Application Shutdown Changes in Windows Vista

- 02/08/2011
- 15 minutes to read

To make shutdown more reliable and robust in Windows Vista, several aspects of how applications are processed at shutdown will change. This topic outlines these changes and provides guidance on how applications should handle shutdown in Windows Vista and future versions of Windows.

Application Shutdown in Windows XP

In Windows XP, each running application is sent the WM_QUERYENDSESSION message at shutdown. Applications can return TRUE to indicate that they can be closed, or FALSE to indicate that they should not be closed (e.g., because doing so would cause the user to lose data or destroy a CD being burned). If an application returns FALSE, in most cases, shutdown will be cancelled (and the application that cancelled shutdown is sent WM_ENDSESSION with wParam == FALSE). Microsoft® guidance for applications that return FALSE is that they should display UI indicating why they needed to cancel shutdown.

Applications can also delay responding to WM_QUERYENDSESSION in order to display UI asking what the user would like to do. For example, when Notepad has unsaved data and displays a "Would you like to save your data?" dialog during shutdown, this is what it is doing.

By default, applications can delay responding to WM_QUERYENDSESSION for up to 5 seconds. After 5 seconds, Windows XP will display a dialog that indicates that the application is not responding and allows the user to terminate it. Until the user responds to this dialog, applications can block WM_QUERYENDSESSION (and, consequently, shutdown) indefinitely. If the user clicks the "End Now" button on the dialog, the application is immediately terminated and shutdown will continue. If the user clicks the "Cancel" button, shutdown is canceled and the application will continue running.

If any top level window returns FALSE to WM_QUERYENDSESSION, the shutdown is cancelled, and each top level window that was sent WM_QUERYENDSESSION will be sent WM_ENDSESSION with wParam == FALSE.

If every top level window returns TRUE to WM_QUERYENDSESSION, then each is sent WM_ENDSESSION with wParam == TRUE, in turn.

Applications have 5 seconds to respond to WM_ENDSESSION before Windows displays a system dialog allowing the user to terminate the application. Unlike with WM_QUERYENDSESSION, applications cannot cancel shutdown by responding FALSE to WM_ENDSESSION. Once an application responds to WM_ENDSESSION, Windows closes it. Shutdown then continues, as Windows sends WM_QUERYENDSESSION to the remaining running applications in turn.

Application Shutdown in Windows Vista

Application shutdown in Windows Vista has changed relative to Windows XP in the following respects:

New user interface for applications that block shutdown

In Windows XP, the UI displayed when applications block shutdown is a system dialog. Windows Vista will display visually distinctive full-screen UI. This UI more clearly communicates information about applications blocking shutdown, making it easier for users-many of whom are in a hurry when shutting down-to quickly decide whether to cancel or continue shutdown.

In addition, the Windows Vista UI is capable of identifying multiple applications blocking shutdown and their reasons for doing so, even though the system sends WM_QUERYENDSESSION and WM_ENDSESSION serially to applications. Windows Vista does this through a new API, discussed later in this document, that allows applications to proactively indicate that they need to block shutdown and provide a string explaining why.

Ability for users to forcefully shut down

In Windows XP, the UI for blocking applications allows users to either cancel shutdown or terminate the blocking application. If subsequent applications also block shutdown, the system displays identical UI for each blocking application. This is frustrating for many users, who, when shutting down, "just want" their computers to turn off.

Windows Vista will solve this by allowing users to terminate the blocking application and make shutdown "forceful." In a forceful shutdown, Windows will send applications WM_QUERYENDSESSION with the ENDSESSION_CRITICAL flag. If an application responds FALSE, Windows will continue shutdown instead of canceling it, and will send the application WM_ENDSESSION. If an application times out responding to WM_QUERYENDSESSION or WM_ENDSESSION, Windows will terminate it.

Silent shutdown cancellations will no longer be allowed

In Windows XP, applications are allowed to veto WM_QUERYENDSESSION without displaying any UI indicating why they need to cancel shutdown. These "silent shutdown failures" are highly frustrating to users, who often take a minute or two to realize that shutdown has failed because no UI was displayed.

Windows Vista will eliminate this possibility by displaying UI even if an application vetoes WM_QUERYENDSESSION.

New API for proactively registering shutdown objections

In Windows XP, if applications object to shutdown (for example, because a CD burn is in progress), they have no mechanism for proactively indicating this.

Windows Vista provides an API that applications can use to proactively register a string

explaining their need to block shutdown. Applications can unregister these reasons when they no longer need to block shutdown.

When the user shuts down, Windows Vista displays these reasons in a single UI, allowing users to easily identify which applications are blocking shutdown and why, and quickly decide whether to cancel or continue shutdown. This is an improvement over Windows XP, where, if multiple applications need to block shutdown, they are identified to the user in a succession of system dialogs.

Certain types of applications will no longer be allowed to block shutdown.

At shutdown, Windows Vista will check whether each running application is not responding (an application is defined as not responding if it has not responded to any of its window messages in the last 5 seconds), and, if so, automatically terminate it.

Windows Vista will also not allow console applications or applications that have no visible top-level windows to block shutdown. In most cases, such applications are less important to users at shutdown than applications that do have visible top-level windows. If an application without a visible top-level window blocks shutdown by vetoing WM_QUERYENDSESSION, or takes over 5 seconds to respond to WM_QUERYENDSESSION or WM_ENDSESSION, Windows will automatically terminate it.

However, if an application with no visible top-level windows uses the new API to proactively indicate that it needs to block shutdown, Windows Vista will not automatically terminate it, and will instead treat it like an application that does have a visible top-level window.

Default timeouts change in Windows Vista

The default timeouts around WM_QUERYENDSESSION and WM_ENDSESSION at shutdown have been reduced in some cases to provide the user with a more responsive shutdown experience. The two tables below summarize the changes around these timeouts.

In a non-critical shutdown (i.e., a situation where the user has not yet initiated a critical shutdown by clicking the "Shut down now" button in the new Windows UI discussed earlier), the following default timeouts have been implemented:

Value	Description
[1]	No change from Windows XP behavior.
[2]	Change from Windows XP behavior, but Windows does not terminate the application at shutdown.
[3]	Change from Windows XP behavior, and Windows may terminate the application at shutdown.

Message	Application Category	
	Visible top-level window or reason string specified	*No visible top-level window and no reason string specified*

WM_QUERYENDSESSION	Application can take as much time as needed to respond to WM_QUERYENDSESSION. Windows displays new UI after 5 seconds. [1]	Application has 5 seconds to respond to WM_QUERYENDSESSION, and then Windows terminates the application if the application is unresponsive. [3]
WM_QUERYENDSESSION	Windows displays new UI if the application responds **FALSE** to WM_QUERYENDSESSION. [2]	Windows sends WM_ENDSESSION message to the application if the application responds **FALSE** to WM_QUERYENDSESSION. [3]
WM_ENDSESSION	Application can take as much time as needed to respond to WM_ENDSESSION. Windows displays new UI after 5 seconds. [1]	Application has 5 seconds to respond to WM_ENDSESSION, and then Windows terminates the application if the application is unresponsive. [3]

Table 1. Application shutdown procedure in a non-critical shutdown.

In a critical shutdown, where the user has clicked the **Shut down now** button in the new Windows UI and applications will no longer be allowed to block shutdown, the following default timeouts have been implemented:

Message	Application Category	
	Visible top-level window or reason string specified	*No visible top-level window and no reason string specified*
WM_QUERYENDSESSION	Application has 1 second to respond to WM_QUERYENDSESSION, and then Windows terminates the application if the application is unresponsive. [3]	Application has 1 second to respond to WM_QUERYENDSESSION, and then Windows terminates the application if the application is unresponsive. [3]
WM_QUERYENDSESSION	If the application responds **FALSE** to WM_QUERYENDSESSION, Windows sends WM_ENDSESSION message to the application. [2]	If the application responds **FALSE** to WM_QUERYENDSESSION, Windows sends WM_ENDSESSION message to the application. [2]
WM_ENDSESSION	Application has 30 seconds to respond to WM_ENDSESSION, and then Windows terminates the application if the application is unresponsive. [3]	Application has 5 seconds to respond to WM_ENDSESSION, and then Windows terminates the application if the application is unresponsive. [3]

Table 2. Application shutdown procedure in a critical shutdown.

Best Practices for Handling Shutdown in Windows Vista

As a result of the above changes, how should applications handle shutdown in Windows Vista differently relative to Windows XP? The following are recommended best practices.

Applications should not block shutdown

If you take only one thing away from reading this topic, it should be this one. You will be presenting the best experience to your users if your application does not block shutdown. When users initiate shutdown, in the vast majority of cases, they have a strong desire to

see shutdown succeed; they may be in a hurry to leave the office for the weekend, for example. Applications should respect this desire by not blocking shutdown if at all possible.

In many cases, applications that block shutdown can be redesigned so that they no longer need to do so. For example, applications with unsaved data at shutdown can be designed to automatically save their data and restore it when the user next starts the application, instead of blocking shutdown to ask the user what to do. Microsoft Office 2007 will do this in certain scenarios.

In other cases, there is usually a sensible default action that applications should automatically perform, instead of blocking shutdown to ask the user what to do. In many cases, this is true even if the default action could be slightly destructive. For example, if an application is in the middle of burning a CD when shutdown occurs, it should cancel the CD burn instead of blocking shutdown to ask the user what to do. Most users tend not to forget that they have a CD burn in progress, and if they shut down their computers while burning a CD, it would be sensible to assume that they intend to cancel the CD burn. This is what Windows Media Player 11 will do if the user shuts down while a CD burn is in progress.

Applications that must block shutdown should use the new shutdown reason API

In Windows XP, Microsoft recommended that if an application needed to block shutdown, it should display its own UI explaining why it needed to do so, so users would be less frustrated when shutdown failed. As discussed earlier, Windows Vista will reduce user frustration when shutdown fails even more, by displaying new UI that lists all the reasons applications have provided for blocking shutdown.

In Windows Vista, if your application must block shutdown, in addition to returning FALSE or not responding to WM_QUERYENDSESSION, it should leverage this new UI by using a simple API to provide Windows with a reason string explaining why it is blocking shutdown. This API is straightforward:

```
BOOL ShutdownBlockReasonCreate(HWND hWnd, LPCWSTR pwszReason);
```

```
BOOL ShutdownBlockReasonDestroy(HWND hWnd);
```

```
BOOL ShutdownBlockReasonQuery(HWND hWnd, LPWSTR pwszBuff, DWORD *pcchBuff);
```

Use of this API is detailed more fully later in this topic, as well as in the MSDN documentation for the individual ShutdownBlockReason() functions.

Again, note that this API does not replace the need to return FALSE (or delay responding) to WM_QUERYENDSESSION to block shutdown. Applications need to do this in addition to using the API. Applications that return TRUE to WM_QUERYENDSESSION will be closed at shutdown, regardless of whether they have used the API.

Also note that if your application has no visible top-level windows, it must use this API if it needs to successfully block shutdown. Such applications will automatically be terminated if they block shutdown without using the API.

Applications should no longer rely on always being able to block shutdown

If the user initiates a critical shutdown by clicking the "Shut down now" button from the new Windows UI discussed earlier, applications will not be allowed to block shutdown. In a critical shutdown, applications that return FALSE to WM_QUERYENDSESSION will be sent WM_ENDSESSION and closed, while those that time out in response to WM_QUERYENDSESSION will be terminated. The different procedure that is followed in a critical shutdown is summarized in Table 2 above.

In most cases, shutdown will not be critical, but applications should be prepared for the possibility of a critical shutdown. By following these recommendations, you can ensure that your application will handle critical shutdowns gracefully, without being terminated:

1. Make sure your application is prepared to respond to WM_ENDSESSION even if it responded FALSE to WM_QUERYENDSESSION.
2. Make sure your application responds to WM_QUERYENDSESSION as quickly as possible. If you need to do any time-consuming processing at shutdown, you should do it in response to WM_ENDSESSION. As indicated in Table 2, applications that respond TRUE to WM_QUERYENDSESSION will by default be given 30 seconds to do shutdown processing and respond to WM_ENDSESSION.
3. If your application needs to determine whether a shutdown is critical, it can check the ENDSESSION_CRITICAL bit in the lParam parameter of WM_QUERYENDSESSION.

Using the New Shutdown Reason API

The new shutdown reason API consists of three functions:

- `BOOL ShutdownBlockReasonCreate(HWND hWnd, LPCWSTR pwszReason);`
- `BOOL ShutdownBlockReasonDestroy(HWND hWnd);`
- `BOOL ShutdownBlockReasonQuery(HWND hWnd, LPWSTR pwszBuff, DWORD *pcchBuff);`

Again, the best practice for Windows Vista applications at shutdown is that they should never block shutdown. However, if your application must block shutdown, Microsoft recommends that you use this API. The functions are simple to use.

1. When your application needs to block shutdown, it should call ShutdownBlockReasonCreate() to register a reason string, and pass in a handle to the window it uses to handle WM_QUERYENDSESSION.
2. When your application no longer needs to block shutdown, it should call ShutdownBlockReasonDestroy() to unregister its reason string.
3. If your application needs to determine what reason string it registered earlier, it should call ShutdownBlockReasonQuery() to retrieve it.

Windows will store the reason string associated with your application for the duration of the user's session. The reason string store is not persisted when the user successfully shuts down; new sessions initialize an empty reason string store.

There are three main usage models for these functions.

1. The application knows ahead of time when it will need to block shutdown (preferred

usage model)
2. The application only knows whether it will need to block shutdown during the shutdown process
3. The application has no visible top-level windows and needs more than 30 seconds to handle WM_ENDSESSION and perform shutdown processing

These are described in detail below.

The application knows ahead of time when it will need to block shutdown (preferred usage model)

An example of this is an application that performs a critical operation and knows that it will need to block shutdown during that operation. If that operation is performed in a single block of code within the application, then ShutdownBlockReasonCreate() should be called at the start of that block of code, and ShutdownBlockReasonDestroy() at the end. In addition, the application should return FALSE to WM_QUERYENDSESSION if the operation is in progress when WM_QUERYENDSESSION is received.

This way, the application will block shutdown and its reason for blocking shutdown will be displayed if and only if the critical operation was in progress when shutdown was initiated.

This is the preferred usage model for this API, because it has the advantage that if another application blocks shutdown before your application and the new Windows UI is displayed, your application will be listed as needing to block shutdown and its reason string will be shown. This is not true for the other usage models discussed below.

The application only knows whether it will need to block shutdown during the shutdown process

In some cases, an application can only know whether it will need to block shutdown when it is responding to WM_QUERYENDSESSION. In this case, it should call ShutdownBlockReasonCreate() within its WM_QUERYENDSESSION handler, then respond FALSE to WM_QUERYENDSESSION.

This way, the application will block shutdown and its reason for blocking shutdown will be displayed.

There are two important points to note in this usage model:

1. It is especially important to ensure that your application unregisters its reason string by calling ShutdownBlockReasonDestroy() later, when it no longer needs to block shutdown. Otherwise, Windows may display a stale string for your application if a future shutdown is blocked, and your users may be confused.
2. Your application will not always be presented to the user as needing to block shutdown. Specifically, if another application blocks shutdown before your application and the new Windows UI is displayed, your application will not be identified as needing to block shutdown, because it will not have been sent WM_QUERYENDSESSION yet.

The application has no visible top-level windows and needs more than 30

seconds to handle WM_ENDSESSION and perform shutdown processing

By default, applications without any visible top-level windows will be given 5 seconds to handle WM_ENDSESSION before being terminated.

If your application may need more than 5 seconds to complete its shutdown processing in response to WM_ENDSESSION, it should call ShutdownBlockReasonCreate() in its WM_QUERYENDSESSION handler, and promptly respond TRUE to WM_QUERYENDSESSION so as not to block shutdown. It should then perform all shutdown processing in its WM_ENDSESSION handler.

This way, Windows will treat your application as if it had visible top-level windows and will give it 30 seconds to handle WM_ENDSESSION.

Best Practices For Shutdown Reason Strings

When users are shutting down Windows, they are typically in a hurry and will spend just a few seconds looking at the new Windows UI listing application shutdown reasons. Because of this, it is imperative that the reason strings your application registers are easy to read and understand.

Below are examples of reason strings, as well as revisions to those strings that make them more concise and therefore easier to understand.

Less Clear	Clear
This application is blocking shutdown because a CD burn is in progress.	A CD burn is in progress.
A TV show recording is currently occurring.	A TV show is being recorded.
A transaction is in progress. Do not shut down.	A transaction is in progress.

Send comments about this topic to Microsoft

Build date: 2/8/2011