ReadFileEx function

Reads data from the specified file or input/output (I/O) device. It reports its completion status asynchronously, calling the specified completion routine when reading is completed or canceled and the calling thread is in an alertable wait state.

To read data from a file or device synchronously, use the **ReadFile** function.

Syntax

C++

```
BOOL WINAPI ReadFileEx(
  _In_      HANDLE                          hFile,
  _Out_opt_ LPVOID                          lpBuffer,
  _In_      DWORD                           nNumberOfBytesToRead,
  _Inout_   LPOVERLAPPED                    lpOverlapped,
  _In_      LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

Parameters

hFile [in]

A handle to the file or I/O device (for example, a file, file stream, physical disk, volume, console buffer, tape drive, socket, communications resource, mailslot, or pipe).

This parameter can be any handle opened with the **FILE_FLAG_OVERLAPPED** flag by the **CreateFile** function, or a socket handle returned by the **socket** or **accept** function.

This handle also must have the **GENERIC_READ** access right. For more information on access rights, see **File Security and Access Rights**.

lpBuffer [out, optional]

A pointer to a buffer that receives the data read from the file or device.

This buffer must remain valid for the duration of the read operation. The application should not use this buffer until the read operation is completed.

nNumberOfBytesToRead [in]

The number of bytes to be read.

lpOverlapped [in, out]

A pointer to an **OVERLAPPED** data structure that supplies data to be used during the asynchronous (overlapped) file read operation.

For files that support byte offsets, you must specify a byte offset at which to start reading from the file. You specify this offset by setting the **Offset** and **OffsetHigh** members of the **OVERLAPPED** structure. For files or devices that do not support byte offsets, **Offset** and **OffsetHigh** are ignored.

The **ReadFileEx** function ignores the **OVERLAPPED** structure's **hEvent** member. An application is free to use that member for its own purposes in the context of a **ReadFileEx** call. **ReadFileEx** signals completion of its read operation by calling, or queuing a call to, the completion routine pointed to by *lpCompletionRoutine*, so it does not need an event handle.

The **ReadFileEx** function does use the **OVERLAPPED** structure's **Internal** and **InternalHigh** members. An application should not set these members.

The **OVERLAPPED** data structure must remain valid for the duration of the read operation. It should not be a variable that can go out of scope while the read operation is pending completion.

lpCompletionRoutine [in]

A pointer to the completion routine to be called when the read operation is complete and the calling thread is in an alertable wait state. For more information about the completion routine, see **FileIOCompletionRoutine**.

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 function succeeds, the calling thread has an asynchronous I/O operation pending: the overlapped read operation from the file. When this I/O operation completes, and the calling thread is blocked in an alertable wait state, the system calls the function pointed to by *lpCompletionRoutine*, and the wait state completes with a return code of **WAIT_IO_COMPLETION**.

If the function succeeds, and the file reading operation completes, but the calling thread is not in an alertable wait state, the system queues the completion routine call, holding the call until the calling thread enters an alertable wait state. For information about alertable waits and overlapped input/output operations, see **About Synchronization**.

If **ReadFileEx** attempts to read past the end-of-file (EOF), the call to **GetOverlappedResult** for that operation returns **FALSE** and **GetLastError** returns **ERROR_HANDLE_EOF**.

Remarks

When using **ReadFileEx** you should check **GetLastError** even when the function returns "success" to check for conditions that are "successes" but have some outcome you might want to know about. For example, a buffer

overflow when calling **ReadFileEx** will return **TRUE**, but **GetLastError** will report the overflow with **ERROR_MORE_DATA**. If the function call is successful and there are no warning conditions, **GetLastError** will return **ERROR_SUCCESS**.

The **ReadFileEx** function may fail if there are too many outstanding asynchronous I/O requests. In the event of such a failure, GetLastError can return **ERROR_INVALID_USER_BUFFER** or **ERROR_NOT_ENOUGH_MEMORY**.

To cancel all pending asynchronous I/O operations, use either:

- CancelIo—this function only cancels operations issued by the calling thread for the specified file handle.
- CancelIoEx—this function cancels all operations issued by the threads for the specified file handle.

Use CancelSynchronousIo to cancel pending synchronous I/O operations.

I/O operations that are canceled complete with the error **ERROR_OPERATION_ABORTED**.

If part of the file specified by *hFile* is locked by another process, and the read operation specified in a call to **ReadFileEx** overlaps the locked portion, the call to **ReadFileEx** fails.

When attempting to read data from a mailslot whose buffer is too small, **ReadFileEx** returns **FALSE**, and GetLastError returns **ERROR_INSUFFICIENT_BUFFER**.

Accessing the input buffer while a read operation is using the buffer may lead to corruption of the data read into that buffer. Applications must not read from, write to, reallocate, or free the input buffer that a read operation is using until the read operation completes.

An application uses the MsgWaitForMultipleObjectsEx, WaitForSingleObjectEx, WaitForMultipleObjectsEx, and SleepEx functions to enter an alertable wait state. For more information about alertable waits and overlapped input/output, see About Synchronization.

There are strict requirements for successfully working with files opened with **CreateFile** using **FILE_FLAG_NO_BUFFERING**. For details see File Buffering.

In Windows 8 and Windows Server 2012, this function is supported by the following technologies.

Technology	Supported
Server Message Block (SMB) 3.0 protocol	Yes
SMB 3.0 Transparent Failover (TFO)	Yes
SMB 3.0 with Scale-out File Shares (SO)	Yes
Cluster Shared Volume File System (CsvFS)	Yes
Resilient File System (ReFS)	Yes

Transacted Operations

If there is a transaction bound to the file handle, then the function returns data from the transacted view of the file. A transacted read handle is guaranteed to show the same view of a file for the duration of the handle. For additional information, see About Transactional NTFS.

Examples

For an example, see Named Pipe Server Using Completion Routines.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	FileAPI.h (include Windows.h); WinBase.h on Windows Server 2008 R2, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CancelIo
CancelIoEx
CancelSynchronousIo
CreateFile
File Management Functions
FileIOCompletionRoutine
MsgWaitForMultipleObjectsEx
ReadFile

SetErrorMode
SleepEx
WaitForMultipleObjectsEx
WaitForSingleObjectEx
WriteFileEx