

# Microsoft Windows Logo Program Software Requirements

---

**Chapter 9****OnNow/ACPI Support**

These guidelines define the Gold Logo Program requirements for ensuring that the application can participate in system-wide power management.

---

**Summary of OnNow/ACPI Requirements**

---

**Requirements**

1. Respond to sleep requests from the operating system
2. Respond to sleep notifications properly
3. Handle non-critical wake notifications without losing data
4. Handle critical sleep and wake notifications properly
5. Use SetThreadExecution() to indicate a busy application
6. Reconcile multi-user edits on network files

**Rationale**

Applications must participate in system-wide power management decision-making to ensure error-free handling of power down and power up scenarios. Applications must be able to put themselves to sleep on system or user request to support a low-power state, and then they must respond to wake notifications, preserving data appropriately.

In this chapter, the term "sleep" means that the system is on standby or is in hibernation. To the application, standby and hibernation are the same. The difference occurs in how the operating system determines what gets powered down. The application does not need to provide any additional feedback to make this determination.

The OnNow design initiative is a set of design specifications which, when applied to system hardware and software applications, enable a PC to deliver the instantly available capabilities consumers expect from TVs, VCRs, stereos and other appliances.

**Reference**

For more information about the OnNow design initiative, see the web site available at <http://www.microsoft.com/hwdev/onnow.htm>.

**Customer Benefits**

- OnNow/Advance Configuration and Power Interface (ACPI) reduces power consumption, whether power comes from a wall outlet or a battery. When the computer is turned "off," it goes into a lower power state and can then be wakened by a device in the system, such as the network card, modem, or keyboard.
- The PC is consistently available to the user because it can rapidly return from a low power state to a fully-functional state. For example, network administrators can manage computers late at night, and home users can receive faxes, without being there to turn on and tend to the machine.

- Customers can control what happens when their PCs power down in a way that is easily understood and predictable.

## How to Comply with OnNow/ACPI Requirements

This section defines the specific OnNow/ACPI requirements for applications.

### 1. Respond to sleep requests from the operating system

Your application must always accept a sleep request in order for the operating system to go into a lower power state. Whenever the application receives a WM\_POWERBROADCAST message with a *wParam* value of PBT\_APMQUERYSPEND, the application must get ready to go to sleep by ensuring that all outgoing operations are completed and no data loss will occur on resumption. The application must then return a value of TRUE for this message.

#### Exceptions

1. The application may display a dialog box and enlist the user's help if both of the following conditions are true:
  - The UI bit is set in the WM\_POWERBROADCAST message.
  - Putting the computer to sleep will cause user data loss or corruption.

The application may return a value of BROADCAST\_QUERY\_DENY in response to the WM\_POWERBROADCAST message if the user answers "no" when the application warns of a loss of data that might occur if the computer is put to sleep. Examples include an indexing or database query that cannot be interrupted, but is expected to finish in a short period of time. This example will result in the operating system canceling the sleep request and the application being notified with a WM\_POWERBROADCAST PBT\_APMQUERYSPENDFAILED message.

The dialog confirmation box must have a built-in time limit of about 30 seconds or less, so that if the user does not respond within a given period of time, the application will once again attempt to go to sleep.

2. If the UI bit is not set and if the application has file handles open across the network which might lead to loss of data, the application may refuse the sleep request if it is complex or impossible to design a mechanism by which the application can recover using a local temporary file upon resuming.

In the instances where applications have file handles open across the network and a sleep request is received, some special handling may be needed. An example of this is the case when a user may put a computer to sleep and then undock it, or put a computer to sleep and unplug the network card.

To handle these situations, it is recommended, but not required, that the application do the following: If the user has not explicitly requested that the modified data be saved to the open file, the application should save a copy of this data in a temporary file on local non-volatile media. The application will close all file handles at the time of suspending. When resuming, if the network is not present, the application should prompt the user of this failure and then offer to allow the user to continue to work by restoring context from the copy of the temporary file saved on local media.

**Note:** If the client-side caching service is running on the computer, it will provide exactly this functionality. In this case, the call to open a network file in the application will succeed because client-side caching will provide data from a local copy of the file.

In all other cases, the application may *not* refuse a sleep request if the UI bit is not set. This situation occurs, for example, when the computer is going to sleep as a result of a user closing the lid on a laptop, and then cannot respond to any UI messages. In this case, the application must attempt to save data, get ready to stop all activity, and return TRUE in response to the WM\_POWERBROADCAST/PBT\_APMQUERYSPEND message.

## ***2. Respond to sleep notifications properly***

---

Once all applications have accepted the sleep request, the operating system will send a WM\_POWERBROADCAST / PBT\_APMSPEND message. In response, your application must allow the hardware to completely power down by taking the appropriate actions. These actions include:

- Saving all data and closing all open files, including network files
- Pausing sound
- Pausing all play in games
- Restoring any drivers that the application modified to their initial state

Although your application is required to prevent data loss, it is up to each individual application to determine the appropriate implementation. We do, however, advise that your application do so in the following manner:

- Flush any user data to local non-volatile storage that will persist after the power supply is shut down.
- Write user data to temporary storage when the user has not requested that the original file be overwritten.

**Note:** In some circumstances, the sleep request may be canceled. If this occurs, the operating system follows the WM\_POWERBROADCAST/PBT\_APMQUERYSPEND message with WM\_POWERBROADCAST / APM\_QUERYSPENDFAILED message. In this case, the application should restore all its data to a working state and continue all operations normally.

## ***3. Handle non-critical wake notifications without losing data***

---

The application must be able to handle non-critical wake notifications without losing data and return the application to an unambiguous state.

Applications that were open when the computer went into the sleep state should be open when the system wakes. The operating system will notify the applications with a WM\_POWERBROADCAST / PBT\_APMRESUMESPEND message when the computer wakes up from sleep. The applications should attempt to restore all context to the state that existed just before the computer went to sleep. If the data cannot be fully restored, the application should notify the user, and it may seek the user's aid in restoring data to a state acceptable to the user.

The application must attempt to recover back to a stable state. The application must not stall, cause the system to crash, destabilize the system, corrupt existing data files, or knowingly lose data without notifying the user.

It is not required for games to restore context to the point when the computer went to sleep, but we recommend that the state is restored to a point that provides a good user experience. The game still must follow all the other robustness requirements—no system crashes and so on.

#### **4. Handle critical sleep and wake notifications properly**

---

Applications must respond to critical wake notifications at least as well as they recover from power loss.

In certain situations, the operating system may need to perform a critical sleep operation—for example, if battery capacity is critically low, or if the computer temperature is critically high and must be shut down to prevent hardware damage. A user may also initiate a critical sleep in an urgent situation—for example, the user has to board a plane and must shut the computer down instantly.

In cases of critical sleep, the applications will not be notified by the operating system of the impending sleep event. Your application will not get the opportunity to perform any actions defined in the previous requirements.

When the computer is wakened, the operating system will indicate in its wake notification whether the shut down was critical. The application should not crash and should wake to a stable state—no stalls, crashes, or corruption of files that were not opened by the application. Data may be lost, but the application should notify the user of the data loss that occurred.

#### **5. Use `SetThreadExecution()` to indicate a busy application**

---

Applications which have long operations that need to continue running even though the PC appears idle must use the `SetThreadExecution()` API function to mark the computer as busy. When marked as busy, the operating system will not send sleep requests as a result of the computer being idle.

The applications should use the `ES_CONTINUOUS` flag only for cases when the application is performing an operation which, if interrupted, might result in a loss of data. Performing a query and an update of a database is one such case. In all other cases, the applications should only use the `ES_SYSTEM_REQUIRED` flag. See the Microsoft Platform SDK for details.

Please note that using the `SetThreadExecution()` call does not stop explicit sleep requests made by the user; these requests must still be handled as specified in all the other requirements in this chapter.

Examples of applications that need to use this function are video playback or presentation applications to keep the display on. Such applications should use the `SetThreadExecutionState()` API with the `ES_DISPLAY_REQUIRED` flag set to prevent the operating system from turning off the display.

#### **6. Reconcile multi-user edits on network files**

---

If an application accepts a sleep request while network file handles are open, edit conflicts can occur that would not normally happen in non-sleep/wake scenarios.

When a computer is placed in a sleep state, locks on network files may be released. This allows other users to edit a file that would otherwise remain locked. When your application responds to a wake notification and tries to reopen file handles in such cases, it must (in sequence):

1. Detect that someone has modified or has locked one or more of the files.
2. Alert the user to this condition.
3. Allow the user to keep working while other users have the original files locked.
4. Offer to save user's work by saving the old file with user's modification to another directory and with another file name. It is recommended, but not

required, that in the interest of providing a better user experience, applications add functionality to assist the user to merge the differences between the two versions of a file.

## ***How to Pretest Applications for OnNow/ACPI Support***

---

This section presents pretesting guidelines for OnNow/ACPI requirements.

### ***To pretest productivity applications for OnNow/ACPI compliance:***

---

- Open a file on a local hard disk, edit it—don't save—and put the computer to sleep, and then wake the computer. Verify that the application continues to run, the file can be saved, and so forth.
- Open a file on a hard disk in a docking station, edit it—don't save—and put the computer to sleep, undock the computer, and then wake the computer. Verify that the application correctly informs the user of the problem and leads the user through steps to correct the problem.
- Open a file on the network, edit it—don't save—and put the computer to sleep, wait a few minutes for the network connection to time out, and then wake the computer. Verify that the application continues to run, that the file can be saved, and so forth.
- Open a file on the network, edit it—don't save—and put the computer to sleep. Modify the file on the network from some other computer and make modifications so that some of them conflict with the modifications made previously on the first computer, and some of them do not conflict. Save the file. Wake the first computer. The application should correctly detect that the file was modified and handle the non-conflicting modifications automatically and prompt to user to reconcile the conflicting modifications.

### ***To pretest event-handling applications for OnNow/ACPI compliance:***

---

1. In the Power control panel, set the system idle time as low as possible.
2. Start the application and its event handling feature.
3. Put the computer to sleep using the Sleep button or the Start menu.
4. Start an event that will take longer to process than the system idle time. If necessary, start a series of such operations so that the computer continues to stay busy.
5. Verify that the computer does not go to sleep while the event is being processed and that, after the event has been processed, the computer goes to sleep immediately.

### ***To pretest presentation applications for OnNow/ACPI compliance:***

---

1. In the Power control panel, set the system idle time as low as possible.
2. Start the application and play a presentation.
3. Wait for a time longer than the system idle time, and verify that the computer does not go to sleep and that the display does not go blank during that time

### ***To pretest applications that may lose data if the computer is put to sleep at a particular time:***

---

1. Start the application and get it to a point where it starts operations that might corrupt data if interrupted.
2. Press the power button to put the computer to sleep

3. Verify that the application puts up a dialog box to prompt the user about possible loss of data. There are several cases to be tested under this circumstance:
  - Respond “NO” to the dialog box and verify that the computer doesn’t go to sleep and that the application continues as normal without losing any data.
  - Respond “YES” to the dialog box and verify that the computer goes to sleep. Wake the computer and ensure that the computer doesn’t hang or cause the operating system to crash; also ensure that the application doesn’t hang or crash, but that it notifies the user of any possible data loss and attempts to recover all possible data.
  - Do not respond to the dialog box, then ensure that the application times out correctly and attempts to go to sleep again at the end of the time period.

***To verify response to critical sleep requests:***

---

1. Click on the “start” button and select the “standby” option while holding down the CTRL key. This will force a critical standby to occur.
2. Wake the computer by pressing the power button. Verify that your application does not cause any destabilizing behavior such as hanging or crashing, and that it attempts to recover all possible data.

