WaitForMultipleObjects function

Waits until one or all of the specified objects are in the signaled state or the time-out interval elapses.

To enter an alertable wait state, use the WaitForMultipleObjectsEx function.

Syntax

C++

```
DWORD WINAPI WaitForMultipleObjects(
  _In_       DWORD  nCount,
  _In_ const HANDLE *lpHandles,
  _In_       BOOL   bWaitAll,
  _In_       DWORD  dwMilliseconds
);
```

Parameters

nCount [in]

The number of object handles in the array pointed to by *lpHandles*. The maximum number of object handles is **MAXIMUM_WAIT_OBJECTS**. This parameter cannot be zero.

lpHandles [in]

An array of object handles. For a list of the object types whose handles can be specified, see the following Remarks section. The array can contain handles to objects of different types. It may not contain multiple copies of the same handle.

If one of these handles is closed while the wait is still pending, the function's behavior is undefined.

The handles must have the **SYNCHRONIZE** access right. For more information, see Standard Access Rights.

bWaitAll [in]

If this parameter is **TRUE**, the function returns when the state of all objects in the *lpHandles* array is signaled. If **FALSE**, the function returns when the state of any one of the objects is set to signaled. In the latter case, the return value indicates the object whose state caused the function to return.

dwMilliseconds [in]

The time-out interval, in milliseconds. If a nonzero value is specified, the function waits until the specified objects are signaled or the interval elapses. If *dwMilliseconds* is zero, the function does not enter a wait state if the specified objects are not signaled; it always returns immediately. If *dwMilliseconds* is **INFINITE**, the function will return only when the specified objects are signaled.

Return value

If the function succeeds, the return value indicates the event that caused the function to return. It can be one of the following values. (Note that **WAIT_OBJECT_0** is defined as 0 and **WAIT_ABANDONED_0** is defined as 0x00000080L.)

Return code/value	Description
WAIT_OBJECT_0 to (**WAIT_OBJECT_0** + *nCount*– 1)	If *bWaitAll* is **TRUE**, the return value indicates that the state of all specified objects is signaled. If *bWaitAll* is **FALSE**, the return value minus **WAIT_OBJECT_0** indicates the *lpHandles* array index of the object that satisfied the wait. If more than one object became signaled during the call, this is the array index of the signaled object with the smallest index value of all the signaled objects.
WAIT_ABANDONED_0 to (**WAIT_ABANDONED_0** + *nCount*– 1)	If *bWaitAll* is **TRUE**, the return value indicates that the state of all specified objects is signaled and at least one of the objects is an abandoned mutex object.

	If *bWaitAll* is **FALSE**, the return value minus **WAIT_ABANDONED_0** indicates the *lpHandles* array index of an abandoned mutex object that satisfied the wait. Ownership of the mutex object is granted to the calling thread, and the mutex is set to nonsignaled. If a mutex was protecting persistent state information, you should check it for consistency.
WAIT_TIMEOUT 0x00000102L	The time-out interval elapsed and the conditions specified by the *bWaitAll* parameter are not satisfied.
WAIT_FAILED (**DWORD**)0xFFFFFFFF	The function has failed. To get extended error information, call GetLastError.

Remarks

The **WaitForMultipleObjects** function determines whether the wait criteria have been met. If the criteria have not been met, the calling thread enters the wait state until the conditions of the wait criteria have been met or the time-out interval elapses.

When *bWaitAll* is **TRUE**, the function's wait operation is completed only when the states of all objects have been set to signaled. The function does not modify the states of the specified objects until the states of all objects have been set to signaled. For example, a mutex can be signaled, but the thread does not get ownership until the states of the other objects are also set to signaled. In the meantime, some other thread may get ownership of the mutex, thereby setting its state to nonsignaled.

When *bWaitAll* is **FALSE**, this function checks the handles in the array in order starting with index 0, until one of the objects is signaled. If multiple objects become signaled, the function returns the index of the first handle in the array whose object was signaled.

The function modifies the state of some types of synchronization objects. Modification occurs only for the object or objects whose signaled state caused the function to return. For example, the count of a semaphore object is decreased by one. For more information, see the documentation for the individual synchronization objects.

To wait on more than **MAXIMUM_WAIT_OBJECTS** handles, use one of the following methods:

- Create a thread to wait on **MAXIMUM_WAIT_OBJECTS** handles, then wait on that thread plus the other handles. Use this technique to break the handles into groups of **MAXIMUM_WAIT_OBJECTS**.
- Call RegisterWaitForSingleObject to wait on each handle. A wait thread from the thread pool waits on **MAXIMUM_WAIT_OBJECTS** registered objects and assigns a worker thread after the object is signaled or the time-out interval expires.

The **WaitForMultipleObjects** function can specify handles of any of the following object types in the *lpHandles* array:

- Change notification
- Console input
- Event
- Memory resource notification
- Mutex
- Process
- Semaphore
- Thread
- Waitable timer

Use caution when calling the wait functions and code that directly or indirectly creates windows. If a thread creates any windows, it must process messages. Message broadcasts are sent to all windows in the system. A thread that uses a wait function with no time-out interval may cause the system to become deadlocked. Two examples of code that indirectly creates windows are DDE and the CoInitialize function. Therefore, if you have a thread that creates windows, use MsgWaitForMultipleObjects or MsgWaitForMultipleObjectsEx, rather than **WaitForMultipleObjects**.

Examples

For an example, see Waiting for Multiple Objects.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinBase.h (include Windows.h)

Library	Kernel32.lib
DLL	Kernel32.dll

See also

Synchronization Functions
Wait Functions

Community Additions

timeout event reset

Please state if the timeout event is auto reset. I don't see anything available to call with the ResetEvent() method so the presumption is auto reset. But presumptions like that can be dangerous.

JAG77
6/22/2014

Use WaitHandle class for managed code

For managed code like Visual Basic or C#

It is better to use http://msdn.microsoft.com/en-us/library/system.threading.waithandle.aspx instead of CreateEvent / SetEvent / WaitFor(Multiple/Single)Object(s).

Mike de Klerk
9/7/2012

No duplicate HANDLEs allowed with bWaitAll==TRUE

If bWaitAll is true, this function will return WAIT_FAILED with ERROR_INVALID_PARAMETER if the HANDLE array contains any duplicates (direct or indirect).

Ref Raymond Chen's blog: http://blogs.msdn.com/b/oldnewthing/archive/2011/02/25/10133817.aspx

Steve Friedl
3/5/2011

C# syntax

```
[DllImport("kernel32.dll", SetLastError=true)]
public static extern uint WaitForMultipleObjects(uint nCount, IntPtr[] handles, bool bWaitAll, uint dwMilliseconds);
```

dmex
5/12/2009

vb.net syntax

```
<DllImport("kernel32.dll", SetLastError:=True)> _
Public Shared Function WaitForMultipleObjects(ByVal nCount As UInt32, ByVal [handles] As IntPtr(), ByVal bWaitAll As Boolean, ByVal
dwMilliseconds As UInt32) As UInt32
End Function
```

dmex
5/12/2009