Power Availability Requests

June 23, 2009

Abstract

By default, Windows®-based platforms enable device and system power management technologies to help improve energy efficiency and reduce power consumption. Two of the most effective power management features are display power management and automatic sleep. However, in some scenarios, applications or drivers must temporarily disable these power management technologies to perform tasks as the user expects. Applications can temporarily prevent display power management and automatic sleep by making power availability requests.

This paper describes when to use availability requests, the user-mode and kernel-mode functions that support such requests, and new options in the PowerCfg tool to manage availability requests on Windows 7 or Windows Server® 2008 R2.

This information applies to the following operating systems:

Windows Server 2008 R2
Windows 7

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

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

Document History

Date	Change
June 23, 2009	First publication

Contents

Introduction 3
Scenarios for Availability Requests 3
Temporarily Delay Display Power Management 4
Temporarily Delay Automatic Sleep 5
Enable Away Mode 5
New Functions for Availability Requests 5
Win32 Functions 6
Kernel-Mode Functions 10
Using the Win32 Power Availability Functions in Managed Environments 12
Developing Software for Windows 7 and Earlier Versions of Windows 13
Common Windows Availability Requests 14
Managing Availability Requests 14
Enumerating Outstanding Requests 14
Analyzing System Energy Efficiency 15
Overriding Availability Requests 16
Best Practices for Availability Requests 17
Resources 17

Introduction

Windows® platforms provide a wide variety of power management technologies to help extend mobile PC battery life and improve energy efficiency. Two of the most common and effective technologies are display power management and automatic sleep. Display power management—powering off or reducing brightness of the display after a period of user inactivity—is effective because the display consumes a large amount of the total system power budget. Similarly, automatic sleep after user inactivity is an effective power-saving feature because it can reduce the power consumption of an idle PC to only a few watts.

Beginning with Windows Vista®, both display power management and automatic sleep are enabled by default for all Windows client operating systems. However, some user scenarios conflict with these power-saving modes and require display power management and automatic sleep to be temporarily disabled. A good example is video playback. If the computer plays a DVD for several hours, the user does not interact with the system via keyboard or mouse. The display and sleep idle timeouts must be temporarily disabled so that the DVD movie can play without interruption.

By using power availability requests (or simply *availability requests*), applications, services, and drivers can temporarily disable power management features to accomplish user scenarios. Availability requests can prevent the display from turning off after inactivity, prevent the system from automatically sleeping, and enable Away Mode.

Developers of applications, services, and drivers should be aware of availability requests and the best practices for their use. Correct use of availability requests is critical because end users and IT administrators expect the computer to use display power management and automatic sleep as configured in Power Options in Control Panel.

Although Windows 7 and Windows Server® 2008 R2 provide innovations in availability request APIs and diagnostics, many earlier versions of Windows have supported availability requests. Developers might be familiar with the **SetThreadExecutionState** function, which allows user-mode Win32 applications to create availability requests.

This paper details the improvements in availability request APIs and diagnostics for Windows 7 and Windows Server 2008 R2. It covers new Win32 application programming interface (API) functions and kernel-mode device driver interface (DDI) functions that can create availability requests that have rich textual diagnostic information. The paper also describes enhancements to the PowerCfg utility that enable administrators to enumerate outstanding availability requests to help determine why display power management or automatic sleep is not functioning as expected.

Scenarios for Availability Requests

Applications use availability requests for the following reasons:

- To temporarily disable display power management

- To temporarily disable automatic sleep after a period of inactivity
- To enable Away Mode

You should use availability requests only when essential to complete a user scenario. Users expect automatic sleep or display power management to function as enabled in Power Options. If an availability request has disabled one of these power management features, the user might not be aware which application or driver has temporarily disabled the functionality.

Applications should create the availability request when the user scenario begins and release it promptly when the user scenario completes. You must test and validate the use of availability requests to ensure that power management features are disabled only when necessary. Validation should ensure that the availability request is created when the user scenario begins and is promptly released when the user scenario is completed.

Temporarily Delay Display Power Management

To reduce power consumption, Windows automatically turns off the display after a period of user inactivity. Windows 7 also automatically reduces the brightness of the integrated display in a mobile PC after a period of user inactivity.

To determine user activity, Windows tracks user input through the attached human input devices. If the elapsed time since the last user input is greater than the display idle timeout that the current power policy specifies, Windows turns off the display. Similarly, if the elapsed time since the last user input is greater than the display dim timeout specified in power policy, Windows dims the display.

Applications can 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. Common scenarios include:

- Delivering a full-screen presentation
- Viewing video content
- Displaying a picture slide show

However, when preventing display power management, developers should remember that the display consumes a large amount of power. Display power consumption is particularly important on mobile PCs where the LCD backlight consumes a large percentage of the overall system power budget.

Display power management is important for correct operation of media playback applications, including Web-based applications. Many applications correctly prevent display power management when rendering video content in full-screen modes. However, few applications correctly prevent the display from dimming or turning off when rendering video content that is embedded into a Web page.

The ideal design for the Web-integrated video scenario is to make an availability request when the user initiates video playback and to clear the request when video playback stops. Use an availability request only when rendering video content that

the user has initiated. Availability requests are not appropriate for preventing display power management during display of an animated advertisement.

Temporarily Delay Automatic Sleep

To help reduce system power consumption, Windows automatically places the computer in the sleep state after a period of user inactivity. Sleep consumes much less power than in the working state, allowing for large potential power savings when the system is idle. Typically, the computer automatically enters sleep after 30 or 60 minutes of user inactivity. The user can increase or decrease the sleep idle timeout in Power Options.

Applications typically might need to prevent automatic sleep when they are consuming system resources to deliver a user scenario, but the user is not currently interacting with the system. Common scenarios include:

- Delivering a full-screen presentation
- Viewing video content
- Saving files to an optical disk
- Downloading files from the Internet or another computer on the network
- Recording television content

However, when preventing automatic sleep, developers should keep in mind the large potential power savings for automatic sleep. The user or system administrator expects the system to automatically enter sleep after the period of inactivity specified in Power Options. Developers must ensure that their applications are delivering critical user scenarios when creating availability requests to prevent automatic sleep.

Enable Away Mode

You can also use availability requests to enable Away Mode on Windows Vista and later versions of Windows. Away Mode is designed for entertainment and media PCs. When Away Mode is enabled, if the user places the system in sleep, the computer remains on (ACPI S0 state) and the display is turned off and audio is muted. Away Mode provides the illusion that the system turned off, while it remains powered on to deliver media content to networked PCs or Windows Media Center Extender devices.

Enable Away Mode only when essential for media and connected entertainment scenarios. End users expect the system to enter sleep when they select **Sleep** from the Windows **Start** menu or press the sleep button on the computer or a connected remote control. Similarly, use of automatic sleep and Wake-on-LAN technologies is often preferable to Away Mode because sleep provides greater energy efficiency than keeping the computer in the On state with Away Mode.

New Functions for Availability Requests

In previous versions of Windows, applications used the **SetThreadExecutionState** function to make an availability request to prevent display power management or prevent automatic sleep. **SetThreadExecutionState** requires that requests be set and cleared on the same Win32 thread. Additionally, **SetThreadExecutionState** callers

cannot specify the reason for the availability request, which could aid administrators in diagnosing why PCs are not using display power management and automatic sleep as expected. These challenges made its use cumbersome and diagnosis difficult.

Windows 7 and Windows Server 2008 R2 include new user-mode and kernel-mode functions that are easier to use. The new functions use an object and handle model for each availability request. These functions also improve system-wide diagnostics for availability requests by allowing you to provide a textual string denoting the reason for the request.

Win32 Functions

The user-mode Win32 API includes three new functions for creating and managing availability requests. These functions are defined in Winbase.h and listed in Table 1.

Table 1. Power Availability Request Functions in Win32 API

Function name	Description
PowerCreateRequest	Creates a power context object and returns a handle to it.
PowerClearRequest	Removes an outstanding availability request on a particular request context object
PowerSetRequest	Activates a power availability request and indicates the type of request.

You can use the new **PowerCreateRequest**, **PowerSetRequest**, and **PowerClearRequest** functions to create availability requests in Windows 7 and Windows Server 2008 R2. These functions improve on the **SetThreadExecutionState** function by removing the per-thread restriction and removing the inconsistency when **SetThreadExecutionState** is used with the ES_CONTINUOUS flag.

The **PowerCreateRequest**, **PowerSetRequest**, and **PowerClearRequest** functions use a handle and object–based model. When an application requires an availability request, it calls **PowerCreateRequest** to create an object that contains context about the availability request. The context object includes caller-provided diagnostic information about the request, including a textual string indicating the reason for the request.

Winbase.h defines the context object and the **PowerCreateRequest** function as follows:

```
//
// Power Request Context Object
//
typedef struct _REASON_CONTEXT {
    ULONG Version;
    DWORD Flags;
    union {
        struct {
            HMODULE LocalizedReasonModule;
            ULONG LocalizedReasonId;
            ULONG ReasonStringCount;
            LPWSTR *ReasonStrings;

        } Detailed;

        LPWSTR SimpleReasonString;
```

```
    } Reason;
} REASON_CONTEXT, *PREASON_CONTEXT;

typedef REASON_CONTEXT POWER_REQUEST_CONTEXT, *PPOWER_REQUEST_CONTEXT,
*LPPOWER_REQUEST_CONTEXT;

//
// Version and Reason Type Constants
//
#define POWER_REQUEST_CONTEXT_VERSION          0
#define POWER_REQUEST_CONTEXT_SIMPLE_STRING    0x00000001
#define POWER_REQUEST_CONTEXT_DETAILED_STRING  0x00000002

//
// PowerCreateRequest API
//
WINBASEAPI
HANDLE
WINAPI
PowerCreateRequest (
    __in PREASON_CONTEXT Context
    );
```

The diagnostic string can be simple or detailed. In the simple case, the caller provides a string value such as “Application xxx Is Downloading Files” in the **SimpleReasonString** member of the REASON_CONTEXT object. For example:

```
//
// Simple, non-localized availablity request diagnostic string
//
POWER_REQUEST_CONTEXT SimplePowerRequest;
SimplePowerRequest.Version = POWER_REQUEST_CONTEXT_VERSION;
SimplePowerRequest.Flags = POWER_REQUEST_CONTEXT_SIMPLE_STRING;
SimplePowerRequest.SimpleReasonString = L”Sample Reason String.”;
```

Detailed strings enable localization of the diagnostic string and string substitution in the **Detailed** member of the object. If you provide a module and resource ID in the **Detailed** member, Windows automatically displays the string in the language of the user enumerating outstanding availability requests, assuming localized modules have been provided. The following shows how to supply a diagnostic string that Windows localizes:

```
//
// Localized availablity request diagnostic string
// without substitution
//
POWER_REQUEST_CONTEXT LocalizedPowerRequest;
LocalizedPowerRequest.Version = POWER_REQUEST_CONTEXT_VERSION;
LocalizedPowerRequest.Flags = POWER_REQUEST_CONTEXT_DETAILED_STRING;
LocalizedPowerRequest.LocalizedReasonModule = HandleToResourceDLL;
LocalizedPowerRequest.LocalizedReasonId = -1040
LocalizedPowerRequest.ReasonStringCount = 0;
LocalizedPowerRequest.ReasonStrings = NULL;
```

The detailed string also enables substitution at run time to provide rich diagnostic information. For example, by using substitution you can provide a string such as "Application is downloading abc.zip from ftp://www.microsoft.com". Within Windows, detailed string substitution is used to display the path of any open network files that cause automatic sleep to be temporarily disabled. The following shows how an application can use substitution to provide additional detail:

```
//
// Localized availablity request diagnostic string
// with string substituion.  Assume string resource -1041 contains
// L"My Application is recording television on channel %1".
//
POWER_REQUEST_CONTEXT LocalizedPowerRequest;
LocalizedPowerRequest.Version = POWER_REQUEST_CONTEXT_VERSION;
LocalizedPowerRequest.Flags = POWER_REQUEST_CONTEXT_DETAILED_STRING;
LocalizedPowerRequest.LocalizedReasonModule = HandleToResourceDLL;
LocalizedPowerRequest.LocalizedReasonId = -1041
LocalizedPowerRequest.ReasonStringCount = 1;
LocalizedPowerRequest.ReasonStrings = "35";
```

After the application has created a context object for an availability request, the application can set or activate the request by using the **PowerSetRequest** function.

The following shows the definition of this function from Winbase.h:

```
//
// Power Request Type Enumeration
//
typedef enum _POWER_REQUEST_TYPE {
    PowerRequestDisplayRequired,
    PowerRequestSystemRequired,
    PowerRequestAwayModeRequired
} POWER_REQUEST_TYPE, *PPOWER_REQUEST_TYPE;

//
// PowerSetRequest API
//
WINBASEAPI
BOOL
WINAPI
PowerSetRequest (
    __in HANDLE PowerRequest,
    __in POWER_REQUEST_TYPE RequestType
    );
```

Table 2 describes the availability request types that an application can set by using **PowerSetRequest**.

Table 2. Availability Request Types

Power request type	Description
PowerRequestDisplayRequired	Prevents display power management, including automatic display dimming.
PowerRequestSystemRequired	Prevents automatic sleep.
PowerRequestAwayModeRequired	Enables Away Mode.

An application can use **PowerSetRequest** to set multiple and different types of availability requests on the same request context object. For example, an application

can create a single context object to prevent both display power management and automatic sleep while displaying full-screen video content. To make the availability requests, the application calls **PowerSetRequest** twice: once with the **PowerRequestDisplayRequired** type and once with the **PowerRequestSystemRequired** type. Windows displays the same diagnostic context information for both requests.

An application can also call **PowerSetRequest** multiple times for the same type of request. For example, consider an application that downloads multiple files from the network. The application can create a single request context object and call **PowerSetRequest** each time it starts to download a file. Windows displays the same diagnostic context information for each request.

The **PowerClearRequest** function removes or clears an outstanding availability request on a given request context object. Winbase.h defines this function as follows:

```
WINBASEAPI
BOOL
WINAPI
PowerClearRequest (
    __in HANDLE PowerRequest,
    __in POWER_REQUEST_TYPE RequestType
    );
```

For each request that an application initiates by using **PowerSetRequest**, the application must call **PowerClearRequest** to clear the request when it is no longer required. As with **PowerSetRequest**, an application can call **PowerClearRequest** more than once on a single request context object to clear more than one type of request. The application must call **PowerClearRequest** exactly once for each previous call to **PowerSetRequest** on a given request context object.

When the application has cleared all availability requests and no longer requires the request context object, it must close the handle to the request object by calling the **CloseHandle** function.

The **PowerCreateRequest, PowerSetRequest** and **PowerClearRequest** functions are safe for use in Win32 services in the Service Control Manager (SCM) programming model. These functions are also safe for use within a thread pool.

The following code sample demonstrates how to use the **PowerCreateRequest**, **PowerSetRequest**, and **PowerClearRequest** functions to create an availability request for an application that downloads files from the network. The application creates a system availability request to prevent automatic sleep while the file is being downloaded:

```
//
// Create a system availablity request to keep the system from
// automatically sleeping while downloading a file.
//
POWER_REQUEST_CONTEXT DownloadPowerRequestContext;
HANDLE DownloadPowerRequest;

//
// Set up the diagnostic string
```

```
//
DownloadPowerRequestContext.Version = POWER_REQUEST_CONTEXT_VERSION;
DownloadPowerRequestContext.Flags =
     POWER_REQUEST_CONTEXT_SIMPLE_STRING;
DownloadPowerReqeustContext.Reason.SimpleReasonString =
     L”My application is downloading files.”

//
// Create the request, get a handle
//
DownloadPowerRequest =
     PowerCreateRequest(&DownloadPowerRequestContext);

//
// Set a system request to prevent automatic sleep
//
PowerSetRequest(
       DownloadPowerRequest,
       PowerRequestSystemRequired
       );

//
// Download the file...
//

//
// Clear the request
//
PowerClearRequest(DownloadPowerRequest);
```

Kernel-Mode Functions

In Windows 7 and Windows Server 2008 R2, kernel-mode code can call the **PoCreatePowerRequest, PoSetPowerRequest**, **PoClearPowerRequest,** and **PoDeletePowerRequest** DDI functions to create and manage availability requests. These functions are defined in Wdm.h and must be called at IRQL<DISPATCH_LEVEL. Table 3 lists these functions and the user-mode equivalents.

Table 3. Kernel-Mode Availability Request Functions and Equivalent Win32 Functions

Kernel-mode function	User-mode function
PoCreatePowerRequest	**PowerCreateRequest**
PoSetPowerRequest	**PowerSetRequest**
PoClearPowerRequest	**PowerClearRequest**
PoDeletePowerRequest	No equivalent

The kernel-mode functions work in almost exactly the same manner as the corresponding user-mode Win32 functions. The primary difference is that the Windows kernel creates the request object and allocates the memory that backs it. Therefore, the driver must call into the kernel to delete the request object when the availability request is no longer required. The driver must call **PoDeletePowerRequest** when it no longer requires use of the power request object so that Windows can free the memory. The other functions work in exactly the same way as their Win32 counterparts. For additional details about how to use these functions, see the previous section.

The following shows the function and structure prototypes for **PoCreatePowerRequest**, **PoSetPowerRequest**, **PoClearPowerRequest** and **PoDeletePowerRequest**, as defined in Wdm.h:

```
//
// PoCreatePowerRequest function
//
NTKERNELAPI
NTSTATUS
PoCreatePowerRequest (
    __deref_out PVOID *PowerRequest,
    __in PDEVICE_OBJECT DeviceObject,
    __in PCOUNTED_REASON_CONTEXT Context
    );

//
// PoSetPowerRequest function
//
NTKERNELAPI
NTSTATUS
PoSetPowerRequest (
    __inout PVOID PowerRequest,
    __in POWER_REQUEST_TYPE Type
    );

//
// PoClearPowerRequest function
//
NTKERNELAPI
NTSTATUS
PoClearPowerRequest (
    __inout PVOID PowerRequest,
    __in POWER_REQUEST_TYPE Type
    );

//
// PoDeletePowerRequest function
//
NTKERNELAPI
VOID
PoDeletePowerRequest (
    __inout PVOID PowerRequest
    );
```

The kernel-mode context structure is essentially similar to the user-mode structure, and the constants and enumerations are identical, as the following definitions show:

```
//
// Power Request Reason Context Structure
//
typedef struct _COUNTED_REASON_CONTEXT {
    ULONG Version;
    ULONG Flags;
    union {
        struct {
            UNICODE_STRING ResourceFileName;
            USHORT ResourceReasonId;
            ULONG StringCount;
            PUNICODE_STRING __field_ecount(StringCount) ReasonStrings;
        } DUMMYSTRUCTNAME;
```

```
        UNICODE_STRING SimpleString;
    } DUMMYUNIONNAME;
} COUNTED_REASON_CONTEXT, *PCOUNTED_REASON_CONTEXT;

//
// Power Request Constants
//
#define POWER_REQUEST_CONTEXT_VERSION          0
#define POWER_REQUEST_CONTEXT_SIMPLE_STRING    0x00000001
#define POWER_REQUEST_CONTEXT_DETAILED_STRING  0x00000002

//
// Power Request Enumerations
//
typedef enum _POWER_REQUEST_TYPE {
    PowerRequestDisplayRequired,
    PowerRequestSystemRequired,
    PowerRequestAwayModeRequired
} POWER_REQUEST_TYPE, *PPOWER_REQUEST_TYPE;
```

Using the Win32 Power Availability Functions in Managed Environments

You can easily use the **PowerCreateRequest**, **PowerSetRequest**, and **PowerClearRequest** functions in Windows 7 and Windows Server 2008 R2 in managed programming environments by using the Platform Invocation Services (PInvoke) capability. The following example C# definitions can be used directly in a managed application or service:

```
//
// Availability Request Functions
//
[DllImport("kernel32.dll")]
static extern IntPtr PowerCreateRequest(
    ref POWER_REQUEST_CONTEXT Context
    );

[DllImport("kernel32.dll")]
static extern bool PowerSetRequest(
    IntPtr PowerRequestHandle,
    PowerRequestType RequestType
    );

[DllImport("kernel32.dll")]
static extern bool PowerClearRequest(
    IntPtr PowerRequestHandle,
    PowerRequestType RequestType
    );

//
// Availablity Request Enumerations and Constants
//
enum PowerRequestType {
    PowerRequestDisplayRequired = 0,
    PowerRequestSystemRequired,
    PowerRequestAwayModeRequired,
    PowerRequestMaximum
}

const int POWER_REQUEST_CONTEXT_VERSION = 0;
const int POWER_REQUEST_CONTEXT_SIMPLE_STRING = 0x1;
```

```
const int POWER_REQUEST_CONTEXT_DETAILED_STRING = 0x2;

//
// Availablity Request Structures
//

//
// Note:
//
// Windows defines the POWER_REQUEST_CONTEXT structure with an
// internal union of SimpleReasonString and Detailed information.
// To avoid runtime interop issues, this version of
// POWER_REQUEST_CONTEXT only supports SimpleReasonString.
// To use the detailed information,
// define the PowerCreateRequest function with the first
// parameter of type POWER_REQUEST_CONTEXT_DETAILED.
//
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Unicode)]
public struct POWER_REQUEST_CONTEXT
{
     public UInt32 Version;
     public UInt32 Flags;
     [MarshalAs(UnmanagedType.LPWStr)] public string
          SimpleReasonString;
}

[StructLayout(LayoutKind.Sequential)]
public struct PowerRequestContextDetailedInformation
{
    public IntPtr LocalizedReasonModule;
    public UInt32 LocalizedReasonId;
    public UInt32 ReasonStringCount;
    [MarshalAs(UnmanagedType.LPWStr)] public string[] ReasonStrings;
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
public struct POWER_REQUEST_CONTEXT_DETAILED
{
    public UInt32 Version;
    public UInt32 Flags;
    public PowerRequestContextDetailedInformation DetailedInformation;
}
```

Developing Software for Windows 7 and Earlier Versions of Windows

If your application software must run on both Windows 7 and earlier versions of Windows, you should use the new power availability function **PowerCreateRequest** and associated functions whenever possible. In user mode, the best way to determine whether these functions are available on the current Windows instance is to call the **GetProcAddress** function to dynamically obtain a pointer to **PowerCreateRequest** and the associated functions. If **GetProcAddress** returns a pointer that is not equal to NULL, the requested function is available and the application should use it.

Kernel-mode components can use the **MmGetSystemRoutineAddress** function for the same purpose.

Common Windows Availability Requests

Table 4 lists Windows components that make availability requests based on user scenarios or policy configuration. Administrators can use this information to understand the most common availability requests on Windows platforms.

Table 4. Common Windows Availability Requests

Windows component	Request type	Description
Network File Sharing	SYSTEM	Created on both the client and server when a file that is open across the network is not backed by a client-side cache.
Audio Playback	SYSTEM	Created when any audio stream is active.
Media Sharing	SYSTEM	Created when remote devices are configured and are not compatible with Wake-on-LAN (WoL).
Windows Explorer	SYSTEM	Created during file copy operations.
Windows Print Spooler	SYSTEM	Created when print spooler is active.
Media Center / Extender	SYSTEM + AWAY MODE	Created when streaming content to remote extender devices. Away Mode request is based on user policy selection during extender setup.
Windows Media Player	SYSTEM + DISPLAY	Created when playing back full-screen video content.

Managing Availability Requests

Administrators often focus on enabling PC power-saving features, including display power management and automatic sleep. When such features do not work as expected, it can be difficult to identify the application, service, or driver that is responsible. Windows includes tools that can help you resolve any problems and ensure that all managed PCs use display power management and automatic sleep efficiently.

In Windows 7 Windows Server 2008 R2, the PowerCfg utility includes new options that provide information about outstanding availability requests:

- **/requests**
- **/energy**
- **/requestsoverride**

If a component creates the availability request by using the new **PowerCreateRequest** and related functions, Windows reports additional diagnostic information as described in the following sections. Only the process name and service name are available if the request is created using the older **SetThreadExecutionState** function.

Enumerating Outstanding Requests

In Windows 7 and Windows Server 2008 R2, the PowerCfg utility supports the **/requests** option, which enumerates all outstanding availability requests. To use this option, you must run PowerCfg from an elevated command prompt.

In the following example, Windows Explorer has a single outstanding system availability request because of an ongoing file copy operation:

```
C:\>PowerCfg /REQUESTS
DISPLAY:
None.

SYSTEM:
[PROCESS] \Device\HarddiskVolume2\Windows\explorer.exe
A file copy operation is in progress.

AWAYMODE:
None.
```

As the example shows, PowerCfg displays all outstanding availability requests for each type of request: Display, System, and Away Mode. For each request, it shows in brackets whether the requestor is a process, service, or driver, and the path to the process executable file on disk, the name of the service, or the name of the driver.

If the requestor uses the new availability request function and provides a context string, PowerCfg shows the string. In the example, Windows Explorer (Explorer.exe) was updated to provide the string "A file copy operation is in progress." as the reason for the system availability request.

PowerCfg /requests enumerates availability requests that user-mode applications make by using the older **SetThreadExecutionState** function and displays the type of request and "kernel" as the requestor. The utility displays [LEGACYKERNELCALLER] for availability requests that a kernel-mode caller created by using the older **PoRegisterSystemState** function.

Analyzing System Energy Efficiency

In Windows 7 and Windows Server 2008 R2, the PowerCfg utility supports the **/energy** option, which analyzes overall system energy efficiency.

The **/energy** option generates a report that contains detailed results and includes information about any outstanding availability requests at the time of analysis. Such requests are logged as errors, because they prevent optimal system energy efficiency. For complete information about the **/energy** option, see "Using PowerCfg to Evaluate System Energy Efficiency," which is listed in "Resources" later in this paper.

Figure 1 shows a snippet from the PowerCfg **/energy** report that describes an outstanding availability request.

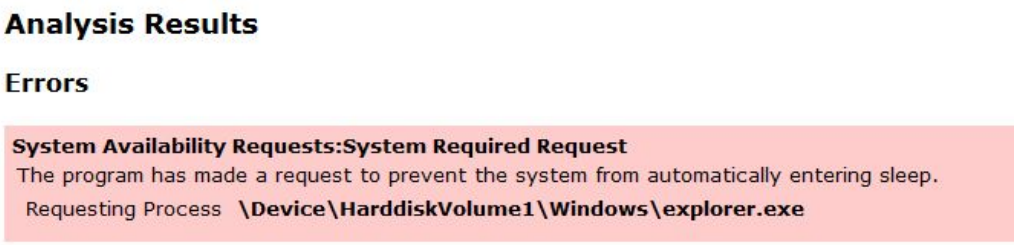

Figure 1 Sample PowerCfg /energy output

Overriding Availability Requests

To enable system administrators to work around software that makes availability requests incorrectly, the PowerCfg utility on Windows 7 and Windows Server 2008 R2 provides the **/requestsoverride** option. By using this option, you can override any availability request on the system. To use this option, you must run PowerCfg from an elevated command prompt.

The following shows the help message for this option:

```
Usage: POWERCFG -REQUESTSOVERRIDE <CALLER_TYPE> <NAME> <REQUEST>
       <CALLER_TYPE>    Specifies one of the following caller type:
                        PROCESS, SERVICE, DRIVER. This is obtained by
                        calling the POWERCFG -REQUESTS command.
       <NAME>           Specifies the caller name. This is the name
                        returned from calling POWERCFG -REQUESTS
                        command.
       <REQUEST>        Specifies one or more of the following Power
                        Request Types: Display, System, Awaymode.

Example:
     POWERCFG -REQUESTSOVERRIDE PROCESS wmplayer.exe Display System
```

You can use the **/requestsoverride** option to override any availability request on the system. For example, if an application incorrectly makes an availability request, but cannot be recompiled, you can use **/requestsoverride** to restore display power management or automatic sleep behaviors. To correct such a problem in an administrative environment, you can call PowerCfg from a script that is deployed to each system by using Windows Group Policy.

The following example shows how to override the system availability request that Windows Explorer makes when it copies a file. The first command in the example sets the override on the Explorer.exe process and the second command enumerates the outstanding overrides:

```
C:\>powercfg /requestsoverride PROCESS explorer.exe System

C:\>powercfg /requestsoverride
[SERVICE]

[PROCESS]
explorer.exe SYSTEM

[DRIVER]
```

To remove the power request override, use the **/requestoverride** option, but do not specify any type of override (System, Display, AwayMode) as in the following example:

```
C:\>powercfg /requestsoverride PROCESS explorer.exe

C:\>powercfg /requestsoverride
[SERVICE]

[PROCESS]

[DRIVER]
```

Best Practices for Availability Requests

Applications, services, and drivers should use availability requests to temporarily disable display power management, to temporarily disable automatic sleep, and to enable Away Mode for entertainment and media PC scenarios. The following best practices apply for the programmatic use of availability requests:

- Use **PowerCreateRequest** and related functions in user-mode code if possible. Use **GetProcAddress** to determine whether **PowerCreateRequest** is available on the current operating system. If the system does not support this function, use **SetThreadExecutionState** if required.
- Set the availability request using **PowerSetRequest** only when the availability request is required. Clear the request by calling **PowerClearRequest** as soon as the scenario is completed.
- Provide a localized, textual reason for the availability request when you call **PowerCreateRequest** to create the request context.
- Clean up all request objects and associated handles before process exit or service stop.
- Enable Away Mode only for entertainment and media PC scenarios.

Administrators can use the new options for the PowerCfg utility in Windows 7 and Windows Server 2008 R2 to identify problems with display power management and automatic sleep. The following best practices apply for administrative management of availability requests:

- Use PowerCfg with the **/requests** option to identify outstanding availability requests that prevent display power management or automatic sleep. Run PowerCfg from an elevated command prompt.
- Use PowerCfg with the **/requestsoverride** option to override availability requests. Override only the minimally required set of requests to achieve the desired system behavior. Run PowerCfg from an elevated command prompt.
- Distribute PowerCfg commands to each system in the enterprise by using a Group Policy script.

Resources

MSDN

Platform Invoke Tutorial
http://msdn.microsoft.com/en-us/library/aa288468.aspx

PowerCreateRequest
http://msdn.microsoft.com/en-us/library/dd405533(vs.85).aspx

Windows Driver Kit (WDK)

PoCreatePowerRequest
http://msdn.microsoft.com/en-us/library/dd568005.aspx

PoRegisterSystemState
http://msdn.microsoft.com/en-us/library/ms806573.aspx

PoSetSystemState
http://msdn.microsoft.com/en-us/library/ms806601.aspx

SetThreadExecutionState
http://msdn.microsoft.com/en-us/library/aa373208(VS.85).aspx

Windows Hardware Developer Central (WHDC) Web site

Application Power Management Best Practices for Windows Vista
http://www.microsoft.com/whdc/system/pnppwr/powermgmt/PM_apps.mspx

Using PowerCfg to Evaluate System Energy Efficiency
www.microsoft.com/whdc/system/pnppwr/powermgmt/PowerCfg.mspx