

CloseHandle function

Closes an open object handle.

Syntax

C++

```
BOOL WINAPI CloseHandle(
    _In_ HANDLE hObject
);
```

Parameters

hObject [in]

A valid handle to an open object.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

If the application is running under a debugger, the function will throw an exception if it receives either a handle value that is not valid or a pseudo-handle value. This can happen if you close a handle twice, or if you call **CloseHandle** on a handle returned by the [FindFirstFile](#) function instead of calling the [FindClose](#) function.

Remarks

The **CloseHandle** function closes handles to the following objects:

- Access token
- Communications device
- Console input
- Console screen buffer
- Event
- File
- File mapping
- I/O completion port
- Job
- Mailslot
- Memory resource notification
- Mutex
- Named pipe
- Pipe
- Process
- Semaphore
- Thread
- Transaction
- Waitable timer

The documentation for the functions that create these objects indicates that **CloseHandle** should be used when you are finished with the object, and what happens to pending operations on the object after the handle is closed. In general, **CloseHandle** invalidates the specified object handle, decrements the object's handle count, and performs object retention checks. After the last handle to an object is closed, the object is removed from the system. For a summary of the creator functions for these objects, see [Kernel Objects](#).

Generally, an application should call **CloseHandle** once for each handle it opens. It is usually not necessary to call **CloseHandle** if a function that uses a

handle fails with `ERROR_INVALID_HANDLE`, because this error usually indicates that the handle is already invalidated. However, some functions use `ERROR_INVALID_HANDLE` to indicate that the object itself is no longer valid. For example, a function that attempts to use a handle to a file on a network might fail with `ERROR_INVALID_HANDLE` if the network connection is severed, because the file object is no longer available. In this case, the application should close the handle.

If a handle is transacted, all handles bound to a transaction should be closed before the transaction is committed. If a transacted handle was opened by calling [CreateFileTransacted](#) with the `FILE_FLAG_DELETE_ON_CLOSE` flag, the file is not deleted until the application closes the handle and calls [CommitTransaction](#). For more information about transacted objects, see [Working With Transactions](#).

Closing a thread handle does not terminate the associated thread or remove the thread object. Closing a process handle does not terminate the associated process or remove the process object. To remove a thread object, you must terminate the thread, then close all handles to the thread. For more information, see [Terminating a Thread](#). To remove a process object, you must terminate the process, then close all handles to the process. For more information, see [Terminating a Process](#).

Closing a handle to a file mapping can succeed even when there are file views that are still open. For more information, see [Closing a File Mapping Object](#).

Do not use the **CloseHandle** function to close a socket. Instead, use the [closesocket](#) function, which releases all resources associated with the socket including the handle to the socket object. For more information, see [Socket Closure](#).

Windows Phone 8: This API is supported.

Windows Phone 8.1: This API is supported.

Examples

For an example, see [Taking a Snapshot and Viewing Processes](#).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps Windows Store apps]
Minimum supported server	Windows 2000 Server [desktop apps Windows Store apps]
Header	Winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

- [CreateFile](#)
- [CreateFileTransacted](#)
- [DeleteFile](#)
- [FindClose](#)
- [FindFirstFile](#)
- [Handle and Object Functions](#)
- [Kernel Objects](#)
- [Object Interface](#)

Community Additions

CloseHandle() behavior with invalid handles

`CloseHandle(0)` and `CloseHandle(INVALID_HANDLE_VALUE)` will both return `FALSE`, with `GetLastError()` returning

ERROR_INVALID_HANDLE (= 6).



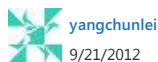
simplified macro for CloseHandle

```
#define CLOSE_HANDLE(x) CloseHandle(x); \  
x = NULL;
```



Should we call CloseHandle(hMutex) after calling OpenMutex(SYNCHRONIZE,FALSE,_T("SomeName"))?

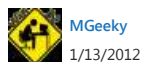
as title



CloseHandle and deadlocks ?

Hi there,

I am writing because today I had an interesting, but rather annoying issue with CloseHandle calls. Haven't got time to perform further investigations, but I am pretty sure that this function was a culprit. But let's back to the point, CloseHandle with already closed handle (or handle = 0) passed as an argument behaved strangely - led to the process deadlock. Because of my negligence in DllMain routine, where after THREAD_ATTACH/THREAD_DETACH cases I didn't set a break command, some cleanups were performing just after valid PROCESS_ATTACH. This may sound obvious - but I recommend checking passed variables to the cleanup routines...



VB.NET SafeCloseHandle by EGL

Kernel Objects

[http://msdn.microsoft.com/en-us/library/ms724485\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms724485(VS.85).aspx)

Object Categories

[http://msdn.microsoft.com/en-us/library/ms724515\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms724515(VS.85).aspx)

Some API use different destroyers such as FindClose , FreeLibrary , DestroyIcon.

It should be noted here that you can still use IntPtr but if you need to make sure a handle remains valid until your finished using it like ReadFile,WriteFile or Volume Handle then you should use this approach instead. OpenProcessToken is not the best example but was the easiest example to visually show how you can use the class. You could safely use IntPtr instead for that API because you don't really need to keep the handle valid for a long period of time you can close it immediatley.

Safe Handles and Critical Finalization

<http://msdn.microsoft.com/en-us/library/fh21e17c.aspx>

```
Imports Microsoft.Win32.SafeHandles  
Imports System.Security.Permissions  
Imports System.Runtime.ConstrainedExecution  
Imports System.Runtime.InteropServices  
Imports System.ComponentModel
```

```
Module NativeMethods
```

```
<DllImport("kernel32.dll", CharSet:=CharSet.Auto, SetLastError:=True, ExactSpelling:=True)> _  
<ReliabilityContract(Consistency.WillNotCorruptState, Cer.MayFail)> _  
Public Function CloseHandle(ByVal hObject As IntPtr) As Integer  
End Function
```

```
End Module
```

```
<SecurityPermission(SecurityAction.InheritanceDemand, UnmanagedCode:=True)> _  
<SecurityPermission(SecurityAction.Demand, UnmanagedCode:=True)> _  
Friend Class SafeCloseHandle  
Inherits SafeHandleZeroOrMinusOneIsInvalid  
Private Sub New()
```

```

MyBase.New(True)
End Sub
<ReliabilityContract(Consistency.WillNotCorruptState, Cer.MayFail)> _
Protected Overloads Overrides Function ReleaseHandle() As Boolean
Return NativeMethods.CloseHandle(handle)
End Function
End Class

```

Example of using this class might be as follows:

```

<DllImport("advapi32.dll", CharSet:=CharSet.Auto, SetLastError:=True)> _
Public Function OpenProcessToken(ByVal ProcessHandle As IntPtr, ByVal DesiredAccess As Integer, <Out()> ByRef TokenHandle As
SafeCloseHandle) As Integer
End Function

Public Class Class1

Private _hToken As SafeCloseHandle

Public Sub OpenToken()

Dim hToken As SafeCloseHandle = Nothing

If OpenProcessToken(GetCurrentProcess, TokenAccessLevels.AdjustPrivileges Or TokenAccessLevels.Query, hToken) Then
_hToken = hToken '// Makes _handle point to a critical finalizable object.
If _hToken.IsInvalid <> True Then
'// TODO: Stuff here
Console.WriteLine(_hToken.DangerousGetHandle())
'// close token handle
_hToken.Close()
Else
Throw New Win32Exception(Marshal.GetLastWin32Error())
End If
End If
End Sub
End Class

```




CloseHandle & synchronous I/O on communication devices

A small caveat: closing the last handle to a file object counts as a synchronous I/O operation, and it will synchronize with other synchronous I/O operations if the file wasn't opened with FILE_FLAG_OVERLAPPED. Closing the last handle to a pipe, socket or other device that's being currently used in a synchronous I/O operation will block until the operation completes

This is due to a small design oversight in the Windows I/O subsystem: the synchronous I/O lock of a file object isn't dropped before entering the wait for the operation's completion, causing the nesting of two locks and the potential for deadlocks

In Windows Vista and later, **CancelSynchronousIo** can be used to interrupt and cancel stuck I/O operations




Redart

10/1/2009

C# syntax

```
[DllImport("kernel32", CharSet=CharSet.Auto, SetLastError=true, ExactSpelling=true)]
internal static extern int CloseHandle(IntPtr hObject);
```




dmex

5/7/2009

vb.net syntax

```
<DllImport("kernel32.dll", CharSet:=CharSet.Auto, SetLastError:=True, ExactSpelling:=True)> _
Public Shared Function CloseHandle(ByVal hObject As IntPtr) As Integer
End Function
```



dmex

5/7/2009