Memory Management

The memory manager implements virtual memory, provides a core set of services such as memory mapped files, copy-on-write memory, large memory support, and underlying support for the cache manager.

# About Memory Management

Each process on 32-bit Microsoft Windows has its own virtual address space that enables addressing up to 4 gigabytes of memory. Each process on 64-bit Windows has a virtual address space of 8 terabytes. All threads of a process can access its virtual address space. However, threads cannot access memory that belongs to another process, which protects a process from being corrupted by another process.

# Virtual Address Space

The virtual address space for a process is the set of virtual memory addresses that it can use. The address space for each process is private and cannot be accessed by other processes unless it is shared.

A virtual address does not represent the actual physical location of an object in memory; instead, the system maintains a *page table* for each process, which is an internal data structure used to translate virtual addresses into their corresponding physical addresses. Each time a thread references an address, the system translates the virtual address to a physical address.

The virtual address space for 32-bit Windows is 4 gigabytes (GB) in size and divided into two partitions: one for use by the process and the other reserved for use by the system. By default, 64-bit Microsoft Windows-based applications have a user-mode address space of several terabytes.

However, applications can specify that the system should allocate all memory for the application below 2 gigabytes. This feature is beneficial for 64-bit applications if the following conditions are true:

* A 2 GB address space is sufficient.
* The code has many pointer truncation warnings.
* Pointers and integers are freely mixed.
* The code has polymorphism using 32-bit data types.

All pointers are still 64-bit pointers, but the system ensures that every memory allocation occurs below the 2 GB limit, so that if the application truncates a pointer, no significant data is lost. Pointers can be truncated to 32-bit values, then extended to 64-bit values by either sign extension or zero extension.

To specify this memory limitation, use the **/LARGEADDRESSAWARE:NO** linker option. Note that **/LARGEADDRESSAWARE:NO** is ignored for an ARM64 binary. However, be aware that problems can occur when using this option. If you build a DLL that uses this option and the DLL is called by an application that does not use this option, the DLL could truncate a 64-bit pointer whose upper 32 bits are significant. This can cause application failure without any warning.

# Memory Pools

 nonpaged pool and paged pool. Both memory pools are located in the region of the address space that is reserved for the system and mapped into the virtual address space of each process. The nonpaged pool consists of virtual memory addresses that are guaranteed to reside in physical memory as long as the corresponding kernel objects are allocated. The paged pool consists of virtual memory that can be paged in and out of the system. To improve performance, systems with a single processor have three paged pools, and multiprocessor systems have five paged pools.

The handles for [kernel objects](https://msdn.microsoft.com/en-us/library/windows/desktop/ms724485(v=vs.85).aspx) are stored in the paged pool, so the number of handles you can create is based on available memory.

The system records the limits and current values for its nonpaged pool, paged pool, and page file usage.

**VIRTUAL MEMORY**

# Virtual memory may at the flash or disk. The application program uses the physical addresses at the RAM.A virtual memory management systems maps the virtual addresses of the pages with the physical addresses of the pages after the pages of the program has been loaded at RAM.

**PAGING**

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

**KERNEL MEMORY**

The total **memory** consists of the RAM (random-access **memory**) and the virtual**memory**. The "**Paged**" under **kernel memory** in the task manager refers to the part of the virtual **memory** that is dedicated to the **kernel**, while the "**Nonpaged**" refers to the dedicated **kernel memory** in RAM.

**CACHED MEMORY**

A **memory cache**, sometimes called a **cache** store or **RAM cache**, is a portion of**memory** made of high-speed static **RAM** (SRAM) instead of the slower and cheaper dynamic **RAM** (DRAM) used for main **memory**. **Memory caching** is effective because most programs access the same data or instructions over and over.

### Type of Paging

ATG Search supports two types of paging, normal paging and fast paging.

Normal paging is the default. In this mode, the search engine returns (in the form handler’s pagesAvailable property), the total number of pages of results. You can create links that enable the customer to go directly to any page.

Fast paging is specified by setting the fastPaging property of the search request to true. In this mode, the search engine does not return information about the total number of pages of results; pagesAvailable is set to the highest-numbered page that has been rendered so far. You can enable customers to go to the next page or to any page previously rendered.

For a single-partition index, normal paging is always enabled (the fastPaging property is ignored). For a multi-partition index, you can choose between normal paging and fast paging, but fast paging is recommended. Fast paging is much less resource-intensive than normal paging. On multi-partition indexes, normal paging can be very memory- and CPU-intensive, because results from the partitions must be merged.

**Page-in & Page-out**

If the paged out information for the least active process is needed back into the memory, the page is retrieved from the disk and brought back into the memory. This is called page-in.

Similarly, when the least active frame is moved out from the memory on to the disk, the process is called page-out.

# Working Set

The working set of a process is the set of pages in the virtual address space of the process that are currently resident in physical memory. The working set contains only pageable memory allocations; nonpageable memory allocations such as [Address Windowing Extensions](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366527(v=vs.85).aspx)(AWE) or [large page allocations](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366720(v=vs.85).aspx) are not included in the working set.

When a process references pageable memory that is not currently in its working set, a page fault occurs. The system page fault handler attempts to resolve the page fault and, if it succeeds, the page is added to the working set. (Accessing AWE or large page allocations never causes a page fault, because these allocations are not pageable .)

A hard page fault must be resolved by reading page contents from the page's backing store, which is either the system paging file or a memory-mapped file created by the process. A soft page fault can be resolved without accessing the backing store. A soft page fault occurs when:

* The page is in the working set of some other process, so it is already resident in memory.
* The page is in transition, because it either has been removed from the working sets of all processes that were using the page and has not yet been repurposed, or it is already resident as a result of a memory manager prefetch operation.
* A process references an allocated virtual page for the first time (sometimes called a demand-zero fault).

# Page State

The pages of a process's virtual address space can be in one of the following states.

|  |  |
| --- | --- |
| **State** | **Description** |
| Free | The page is neither committed nor reserved. The page is not accessible to the process. It is available to be reserved, committed, or simultaneously reserved and committed. Attempting to read from or write to a free page results in an access violation exception.  A process can use the **[VirtualFree](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366892(v=vs.85).aspx)** or **[VirtualFreeEx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366894(v=vs.85).aspx)** function to release reserved or committed pages of its address space, returning them to the free state. |
| Reserved | The page has been reserved for future use. The range of addresses cannot be used by other allocation functions. The page is not accessible and has no physical storage associated with it. It is available to be committed.  A process can use the **[VirtualAlloc](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366887(v=vs.85).aspx)** or **[VirtualAllocEx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890(v=vs.85).aspx)** function to reserve pages of its address space and later to commit the reserved pages. It can use **[VirtualFree](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366892(v=vs.85).aspx)** or **[VirtualFreeEx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366894(v=vs.85).aspx)** to decommit committed pages and return them to the reserved state. |
| Committed | Memory charges have been allocated from the overall size of RAM and paging files on disk. The page is accessible and access is controlled by one of the [memory protection constants](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366786(v=vs.85).aspx). The system initializes and loads each committed page into physical memory only during the first attempt to read or write to that page. When the process terminates, the system releases the storage for committed pages.  A process can use **[VirtualAlloc](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366887(v=vs.85).aspx)** or **[VirtualAllocEx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890(v=vs.85).aspx)** to commit physical pages from a reserved region. They can also simultaneously reserve and commit pages.  The **[GlobalAlloc](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366574(v=vs.85).aspx)** and **[LocalAlloc](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366723(v=vs.85).aspx)** functions allocate committed pages with read/write access. |