



The Art of Graceful Application Suspension

by Lynn Merrill

February 3, 2004

The information contained in this document is provided for informational purposes only and represents the current view of Intel Corporation Intel on the date of publication. Intel makes no commitment to update the information contained in this document, and Intel reserves the right to make changes at any time, without notice.

DISCLAIMER. THIS DOCUMENT AND ALL INFORMATION CONTAINED HEREIN IS PROVIDED AS IS. INTEL MAKES NO REPRESENTATIONS OF ANY KIND WITH RESPECT TO PRODUCTS REFERENCED HEREIN, WHETHER SUCH PRODUCTS ARE THOSE OF INTEL OR THIRD PARTIES. INTEL EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, IMPLIED OR EXPRESS, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, AND ANY WARRANTY ARISING OUT OF THE INFORMATION CONTAINED HEREIN, INCLUDING WITHOUT LIMITATION, ANY PRODUCTS, SPECIFICATIONS, OR OTHER MATERIALS REFERENCED HEREIN. INTEL DOES NOT WARRANT THAT THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN IS FREE FROM ERRORS, OR THAT ANY PRODUCTS OR OTHER TECHNOLOGY DEVELOPED IN CONFORMANCE WITH THIS DOCUMENT WILL PERFORM IN THE INTENDED MANNER, OR WILL BE FREE FROM INFRINGEMENT OF THIRD PARTY PROPRIETARY RIGHTS, AND INTEL DISCLAIMS ALL LIABILITY THEREFOR.

INTEL DOES NOT WARRANT THAT ANY PRODUCT REFERENCED HEREIN OR ANY PRODUCT OR TECHNOLOGY DEVELOPED IN RELIANCE UPON THIS DOCUMENT, IN WHOLE OR IN PART, WILL BE SUFFICIENT, ACCURATE, RELIABLE, COMPLETE, FREE FROM DEFECTS OR SAFE FOR ITS INTENDED PURPOSE, AND HEREBY DISCLAIMS ALL LIABILITIES THEREFOR. ANY PERSON MAKING, USING OR SELLING SUCH PRODUCT OR TECHNOLOGY DOES SO AT HIS OR HER OWN RISK.

Licenses may be required. Intel and others may have patents or pending patent applications, trademarks, copyrights or other intellectual proprietary rights covering subject matter contained or described in this document. No license, express, implied, by estoppels or otherwise, to any intellectual property rights of Intel or any other party is granted herein. It is your responsibility to seek licenses for such intellectual property rights from Intel and others where appropriate.

Limited License Grant. Intel hereby grants you a limited copyright license to copy this document for your use and internal distribution only. You may not distribute this document externally, in whole or in part, to any other person or entity.

LIMITED LIABILITY. IN NO EVENT SHALL INTEL HAVE ANY LIABILITY TO YOU OR TO ANY OTHER THIRD PARTY, FOR ANY LOST PROFITS, LOST DATA, LOSS OF USE OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OF THIS DOCUMENT OR RELIANCE UPON THE INFORMATION CONTAINED HEREIN, UNDER ANY CAUSE OF ACTION OR THEORY OF LIABILITY, AND IRRESPECTIVE OF WHETHER INTEL HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

Intel, the Intel logo, Pentium, Intel Xeon, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2005 Intel Corporation

Introduction

Does this sound familiar? You're working on an application with critical data ready for submission or publication, and suddenly you need to step away for an urgent matter that takes longer than expected. You return to your desk only to find that the computer has gone into a suspended state, or has even shut down completely. You might have lost some or all of your critical data, simply because the application did not "save" before being halted in its execution. Do you know anyone who hasn't had an experience like this? Application developers can help us avoid this scenario by paying attention to the messages and events that are sent by the operating system prior to the suspension or hibernation of the system.

This paper provides examples of what happens when a system prepares to suspend operations, suspends operations and resumes operations from a suspended state, within a Windows* environment. We address the purpose of the messages and events that are sent and describe their proper use and timing while offering suggestions for how to use them efficiently to avoid possible data loss. In addition, we explore what types of applications should be paying attention to these messages and events. Focusing on the messaging specific to the Windows XP* Professional operating systems, coding examples are provided to illustrate ways to prepare for and recover from the suspension and to suggest ways to either prevent suspension from occurring, or to work around the current limitations. A basic understanding of Windows messaging is assumed for this discussion.

Definitions

WM_POWERBROADCAST – WindowProc()	Message broadcast to the application through the function indicating that a power-management event has or is occurring.
PBT_APMQUERYSPEND –	Event that requests permission to suspend
PBT_APMQUERYSPENDFAILED –	Event that informs of a failed request to suspend
PBT_APMRESUMEAUTOMATIC –	Event received on automatic resume from suspended state
PBT_APMRESUMECRITICAL –	Event that signals resume from an unknown and volatile state
PBT_APMRESUMESPEND –	Event that signals a resume from a suspend
PBT_APMSPEND –	Event received just prior to system suspending

Motivation

Developing applications can be tricky business. With the advent of the mobile environment, that business became even trickier. Deciding which applications need to pay attention to the suspension of the computer, and which ones need not, is the first step. To begin, any application that would be effected by a change in operating environment, such as network, USB, or Firewire* connections, those that cannot be interrupted before they complete their operation or any application that displays information on the monitor without user interaction are all applications that should be suspension aware.

What happens if your application is making use of a file on the network when system suspension occurs? Upon resuming operation, the network connection is no longer there and unless your application is aware of that you will have problems. How about if you are burning a CD or DVD over Firewire or USB and the system suspends, what do you do? Well, unfortunately you have just created another nice coaster for your desk. Have you ever made a presentation to an important client and been side tracked by an extended comment or question-- during which time your system suspends and terminates the display? Embarrassed, you waste precious minutes while you resetting the machine and finding the appropriate location within your presentation. It's easy to see just how important it is to have your application paying attention

These and other examples illustrate the importance of developing applications to take advantage of system generated messages. Making your application power message aware also makes it more robust and resilient to the occasional occurrence of when a machine fails to recover from suspension and must be rebooted. By making a backup file, the data stored prior to suspension can be retrieved more easily, making recovery less painful.

System Suspension

The system suspension process is used primarily to save power on mobile devices or turn off a hard disk drive prior to moving a device. It is generally tied to settings that the user has defined for when their device is not used for a given period of time. If power settings are set appropriately the system suspension process can also be invoked by shutting the lid on a laptop. To demonstrate the message patterns that are associated with system suspension, we connected via Remote Control to the test system, invoked Spy++ on the test system to examine and capture the WM_POWERBROADCAST messages and events that are broadcast by the test system. In general, the following is the pattern of messages that are sent to and received from an application:

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMQUERYSPEND
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMSPEND
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

(Note: Lines that have an **S** at the beginning denote messages that are sent with SendMessage(). Those that have an **R** at the beginning denote the message return value from the application.)

The above pattern indicates that the system sent a WM_POWERBROADCAST message with a PBT_APMQUERYSPEND event to request permission of the application to suspend the system. This message and event are sent to all applications, threads, and processes that are currently running on the system. This is generally the first event leading to a suspension. Upon receiving this message, the application should make preparations for closing down before returning. These preparations are varied depending on the application, but could include things like flushing database tables to disk, saving files to disk, releasing memory or shared resources, closing network connections, or any other task that would make the application safe to shutdown. Generally the system allows up to 20 seconds for this message to be pulled from the application's message queue. Therefore, any shutdown operations that are necessary should be started as soon as this event is received. In this example the application returned TRUE indicating that suspension could continue.

The message sent just before the system suspends is WM_POWERBROADCAST with a PBT_SPMSPEND event. The return value of TRUE is returned by the application.

Denying Suspension Request

If the application cannot allow or does not want the system to suspend, there is a way to tell the system that the application denies the request. To do this, upon receiving the PBT_APMQUERYSPEND event, the application would return the value BROADCAST_QUERY_DENY. This tells the system that there is an application or process that cannot or will not allow the suspension. In the case where an application denies suspension, the message pattern would be as follows:

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMQUERYSPEND
```

```
R WM_POWERBROADCAST fSucceeded:True [HRESULT:424D5144]
```

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMQUERYSPENDFAILED
```

Note that in this example the value returned by the application in HRESULT is 424D5144, which is the value of BROADCAST_QUERY_DENY. The system then sends out the PBT_APMQUERYSPENDFAILED event, telling all applications that the suspension request has failed and that normal operations within the application can resume.

Another way of preventing the system from suspending is by using the function SetThreadExecutionState(). The role of this function is to inform the system that it is in use and, as our testing indicates, to prevent WM_POWERBROADCAST messages and events dealing with suspension from being sent by the system at all. The system will, however, continue to send events dealing with battery and power state changes so that those things can still be monitored. There are three flags associated with this function: ES_SYSTEM_REQUIRED, indicating that some operation is being performed; ES_DISPLAY_REQUIRED, indicating that the display is required by some operation;

and ES_CONTINUOUS, telling the system that the accompanying state should remain in effect until changed. As a recommendation, applications such as fax servers, answering machines, backup agents, and network management applications should use ES_SYSTEM_REQUIRED and multimedia applications, such as video players and presentation applications should use ES_DISPLAY_REQUIRED. If ES_CONTINUOUS is not used, the idle timer is simply reset; therefore, periodic calls to this function should be made to reset the timer. Note however, that this function does not prevent the user from suspending the system by closing the lid on a laptop. The only way to prevent the user from suspending the system is to deny the PBT_APMQUERYSUSPEND event as described above. In addition, this function does not always prevent a screen saver from executing. Other means of preventing this can be used, such as defining a time and using SystemParametersInfo() to query and reset the screen save time-out value.

Resuming Operations from a Suspended State

The resumption of operations from a suspended state is just as critical to an application's behavior as putting it into suspension, and care should be taken by application developers to handle the messages that are received during this process. Restoring memory, disk status, and database tables, are just some examples of elements within an application that need to be refreshed or reconfigured when resuming from a suspended state. Without these elements being in a healthy condition, normal application behavior would be hit and miss at best. As with suspension, the WM_POWERBROADCAST message is used to alert the application that a resume is either underway, or has completed. Generally, the following messages and events are sent by the operating system:

```
S WM_POWERBROADCAST (long)dwPowerEvent:0012
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMRESUMESUSPEND
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

The first event is 0x12, which corresponds to PBT_APMRESUMEAUTOMATIC. This event simply informs the application that an automatic resume is underway. In general, no response to this event is needed, however, there is one case, discussed below, where an application might want to pay attention to this event. The next event that is sent is PBT_APMRESUMESUSPEND. This event is sent only if a PBT_APMRESUMESUSPEND was sent during the suspension process. This indicates that the system has safely resumed operations and that the application can carry on. If while suspended the application releases memory, database tables, or other critical information, this is where the developer would want to reinitialize or reallocate those working elements of the application. As mentioned above placing the application back in a healthy condition is greatly desired.

Other resume events that can be received are PBT_APMQUERYSUSPENDFAILED and PBT_APMRESUMECRITICAL. As discussed above, the PBT_APMQUERYSUSPENDFAILED is received if some other process has denied the systems' request to suspend, signaling that normal operations have resumed. PBT_APMRESUMECRITICAL is sent when some or all application did not receive a PBT_APMRESUMESUSPEND event, such as you might see after a battery failure. Because of the critical nature of this message, resources and data previously available may no longer be present. Network

connections are particularly vulnerable; therefore applications should take action with respect to any files or data that reside on the network. In short, applications should restore their state to the best of their ability when receiving this event.

Hibernation

Hibernation saves the complete state of the machine when the system shuts down so that when power is reestablished the system is returned to the same state as when hibernation occurred. An interesting anomaly in the message pattern occurs when a system goes from suspend to hibernation and back to a resumed state. The following is an example of this pattern:

Going to Suspend

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMQUERYSPEND
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMSPEND
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

Going to Hibernation

```
S WM_POWERBROADCAST (long)dwPowerEvent:0012
```

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMQUERYSPEND
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMSPEND
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

```
S WM_POWERBROADCAST (long)dwPowerEvent:PBT_APMSPEND
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

Power Button Pressed

```
S WM_POWERBROADCAST (long)dwPowerEvent:0012
```

```
R WM_POWERBROADCAST fSucceeded:True [IResult:00000001]
```

```
S WM_POWERBROADCAST (long)dwPowerEvent:0012
```

The difference in event order here is that as the system goes to hibernate, and again after the power button is pressed to turn the system back on, event PBT_APMAUTOMATICRESUE, 0x12, is received. What is missing is the PBT_APMRESUMESPEND event signaling that a resume has occurred. Our testing revealed that in some cases there is only one 0x12 code received, and in others there are two received. In no cases did we receive the PBT_APMRESUMESPEND when hibernation resumed, even though documentation indicates that this event will be sent. For this

reason there may be a need for some coordination between the PBT_APMRESUMESUSPEND and the PBT_APMAUTOMATICRESUME events to ensure that a healthy state of execution is reached by the application following a suspension.

Coding Example

To illustrate, we'll examine an example of how to handle the messages and events associated with suspend and resume operations.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
```

```
{
```

```
    int wmlD, wmEvent;
```

```
    PAINTSTRUCT ps;
```

```
    HDC hdc;
```

```
    switch (message)
```

```
    {
```

```
        .
```

```
        .
```

```
        .
```

```
    case WM_POWERBROADCAST:
```

```
        switch (wParam)
```

```
        {
```

```
            case PBT_APMQUERYSPEND:
```

```
                PrepToSuspend(TRUE);
```

```
                return (TRUE);
```

```
            case PBT_APMQUERYSPENDFAILED:
```

```
                ReInitVars();
```

```
        break;

    case PBT_APMRESUMESUSPEND:

        ReInitVars();

        break;

    case PBT_APMSUSPEND:

        break;

    case PBT_APMRESUMECRITICAL:

        ReInitVarsCritical();

        break;

    case PBT_APMBATTERYLOW:

        MessageBox(hWnd,"Battery is Critical","Low Battery",MB_OK);

        break;

    case PBT_APMPOWERSTATUSCHANGE:

        AdvisePowerChange();

        Break;

    case PBT_APMOEMEVENT:

        break;

    case PBT_APMRESUMEAUTOMATIC:

        break;

}

.

.

.

default:
```

```

        return DefWindowProc(hWnd, message, wParam, lParam);

    }

    return 0;
}

```

This is just one way you could treat each of the events that are associated with the WM_POWERBROADCAST message. Note that a call to PrepToSuspend(TRUE) is made when the PBT_APMQUERYSUSPEND event is received and TRUE is returned. If the application wanted to deny suspension, it would simply return BROADCAST_QUERY_DENY here and have not call to the function as illustrated. Note also that each of the non-critical resume related messages invokes RelnitVars(), and that the critical resume event calls RelnitVarsCritical(). The difference in function content could be reflected in a more robust initialization for the critical resume as we have discussed.

The example above also lists three events not discussed in this paper. PBT_APMOEMEVENT is not a consistent event but is left to the discretion of the OEM vendor to perhaps capture their own events. In most cases it is not necessary to respond to this event. PBT_APMBATTERYLOW is received when this condition is signaled by the system. Other appropriate actions could take place here other than just informing the user that the battery is critical. PBT_POWERSTATUSCHANGE is an event that should be handled by the application. It signals a change in the power state of the machine, communicating battery information and other important data to the application. Its details are left to the reader to discover.

Conclusion

The age old question of “Why should we do this in our application?” comes up whenever we examine new functionality and need to determine whether or not it is relevant to a particular product. In the case of suspension, as more and more people purchase mobile notebook PCs to serve as a primary system both at home and in the business environment, it becomes more important for applications to pay particular attention to this occurrence. The customer has begun to expect applications that suspend and resume to a working state, more gracefully, more error and problem free than ever before. Applications that rely on files or content on the network, that operate for long periods of time without user interaction but with display, that capture input from the user for storage or use at a later time, or that keep large amounts of data in memory are examples of applications that should handle these messages and events. Software architects and strategists can design more robust, better performing applications by designing their software to overcome the challenges of the mobile workforce and lifestyle.

Today’s software requirements are undergoing a major redesign because of the growing need to support mobile computing users. The WM_POWERBROADCAST message is used by the operating system to signal an imminent change in the operations of a system. As the role of suspension is to conserve the battery life, this feature will have more impact on the application than it has in the past. Applications should therefore become more aware of power consumption, and make every

effort to preserve battery power, handling properly these messages and events that are targeted to aid in this endeavor.

Additional Resources

Articles

[Intel® Mobile Application Architecture Guide](#)

[Windows WM_POWERBROADCAST Messages](#)

[Power Management: Designing Applications to Conserve Battery Life](#)

[Mobile Computing Increases Demands on Hardware](#)

Developer Centers

[Mobilized Software](#)

[Intel® Centrino™ Mobile Technology](#)

[Intel® PCA processors](#)

Community

[Handheld & Wireless Application Development](#)

[Intel® Software Development Products](#)

Other Resources

[Intel® Early Access Program for Mobility](#)

[CMP Mobilized Software Site*](#)

