OVERLAPPED structure

Contains information used in asynchronous (or *overlapped*) input and output (I/O).

Syntax

C++

```
typedef struct _OVERLAPPED {
  ULONG_PTR Internal;
  ULONG_PTR InternalHigh;
  union {
    struct {
      DWORD Offset;
      DWORD OffsetHigh;
    };
    PVOID  Pointer;
  };
  HANDLE    hEvent;
} OVERLAPPED, *LPOVERLAPPED;
```

Members

Internal

The status code for the I/O request. When the request is issued, the system sets this member to **STATUS_PENDING** to indicate that the operation has not yet started. When the request is completed, the system sets this member to the status code for the completed request.

The **Internal** member was originally reserved for system use and its behavior may change.

InternalHigh

The number of bytes transferred for the I/O request. The system sets this member if the request is completed without errors.

The **InternalHigh** member was originally reserved for system use and its behavior may change.

Offset

The low-order portion of the file position at which to start the I/O request, as specified by the user.

This member is nonzero only when performing I/O requests on a seeking device that supports the concept of an offset (also referred to as a file pointer mechanism), such as a file. Otherwise, this member must be zero.

For additional information, see Remarks.

OffsetHigh

The high-order portion of the file position at which to start the I/O request, as specified by the user.

This member is nonzero only when performing I/O requests on a seeking device that supports the concept of an offset (also referred to as a file pointer mechanism), such as a file. Otherwise, this member must be zero.

For additional information, see Remarks.

Pointer

Reserved for system use; do not use after initialization to zero.

hEvent

A handle to the event that will be set to a signaled state by the system when the operation has completed. The user must initialize this member either to zero or a valid event handle using the CreateEvent function before passing this structure to any overlapped functions. This event can then be used to synchronize simultaneous I/O requests for a device. For additional information, see Remarks.

Functions such as **ReadFile** and **WriteFile** set this handle to the nonsignaled state before they begin an I/O operation. When the operation has completed, the handle is set to the signaled state.

Functions such as GetOverlappedResult and the synchronization **wait functions** reset auto-reset events to the nonsignaled state. Therefore, you should use a manual reset event; if you use an auto-reset event, your application can stop responding if you wait for the operation to complete and then call **GetOverlappedResult** with the *bWait* parameter set to **TRUE**.

Remarks

Any unused members of this structure should always be initialized to zero before the structure is used in a function call. Otherwise, the function may fail and return **ERROR_INVALID_PARAMETER**.

The **Offset** and **OffsetHigh** members together represent a 64-bit file position. It is a byte offset from the start of the file or file-like device, and it is specified by the user; the system will not modify these values. The calling process must set this member before passing the **OVERLAPPED** structure to functions that use an offset, such as the **ReadFile** or **WriteFile** (and related) functions.

You can use the HasOverlappedIoCompleted macro to check whether an asynchronous I/O operation has completed if GetOverlappedResult is too cumbersome for your application.

You can use the CancelIo function to cancel an asynchronous I/O operation.

A common mistake is to reuse an **OVERLAPPED** structure before the previous asynchronous operation has been completed. You should use a separate structure for each request. You should also create an event object for each thread that processes data. If you store the event handles in an array, you could easily wait for all events to be signaled using the WaitForMultipleObjects function.

For additional information and potential pitfalls of asynchronous I/O usage, see **CreateFile**, **ReadFile**, **WriteFile**, and related functions.

For a general synchronization overview and conceptual **OVERLAPPED** usage information, see Synchronization and Overlapped Input and Output and related topics.

For a file I/O–oriented overview of synchronous and asynchronous I/O, see Synchronous and Asynchronous I/O.

Examples

For an example, see Named Pipe Server Using Overlapped I/O.

Requirements

Minimum supported client	Windows XP [desktop apps | Windows Store apps]
Minimum supported server	Windows Server 2003 [desktop apps | Windows Store apps]
Header	WinBase.h on Windows XP, Windows Server 2003, Windows Vista, Windows 7, Windows Server 2008, and Windows Server 2008 R2 (include Windows.h); MinWinBase.h on Windows 8 and Windows Server 2012

See also

CancelIo
CreateFile
GetOverlappedResult
HasOverlappedIoCompleted
ReadFile
Synchronization and Overlapped Input and Output
Synchronous and Asynchronous I/O
WriteFile