VirtualAllocEx function

Reserves, commits, or changes the state of a region of memory within the virtual address space of a specified process. The function initializes the memory it allocates to zero.

To specify the NUMA node for the physical memory, see **VirtualAllocExNuma**.

Syntax

C++

```
LPVOID WINAPI VirtualAllocEx(
  _In_     HANDLE hProcess,
  _In_opt_ LPVOID lpAddress,
  _In_     SIZE_T dwSize,
  _In_     DWORD  flAllocationType,
  _In_     DWORD  flProtect
);
```

Parameters

hProcess [in]

The handle to a process. The function allocates memory within the virtual address space of this process.

The handle must have the **PROCESS_VM_OPERATION** access right. For more information, see **Process Security and Access Rights**.

lpAddress [in, optional]

The pointer that specifies a desired starting address for the region of pages that you want to allocate.

If you are reserving memory, the function rounds this address down to the nearest multiple of the allocation granularity.

If you are committing memory that is already reserved, the function rounds this address down to the nearest page boundary. To determine the size of a page and the allocation granularity on the host computer, use the **GetSystemInfo** function.

If *lpAddress* is **NULL**, the function determines where to allocate the region.

If this address is within an enclave that you have not initialized by calling **InitializeEnclave**, **VirtualAllocEx** allocates a page of zeros for the enclave at that address. The page must be previously uncommitted, and

will not be measured with the EEXTEND instruction of the Intel Software Guard Extensions programming model.

If the address in within an enclave that you initialized, then the allocation operation fails with the **ERROR_INVALID_ADDRESS** error.

dwSize [in]

The size of the region of memory to allocate, in bytes.

If *lpAddress* is **NULL**, the function rounds *dwSize* up to the next page boundary.

If *lpAddress* is not **NULL**, the function allocates all pages that contain one or more bytes in the range from *lpAddress* to *lpAddress+dwSize*. This means, for example, that a 2-byte range that straddles a page boundary causes the function to allocate both pages.

flAllocationType [in]

The type of memory allocation. This parameter must contain one of the following values.

Value	Meaning
MEM_COMMIT 0x00001000	Allocates memory charges (from the overall size of memory and the paging files on disk) for the specified reserved memory pages. The function also guarantees that when the caller later initially accesses the memory, the contents will be zero. Actual physical pages are not allocated unless/until the virtual addresses are actually accessed. To reserve and commit pages in one step, call **VirtualAllocEx** with `MEM_COMMIT | MEM_RESERVE`. Attempting to commit a specific address range by specifying **MEM_COMMIT** without **MEM_RESERVE** and a non-**NULL** *lpAddress* fails unless the entire range has already been reserved. The resulting error code is **ERROR_INVALID_ADDRESS**. An attempt to commit a page that is already committed does not cause the function to fail. This means that you can commit pages without first determining the current commitment state of each page. If *lpAddress* specifies an address within an enclave, *flAllocationType* must be **MEM_COMMIT**.
MEM_RESERVE 0x00002000	Reserves a range of the process's virtual address space without allocating any actual physical storage in memory or in the paging file on disk. You commit reserved pages by calling **VirtualAllocEx** again with **MEM_COMMIT**. To reserve and commit pages in one step, call **VirtualAllocEx** with `MEM_COMMIT | MEM_RESERVE`.

	Other memory allocation functions, such as **malloc** and LocalAlloc, cannot use reserved memory until it has been released.
MEM_RESET 0x00080000	Indicates that data in the memory range specified by *lpAddress* and *dwSize* is no longer of interest. The pages should not be read from or written to the paging file. However, the memory block will be used again later, so it should not be decommitted. This value cannot be used with any other value. Using this value does not guarantee that the range operated on with **MEM_RESET** will contain zeros. If you want the range to contain zeros, decommit the memory and then recommit it. When you use **MEM_RESET**, the **VirtualAllocEx** function ignores the value of *fProtect*. However, you must still set *fProtect* to a valid protection value, such as **PAGE_NOACCESS**. **VirtualAllocEx** returns an error if you use **MEM_RESET** and the range of memory is mapped to a file. A shared view is only acceptable if it is mapped to a paging file.
MEM_RESET_UNDO 0x1000000	**MEM_RESET_UNDO** should only be called on an address range to which **MEM_RESET** was successfully applied earlier. It indicates that the data in the specified memory range specified by *lpAddress* and *dwSize* is of interest to the caller and attempts to reverse the effects of **MEM_RESET**. If the function succeeds, that means all data in the specified address range is intact. If the function fails, at least some of the data in the address range has been replaced with zeroes. This value cannot be used with any other value. If **MEM_RESET_UNDO** is called on an address range which was not **MEM_RESET** earlier, the behavior is undefined. When you specify **MEM_RESET**, the **VirtualAllocEx** function ignores the value of *flProtect*. However, you must still set *flProtect* to a valid protection value, such as **PAGE_NOACCESS**. **Windows Server 2008 R2, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP:** The **MEM_RESET_UNDO** flag is not supported until Windows 8 and Windows Server 2012.

This parameter can also specify the following values as indicated.

Value	Meaning
MEM_LARGE_PAGES	Allocates memory using large page support. The size and alignment must be a multiple of the large-page minimum. To obtain this

0x20000000	value, use the **GetLargePageMinimum** function.
MEM_PHYSICAL 0x00400000	Reserves an address range that can be used to map **Address Windowing Extensions** (AWE) pages. This value must be used with **MEM_RESERVE** and no other values.
MEM_TOP_DOWN 0x00100000	Allocates memory at the highest possible address. This can be slower than regular allocations, especially when there are many allocations.

flProtect [in]

The memory protection for the region of pages to be allocated. If the pages are being committed, you can specify any one of the **memory protection constants**.

If *lpAddress* specifies an address within an enclave, *flProtect* cannot be any of the following values:

- PAGE_NOACCESS
- PAGE_GUARD
- PAGE_NOCACHE
- PAGE_WRITECOMBINE

Return value

If the function succeeds, the return value is the base address of the allocated region of pages.

If the function fails, the return value is **NULL**. To get extended error information, call **GetLastError**.

Remarks

Each page has an associated **page state**. The **VirtualAllocEx** function can perform the following operations:

- Commit a region of reserved pages
- Reserve a region of free pages
- Simultaneously reserve and commit a region of free pages

VirtualAllocEx cannot reserve a reserved page. It can commit a page that is already committed. This means you can commit a range of pages, regardless of whether they have already been committed, and the function will not fail.

You can use **VirtualAllocEx** to reserve a block of pages and then make additional calls to **VirtualAllocEx** to

commit individual pages from the reserved block. This enables a process to reserve a range of its virtual address space without consuming physical storage until it is needed.

If the *lpAddress* parameter is not **NULL**, the function uses the *lpAddress* and *dwSize* parameters to compute the region of pages to be allocated. The current state of the entire range of pages must be compatible with the type of allocation specified by the *flAllocationType* parameter. Otherwise, the function fails and none of the pages is allocated. This compatibility requirement does not preclude committing an already committed page; see the preceding list.

To execute dynamically generated code, use **VirtualAllocEx** to allocate memory and the VirtualProtectEx function to grant **PAGE_EXECUTE** access.

The **VirtualAllocEx** function can be used to reserve an Address Windowing Extensions (AWE) region of memory within the virtual address space of a specified process. This region of memory can then be used to map physical pages into and out of virtual memory as required by the application. The **MEM_PHYSICAL** and **MEM_RESERVE** values must be set in the *AllocationType* parameter. The **MEM_COMMIT** value must not be set. The page protection must be set to **PAGE_READWRITE**.

The VirtualFreeEx function can decommit a committed page, releasing the page's storage, or it can simultaneously decommit and release a committed page. It can also release a reserved page, making it a free page.

When creating a region that will be executable, the calling program bears responsibility for ensuring cache coherency via an appropriate call to FlushInstructionCache once the code has been set in place. Otherwise attempts to execute code out of the newly executable region may produce unpredictable results.

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

Memory Management Functions

ReadProcessMemory
Virtual Memory Functions
VirtualAllocExNuma
VirtualFreeEx
VirtualLock
VirtualProtect
VirtualQuery
WriteProcessMemory