering the sleep state while applications are performing a background task or undertaking an operation that the user expects the system to continue until the operation is complete. Applications might need to prevent the sleep idle timeout in the following scenarios:

* A media application that displays full-screen content or visualizations
* Background applications such as hard disk defragmentation or antivirus utilities while a system scan is in progress
* A media application that records television content
* An application or service that streams content to other devices on the network
* An application such as a Web browser while downloading large files from the internet or network
* Applications that synchronize content to another device or computer

When preventing the sleep idle timeout in their applications, developers must exercise caution because the user expects the system to idle to sleep if it is enabled in power policy. Therefore, applications should prevent the sleep idle timeout only when absolutely necessary.

## Using SetThreadExecutionState

Windows applications can request that the display idle timeout and the sleep idle timeout be prevented by using the SetThreadExecutionState function. SetThreadExecutionState takes a single DWORD parameter that indicates which resources (such as display or system) are requested to remain available.

EXECUTION\_STATE SetThreadExecutionState(EXECUTION\_STATE esFlags)

Return Value

EXECUTION\_STATE

A typedef for a DWORD that is treated as a set of binary flags that indicate the current execution state of the current thread. If the function succeeds, the previous thread execution state is returned. Otherwise, the return value is NULL. Typically, applications do not need to reference the return value.

Parameters

EXECUTION\_STATE esFlags

Defines the flags that should be set in the execution state. Possible input values include the following:

| **EXECUTION\_STATE flags** | **Meaning** |
| --- | --- |
| ES\_SYSTEM\_REQUIRED | Requests system availability (sleep idle timeout is prevented). |
| ES\_DISPLAY\_REQUIRED | Requests display availability (display idle timeout is prevented). |
| ES\_AWAY\_MODE\_REQUIRED | Requests Away Mode to be enabled. |
| ES\_CONTINUOUS | Requests that the other EXECUTION\_STATE flags set remain in effect until SetThreadExecutionState is called again with the ES\_CONTINUOUS flag set and one of the other EXECUTION\_STATE flags cleared. |

The basic usage pattern for SetThreadExecutionState is to prevent the display or sleep idle timeout while completing a task:

1. Call SetThreadExecutionState with the ES\_CONTINUOUS flag set and one or more of the ES\_DISPLAY\_REQUIRED or ES\_SYSTEM\_REQUIRED flags set.

2. Process the task that requires the display or sleep idle timeout to be prevented.

3. After the task has completed, call SetThreadExecutionState with only the ES\_CONTINUOUS flag set. This indicates that display or system availability is no longer required.

The following example code demonstrates how to disable the display and sleep idle timeouts while rendering full-screen video or other media content:

//

// Playback of full-screen media content is beginning:

//

SetThreadExecutionState(ES\_CONTINUOUS | ES\_SYSTEM\_REQUIRED | ES\_DISPLAY\_REQUIRED);

//

// Wait until full-screen media content is paused, stopped,

// or the application is exited.

//

//

// Clear EXECUTION\_STATE flags so the system can return to

// normal display and sleep idle operation.

//

SetThreadExecutionState(ES\_CONTINUOUS);

Applications can not use SetThreadExecutionState to attempt to prevent the system from entering the sleep state in response to a user-initiated sleep action, such as pressing the system sleep button or closing the lid on a mobile PC. SetThreadExecutionState does not prevent user-initiated sleep transitions.

Additionally, applications should not depend on their processor usage alone to prevent the system from idling to sleep. If an application requires the system to stay awake, it should prevent the sleep idle timeout by using SetThreadExecutionState.

## SetThreadExecutionState Notes

When using SetThreadExecutionState, developers should remember the following:

* SetThreadExecutionState does not prevent the completion of user-initiated sleep transitions. A sleep transition that is initiated when a user presses the system power or sleep buttons completes successfully, and an application cannot blocked the transition by using SetThreadExecutionState with the ES\_SYSTEM\_REQUIRED flag.
* A call to SetThreadExecutionState with the ES\_DISPLAY\_REQUIRED flag causes the display to turn on. Users might be frustrated if the display turns on for no apparent reason. Additionally, if the system is currently in Away Mode, calling SetThreadExecutionState with the ES\_DISPLAY\_REQUIRED flag causes the system to exit Away Mode, which includes turning on the display and unmuting audio playback.
* Calling SetThreadExecutionState without the ES\_CONTINUOUS flag resets the idle timers. If SetThreadExecutionState is called with the ES\_DISPLAY\_REQUIRED or ES\_SYSTEM\_REQUIRED flag but without the ES\_CONTINUOUS flag, the display idle countdown or sleep idle countdown timer resets. For example, if the display idle timeout is set to 20 minutes and 10 minutes have elapsed since the last user input, a call to SetThreadExecutionState with only the ES\_DISPLAY\_REQUIRED flag resets the idle countdown and the display turns off 20 minutes later, assuming no additional user input.
* Any call to SetThreadExecutionState with the ES\_AWAYMODE\_REQUIRED flag also requires the ES\_CONTINUOUS flag.
* The Windows kernel power manager tracks execution state flags on a per-thread basis. Applications or services using threadpools must ensure that corresponding calls to **SetThreadExecutionState** are made on the same thread and that execution state is cleared when returning a thread to the threadpool.
* Applications must clean up their thread execution state before the application is closed or exits. Although the Windows kernel power manager cleans up after applications terminate unexpectedly with outstanding thread execution state references, applications that use SetThreadExecutionState should clear their outstanding execution state references by calling SetThreadExecutionState with only the ES\_CONTINUOUS flag set.

# Designing for Extended Battery Life

Applications that are designed for Windows should expect to run on many form factors, including mobile PCs, desktop PCs, servers, Tablet PCs, and ultra-mobile PCs (UMPCs). Many of these form factors are designed to run on battery power, and users expect extended battery life from their systems. Applications and hardware and devices play a key role in enabling extended battery life.

## Best Practices for Extending Battery Life

For extended battery life on Windows platforms, application developers should follow these best practices when designing their applications or services:

* **Invest in performance optimizations.** Every performance optimization is a battery life optimization. Reductions in required resources, such as using less processor time, allow system hardware to become idle and enter low-power modes.
* **Adjust to user power policy**. Windows Vista makes it easy for the user to choose the overall power-savings or performance behavior of the system. Applications should respond to changes in power policy and reduce their resource usage or increase their performance accordingly.
* **Reduce resource usage when the system is on battery power.** Applications should reduce their resource usage (such as background update frequency) when the system is on battery power.
* **Do not render to the display when it is off.** The system display might be off for power savings. Applications should not perform unnecessary graphics rendering when the display is turned off because this wastes system resources and power.
* **Avoid polling and spinning in tight loops.** Heavy processor usage reduces the effectiveness of processor power management technologies such as processor idle states and processor performance states.
* **Do not use high-resolution periodic timers.** Using high-resolution (<10‑millisecond) periodic timers reduces the efficiency of processor power management technologies.
* **Allow the system to idle the display and idle to sleep**. Follow the best practices in this paper to only disable the display and sleep idle timeouts when absolutely necessary.
* **Respond to common power management events.** Application developers should use the information in the rest of this paper to register for and respond to common power management events, such as system power source transitions and monitor on and off notifications.

Application developers should also test their applications on different hardware platforms to ensure that they do not impact system battery life.

## Battery Life Impact of High-Resolution Periodic Timers

Developers must be aware that the use of high-resolution periodic timers may increase overall system power consumption and reduce system battery life. Applications might knowingly use high-resolution periodic timers through the Windows Multimedia API or inadvertently through the use of external libraries and application development frameworks. Windows Vista logs an event to the system event log when an application’s use of periodic timers causes the system timer interval to change.

The most common application use of high-resolution periodic timers is through the **timeBeginPeriod** Windows Multimedia API. **timeBeginPeriod** is used to specify a minimum timer resolution for an application, and many applications call **timeBeginPeriod** with a value of 1 to increase the timer resolution to the maximum of 1 millisecond. The timer resolution may be increased to support graphical animations, audio playback, or video playback.

Although timer resolution for the application increases to 1 millisecond. The impact is that the Windows kernel must increase the resolution of the system timer to at least the resolution that any application requests. Therefore, if only one application requests a timer resolution of 1 millisecond, the system timer decreases its interval (also called the “system timer tick”) to at least 1 millisecond.

Modern processors and chipsets, particularly in mobile platforms, use the idle time between system timer intervals to reduce system power consumption. Various processor and chipset components are placed into low-power idle states between each timer interval. However, these low-power idle states are often ineffective at lowering system power consumption when the system timer interval is less than 10 milliseconds.

When the system timer interval is decreased to less than 10 milliseconds, including when an application calls **timeBeginPeriod** with a resolution of 1 ms, the low-power idle states are ineffective at reducing system power consumption and system battery life suffers. System battery life may be reduced as much as 25 percent, depending on the hardware platform.

In addition to explicitly using the Windows Multimedia APIs to increase timer resolution, application developers should validate that any external libraries or application development frameworks do not unexpectedly change the system timer interval on the application’s behalf.

To help application developers and users to determine if an application has changed the system timer interval, Windows Vista automatically generates a system event when the system timer interval changes, logging the process name and the requested interval. This event is written to the kernel power diagnostic log. The event has event ID 63:

The application or service *[path-to-process-image]* is attempting to update the system timer resolution to a value of *[timer-interval-in-100-ns-units]*.

For example, a test application named **test** generates the following event when it calls the **timeBeginPeriod** API with a value of 1 millisecond:

The application or service \Device\harddiskVolume1\test.exe is attempting to update the system timer resolution to a value of 1000.

To enable the kernel power diagnostic event log

1. Open the Windows Event Viewer.

2. On the View menu, enable Show Analytic and Debug Logs.

3. In the left tree view, navigate to Applications and Services Logs > Microsoft > Windows > Kernel-Power.

4. Right-click Diagnostic and select Enable Log.

5. Restart the system.

After the system is restarted, various events will be viewable in the kernel powerdiagnostic event log. The system timer interval change events have event ID 63.

Application developers should follow these best practices when using high-resolution periodic timers:

* Understand the system power consumption impact of high-resolution periodic timers. A single application can cause the system timer interval to change and reduce system battery life as much as 25 percent.
* Validate that your application does not unexpectedly change the system timer interval by searching for event ID 63 in the Microsoft Windows Kernel Power diagnostic event log.
* If your application must use a high-resolution periodic timer, enable only the periodic timer while the required functionality is active. For example, if the high-resolution periodic timer is required for animation, disable the periodic timer when the animation is complete.
* If your application must use a high-resolution periodic timer, consider disabling use of the periodic timer and associated functionality when a Power Saver power plan is active on the system or the system is on battery power.

# Responding to Common Power Events

Windows Vista enables applications to easily register for notification of common power events such as system power source (AC/DC) transitions and user power policy changes. Applications should use these events to dynamically change their resource usage to help extend system battery life or increase their performance according to user power policy.

## Registering for Power Events

Applications can register for power management events by using RegisterPowerSettingNotification.

HANDLE RegisterPowerSettingNotification(
 LPCGUID PowerSettingGuid,
 DWORD Flags)

Return Value

HANDLE hRecipient

Returns a handle to this event registration so that it can be unregistered for by calling UnregisterPowerSettingNotification.

Parameters

LPCGUID PowerSettingGuid

A pointer to a globally unique identifier (GUID) that indicates the power management setting or event for which to register. Possible values are defined in winnt.h.

DWORD Flags

Indicates whether the registrant is a windowed application or a service.

| **Flag value** | **Meaning** |
| --- | --- |
| DEVICE\_NOTIFY\_WINDOW\_HANDLE | Registrant is a windowed application. |
| DEVICE\_NOTIFY\_SERVICE\_HANDLE | Registrant is a service. |

The PowerSettingGuid parameter to RegisterPowerSettingNotification indicates a specific power management event. This allows applications to register for only the events that they require.

In the following code sample, RegisterPowerSettingNotiication is used to register for system power source notifications that indicate whether the system is on AC or DC (battery) power.

//

// Registration handle for AC/DC power transitions. Tracking this

// handle allows for deregistration upon application termination.

//

HPOWERNOTIFY ACDCNotificationHandle;

//

// Register for system power source change notifications.

//

ACDCNotificationHandle = RegisterPowerSettingNotification(

 GUID\_ACDC\_POWER\_SOURCE,

 DEVICE\_NOTIFY\_WINDOW\_HANDLE

 );

Power event notifications are delivered to the application through the WM\_POWERBROADCAST message. A PBT\_POWERSETTINGCHANGE event is delivered for the power event notification. The lparam of the event is a POWERSETTING\_BROADCAST structure that contains information identifying the specific power event and additional information about the event.

## Responding to Power Events

Applications receive a PBT\_POWERSETTINGCHANGE event when the registered power event notification occurs. The lparam of this event should be interpreted as a POWERBROADCAST\_SETTING structure, which contains a GUID that indicates which power event notification is delivered and includes additional data about the event notification, such as the new system power source state (AC/DC) or the display on-or-off state.

typedef struct {
 GUID PowerSetting;
 DWORD DataLength;
 UCHAR Data[1];
} POWERBROADCAST\_SETTING, \*PPOWERBROADCAST\_SETTING;

Fields

PowerSetting

Indicates the power event or setting of this notification. This GUID matches the PowerSettingGuid parameter in RegisterPowerSettingNotification and is defined in winnt.h.

DataLength

The length of the binary data in the Data parameter.

Data

A byte buffer that contains extra information about the event. The buffer length is equal to the DataLength parameter.

Headers

Declared in winuser.h.

The following sample code demonstrates how an application can receive the PBT\_POWERSETTINGCHANGE event and interpret the POWERSETTING\_BROADCAST structure to determine the specific power management event and update a global value for the event.

//

// Track if we're on AC or DC power. We will update this

// global when the power event notification occurs. We can

// then refer to this state anywhere in the application at

// any time.

//

DWORD PowerSource;

LRESULT

CALLBACK

UserWindProc(

 HWND hWnd,

 UINT uMsg,

 WPARAM wParam,

 LPARAM lParam)

{

 POWERBROADCAST\_SETTING \*PowerEventInformation;

 switch(wParam) {

 //

 // Handle power management messages

 //

 case WM\_POWERBROADCAST:

 switch(LOWORD(wParam)) {

 //

 // Handle power event notifications

 //

 case PBT\_POWERSETTINGCHANGE:

 //

 // Inspect the power event information

 //

 PowerEventInfomration = (POWERBROADCAST\_SETTING \*)lParam;

 //

 // This is the AC/DC transition that we registered

 // for. Derive the power source for later processing.

 //

 if (IsEqualGUID(PowerSetting->PowerSetting,

 GUID\_ACDC\_POWER\_SOURCE) &&

 (PowerSetting->DataLength == sizeof(DWORD))) {

 //

 // Update our global

 //

 PowerSource = (DWORD\*)PowerSetting->Data;

 }

 break;

 }

 break;

 }

}

## Unregistering for Power Events

Applications must unregister for power event notifications before the application exits by using UnRegisterPowerSettingNotification.

BOOL UnregisterPowerSettingNotification(
 HPOWERNOTIFY Handle)

Return Value

Returns TRUE if UnregisterPowerSettingNotification succeeded or FALSE if it did not succeed.

Parameters

HPOWERNOTIFY Handle

A handle to the registration to be closed. This handle is provided as the return value in the initial call to RegisterPowerSettingNotification.

In the following code sample, UnRegisterPowerSettingNotification is used to unregister for the system power source (AC/DC) event in the previous examples.

case WM\_DESTROY:

 //

 // Unregister for our power event notifications before

 // the window handle becomes invalid.

 //

 if (ACDCNotificationHandle != NULL) {

 UnregisterPowerSettingNotification(ACDCNotificationHandle);

 }

 break;

## System Power Source (AC/DC) Notifications

Applications can register for a power event notification of changes in the system power source. This notification is useful for applications to know when the system is on battery (DC) power and scale back their resource usage accordingly.

For example, applications might respond in the following ways after receiving the system power source notification:

* When on battery power, disable or delay resource-intensive operations such as system file scans.
* When on AC power, reenable resource-intensive operations such as real-time indexing.
* Automatically enable or disable high-resolution visualizations depending on current system power source.
* Warn the user to attach AC power before undertaking a power-sensitive task such as a firmware update or optical media creation.

Applications can also register for power policy change notifications (as explained in the following section) that provide an indication of overall system power-saving behavior, independent of current system power source.

To register for system power source change notifications, use the GUID\_ACDC\_POWER\_SOURCE event GUID.

GUID\_ACDC\_POWER\_SOURCE

Data Type

DWORD

Notification Occurs

Once immediately after registration.

Any time the system switches between AC (plugged in) or DC (on battery) power.

Possible Data Values

| **Value** | **Defines** | **Meaning** |
| --- | --- | --- |
| 0x0 | System is on AC power. | System is plugged into utility power. |
| 0x1 | System is on DC power. | System is running on battery power. |

## Power Plan Personality Notifications

Power policy in Windows Vista is organized into power plans, which contain preferences for each power setting on the system. Changing power plans allows the user to choose overall system power-saving or performance behavior, without having to change each individual power setting.

Windows Vista power plans have a personality attribute. The following personality attribute indicates the plan’s overall power-saving or performance behavior:

* **High Performance** delivers maximum performance and responsiveness at the expense of energy-savings.
* **Balanced** delivers a balance of performance and energy savings. It dynamically changes performance or energy-savings based on current workload on capable hardware and software.
* **Power Saver** delivers maximum energy-savings at the expense of system performance and responsiveness.

Although more than three power plans can be installed on a system, each power plan has one of these three power plan personalities.

Applications can register for power plan personality change notifications and use the event to direct power-saving or performance behaviors, similar to using system power-source notifications. The advantage to using power plan personality change notifications instead of system power source notifications is that the power plan personality indicates the user’s preference for overall system power management behavior regardless of current power source.

For example, the user might require High Performance behavior when editing video media, even though the system is currently battery powered. Similarly, the user might require Power Saver behavior, even on a desktop or server system, to conserve energy.

Applications might respond in the following ways after receiving the power plan personality notification:

* If the current power plan personality is High Performance, then enable resource-intensive operations such as real-time indexing and high-resolution visualizations.
* If the current power plan personality is Power Saver, then disable background updates, lower the fidelity of visualizations or audio, and postpone other unnecessary work.
* If the current power plan personality is Balanced, then choose an intermediate state for background and other work that moderates resource usage.

Windows Vista uses the Balanced power plan by default and applications must correctly deliver a balance of energy-savings and performance when the Balanced plan is active.

Optionally, applications can also register for the system power-source notification (AC/DC) and use it when the current power plan personality is Balanced to enable or disable binary power-saving and performance behaviors. For example, if the application supports enabling or disabling automatic spell-checking, the spell-checking might be enabled when the power plan personality is Balanced and the system power source is AC. Similarly, automatic spell-checking might be disabled when the power plan personality is Balanced and the system power source is DC.

To register for power plan personality change notifications, use the GUID\_POWERSCHEME\_PERSONALITY event GUID.

GUID\_POWERSCHEME\_PERSONALITY

Data Type

GUID

Notification Occurs

Once immediately after registration.

Any time the active power plan personality changes. This can occur when the user changes power plans by using the battery meter in the taskbar.

Possible Data Values

| **Value** | **Meaning** |
| --- | --- |
| GUID\_MAX\_POWER\_SAVINGS | Current power plan personality is Power Saver. Applications should reduce their resource usage as much as possible. |
| GUID\_MIN\_POWER\_SAVINGS | Current power plan personality is High Performance. Applications should deliver maximum performance and responsiveness. |
| GUID\_TYPICAL\_POWER\_SAVINGS | Current power plan personality is Balanced. Applications should deliver moderate responsiveness and resource usage behavior. If possible, resource usage should scale on demand to current application workload.  |

## Monitor Power State Notifications

Applications can register for a power event notification that indicates the current power state of the monitor or display device. The notification is delivered any time the monitor powers on or off due to the display idle timeout or entry or exit from Away Mode.

Applications should use the monitor power state change notification to disable graphics rendering or processing when the monitor is off. For example, applications should not be rendering visualizations when the display is off due to the display idle timeout because this consumes unnecessary power.

To register for monitor power state change notifications, use the GUID\_MONITOR\_POWER\_ON event GUID.

**GUID\_MONITOR\_POWER\_ON**

Data Type

DWORD

Notification Occurs

Once immediately after registration.

Any time the system turns on or turns off the monitor. This can be due to the display idle timeout or entry or exit from Away Mode.

Possible Data Values

| **Value** | **Meaning** |
| --- | --- |
| 0x0 | Monitor is currently off. |
| 0x1 | Monitor is currently on. |

## Battery Capacity Remaining Notifications

Applications can register for a power event notification that indicates the remaining battery capacity. The event notification includes the current remaining battery capacity as a percentage of maximum battery capacity.

Applications might use the battery capacity remaining event to implement a degrade behavior, where resource usage is continually decreased as remaining battery capacity decreases. This helps to further extend battery life, particularly when the battery has little remaining capacity.

Applications might also use the remaining battery capacity notification to warn the user to fully charge the system battery before undertaking a power-sensitive operation such as a firmware update or optical media creation.

To register for remaining battery capacity change notifications, use the GUID\_BATTERY\_CAPACITY\_REMAINING event GUID.

**GUID\_BATTERY\_CAPACITY\_REMAINING**

Data Type

DWORD

Notification Occurs

Once immediately after registration.

Any time the battery capacity changes either due to discharging or charging.

Possible Data Values

DWORD with a numeric value that indicates the current remaining battery capacity as a percentage of maximum battery capacity.

## Background Task Notifications

Applications can register for a power event notification that indicates that the system is currently active enough to support background tasks that would otherwise prevent the system from idling to a sleep state.

The background task notification is delivered to registered applications under the following conditions:

* The primary hard disk drive is on and spinning.
* The system is on AC power.
* The system is moderately idle. For example, current CPU usage is low enough that background tasks do not impact user responsiveness and foreground applications.
* The background task notification has not been delivered in the last minute.

Applications should use the background task notification as a trigger to perform background work items, including:

* Configuration updates and registry flushes
* Downloading schema and definition updates
* Indexing files and generating metadata

To register for background task notifications, use the GUID\_BACKGROUND\_TASK\_NOTIFICATION event GUID.

GUID\_BACKGROUND\_TASK\_NOTIFICATION

Notification Occurs

When the power manager determines that the system is appropriately idle for background tasks.

The Data field in the POWERSETTING\_BROADCAST structure for the GUID\_BACKGROUND\_TASK\_NOTIFICATION event is empty. The event alone is notification that the system is appropriately idle for background tasks.

# Designing for Entertainment and Media PC Scenarios

Many systems running Windows Vista will be used primarily for entertainment and media scenarios. These PCs might be located in the living room and used with Windows Media Center to watch and record television or used to transmit media content to other PCs and devices in the home.

These scenarios require the PC to stay on to record television content and respond to requests for media content from remote computers and devices. However, users might want the PC to look and feel off when they have finished interacting with the PC at the primary display device.

To meet this scenario, Windows Vista features Away Mode, a technology that allows the system to appear and sound off while it remains on to respond to requests for remote media transmission and record television content. Away Mode enables the PC to deliver an experience similar to that of a consumer electronics device, such as a set-top box.

Away Mode is not a new system power state, but is rather an option that an application can request upon entering the sleep state. When Away Mode is enabled, any sleep action (such as pressing the sleep button or closing the lid on a mobile PC) transitions the system to Away Mode.

Windows Vista applications, specifically media applications, should respond when the system transitions into Away Mode and pause local playback of media content. Pausing playback helps extend the experience that the system powered off when it entered Away Mode.

## Responding to Away Mode Notifications

Applications can register for a power event notification indicating that the system is transitioning into or out of Away Mode.

The system transitions into Away Mode when an application, such as Windows Media Center, requests Away Mode to be enabled and the user places the system in the sleep state. When the system enters Away Mode, local audio is muted and the display is turned off. This gives the illusion to the user that the system has turned off, although it is still on to transmit media content across the network or to record television content.

Media applications should register for the Away Mode transition notification and pause local playback of media content when the system enters Away Mode. By doing so, the application is in the same state when the system exits Away Mode. The user can then continue interacting with the system.

To register for the Away Mode notification, applications should use RegisterPowerSettingNotification and the GUID\_SYSTEM\_AWAYMODE power event GUID.

**GUID\_SYSTEM\_AWAYMODE**

Notification occurs

Any time the system enters or exits Away Mode.

Data Type

DWORD

Possible Data Values

| **Value** | **Meaning** |
| --- | --- |
| 0x0 | System is not in Away Mode. |
| 0x1 | System is in Away Mode. |

Application developers should be cautious about having applications perform tasks that require high processor usage while the system is in Away Mode. A system in Away Mode might be located in a living room, bedroom, or other location where thermal and noise constraints exist. Depending on hardware design, high processor usage might cause the system to increase heat output and increase the speed and noise output of processor and system case fans.

## Requesting Away Mode

Applications can request Away Mode to be enabled by using SetThreadExecutionState. Developers must be extremely cautious about enabling Away Mode in their applications because the user expects the system to transition to a sleep state unless the system is being used for entertainment or media scenarios.

Applications that require Away Mode can enable it by calling SetThreadExecutionState with both the ES\_CONTINUOUS and ES\_AWAYMODE\_REQUIRED flags. When Away Mode is no longer required, the application should call SetThreadExecutionState with the ES\_CONTINUOUS flag but without the ES\_AWAYMODE\_REQUIRED flag. The following code sample demonstrates these calls:

//

// Television recording is beginning, enable Away Mode and prevent

// the sleep idle timeout:

//

SetThreadExecutionState(ES\_CONTINUOUS | ES\_SYSTEM\_REQUIRED | ES\_AWAYMODE\_REQUIRED);

//

// Wait until recording has completed…

//

//

// Clear EXECUTION\_STATE flags so Away Mode is disabled and the system can idle to sleep normally:

//

SetThreadExecutionState(ES\_CONTINUOUS);

Application developers should note that the system might still idle to the sleep state after the system has entered Away Mode. If an application requires both entering Away Mode and preventing the system sleep idle timeout, then it should call SetThreadExecutionState with the ES\_AWAYMODE\_REQUIRED and the ES\_SYSTEM\_REQUIRED flags.

# Using Power Management APIs from Managed Code

Applications that are built with managed code must still be designed and tested for power management best practices. All of the APIs and power management events that are described in this paper are accessible from managed code.

## Responding to Sleep and Resume Events

Managed applications can obtain access to sleep and resume events by overriding the WndProc function in a Windows Forms application, as shown in the following code sample:

//

// Override the WndProc to access the WM\_POWERBROADCAST

// message and power events:

//

protected override void WndProc(ref Message m)

{

 switch (m.Msg)

 {

 //

 // If this is a WM\_POWERBROADCAST message, use

 // a OnPowerBroadcast helper function to handle

 // the sleep, resume, or power event.

 //

 case WM\_POWERBROADCAST:

 OnPowerBroadcast(ref m);

 break;

 }

 //

 // Call the base WndProc function

 //

 base.WndProc(ref m);

}

If the system transitions to or resumes from the sleep state, the overridden WndProc function is called with the WM\_POWERBROADCAST message and the appropriate sleep or resume event.

The sleep and resume events are the same as described earlier in this paper. The managed application can import the event definitions from winuser.h, as shown in the following sample code:

//

// Define the WM\_POWERBROADCAST message and events

// from winuser.h

//

const int WM\_POWERBROADCAST = 0x0218;

//

// Events

//

const int PBT\_APMSUSPEND = 0x0004;

const int PBT\_APMRESUMESUSPEND = 0x0007;

const int PBT\_APMPOWERSTATUSCHANGE = 0x000A;

const int PBT\_APMRESUMEAUTOMATIC = 0x0012;

const int PBT\_POWERSETTINGCHANGE = 0x8013;

Applications determine the specific power event by inspecting the WParam component of the Message argument to WndProc.

After determining the specific power event, the best practices for handling the event are the same for both managed and unmanaged code, as shown in the following code sample:

private void OnPowerBroadcast(ref Message m)

{

//

// Determine which power event was delivered

//

switch ((int)m.WParam)

{

 case PBT\_APMSUSPEND:

 //

 // Handle suspend

 //

 break;

 case PBT\_APMRESUMESUSPEND:

 //

 // Handle resume from suspend

 //

 break;

 // Handle additional events

 . . .

}

}

## Responding to Common Power Events

Managed applications should respond to common power events including power plan personality and system power source changes. After the application has obtained access to the WM\_POWERBROADCAST message by overriding the WndProc using the guidelines below, the application can respond to the other power events that were delivered with the PBT\_POWERSETTINGCHANGE event.

Managed applications must define and import the RegisterPowerSettingNotification and UnregisterPowerSettingNotification functions from user32.dll before they can be used to register for power management events, as shown in the following code sample:

[DllImport(@"User32", EntryPoint = "RegisterPowerSettingNotification",

 CallingConvention = CallingConvention.StdCall)]

private static extern IntPtr RegisterPowerSettingNotification(

 IntPtr hRecipient,

 ref Guid PowerSettingGuid,

 Int32 Flags

 );

[DllImport(@"User32", EntryPoint = "UnregisterPowerSettingNotification",

 CallingConvention = CallingConvention.StdCall)]

private static extern bool UnregisterPowerSettingNotification(

 IntPtr handle

 );

Managed applications can obtain access to the power event and the information it contains by accessing the POWERSETTING\_BROADCAST structure that was delivered with PBT\_POWERSETTINGCHANGE events.

To accomplish this, the managed application must define the layout of the POWERBROADCAST\_SETTING structure and marshal the LParam of the WM\_POWERBROADCAST message to determine which event is delivered and the data it contains, as shown in the following code sample:

//

// Define the POWERBROADCAS\_SETTING structure.

// Note that the Data member of the unmanaged structure is

// omitted and handled separately.

//

[StructLayout(LayoutKind.Sequential, Pack = 4)]

internal struct POWERBROADCAST\_SETTING

{

 public Guid PowerSetting;

 public Int32 DataLength;

}

//

// Handle PBT\_POWERSETTINGCHANGE events

//

private void PowerSettingChange(Message m)

{

 //

 // Marshall the POWERBROADCAST\_SETTING structure

 //

 POWERBROADCAST\_SETTING ps =

 (POWERBROADCAST\_SETTING)Marshal.PtrToStructure(

 m.LParam, typeof(POWERBROADCAST\_SETTING)

 );

 //

 // Handle the Data field of the POWERBROADCAST\_SETTING

// structure

 //

 IntPtr pData = (IntPtr)((int)m.LParam + Marshal.SizeOf(ps));

 //

 // Determine which power event was delivered and

 // handle it. We only handle DWORD-sized event data

 // in this example.

 //

 if (ps.DataLength == Marshal.SizeOf(typeof(Int32)))

 {

 Int32 iData = (Int32)Marshal.PtrToStructure(pData, typeof(Int32));

 if (ps.PowerSetting == GUID\_BATTERY\_PERCENTAGE\_REMAINING)

 {

 //

 // Handle battery capacity remaining events

 //

 }

 If (ps.PowerSetting == GUID\_ACDC\_POWER\_STATUS)

 {

 //

 // Handle system power source (AC/DC)

 // notifications

 //

 }

 // Handle additional events

 }

}

## Additional Managed Code Samples and Information

Application developers seeking more information and code samples for interacting with power management APIs from managed code should consult the Windows Vista Mobile PC Hands-on Labs on MSDN listed at the end of this paper.

# Designing for Earlier Versions of Windows

Although some of the power management APIs that were described in this paper are available only in Windows Vista, many applications will be designed for both Windows Vista and earlier versions of Windows. These applications must also follow the best practices for power management and appropriately respond to sleep and resume events and to system power source changes.

Application developers should follow these best practices when designing for power management in both Windows Vista and earlier versions of Windows:

* **Use the same code for sleep and resume events.** Applications should have the same code for sleep and resume events.The primary difference in sleep and resume events between Windows Vista and earlier versions of Windows is that the PBT\_APMQUERYSUSPEND event is not delivered in Windows Vista. Applications should not block in-progress sleep transitions on any version of Windows, and therefore need not respond to the PBT\_APMQUERYSUSPEND event on earlier versions of Windows.
* **Use the same code for preventing system idle timeouts.** Applications can use the same code for preventing system idle timeouts because the SetThreadExecutionState function is available in all earlier versions of Windows beginning with Windows 2000.
* Use system power source (AC/DC) status to enable power-saving or performance behaviors. The preferred method to enable power-saving or performance behaviors is to respond to power plan personality change events in Windows Vista. However, these events are not available on earlier versions of Windows. Instead, applications should use system power source changes (AC/DC) to enable power-saving or performance behaviors. When the system is on battery (DC) power, power-saving behaviors such as disabling or delaying background activity should be enabled. When the system is on AC power, performance behaviors such as real-time indexing should be enabled.
* **Use the GetProcAddress function to determine if RegisterPowerSettingNotification is available on the current operating system.** GetProcAddress can be used to determine if common power events such as GUID\_ACDC\_POWER\_STATUS and GUID\_POWERSCHEME\_PERSONALITY are available on the current operating system. These events should be used to direct power-saving or performance behaviors if available; otherwise, use PBT\_APMPOWERSTATUSCHANGE and the GetSystemPowerStatus function to determine if the system is on AC or DC (battery) power.

# Responding to Power Events within a Windows Service

Many Windows applications are built as a Windows service or have a service component. Service-based applications, like any other application, must also follow the power management best practices in this paper.

Services can receive the WM\_POWERBROADCAST message and all of the power management events by first registering to receive power management messages. Services register to receive power management events and provide a handler callback function that is executed when a power management event occurs.

Services register for power management events by using the RegisterServiceCtrlHandlerEx function:

SERVICE\_STATUS\_HANDLE WINAPI RegisterServiceCtrlHandlerEx(
 LPCTSTR *lpServiceName*,
 LPHANDLER\_FUNCTION\_EX *lpHandlerProc*,
 LPVOID *lpContext*);

RegisterServiceCtrlHandlerEx requires a pointer to a service-provided callback function, HandlerEx, which is notified when power management events occur and behaves similarly to the UserWndProc function for windowed applications:

DWORD WINAPI HandlerEx(
 DWORD dwControl,
 DWORD *dwEventType*,
 LPVOID *lpEventData*,
 LPVOID *lpContext*);

Services must indicate to the service control manager that they can receive power events. To accomplish this, services must call SetServiceStatus and update the dwControlsAccepted field of the SERVICE\_STATUS structure with the SERVICE\_ACCEPT\_POWEREVENT flag:

typedef struct \_SERVICE\_STATUS {
 DWORD dwServiceType;
 DWORD dwCurrentState;
 DWORD dwControlsAccepted;
 DWORD dwWin32ExitCode;
 DWORD dwServiceSpecificExitCode;
 DWORD dwCheckPoint;
 DWORD dwWaitHint;
} SERVICE\_STATUS, \*LPSERVICE\_STATUS;

When the service-supplied HandlerEx function is notified of power events, the dwControl parameter is equal to SERVICE\_CONTROL\_POWEREVENT. The dwEventType and LpEventData parameters are similar to the wParam and lParam parameters of the UserWindProc function.

The dwEventType parameter will be one of the following:

* PBT\_APMRESUMESUSPEND
* PBT\_APMSUSPEND
* PBT\_APMRESUMEAUTOMATIC
* PBT\_POWERSETTINGCHANGE

If the dwEventType parameter is equal to PBT\_POWERSETTINGCHANGE, then the LpEventData parameter is a pointer to a POWERSETTING\_BROADCAST structure that contains the power event GUID and associated data. Services can interpret the POWERSETTING\_BROADCAST structure and determine the power event GUID and associated data by using the same methods as those used by windowed applications.

# Testing Applications for Power Management

Developers should also test applications to ensure that they follow the best practices for power management. Microsoft-provided tools and utilities can be used to determine the impact of an application on power management and to understand the delivery of power management events and notifications.

## Introduction to the PwrTest Utility

PwrTest is a command-line power management test utility that is included with the Windows Driver Kit (WDK). PwrTest supports several power management test scenarios, including:

* Automating sleep and resume transitions
* Logging calls to SetThreadExecutionState
* Observing the impact on processor power management
* Inspecting battery capacity and the remaining percentage

PwrTest is designed to be easy to integrate into existing test suites and custom test utilities. It is completely command-line driven and records the results of each of the power management scenarios to an XML-based log file with a consistent format.

## Using PwrTest to Automate Sleep and Resume Transitions

Application developers can simulate user-initiated sleep and resume transitions automatically by using the PwrTest utility. PwrTest uses public APIs to transition the system to a sleep state and automatically wake the system a short time later by using a timer.

To use PwrTest to automate sleep and resume transitions, developers can view the usage for the PwrTest SLEEP scenario, as shown in the following sample code:

C:\pwrtest>pwrtest /sleep /?

The SLEEP scenario supports the following options:

 /c:n n indicates number of cycles (1 is default).

 /d:n n indicates delay time (in seconds; 90 is default).

 /p:n n indicates sleep time (in seconds; 60 is default).

 /h:y indicates hybrid sleep should be enabled (default is system policy).

 /h:n indicates hybrid sleep should be disabled (default is system policy).

 /t:y indicates transitions should be critical (default is system policy).

 /t:n indicates transitions should not be critical (default is system policy).

 /s:all indicates cycling through all supported power states in order.

 /s:rnd indicates cycling through all supported power states randomly.

 /s:1 indicates target state is always S1.

 /s:3 indicates target state is always S3 (default).

 /s:4 indicates target state is always S4.

 /unattend indicates not to change system execution state after wakeup.

SLEEP scenario examples

 pwrtest /sleep /c:4 /s:all

 pwrtest /sleep /c:4 /p:120 /d:150 /s:all

Developers should use the following examples to transition the system to sleep and resume automatically:

* Transition to sleep once and automatically wake 90 seconds later:

C:\pwrtest>pwrtest /sleep /s:3 /c:1

* Transition to sleep three times, pausing 2 minutes between each sleep and sleeping for 3 minutes for each transition:

C:\pwrtest>pwrtest /sleep /s:3 /c:3 /d:120 /p:180

* Transition to hibernate once and automatically wake one minute later:

c:\pwrtest>pwrtest /sleep /s:4 /c:1 /p:60

Note:  Not all systems support wake from hibernate on timer.

## Using PwrTest to Observe SetThreadExecutionState Calls

Application developers can observe calls to SetThreadExecutionState by using the PwrTest utility. PwrTest displays the process name and execution state of any SetThreadExecutionState call that was executed while PwrTest is run with the ES scenario.

To observe SetThreadExecutionState calls, run PwrTest with the ES scenario, as shown in the following example:

C:\pwrtest>pwrtest /es

Waiting for SetThreadExecutionState Events. Press 'q' to quit...

------------------------------------

Execution State Changed by a process

TimeStamp: 09/04/2006 13:12:26

Process: \Device\HarddiskVolume1\PROGRA~1\MICROS~2\Office12\POWERPNT.EXE

RawState: 0x80000003

Continuous: 1 Display: 1 System: 1 AwayMode: 0

------------------------------------

------------------------------------

Execution State Changed by a process

TimeStamp: 09/04/2006 13:12:39

Process: \Device\HarddiskVolume1\PROGRA~1\MICROS~2\Office12\POWERPNT.EXE

RawState: 0x80000000

Continuous: 1 Display: 0 System: 0 AwayMode: 0

------------------------------------

In the preceding example, PwrTest recorded two calls to SetThreadExecutionState by Microsoft PowerPoint® (powerpnt.exe).

PowerPoint correctly calls SetThreadExecutionState before and after full-screen presentation. PowerPoint requests both the system and display idle timeout to be prevented during full-screen presentation and correctly clears these execution state flags when full-screen presentation mode is exited.

## Introduction to the Power Event Monitoring Tool

Application developers can use the Power Event Monitoring Tool and the information in this paper to observe the delivery of power events.

The Power Event Monitoring Tool is provided as a sample that can be compiled and built by developers. The tool sample is provided on the WHDC Web site and requires the Windows Vista Software Development Kit (SDK).

The Power Event Monitoring Tool registers for common power events, including system power source and power plan personality change notifications, and displays a message to the screen when the notifications occur, as shown in the following figure:



The Power Event Monitoring Tool also displays current battery capacity information and allows application developers to interact with SetThreadExecutionState and execution state flags by manipulating check boxes at the bottom of the application.

Finally, this tool provides excellent reference code for application developers by demonstrating use of the various power management APIs that are referenced in this paper.

# Call to Action

The Microsoft Windows Vista operating system features significant changes to power management infrastructure, functionality, and user experiences. These changes are designed to enable key goals for Windows power management, including reliable and deterministic sleep transitions and enabling energy-saving features by default. Realizing each of these power management goals requires changes to both the Windows operating system and to hosted applications and services. Application developers should use the following best practices to ensure that their application performs correctly in Windows Vista:

* Design and test applications for power management.
* Expect sleep transitions on all platforms and for every application. Handle sleep transitions gracefully and quickly.
* Prevent system idle timeouts only when absolutely necessary. Prevent the system sleep timeout while completing a critical task. Reenable the system sleep timeout when the task has completed.
* Help extend system battery life by registering for power plan personality and system power source change notifications. Use these notifications to enable power-saving or performance-enhancing behaviors.
* Register for Away Mode notifications and pause local playback of media content when the system enters Away Mode.
* Follow the best practices in this paper even if the application is service-based or built with managed code.
* Use the PwrTest utility to automate sleep and resume transitions and observe calls to SetThreadExecutionState.

# References

ACPI/Power Management – Architecture and Driver Support

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

#### Microsoft Developer Network (MSDN)

Windows Vista Mobile PC Hands-On Labs
<http://msdn2.microsoft.com/windowsvista/Aa904995>

Mobile PC Development Guide – Summary of Usability Guidelines
<http://msdn2.microsoft.com/en-us/library/ms695506.aspx>

Windows Vista Developer Center
<http://msdn2.microsoft.com/en-us/windowsvista/default.aspx>

Windows Software Development Kit (SDK) for Windows Vista

Installation Instructions for the Windows SDK
<http://msdn2.microsoft.com/en-us/library/ms717328.aspx>

#### Windows Driver Kit (WDK) and WHDC Web site

WDK and Developer Tools – Overview
<http://www.microsoft.com/whdc/DevTools/default.mspx>

WDK – How to obtain the current Windows Driver Kit
<http://www.microsoft.com/whdc/devtools/wdk/betaWDK.mspx>

Power Event Monitoring Tool
<http://www.microsoft.com/whdc/system/pnppwr/powermgmt/PM-apps_samp.mspx>
Requires the Windows Vista Software Development Kit (SDK)

PwrTest in the WDK
<http://msdn2.microsoft.com/en-us/library/aa906552.aspx>